Apple Pay Merchant Integration Guide

March 2026

Contents

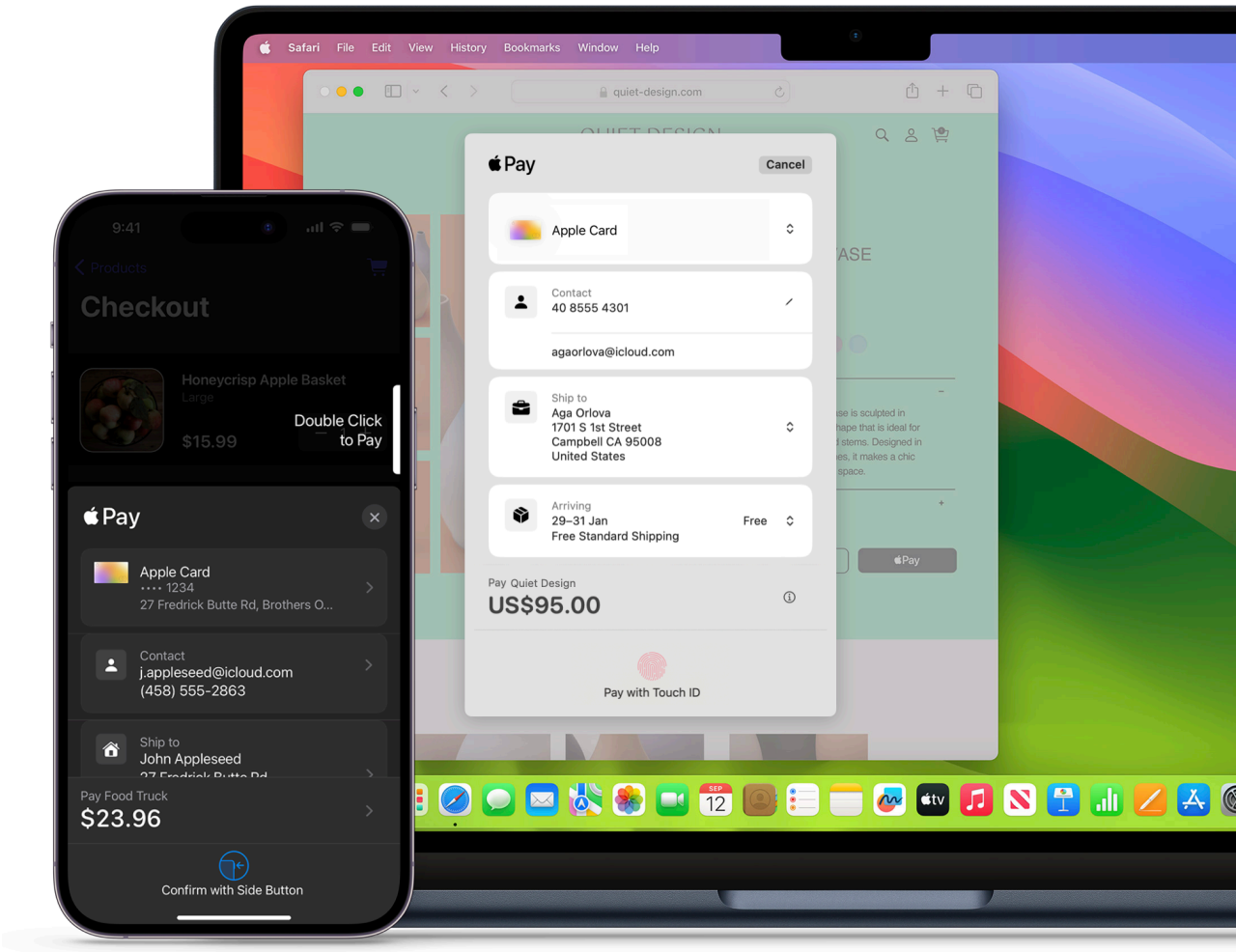
Introduction

Apple Pay provides an easy, secure and private way to make payments in iOS, iPadOS, macOS, visionOS and watchOS apps, and on the web when using compatible browsers. By using Face ID, Touch ID, Optic ID or double-clicking Apple Watch, users can quickly and securely provide their payment, shipping, and contact information to check out.

This guide outlines the steps needed to enable Apple Pay in app and on the web. To experience an Apple Pay test transaction on a compatible device, visit the Apple Pay demo site at applepaydemo.apple.com.

Integrating Apple Pay is often a cross-functional effort involving designers, developers, server administrators and payment teams within an organization. This guide supports each of these roles through the end-to-end process of enabling Apple Pay in your app or on the web, and the typical role responsible for each task will be noted beside the task checkbox.

ACCOUNT ADMIN UX DESIGNER SERVER ADMIN PAYMENTS TEAM DEVELOPER



Getting started

Before enabling Apple Pay, it is important that developers understand how Apple Pay differs from an In-App purchase, and make sure their implementation follows the guidelines. There are many ways to implement Apple Pay, with some of the most popular Payment Service Providers and E-Commerce platforms offering an Apple Pay SDK or JavaScript API as a quick and reliable way to support Apple Pay in an app or on a website.

Guidelines

Apple Pay vs In-App Purchases

Apple Pay can be used in your app to sell physical goods like groceries, clothing, and appliances; for services such as club memberships, hotel reservations, and events tickets; and for donations. In-App Purchase are used to sell virtual goods such as premium content for your app, and subscriptions for digital content.

App Review Guidelines

Before submitting your app for review, make sure it follows these guidelines to help progress smoothly through the review process.

App Review guidelines: Payments

Apple Pay on the Web Acceptable Use Guidelines

Before deploying Apple Pay on your website, make sure your implementation follows these guidelines.

Apple Pay on the Web Acceptable Use Guidelines

E-Commerce Platforms and Payment Service Providers

The most popular e-commerce platforms and payment service providers support Apple Pay within apps and on the web. Using an Apple Pay SDK or JavaScript API from a payment provider is the quickest and easiest way to support Apple Pay in your app or on your website.

View E-Commerce platforms and Payment Service Providers

If using an e-commerce platform or payment service provider's own SDK or API, contact them to get started. The process to integrate using an SDK or API they provide may differ from the steps outlined in the rest of this guide. Refer to their documentation to understand the process required.

Understanding Apple Pay

Apple Pay is available on all Apple devices with a Secure Element — an industry-standard, certified chip designed to store payment information safely. On macOS, users must have a Mac with Touch ID or an Apple Pay-capable iPhone or Apple Watch to authorize the payment. On third party browsers, whether on Mac or other devices, the user must have an Apple Pay-capable iPhone running iOS 18 or newer to authorize the transaction. Any transaction type you currently support for regular debit and credit cards can be performed with Apple Pay, including refunds.

Payment flow

For one-time transactions, Apple Pay uses device-specific tokenized credit or debit card credentials (DPAN) in place of a Payment Account Number (PAN). When users authenticate the payment using their biometric data or passcode, the tokenized card data is returned to your app or website. This token can then be passed to your Payment Service Provider (PSP) to process as you would for a typical online credit or debit card payment.

Customer

Authenticates with biometrics or passcode. Payment Data is encrypted and returned to the app/website

Merchant App/Website

Sends the Apple Pay payment object to merchant server

Merchant Server

Receives payment object and maps data to PSP API or SDK

Payment Service Provider

Decrypts Apple Pay payment object and formats a 3D Secure authorization message

Acquirer

Sends payment for authorization

Payment Network

De-tokenizes payment data and sends PAN to issuer for authorization

Issuer

Authorizes payment

Get set up for Apple Pay

To enable Apple Pay on your app or website, you need to confirm that you have the correct options configured on your server, are set up to accept Apple Pay Payments and register for an Apple Developer account.

PAYMENTS TEAM

Confirm your Payment Service Provider (PSP) supports Apple Pay

Check the list of currently supported gateways at the link below. If you do not see your PSP on the list, contact them directly to confirm availability.

Supported E-Commerce platforms and Payment Service Providers

SERVER ADMIN

Set up your Server

For Apple Pay on web only, ensure that your server meets the set up requirements for secure communications with Apple Pay.

Setting up your server

ACCOUNT ADMIN

Enroll in the Apple Developer Program

Both Apple Pay in apps and Apple Pay on the web require an Apple Developer program membership, which must be renewed yearly. You can enroll as an individual or as an organization, and you can use the same Apple developer account that you use today to publish apps to the App Store.

You cannot use an Apple Developer Enterprise Program account to enable Apple Pay within an app or on the web.

Enroll for an Apple Developer account

Design your Apple Pay solution

Apple Pay creates a streamlined checkout process, allowing customers to authorize payments and complete transactions promptly. Consider where and when in your customers journey would be best to utilize Apple Pay to help drive conversion and enhance the customer experience.

UX DESIGNER
PAYMENTS TEAM

Assess your Checkout flow

To help drive conversion and enhance your user's experience, carefully consider the location of the Apple Pay button. The best user experiences place the Apple Pay button as early in the checkout process as possible in order to leverage Apple Pay provided information, and minimize data entry.

UX DESIGNER

Review the Human Interface Guidelines

Refer to the Apple Pay Human Interface Guidelines for additional information on how to best incorporate Apple Pay in your app or website.

Apple Pay Human Interface Guidelines

UX DESIGNER

Design your Apple Pay Experience

There are several key Apple Pay design principles that help drive conversion, increase usage and engagement, and provide an excellent user experience. Please review our best practice recommendations in Planning for Apple Pay.

Planning for Apple Pay

Design your Apple Pay experience quickly and accurately by using the Apple Pay Design Template, available for for both Figma and Sketch. This can be found in the Technologies section of the Apple Design Resources page below.

Apple Design Resources

Build your Apple Pay solution

Now that you have designed your solution for Apple Pay, it's time to start building your implementation into your app or website. This section will link to relevant sources within the API documentation, as well as provide additional detail to support you in the process. These steps will differ if you are integrating through your PSP or E-commerce platform, so you should refer to their documentation if that is how you have chosen to integrate.

Configure Apple Pay

To support Apple Pay on your website or app, you need to complete a few set up steps in your developer account: registering a merchant ID, creating certificates, and verifying your web domain(s). Completing the setup enables you to use the Apple Pay web APIs and/or App APIs.

Configure Apple Pay for Apps

Configure Apple Pay on the Web

ACCOUNT ADMIN

Create a Merchant ID

Your Merchant ID uniquely identifies your business as a merchant able to take payments. A merchant Identifier never expires, and can be used for multiple Apps and websites. Follow the steps below when signed in to the Apple Developer Account as an account holder or admin role.

1. In Certificates, Identifiers & Profiles, click Identifiers in the sidebar, then click the add button (+) on the top left.
2. Select Merchant IDs, then click Continue.
3. Enter the merchant description and identifier name, then click Continue.
4. Review the settings, then click Register.

Alternatively, you can create a merchant identifier in Xcode.

ACCOUNT ADMIN

Create a Payment Processing Certificate

A Payment Processing Certificate is associated with your merchant ID and used to secure transaction data. To generate a certificate you will need to upload a Certificate Signing Request (CSR) to the Apple Developer portal. If your PSP is decrypting the data, contact them to obtain a properly formatted CSR to upload and create your certificate. If you plan to decrypt the Apple Pay data yourself, you will need to generate your own CSR using an ECC and 256 bit key pair, unless creating certificates for mainland China in which case you will need to generate RSA 2048 key pairs.

1. In Certificates, Identifiers & Profiles, click Identifiers in the sidebar.
2. Under Identifiers, select Merchant IDs using the filter on the top right.
3. On the right, select your merchant identifier.
4. Under Apple Pay Payment Processing Certificate, click Create Certificate.
5. Create a certificate signing request .
6. Click Choose File.
7. In the dialog that appears, select the certificate request file (a file with a `.certSigningRequest` file extension), then click Choose.
8. Click Continue.
9. Click Download. The certificate file (a file with a `.cer` file extension) appears in your Downloads folder.

If your PSP is decrypting the payment data, they may require this certificate file. Follow the guidance they provide on where and how to share this with them.

The Payment Processing Certificate is valid for a period of 25 months. Before it expires, you'll need to create a new certificate by following the same steps as outlined above. This prepares your replacement certificate but doesn't activate it immediately. Work with your PSP to choose the best time to switch over to the renewed certificate, and when you are both ready, press the “activate” button next to the certificate in the Apple Developer Portal. This tells the Apple Pay servers to begin using the new certificate for encryption.

It’s important to complete this process before your current certificate expires to avoid any interruption to your Apple Pay transactions. Most payment processors can guide you through this step-by-step.

Configure Apple Pay on the Web

If you're developing websites using Apple Pay on the Web, you can use the same merchant ID and Payment Processing Certificate for your website and your app. However, Apple Pay on the web requires additional setup. As a reminder, if you are being onboarded by your Payment Service Provider or E-Commerce platform, they may handle this on your behalf, and you should contact them to confirm wether you need to create your own credentials.

Configure Apple Pay on the Web

ACCOUNT ADMIN ☐ **Create a Merchant ID**

Your Merchant ID uniquely identifies your business as a merchant able to take payments. Follow the steps in the previous section to create a Merchant ID.

ACCOUNT ADMIN ☐ **Create a Payment Processing Certificate**

A certificate associated with your merchant ID, used to encrypt Apple Pay transaction data. Follow steps in previous section to create a Payment Processing Certificate.

ACCOUNT ADMIN ☐ **Verify your domain(s)**

You must register and verify all top-level domains and subdomains where you display the Apple Pay button.

1. In Certificates, Identifiers & Profiles, click Identifiers in the sidebar, then select Merchant IDs from the pop-up menu on the top right.
2. On the right, select your merchant identifier.
3. Under Merchant Domains, click Add Domain.
4. Enter the fully qualified domain name, then click Save.
5. Click Download, place the downloaded file in the specified location, then click Verify.
6. Click Done.

For successful domain validation, domains can't be behind a proxy or redirect, and must be accessible to the Apple servers listed in setting up your server.

ACCOUNT ADMIN ☐ **Create a Merchant Identity Certificate**

A certificate associated with your merchant ID, used to authenticate sessions with the Apple Pay servers. The steps to create this certificate are similar to those for the Payment Processing Certificate, but you will need to generate your own CSR using an RSA 2048 bit key pair.

1. In Certificates, Identifiers & Profiles, click Identifiers in the sidebar, then select Merchant IDs from the pop-up menu on the top right.
2. On the right, select your merchant identifier.
3. Under Apple Pay Merchant Identity Certificate, click Create Certificate.
4. Create a certificate signing request.
5. Click Choose File.
6. In the dialog that appears, select the certificate request file (a file with a `.certSigningRequest` file extension), then click Choose.
7. Click Continue.
8. Click Download. The certificate file (a file with a `.cer` file extension) appears in your Downloads folder.
9. Double-click the Merchant Identity Certificate you just downloaded to add it to your Mac Keychain.

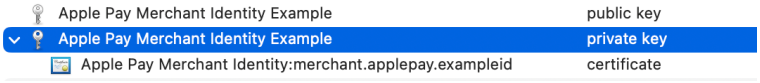
ACCOUNT ADMIN

DEVELOPER

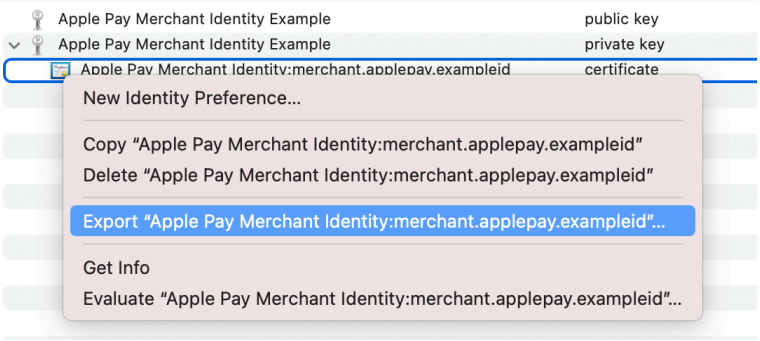
Export your Merchant Identity Certificate

You should now have a private key and corresponding certificate. If you generated the private key in the Keychain App on Mac, the Merchant Identity Certificate and key will need to be exported and packaged into a .p12 file before you are able to use it to generate a session with the Apple Pay servers. Follow the steps below to export and then test your certificate.

1. Open the Keychain app.
2. Click on 'login' from the sidebar, 'Keys' from the menu and search for the key you generated.
3. Click on the disclosure arrow beside the key, which should display your Merchant Identity Certificate.



4. Right-click on the certificate and click Export *<name of certificate>*.



5. Choose a directory to save the certificates, and enter a password to protect the file.

Now you have a .p12 file that contains the key and certificate for Merchant Validation. Open Terminal and use the following commands to split the .p12 into the corresponding key and certificate, where `ApplePayMerchantID_and_privatekey.p12` is the name of your .p12 file:

```
openssl pkcs12 -in ApplePayMerchantID_and_privatekey.p12 -out
ApplePay.crt.pem -nokeys
```

```
openssl pkcs12 -in ApplePayMerchantID_and_privatekey.p12 -out
ApplePay.key.pem -nocerts
```

ACCOUNT ADMIN

DEVELOPER

Test your Merchant Identity Certificate

To test that your Apple Pay Merchant Identity certificate is working correctly, use the following cURL command in terminal to post the certificates and JSON up to Apple servers. Ensure you change the `merchantIdentifier` and `initiativeContext` to the merchant identifier matching your certificate, and a registered and verified domain associated to that merchant identifier. Make sure you are using straight double quotes (") and not curly double quotes (“) to ensure the request can be parsed correctly.

```
curl --location 'https://apple-pay-gateway-cert.apple.com/paymentservices/
paymentSession' \
--header 'Content-Type: text/plain' \
--data '{
    "merchantIdentifier": "merchant.XXXXXX",
    "displayName": "Example Merchant",
    "initiative": "web",
    "initiativeContext": "www.example.com"
}' \
--cert ApplePay.crt.pem \
--key ApplePay.key.pem
```

If the connection succeeds and the session validated, you should see a print out in Terminal similar to the following:

```
{“epochTimestamp”:15445664606792,”expiresAt”:154167344466792,”
merchantSessionIdentifier”:”
SSHC45CBB9C8073415198E3AB7314791C6D_916523AAED1343F5BC5815E12BEE9250AFFDC1A1
7C46B0DE5A943F0F94927C24”,”nonce”:”
8f47a9c1”,"merchantIdentifier":"20C8BB48576962AD936399118741FFEA6130838BDF69
A50B9A3CE4E7","domainName":"www.example.com","displayName":"Example
Merchant”,"signature":"308006092a864886f70d010702a0803080020101310f300d06096
086480165030402010500308006092a864886f70d0107010000a080308203e63082038ba0030
2010202086860f699d9cca70f300a06082a8648ce3d040302307a312e302c06035504030c254
170706c8274854054658569d4170706c65204365727469666963617469 6f6e20417574686f72
697479311330110603550 40a0c0a4170706c6520496e632e310b3009060355040613025553301e170d313630363033313831363430 5a170d32313036303231383.......}
```

This is the *opaque session object*, and is used to start a session for Apple Pay on the Web. It is important that this object is not inspected, parsed or modified in any way. Apple may update the contents of this object from time to time with changes and enhancements.

A general guide to debugging common issues can be found in the troubleshooting section. If your issue is not resolved after reviewing the guide, you can get further assistance by contacting Apple Developer Support directly.

Apple Pay Developer Support

Present Apple Pay as a payment option

With your configuration complete, you can now move on to add an Apple Pay button to your app or website. There are several Apple Pay button types and styles you can use, and it is important that you do not create your own Apple Pay button design or attempt to mimic the system-provided button designs.

DEVELOPER

Display the Apple Pay button

The button will be rendered by the appropriate PassKit or Javascript API, which will display the most up to date version of the button, perform the appropriate localizations and have a system-provided alternative text label that lets VoiceOver describe the button

Display the Apple Pay button - Apple Pay on web

Display the Apple Pay button - PassKit

DEVELOPER

Apple Pay Mark

Use the Apple Pay mark in your app or website to show that Apple Pay is an available payment option. The Apple Pay mark isn't a button and shouldn't be used to launch the payment sheet. Use only the artwork Apple provides, with no alterations other than height.

Apple Pay mark and Marketing Guidelines

DEVELOPER

Check for Apple Pay Availability

To ensure that you only display the Apple Pay button to customers with a supported device, check for Apple Pay availability.

Checking for Apple Pay availability

`canMakePayments()`
Verifies that the device is capable of making Apple Pay payments; it doesn't verify that the user has a provisioned card for use with Apple Pay on the device. You can call this method at any time.

Web - canMakePayments()

App - canMakePayments()

`canMakePayments(usingNetworks:)` App only
Verifies that the device is capable of making Apple Pay payments and that there is a payment card from at least one of the provided networks available.

App - canMakePayments(usingNetworks:)

`applePayCapabilites()` Web only
Verifies on Safari and third-party browsers that the device is capable of making Apple Pay payments. On iOS devices or when using Safari on Mac, it also verifies whether the device has at least one payment card available. Call this method if you want to default to Apple Pay during your checkout flow, or if you want to add an Apple Pay button to your product detail page.

Web - applePayCapabilities()

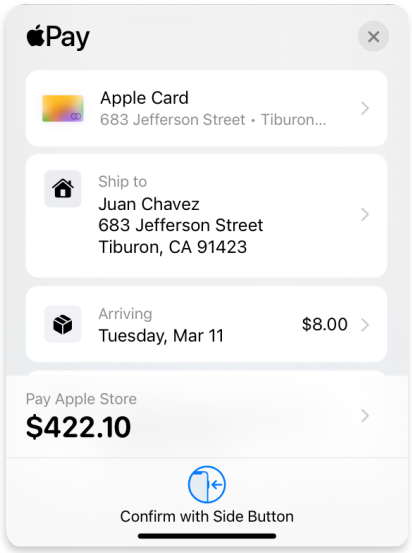
Present the Payment Sheet

To present the payment sheet to the customer, construct a payment request containing information that describes the purchase. This includes merchant information, supported payment networks, line items including the total, currency code, billing and shipping contact, and more.

DEVELOPER

Construct your Payment request

Your app or website specifies what the payment sheet displays, but it doesn't control the users interaction with the sheet. You must decide if it makes sense to present shipping and billing information, shipping methods, and other line items to the user. For the best results, only request the information necessary to process and service the transaction.



DEVELOPER

Complete Merchant Validation Web only

Once the system launches the payment sheet, the merchant validation process is automatically triggered and a call to `onmerchantvalidation` is made. In order for merchant validation to be successful, you'll need to request a merchant session object from the Apple Pay servers.

Inside the `onmerchantvalidation` event handler, make a call to an endpoint on your server asking for a new session to be created. For security purposes the request to the Apple Pay servers needs to come from your server and not from the browser directly. Your server will then need to post your request data alongside the Merchant Identity Certificate to obtain the session object.

```
POST https://apple-pay-gateway.apple.com/paymentservices/paymentSession

{
    merchantIdentifier: "merchant.com.example.mystore",
    displayName: "MyStore",
    initiative: "web",
    initiativeContext: "mystore.example.com"
}
```

Pass the session object into the `complete` method from the event to finish the Merchant Validation process and allow the customer to authenticate the transaction.

When testing with sandbox cards, you'll make the call to a different endpoint. Be sure to make the call to the production endpoint before pushing to your live environment.

DEVELOPER

Shipping and Billing

In the payment request, you can specify the required customer information by setting the relevant `boolean` values in the `paymentOptions` dictionary. Required data can include billing address, shipping address, name, email address, and phone number.

Setting `requestShipping` to `true` presents redacted address information in a callback event prior to the user authenticating the transaction, similar to the example shown below.

```
{
    country: “US”,
    addressLine: [],
    region: “NC”,
    city: “Raleigh”,
    dependentLocality: “”,
    postalCode: “27601”,
    sortingCode: “”,
    recipient: “”,
    phone: “”
}
```

Apple Pay provides the information in a redacted form to protect a person’s privacy, and can vary based on geographic location – in the UK for example, only the first part of the postal code is returned prior to authentication. Use this information to provide relevant shipping options within the `paymentDetails` array, or to calculate any taxes due on the transaction and then update the payment sheet accordingly.

Once a person authenticates the transaction with their biometric data or passcode, you will receive the complete set of requested contact information.

```
{
    country: "US",
    addressLine: ["2399 Elm St"],
    region: "NC",
    city: "Raleigh",
    dependentLocality: "",
    postalCode: "27601",
    sortingCode: "",
    organization: "",
    recipient: "Allison Cain",
    phone: ""
}
```

DEVELOPER

Respond to Payment sheet interactions

When the user interacts with the information they are providing in the payment sheet, events are triggered with relevant information that allows you to respond to the changes. For example, if a customer changes their selected address in the payment sheet, you can update the shipping methods or shipping costs presented.

Web - request.onshippingaddresschange

App - PKPaymentRequestShippingContactUpdate

To display the coupon code field within the payment sheet, add `supportsCouponCode` to your payment request with an empty `couponCode` field if there is no initial coupon code. If a customer enters a coupon code within this field, respond to that event by adding an event listener to the `shippingaddresschange` event and checking the `event.methodDetails.couponCode` property.

DEVELOPER

UX DESIGNER

Error handling

When you determine that there's a problem with an address or contact information on the payment sheet, you can use `ApplePayError` to create a customized error message. Apple Pay highlights the area with an error and displays your message, making it easier for users to correct errors.

Web - ApplePayError

App - PKPaymentError

The following sample code demonstrates Error Handling for a shipping contact

```
const paymentResponse = await paymentRequest.show();
    switch (paymentResponse.methodName) {
        case "https://apple.com/apple-pay":
            if (paymentResponse.details.shippingContact) {
                const errors = validateAddress(paymentResponse.details);
                if (errors.length) {
                    await paymentResponse.retry({
                        paymentMethod: errors,
                    });
                }
            }
            // Handle authorization with valid details here.
            break;

        case "https://...":
            // Other payment method changes here.
            break;
    }
}

function validateAddress({shippingContact}) {
    const errors = [];
    // Validate shipping ZIP code here.
    if (!isValidZipCode(shippingContact.postalCode)) {
        const error = new ApplePayError(
            "shippingContactInvalid",
            "postalCode",
            `ZIP code doesn't match city`
        )
        errors.push(error);
    }
    //validate other parts of address
    return errors;
}
```

The following example demonstrates an implementation of Error Handling for a coupon code

```
paymentRequest.addEventListener("shippingaddresschange", (event) => {
    switch (event.methodName) {
        case "https://apple.com/apple-pay":
            if (event.methodDetails.couponCode) {
                event.updateWith(
                    validCouponCode(event.methodDetails.couponCode)
                );
                return;
            }

            if (event.methodDetails.type) {
                // Handle payment method type changes here.
            }
            break;

        case "https://...":
            // Other payment method changes here.
            break;
    }
});

async function validCouponCode(couponCode) {
    const paymentMethodErrors = [];
    // Validate coupon code here against server.
    if (!(await isValidCouponCode(couponCode))) {
        paymentMethodErrors.push(
            new ApplePayError(
                "couponCodeInvalid",
                undefined,
                `Coupon code "${couponCode}" is invalid`
            )
        );
    }
    return { paymentMethodErrors };
}
```

DEVELOPER

Multi-token payments

Multi token transactions such as a booking site where a user pays for a hotel, flight, and car rental from different merchants, allow you to process and display payment requests with multiple merchants on one payment sheet.

Web - Multi Token Payment Request Modifer

App - PKPaymentTokenContext

DEVELOPER

Disbursement requests

Allow users to transfer funds from your site to an account that is linked to a card in Wallet. The Disbursement Requests are only available as a subset of the Payment Request API.

Web - Disbursement Request Modifer Payment Request API only

App - PKDisbursementRequest

DEVELOPER

Recurring payments and Merchant Tokens

Apple Pay supports the use of merchant tokens (MPANs) to complete a variety of supported payments independent of a device. After the initial transaction, payment details can be stored to create follow-on transactions. A merchant token associates a payment card, merchant, and user, allowing for continuity across multiple devices, even when a device is upgraded or a card is removed. When a payment request is created for a recurring, automatic reload, or a deferred payment, you can provide essential information the customer directly in the payment sheet, potentially helping to decrease cart abandonment and increase conversion rates. Map each payment model you offer to the relevant Apple Pay payment request type.

`recurringPaymentRequest`

Subscriptions such as digital content, streaming services, or membership programs, where the customer is charged the same transaction amount at fixed regular intervals.

Recurring billing such as a utility bill, with a variable transaction amount at fixed regular intervals.

Installment loan repayments such as a buy now, pay later loan, where there are a set number of fixed, regular intervals for a single purchase of goods or services.

Web - Recurring Payment Request Modifer

App - PKRecurringPaymentRequest

`automaticReloadPaymentRequest`

Automatic reloads such as balance top up for transit or stored balance cards, allow you to add funds to a specific account automatically, triggered by a pre-determined threshold.

Web - Automatic Reload Payment Request Modifer

App - PKAutomaticReloadPaymentRequest

`deferredPaymentRequest`

Deferred payments, such as car hire or hotel bookings with incidental charges, allow you to specify a free cancellation period and the date when payment will be taken.

Web - Deferred Payment Request Modifer

App - PKDeferredPaymentRequest

When any of these request types are used to initiate an Apple Pay transaction, a merchant token is automatically requested by Apple, and where supported by the issuer, returned in the Apple Pay payload. If an issuer does not yet support merchant tokens, Apple Pay defaults back to returning the DPAN.

Merchant Token Management API

Merchant tokens are also designed to help prevent or resolve billing issues by giving visibility into important payment lifecycle updates, such as account status, bank card art, and card expiration date. The Apple Pay Merchant Token Management API allows you retrieve and manage these lifecycle updates.

When your app or website creates a payment request using one of the supported request types, it can pass a notification URL in the `tokenNotificationURL` parameter. If a life-cycle event affects the token, Apple Pay sends a notification with an event identifier to that `tokenNotificationURL`. The details of the event can be retrieved by requesting the details from the Apple Pay server with the event identifier.

Receiving and Handling Merchant Token Notifications

When a life-cycle event occurs to a card associated with a merchant token, Apple Pay sends a **GET** request to the endpoint included in the `tokenNotificationURL` parameter of the original payment request.

```
GET https://merchant.example.com/tokenapi/notification/merchantToken/{eventId}
```

When you receive this request, use the `eventId` included in the path to format a **POST** request to retrieve the details of the event. This will include both the Merchant Identifier from the original transaction, alongside the specific eventId that has been triggered.

```
POST https://apple-pay-gateway.apple.com/paymentServices/v1/merchantId/
{merchantId}/merchantToken/event/{eventId}
```

Receiving and handling merchant token notifications

A `merchantTokenEventResponse` object containing the details of the event that triggered the notification will be returned. Consider how to handle different lifecycle events to proactively resolve payments issues. For example, when a consumer revokes the payment authorization on a token and you receive the `UNLINK` event, trigger a notification to the customer to update their payment details before the next billing cycle commences.

```
{
  statusCode: 200,
  merchantTokenEvent: {
    merchantTokenIdentifier: 'MAPL...c5c2',
    eventType: 'UNLINK',
    reason: 'UNLINKDEVICE'
  }
}
```

merchantTokenEventResponse

Authorize payment and map customer information

When a customer authenticates a transaction, the device will create a payment object that contains a payment token with the encrypted payment data, alongside any additional contact data requested from the customer. The information provided by the customer is not validated or verified by Apple.

PAYMENTS TEAM

Validate and Map Customer Information

Once you have validated customer information, map the values provided in Billing and Shipping contact fields to your typical order fulfillment systems.

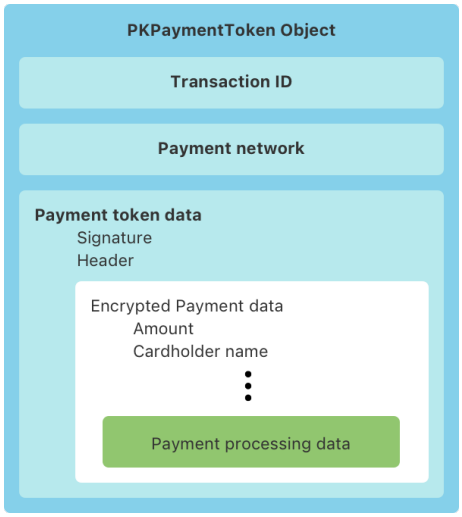

```
{
  "billingContact": {
    "addressLines": [
      "1 First Street"
    ],
    "administrativeArea": "London",
    "country": "United Kingdom",
    "countryCode": "GB",
    "familyName": "Appleseed",
    "givenName": "John",
    "locality": "London",
    "phoneticFamilyName": "",
    "phoneticGivenName": "",
    "postalCode": "AB12 3CD",
    "subAdministrativeArea": "",
    "subLocality": ""
  },
  "shippingContact": {
    "addressLines": [
      "1 First Street"
    ],
    "administrativeArea": "London",
    "country": "United Kingdom",
    "countryCode": "GB",
    "emailAddress": "john.appleseed@apple.com",
    "familyName": "Appleseed",
    "givenName": "John",
    "locality": "London",
    "phoneticFamilyName": "",
    "phoneticGivenName": "",
    "postalCode": "AB12 3CD",
    "subAdministrativeArea": "",
    "subLocality": ""
  },
  "token": {
    "paymentData": {
      "data": "K+kSR...BbAj",
      "signature": "MIAG...AAA=",
      "header": {
        "publicKeyHash": "2xR1...PGQ=",
        "ephemeralPublicKey": "MFkw...1w==",
        "transactionId": "ba9c...7298"
      },
      "version": "EC_v1"
    },
    "paymentMethod": {
      "displayName": "NetworkName 1234",
      "network": "NetworkName",
      "type": "credit"
    },
    "transactionIdentifier": "ba9c...7298"
  }
}
```

DEVELOPER

PAYMENTS TEAM

Authorize Payment with your PSP

Apple Pay passes payment information inside a `PKPaymentToken` object (app) or an `ApplePayPaymentToken` object (web). The payment object has a nested structure that contains a payment token with encrypted payment data, as shown in the figure below.



Depending on your PSP, you can pass the encrypted payment data directly to their systems for authorization and capture. Alternatively, the payment data can be decrypted by you and then sent to your PSP for processing For more information on decrypting the payment data yourself, refer to the documentation at the link below.

Decrypting the Apple Pay Payment Token

DEVELOPER

PAYMENTS TEAM

Complete payment

Once you have passed the data to your PSP and received their response, pass the success or failure back into the Apple Pay APIs. This will inform the user if the payment was successful, and dismisses the payment sheet.

Web - complete

App - PKPaymentAuthorizationResult

Test your Apple Pay integration

Apple provides sandbox accounts and Apple Pay Cards that you can use to run tests in the Apple Pay Sandbox environment. Attempting to use these cards with a live production environment will result in your PSP rejecting the transaction. More information on how to set up the sandbox accounts and cards can be found on the developer website linked below.

Sandbox Testing

It is also important to test Apple Pay in your production environment using real cards to ensure the end-to-end transaction flow is working as expected.

Participating Banks

DEVELOPER
PAYMENTS TEAM

- [ ] **Test your integration on multiple devices and browsers**

Apple Pay is now available on Safari, third party browsers in iOS and MacOS, as well as on other computing platforms. Check that the Apple Pay button is displayed on all the relevant devices and locations, and that you are able to see the Apple Pay button on third party browsers and devices.

DEVELOPER
PAYMENTS TEAM

- [ ] **Data Mapping and formatting**

Confirm that you are able to map all of the relevant data included in the Apple Pay payload, including the shipping and billing contact fields to your order management services. Ensure you can handle this data in the various possible formats. For example, phone numbers may contain numeric values as well as special characters such as “+”.

DEVELOPER
PAYMENTS TEAM

- [ ] **Responding to Events**

When you are supporting handling of Payment Sheet event, such as when the shipping contact or method is updated, ensure you have tested your event handling process and that you are formatting and responding using the relevant completion method.

DEVELOPER
PAYMENTS TEAM

- [ ] **Test all payment flows**

Test that Apple Pay functions as expected across all of the payment flows you are offering.

Good practice guidance when processing Apple Pay transactions

Apple Pay offers multiple layers of security beyond many other payment methods. However, you can apply the same industry best practices for risk mitigation to Apple Pay as you do for other E-commerce payment methods.

Apple Pay uses tokenised credentials (such as a device-specific DPAN or a merchant-specific MPAN) so actual card numbers (PANs) cannot be used as a unique key and each transaction generates a unique cryptogram in place of the CV2. There are many standard checks that are still applicable to Apple Pay. If you do not have an in-house Risk Management tool, please consider reaching out to your Payment Service Provider (PSP) who should be able to help implementing the below Risk Checks.

Velocity and Pattern Checks

A velocity check monitors specific data elements to check and compare a customer's shopping history with current purchases in order to identify any irregularities in their purchasing behavior. This includes sudden bursts of high activity considered inconsistent with standard purchase behaviors for your industry or customer base.

Consider using purchase frequency, repeat billing/delivery address for high value purchases, cardholder name, email, name and IP address - particularly if combinations of these details are used in quick succession. Actions you could take based on a high risk transaction are:

- Decline the transaction.
- Delay shipping and complete outbound checks on address and customer details.

Delivery Checks

Consider implementing risk indicators for delivery methods such as the delivery address being different from the billing address, temporary addresses, instructions to leave goods on doorsteps (or similar), export address or requests for fast delivery. In countries where applicable, use Address Verification Services (AVS) to verify the billing address is correct. Consider looking at data provided by the user and check if it's linked to previous purchases (email addresses, phone numbers, billing and shipping addresses linked to other users). Multiple transactions could be correlated against another order.

Maintain a “Bad Transactions” List

Maintain a list of historical transactions for all payment methods which have either looked suspicious or resulted in chargebacks and compare the details of new transactions against these. If one or more of the transaction details match against historically bad transactions it may be worth putting the transaction aside for manual review or declining it outright. This data may also be available via your Payment Service Provider’s (PSP) Risk Management tool.

Email and IP address Checks

Ensure that email addresses are correct. You can use the Apple Pay Error Handling API's to ensure a customer adheres to your email rules. If the domain looks ok, send an email to the address to confirm it exists. Check the IP address coming from the device to verify the IP country matches the billing country and country where the card was issued.

Country Checks

If you identify certain countries as high risk, you can use the supportedCountries property in the Apple Pay payment request to limit payment cards to those that were issued in specific countries. The supportedCountries list does not affect the currency used for the transaction, and it applies to all payment cards in Wallet. You can also consider limiting IP addresses from high risk regions.

Frequently asked questions

Where does the customer information come from in the payment sheet?
The information comes from Wallet & Apple Pay defaults in Settings, if available, as well as the My Card in Contacts. It could also come from previous Apple Pay transactions. You can set up your My Card by going to Settings > Contacts > My Info.

Is the customer information that comes from Apple Pay verified by Apple?
Customer information is shared as-is, and is not verified by Apple. You will need to validate it on your platform and communicate through the Apple Pay API if fields should be corrected. For more information visit the Error Handling section of this guide.

What customer information can I pull from Apple Pay?
Customer information includes shipping and billing address, name, phone number and email address.

What does the Apple Pay payment token contain?
The structure, format, and data included in the payment token can be found in the Payment Token Format Reference. The payment token only contains information on processing the payment; the customer information is included in the payment object. The payment object encapsulates the customer information and the payment token.

Payment Token Format Reference

What do I do if my PSP is unable to decrypt my payment?
You may need to regenerate your payment processing certificates. Please follow the steps in this guide to generate your payment processing certificates and work with your PSP to understand where the issue may lie.

Can I use the same Merchant ID for multiple PSPs?
This is supported only if you are decrypting the payload yourself and passing on the payment information. Where the PSP's will be performing decryption, they should each have their own, unique merchant identifier to ensure the payments are encrypted with the correct sets of certificates and keys..

Will I get liability shift with Apple Pay?
Please check with your PSP. Depending on your specific agreement, you may get liability shift with certain card networks.

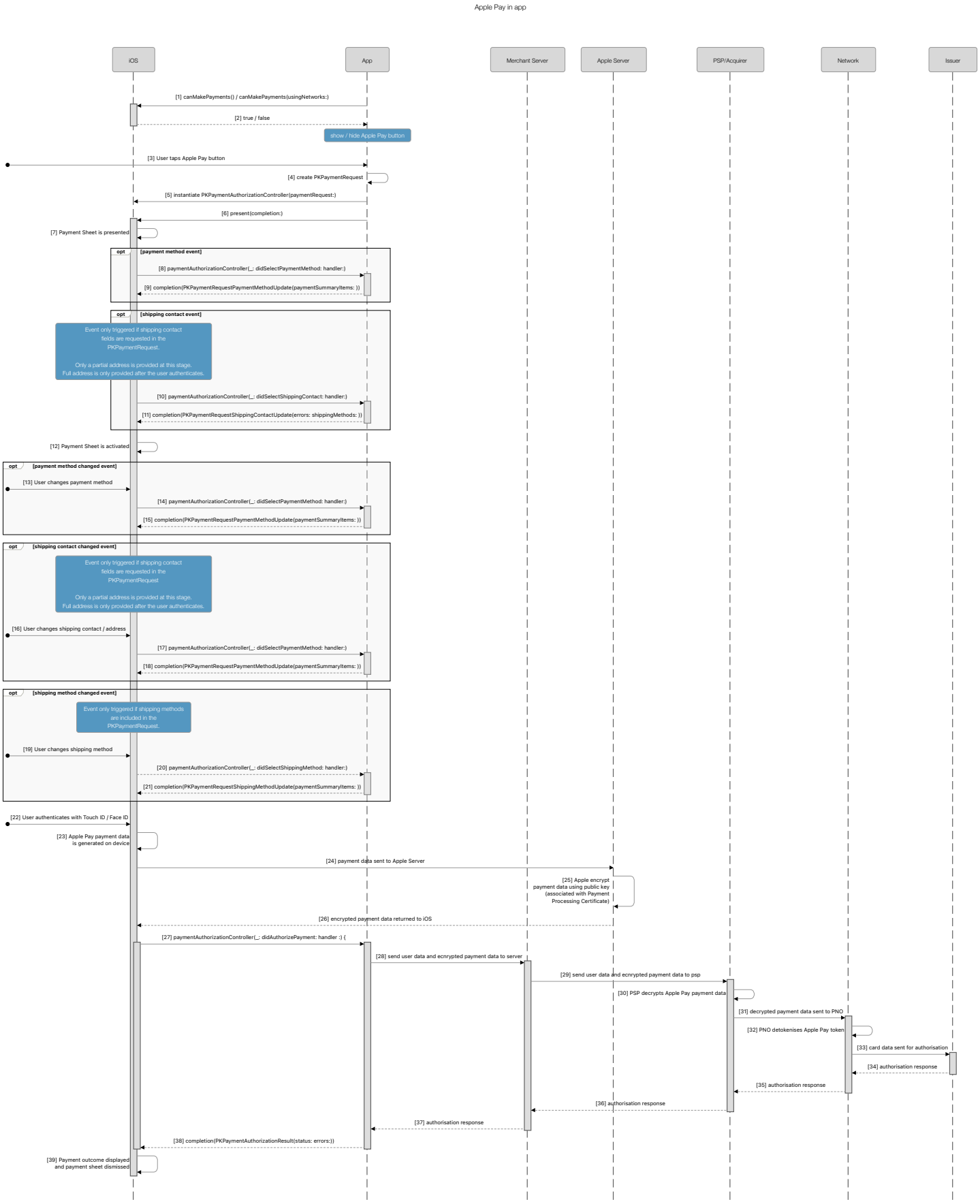
Can I use the same Apple developer account for all countries I process payments in?
Yes; you do not need to have separate Apple developer accounts for multiple markets. You can also use the same Apple Merchant ID if you wish

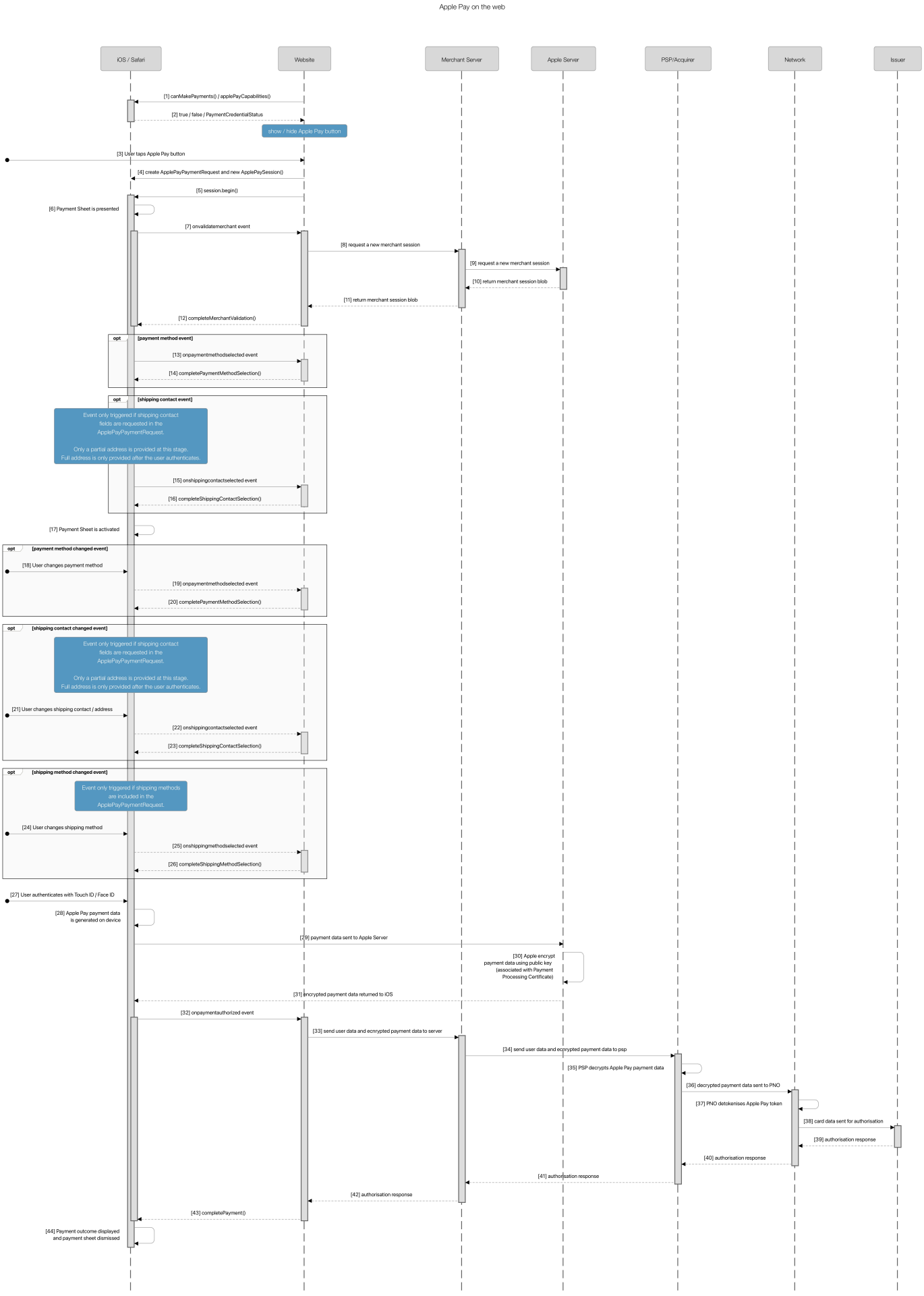
Troubleshooting

Further information and troubleshooting steps to debug common issues with Apple Pay can be found in the below guides.

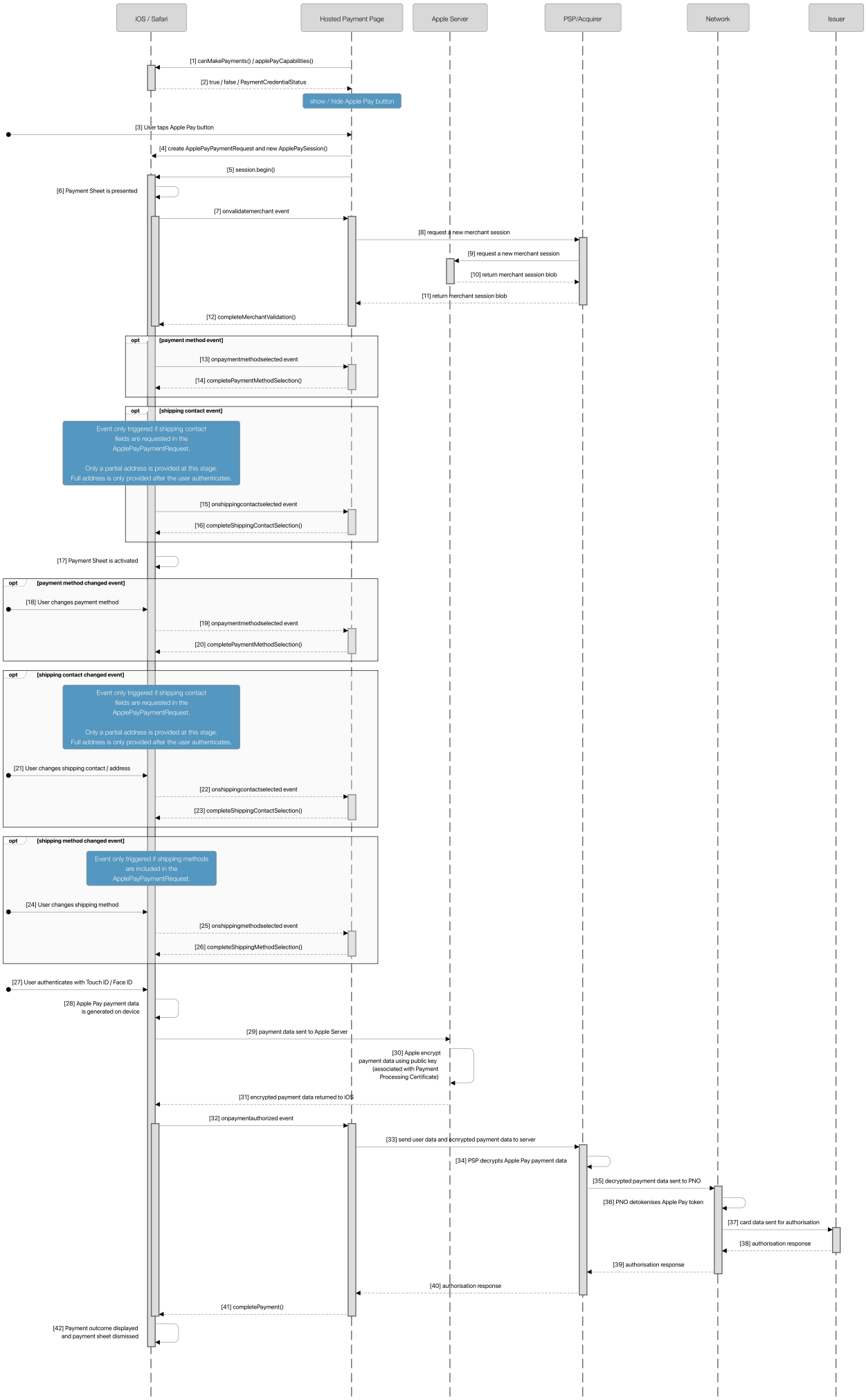
- Apple Pay on the Web Troubleshooting Guide
- Troubleshooting Apple Pay payment processing
- Diagnosing issues with displaying the Apple Pay button on your website
- Diagnosing issues with the Apple Pay payment sheet on your website
- Troubleshooting issues with your Apple Pay merchant identifier configuration

API Diagrams




Apple Pay on the web
iOS / Safari
Website
Merchant Server
Apple Server
PSP/Acquirer
Network
Issuer
[1] canMakePayments() / applePayCapabilities()
[2] true / false / PaymentCredentialStatus
show / hide Apple Pay button
[3] User taps Apple Pay button
[4] create ApplePayPaymentRequest and new ApplePaySession()
[5] session.begin()
[6] Payment Sheet is presented
[7] onvalidatemerchant event
[8] request a new merchant session
[9] request a new merchant session
[10] return merchant session blob
[11] return merchant session blob
[12] completeMerchantValidation()
opt
[payment method event]
[13] onpaymentmethodselected event
[14] completePaymentMethodSelection()
opt
[shipping contact event]
Event only triggered if shipping contact fields are requested in the ApplePayPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[15] onshippingcontactselected event
[16] completeShippingContactSelection()
[17] Payment Sheet is activated
opt
[payment method changed event]
[18] User changes payment method
[19] onpaymentmethodselected event
[20] completePaymentMethodSelection()
opt
[shipping contact changed event]
Event only triggered if shipping contact fields are requested in the ApplePayPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[21] User changes shipping contact / address
[22] onshippingcontactselected event
[23] completeShippingContactSelection()
opt
[shipping method changed event]
Event only triggered if shipping methods are included in the ApplePayPaymentRequest.
[24] User changes shipping method
[25] onshippingmethodselected event
[26] completeShippingMethodSelection()
[27] User authenticates with Touch ID / Face ID
[28] Apple Pay payment data is generated on device
[29] payment data sent to Apple Server
[30] Apple encrypt payment data using public key (associated with Payment Processing Certificate)
[31] encrypted payment data returned to iOS
[32] onpaymentauthorized event
[33] send user data and ecnrypted payment data to server
[34] send user data and ecnrypted payment data to psp
[35] PSP decrypts Apple Pay payment data
[36] decrypted payment data sent to PNO
[37] PNO detokenises Apple Pay token
[38] card data sent for authorisation
[39] authorisation response
[40] authorisation response
[41] authorisation response
[42] authorisation response
[43] completePayment()
[44] Payment outcome displayed and payment sheet dismissed

Apple Pay on the web:
PSP hosted payment page, merchant registered via API



Change Log

March 2026	Minor editorial updates
	`canMakePayments(usingNetworks:)` added
	Merchant token guidance added
	Merchant Token Management API section added
	Example Payment Object added for reference
	Testing recommendations added
Jun 2025	Guidance for `applePayCapabilites` API added
	Good practice when processing Apple Pay transactions guidance added