Apple Pay Programming Guide

Developer

Contents

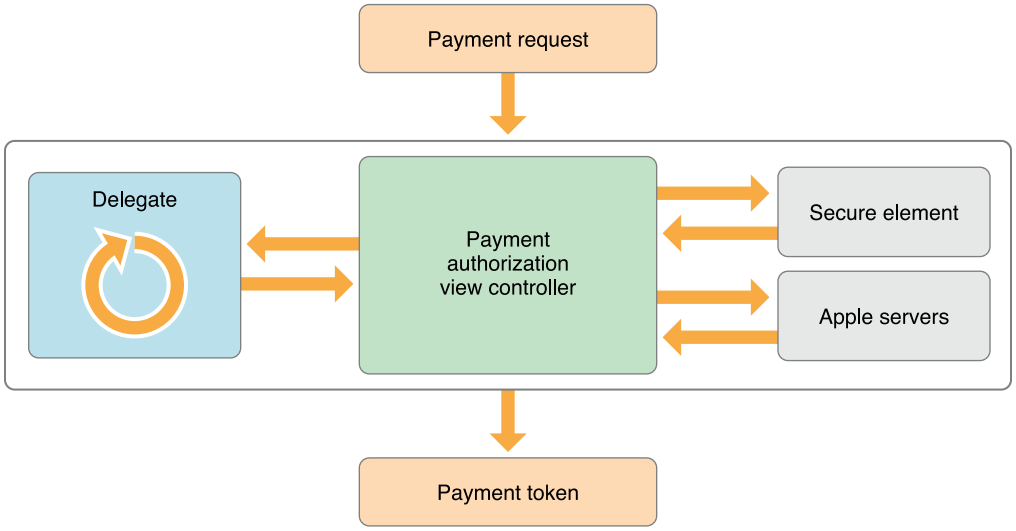
Figures and Listings

Objective-C

About Apple Pay

Apple Pay is a mobile payment technology that lets users give you their payment information for real-world goods and services in a way that is both convenient and secure.

For digital goods and services delivered within the app, see *In-App Purchase Programming Guide*.



You Configure Apple Pay in Xcode and Member Center

Apps that use Apple Pay need to enable the Apple Pay capabilities in Xcode. You also register a merchant identifier and set up cryptographic keys, which are used to securely send payment data to your server.

Relevant Chapter: Configuring Your Environment (page 7)

Users Authorize a Payment Request

A payment request describes the purchase being made, including the payment amount. You pass the payment request to a payment authorization view controller, which displays the request and prompts the user for information you need, such as a shipping and billing address. Your delegate is called to update the payment as the user interacts with the view controller and provides new information.

Relevant Chapters: Creating a Payment Request (page 9), Authorizing a Payment (page 15)

Your Server Processes Payments

Apple Pay encrypts payment information to prevent an unauthorized third party from obtaining the user's payment information. You can handle payments entirely on your server, or your server can use a third-party payment platform to decrypt and process the payment.

For information about payment platforms that support Apple Pay, see developer.apple.com/apple-pay/.

Relevant Chapter: Processing a Payment (page 19)

Configuring Your Environment

A merchant ID identifies you to Apple Pay as being able to accept payments. The public key and certificate associated with your merchant ID is used as part of the payment process to encrypt payment information. Before your app can use Apple Pay, you need to register a merchant ID and configure its certificate.

To register a merchant ID

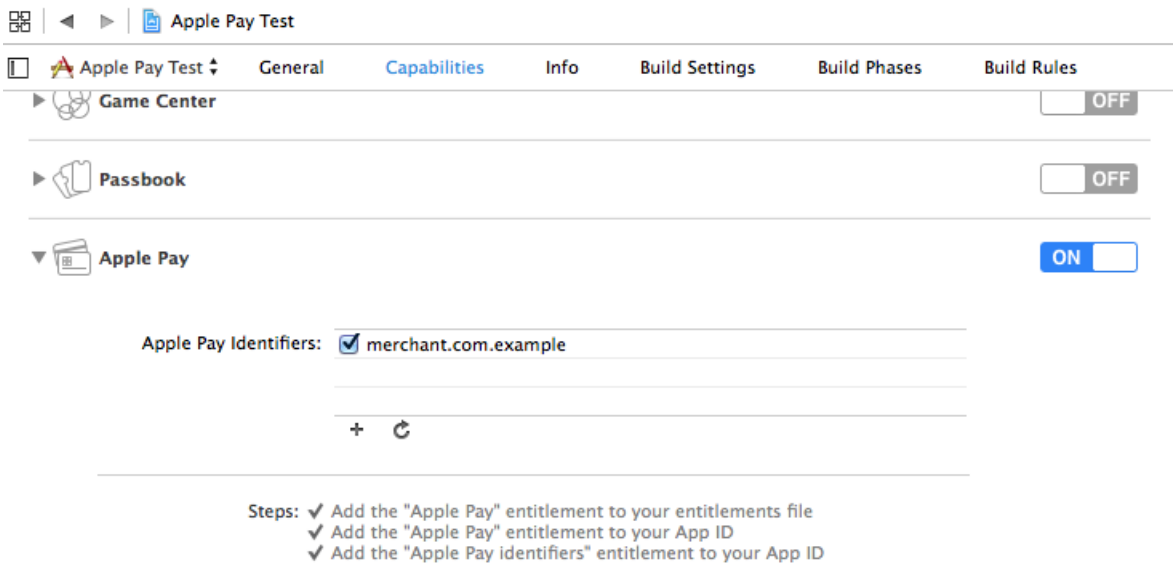
1. In Member Center, select Certificates, Identifiers & Profiles.
2. Under Identifiers, select Merchant IDs
3. Click the Add button (+) in the upper-right corner.
4. Enter a description and identifier, and click Continue.
5. Review the settings, and click Register.
6. Click Done.

To configure a certificate for your merchant ID

1. In Member Center, select Certificates, Identifiers & Profiles.
2. Under Identifiers, select Merchant IDs
3. Select the merchant ID from the list, and click Edit.
4. Click Create Certificate, follow the instructions to obtain or generate your certificate signing request (CSR), and click Continue.
5. Click Choose File, select your CSR, and click Generate.
6. Download the certificate by clicking Download, and click Done.

If you see a warning in Keychain Access that the certificate was signed by an unknown authority or that it has an invalid issuer, make sure you have the WWDR intermediate certificate - G2 and the Apple Root CA - G2 installed in your keychain. You can download them from apple.com/certificateauthority.

To enable Apple Pay for your app in Xcode, open the Capabilities pane. Select the switch in the Apple Pay row, and then select the merchant IDs you want the app to use.



Note: When troubleshooting, it is sometimes helpful to enable Apple Pay manually. Follow these steps to manually enable Apple Pay:

1. In Member Center, select Certificates, Identifiers & Profiles.
2. Under Identifiers, select App IDs
3. Select the app ID from the list, and click Edit.
4. Select Apple Pay, then click Edit.
5. Select the merchant IDs you want to use, and click Continue.
6. Review the settings, and click Assign.
7. Click Done.

Creating a Payment Request

Objective-C

Payment requests are instances of the `PKPaymentRequest` class. A payment request consists of a list of summary items that describe to the user what is being paid for, a list of available shipping methods, a description of what shipping information the user needs to provide, and information about the merchant and payment processor.

Decide Whether the User Can Make Payments

Before creating a payment request, determine whether the user will be able to make payments using a network that you support by calling the `canMakePaymentsUsingNetworks:` method of the `PKPaymentAuthorizationViewController` class. To check whether Apple Pay is supported by this device's hardware and parental controls, use the `canMakePayments` method.

If the user can't make payments, don't show the Apple Pay button. Instead, fall back to another method of payment.

Payment Requests Include Currency and Region Information

All of the summary amounts in a payment request use the same currency, which is specified using the `currencyCode` property of `PKPaymentRequest`. Use a three-character ISO currency code, such as `USD`.

The payment request's country code indicates the country where the purchase took place or will be processed. Use a two-character ISO country code, such as `US`.

The merchant ID you set in a payment request must match one of the merchant IDs in your app's entitlement.

```
request.currencyCode = @"USD";
request.countryCode = @"US";
request.merchantIdentifier = @"merchant.com.example";
```

Payment Requests Have a List of Payment Summary Items

Payment summary items, represented by the `PKPaymentSummaryItem` class, describe the different parts of the payment request to the user. Use a small number of summary items—typically the subtotal, any discount, the shipping, the tax, and the grand total. Provide granular details of the item-by-item costs elsewhere in your app.

Each summary item has a label and an amount, as shown in Listing 3-1. The label is a user-readable description of what the item summarizes. The amount is the corresponding payment amount. All of the amounts in a payment request use the currency specified in the payment request. For a discount or a coupon, set the amount to a negative number.

Listing 3-1 Creating a payment summary item

```
// 12.75 subtotal
NSDecimalNumber *subtotalAmount = [NSDecimalNumber decimalNumberWithMantissa:1275
 exponent:-2 isNegative:NO];
self.subtotal = [PKPaymentSummaryItem summaryItemWithLabel:@"Subtotal"
amount:subtotalAmount];


// 2.00 discount
NSDecimalNumber *discountAmount = [NSDecimalNumber decimalNumberWithMantissa:200
exponent:-2 isNegative:YES];
self.discount = [PKPaymentSummaryItem summaryItemWithLabel:@"Discount"
amount:discountAmount];
```

> **Note:** Payment summary items use the `NSDecimalNumber` class to store the amount as a base-10 quantity. Instances of this class can be created by explicitly specifying the mantissa and exponent (as shown in the code listings) or by providing the quantity as string and specifying a locale. Always use base-10 numbers for financial calculations—for example, to determine the amount of a 5% discount.
>
> Although they may appear to be more convenient, IEEE floating point data types such as `float` and `Double` are not suitable for financial calculations. These data types use a base-2 representation of numbers, which means that some decimal numbers can't be represented exactly—for example, 0.42 must be approximated as 0.41999 repeating. Such approximations can cause financial calculations to return incorrect results.

The last payment summary item in the list is the grand total. Calculate the grand total amount by adding the amounts of all the other summary items. The grand total is displayed differently than the other summary items: Use your company's name as its label, and use the total of all the other summary items' amounts as its amount. Add the payment summary items to the payment request using the `paymentSummaryItems` property.

```
// 10.75 grand total
NSDecimalNumber *totalAmount = [NSDecimalNumber zero];
totalAmount = [totalAmount decimalNumberByAdding:subtotalAmount];
totalAmount = [totalAmount decimalNumberByAdding:discountAmount];
self.total = [PKPaymentSummaryItem summaryItemWithLabel:@"My Company Name"
amount:totalAmount];

self.summaryItems = @[self.subtotal, self.discount, self.total];
request.paymentSummaryItems = self.summaryItems;
```

A Shipping Method Is a Special Payment Summary Item

Create an instance of `PKShippingMethod` for each available shipping method. Just like other payment summary items, shipping methods have a user-readable label such as Standard Shipping or Next Day Shipping, and an amount which is the shipping cost. Unlike other summary items, shipping methods also have a `detail` property—such as "Arrives by July 29" or "Ships in 24 hours"—that explains the difference between shipping methods.

To distinguish shipping methods in your delegate methods, use the `identifier` property. This property is used only by your app—the framework treats it as an opaque value and it doesn't appear in the UI. Assign a unique identifier for each shipping method when you create it. For ease of debugging, use a brief or abbreviated string, such as `"discount"`, `"standard"`, or `"next-day"`.

Some shipping methods aren't available in all areas or have different costs for different addresses. You can update this information when the user selects a shipping address or method, as described in Your Delegate Updates Shipping Methods and Costs (page 16).

Indicating Your Supported Payment Processing Mechanisms

Indicate which payment networks you support by populating the `supportedNetworks` property with an array of sting constants. Indicate which payment processing protocols you support by setting a value for the `merchantCapabilities` property. You must support 3DS; support of EMV is optional.

The merchant capabilities are bit masks and are combined as follows:

```
request.supportedNetworks = @[PKPaymentNetworkAmex, PKPaymentNetworkMasterCard,
PKPaymentNetworkVisa];


// Supports 3DS only
request.merchantCapabilities = PKMerchantCapability3DS;


// Supports both 3DS and EMV
request.merchantCapabilities = PKMerchantCapability3DS | PKMerchantCapabilityEMV;
```

Indicating What Shipping and Billing Information Is Needed

Populate the `requiredBillingAddressFields` and `requiredShippingAddressFields` properties of the payment authorization view controller to indicate what billing and shipping information is needed. When you present this view controller, it prompts the users to supply the needed information. The field constants are combined as follows to set values for these properties:

```
request.requiredBillingAddressFields = PKAddressFieldEmail;
request.requiredBillingAddressFields = PKAddressFieldEmail |
PKAddressFieldPostalAddress;
```

If you already have billing and shipping contact information, set those on the payment request. Although Apple Pay uses this information by default, the user can choose other contact information as part of the payment authorization process.

```
ABRecordRef record = ABPersonCreate();

CFErrorRef error;
BOOL success;

success = ABRecordSetValue(record, kABPersonFirstNameProperty, @"John", &error);
if (!success) { /* ... handle error ... */ }

success = ABRecordSetValue(record, kABPersonLastNameProperty, @"Appleseed", &error);
if (!success) { /* ... handle error ... */ }

ABMultiValueRef shippingAddress =
ABMultiValueCreateMutable(kABMultiDictionaryPropertyType);
NSDictionary *addressDictionary = @{
                                    (NSString *) kABPersonAddressStreetKey: @"1234
 Laurel Street",
                                (NSString *) kABPersonAddressCityKey: @"Atlanta",
                                    (NSString *) kABPersonAddressStateKey: @"GA",
                                    (NSString *) kABPersonAddressZIPKey: @"30303"
                                    };
ABMultiValueAddValueAndLabel(shippingAddress,
                             (__bridge CFDictionaryRef) addressDictionary,
                             kABOtherLabel,
                             nil);
success = ABRecordSetValue(record, kABPersonAddressProperty, shippingAddress,
&error);
if (!success) { /* ... handle error ... */ }

request.shippingAddress = record;

CFRelease(shippingAddress);
CFRelease(record);
```

Storing Additional Information

To store information about the payment request that is specific to your app, such as a shopping cart identifier, use the `applicationData` property. This property is treated as an opaque value by the system. A hash of the application data appears in the payment token after the user authorizes the payment request.

Authorizing a Payment

Objective-C

The payment authorization process is a cooperative effort between the payment authorization view controller and its delegate. A payment authorization view controller does two things: It lets the user select the billing and shipping information that is needed by a payment request, and it lets the user authorize the payment to be made. The delegate methods are called when the user interacts with the view controller so that your app can update the information shown—for example, to update the shipping price when a shipping address is selected. The delegate is also called after the user authorizes the payment request.

> **Note:** As you implement the delegate methods, remember that they can be called multiple times and that the order in which they are called depends on the order of the user's actions.

All of the delegate methods called during the authorization process are passed a completion block as one of their arguments. The payment authorization view controller waits for its delegate to finish responding to one method (by calling the completion block) before it calls any other delegate methods. The `paymentAuthorizationViewControllerDidFinish:` method is the only exception: It doesn't take a completion block and it can be called at any time.

The completion blocks take an argument that lets you specify the current status of the transaction based on the information that's available. If there are no problems with the transaction, you pass the value `PKPaymentAuthorizationStatusSuccess`; otherwise, you pass a value that identifies the problem.

To create an instance of the `PKPaymentAuthorizationViewController` class, pass the payment request to the view controller's initializer. Set a delegate for the view controller, and then present it.

```
PKPaymentAuthorizationViewController *viewController =
[[PKPaymentAuthorizationViewController alloc] initWithPaymentRequest:request];
if (!viewController) { /* ... Handle error ... */ }
viewController.delegate = self;
[self presentViewController:viewController animated:YES completion:nil];
```

As the user interacts with the view controller, the view controller calls its delegate methods.

Your Delegate Updates Shipping Methods and Costs

When the user provides shipping information, the authorization view controller calls your delegate's `paymentAuthorizationViewController:didSelectShippingAddress:completion:` and `paymentAuthorizationViewController:didSelectShippingMethod:completion:` methods. Use these methods to update the payment request based on the new information.

```
- (void) paymentAuthorizationViewController:(PKPaymentAuthorizationViewController
 *)controller
                   didSelectShippingAddress:(ABRecordRef)address
                                 completion:(void (^)(PKPaymentAuthorizationStatus,
 NSArray *, NSArray *))completion
{
    self.selectedShippingAddress = address;
    [self updateShippingCost];
    NSArray *shippingMethods = [self shippingMethodsForAddress:address];
    completion(PKPaymentAuthorizationStatusSuccess, shippingMethods,
self.summaryItems);
}


- (void) paymentAuthorizationViewController:(PKPaymentAuthorizationViewController
 *)controller
                    didSelectShippingMethod:(PKShippingMethod *)shippingMethod
                                 completion:(void (^)(PKPaymentAuthorizationStatus,
 NSArray *))completion
{
    self.selectedShippingMethod = shippingMethod;
    [self updateShippingCost];
    completion(PKPaymentAuthorizationStatusSuccess, self.summaryItems);
}
```

A Payment Token Is Created When Payment Is Authorized

When the user authorizes a payment request, the framework creates a payment token by coordinating with Apple's server and the Secure Element, which is a dedicated chip on the user's device. You send this payment token to your server in the

`paymentAuthorizationViewController:didAuthorizePayment:completion:` delegate method, along with any other information you need to process the purchase—for example, the shipping address and a shopping cart identifier. The process happens as follows:

- The framework sends the payment request to the Secure Element. Only the Secure Element has access to the tokenized device-specific payment card numbers.
- The Secure Element puts together payment data for the specified card and merchant, encrypts it so that only Apple can read it, and sends it to the framework. The framework sends the payment data to Apple's server.
- Apple's server reencrypts the payment data so that only the merchant can read it. The server then signs it, creating the payment token that it sends to the device.
- The framework calls your delegate method with the token, and your delegate sends the token to your server.

The actions on your server vary depending on whether you process your own payments or work with a payment platform. In both cases, your server handles the order and sends a status back to the device, which your delegate passes to its completion handler, as described in Processing a Payment (page 19).

```
- (void) paymentAuthorizationViewController:(PKPaymentAuthorizationViewController
 *)controller
                       didAuthorizePayment:(PKPayment *)payment
                                completion:(void
(^)(PKPaymentAuthorizationStatus))completion
{
    NSError *error;
    ABMultiValueRef addressMultiValue = ABRecordCopyValue(payment.billingAddress,
 kABPersonAddressProperty);
    NSDictionary *addressDictionary = (__bridge_transfer NSDictionary *)
ABMultiValueCopyValueAtIndex(addressMultiValue, 0);
    NSData *json = [NSJSONSerialization dataWithJSONObject:addressDictionary
options:NSJSONWritingPrettyPrinted error: &error];

    // ... Send payment token, shipping and billing address, and order information
 to your server ...

    PKPaymentAuthorizationStatus status;  // From your server
    completion(status);
}
```

Your Delegate Dismisses the Payment Authorization View Controller

After the framework displays the transaction's status, the authorization view controller calls your delegate's `paymentAuthorizationViewControllerDidFinish:` method. In your implementation, dismiss the authorization view controller and then display your own app-specific order-confirmation page.

```
- (void)
paymentAuthorizationViewControllerDidFinish:(PKPaymentAuthorizationViewController
 *)controller
{
    [controller dismissViewControllerAnimated:YES completion:nil];
}
```

Payment Processing

Processing a payment involves several steps:

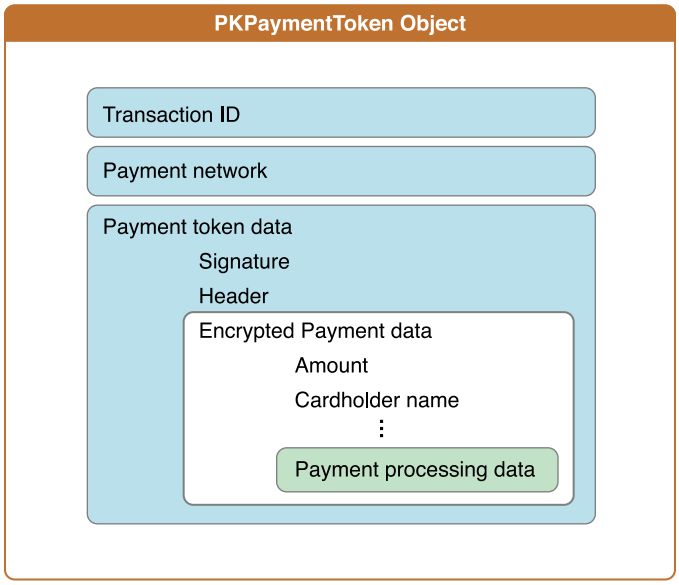
1. Sending the payment information to your server, along with other information needed to process the order
2. Verifying the hashes and signature of the payment data
3. Decrypting the encrypted payment data
4. Submitting payment data to the payment processing network
5. Submitting the order to your order-tracking system

You have two options for processing the payment: You can take advantage of a payment platform to process the payment, or you can implement the payment processing yourself. A payment processing platform typically handles most of the steps listed above.

Reading, verifying, and processing payment information requires an understanding of several areas of cryptography such as calculating an SHA–1 hash, reading and validating a PKCS #7 signature, and performing elliptic curve Diffie-Hellman key exchange. If you don't have a background in cryptography, consider using a payment platform that performs these operations for you. For information about payment platforms that support Apple Pay, see developer.apple.com/apple-pay/.

The information used to process a payment has a nested data structure, as shown in Figure 5-1. A payment token is an instance of the `PKPaymentToken` class. The value of its `paymentData` property is a JSON dictionary, which has a header with information used for validation, and encrypted payment data. The encrypted data includes information such as the amount and cardholder name and other information used for the specific payment processing protocol.

Figure 5-1 Payment data structure



For details on the format of the payment data structure, see *Payment Token Format Reference*.

Document Revision History

This table describes the changes to *Apple Pay Programming Guide*.

Date	Notes
2014-11-18	New document that describes how to use Apple Pay to let users purchase real-world goods and services from within an app.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010