amazon payment services

iOS Integration Guide

Document Version: 3.7

April, 2021

Contents

1 About this Document 5

1.1 Intended audience 5
1.2 Note regarding PayFort / FORT 5

2 Before you start the integration 6

3 Mobile SDK transaction workflow 7

4 Installing the Mobile SDK 8

4.1 Include the SDK in your Xcode project 8
4.1.1 Manual installation 8
4.1.2 Installation using CocoaPods: 8
4.2 Ensuring your application does not disconnect in background mode 9

5 Creating a mobile SDK token 10

5.1 Mobile SDK token URLs 10
5.2 Submitting token request parameters 11

6 Processing transactions with the iOS SDK 13

6.1 Two routes for payment processing 13
6.2 Steps for standard checkout 14
6.3 iOS mobile SDK operations 17
6.3.1 Request parameters 17
6.3.2 Response parameters 21

7 Complete sample code for standard UI checkout: 25

8 Customizing the standard payment UI 27

8.1 Customizing the standard payment UI 27
8.2 Hiding the Amazon Payment Service loading prompt 28
8.3 Changing the presentation style 29
8.4 iOS SDK response 29

9 Using a custom payment processing UI 30

9.1 Stage 1: Generate an SDK token 30
9.2 Stage 2: Create the card components 30
9.3 Stage 3: Initiate the payment 35
9.4 Sample code 35

10 Using Direct Pay 37

10.1 Direct Pay sample code 37

11 Amazon Payment Services iOS SDK transaction feedback 39

11.1 Receiving transaction feedback 39
11.2 Registering Transaction Feedback URLs 39
11.3 Transaction Feedback Implementation 40

12 Validate API 41

13 Migrating from version 2.3 of the SDK 42

14 Appendix: About the Software 47

14.1 About this software 47
14.2 Supported Platforms 47
14.3 Localization 47
14.4 Screen Orientation 47
14.5 Supported Payment Methods 47
14.6 Supported Payment Options 47

Contact Us

integration-ps@amazon.com
https://paymentservices.amazon.com

1 About this Document

This document describes our Mobile SDK for iOS and includes information on how to integrate it with your mobile application.

1.1 Intended audience

This document was created for iOS developers that integrate the Amazon Payment Services iOS mobile SDK with their merchant application.

1.2 Note regarding PayFort / FORT

Amazon Payment Services is the new name for PayFort. PayFort is a leading provider of payment processing services that was acquired by Amazon in 2017.

Throughout this section, and in our API reference and SDK guides, you will see references to PayFort. You may also see references to Fort or FORT.

We continue to use PayFort and Fort in our documentation for the simple reason that the code that powers Amazon Payment Services still contains references to PayFort.

To ensure ongoing stability, and to minimize the development overhead for our customers, we are slowly but steadily changing references to PayFort across our core code and our documentation.

In the meantime, when you see PayFort or Fort, you can safely assume that we are referring to Amazon Payment Services features and benefits.

2 Before you start the integration

Read through the following steps first to understand how the integration process works. This will help you to understand why these steps are required and why you need to follow a specific sequence.

Step 1: Access your test account

You need to make sure that you have access to a test account with Amazon Payment Services. It is a full test environment that allows you to fully simulate transactions.

Step 2: Choose between a standardized or custom payment UI

The Amazon Payment Services iOS SDK provides you with a standard payment UI that you can use to quickly integrate in-app payments. The standard UI offers limited customizability.

Alternatively, you can choose to build your own payment UI using Amazon Payment Services iOS SDK building blocks, we describe this in the section on custom-coding a payment processing UI.

Step 3: Make sure that you are using the correct integration type

Prior to building the integration, you need to make sure that you are selecting and using the proper parameters in the API calls as per the required integration type. All the mandatory parameters are mentioned under every section in the API document

Step 4: Install the Amazon Payment Services iOS SDK in your developer environment

You need to download our iOS SDK from the link provided. Next you need to include the iOS SDK in your Xcode project by following the steps in the next section. You are also required to install the library and to integrate the iOS SDK into your app.

Step 5: Create a test transaction request

You need to create a test transaction request. Processing a valid API request depends on the transaction parameters included, you need to check the documentation and review every parameter to reduce the errors in processing the transaction.

Step 6: Process a transaction response

After every payment, Amazon Payment Services returns the transaction response on the URL configured in your account under **Technical Settings**, **Channel Configuration**.

For more details review the transaction feedback instructions in this section. You need to validate the response parameters returned on this URL by calculating the signature for the response parameters using the SHA response phrase configured in your account under security settings.

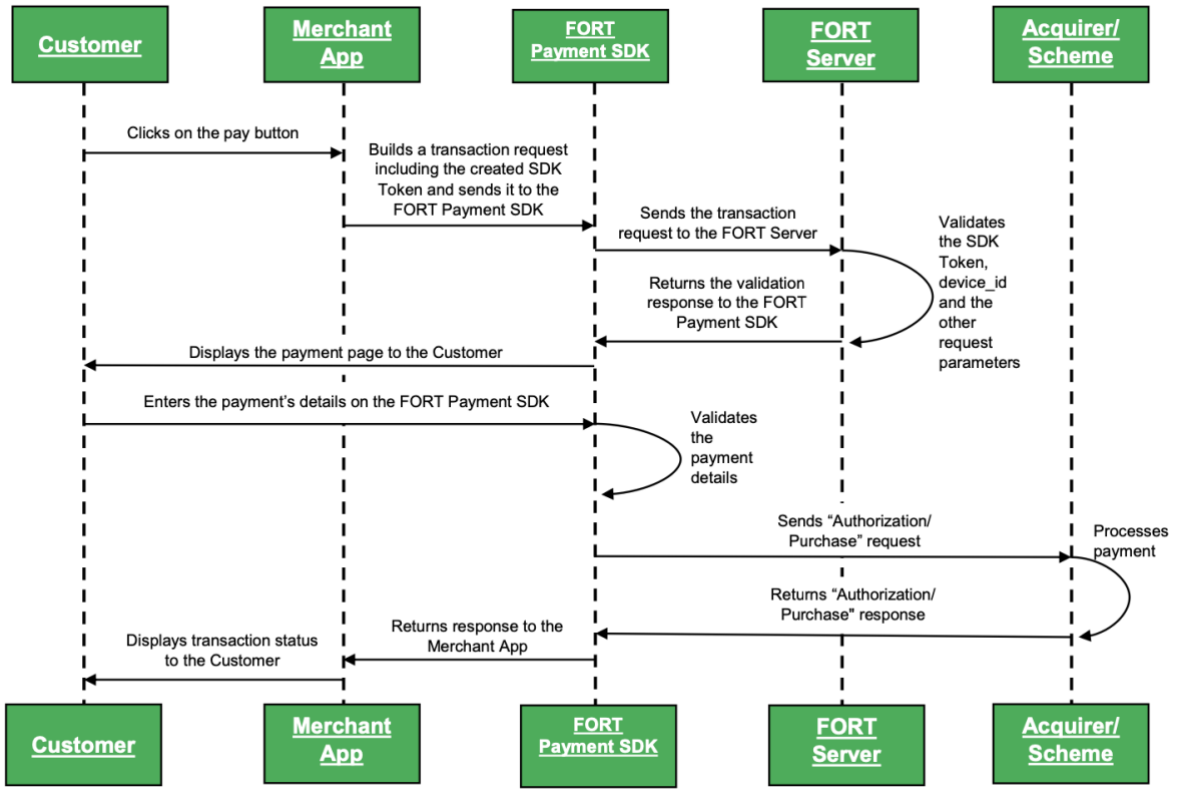
Step 7: Test and Go Live

You can use our test card numbers to test your integration and simulate your test cases. The Amazon Payment Services team may need to test your integration before going live to assure your application integration.

3 Mobile SDK transaction workflow

Below we describe the transaction workflow that occurs when you process a payment using the Amazon Payment Service iOS Mobile SDK.

1. Your customer clicks on the Pay button in your app.
2. Your merchant system (back-end) generates a mobile SDK token using the Amazon Payment Services API.
3. Your app passes the parameters, including the SDK token, to the iOS mobile SDK.
4. The iOS mobile SDK starts a secure connection and sends the transaction request to the Amazon Payment Services server to be validated.
5. The Amazon Payment Services API validates the SDK token, device_ID and other request parameters and returns the validation response the iOS SDK.
6. Assuming validation is passed your merchant app displays a payment page to the customer.
7. Your customer enters their payment details on the payment page in the iOS SDK prompt on their device.
8. The Amazon Payment Services iOS SDK validates your customer's details and sends a transaction (Authorization or Purchase) request to the relevant payment processor and issuing bank.
9. The payment processor processes the transaction request and returns a transaction response to the iOS SDK.
10. The Amazon Payment Services iOS SDK returns the transaction response to your merchant app.
11. Your merchant app displays the response to your customer.



4 Installing the Mobile SDK

These are the first steps you need to follow to install the Amazon Payment Services iOS SDK in your iOS application.

4.1 Include the SDK in your Xcode project

You have two options for including the SDK in your Xcode project. You can do so manually, or you can include the SDK by using CocoaPods

4.1.1 Manual installation

1. Obtain the Amazon Payment Services iOS Mobile SDK by downloading it from the Amazon Payment Services website.
2. Extract the folder you downloaded in the previous step.
3. Drag **`PayFortSDK.xcframework`** into the **Frameworks**, **Libraries**, and **Embedded Content** section of your target.

4.1.2 Installation using CocoaPods:

NOTE: The PayFort SDK is distributed as an XCFramework, therefore you are required to use Cocoapods 1.9.0 or newer.

1. Add the following code to your Podfile (inside the target section):

```
pod 'PayFortSDK'
```

2. Add the following to the bottom of your Podfile:

```
post_install do |installer|
      installer.pods_project.targets.each do |target|
        if ['PayFortSDK'].include? target.name
          target.build_configurations.each do |config|
              config.build_settings['BUILD_LIBRARY_FOR_DISTRIBUTION'] = 'YES'
          end
        end
      end
    end
```


3. Run the following command:

```
pod install
```

NOTE: Ensure it is linked once in the Linked Framework and Libraries or just drag the PayFortSDK.xcframework to Embedded Binaries in the general tab in the project settings.

NOTE: In Xcode, secondary-click your project's .plist file and select Open As -> Source Code. Insert the following XML snippet into the body of your file just before the final, as below:

```
</dict>element
<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key><true/>
</dict>
```

4.2 Ensuring your application does not disconnect in background mode

To ensure that your app does not disconnect from the SDK when it goes into the background make sure to add this code:

Objective-C:

```
(void)applicationDidEnterBackground:(UIApplication *)application {
    block UIBackgroundTaskIdentifier backgroundTask; backgroundTask =
[application beginBackgroundTaskWithExpirationHandler: ^ { [application
endBackgroundTask:backgroundTask];
backgroundTask = UIBackgroundTaskInvalid; }]; }
```

Swift

```
func applicationDidEnterBackground(_ application: UIApplication)
{
var bgTask: UIBackgroundTaskIdentifier = 0 bgTask =
application.beginBackgroundTask(expirationHandler:
{ application.endBackgroundTask(bgTask) bgTask = UIBackgroundTaskInvalid
})
}
```

5 Creating a mobile SDK token

A mobile SDK authentication token is required to authenticate every request sent to the SDK. The token is also significant to process payment operations with Amazon Payment Services through our iOS mobile SDK.

To get started with our iOS mobile SDK you must first establish the ability to generate a mobile SDK token.

NOTE: The creation and initiation of a mobile SDK token happens on your server – your server must generate the token by sending a request to the Amazon Payment Services API.

NOTE: A unique authentication token must be created for each transaction. Each authentication token has a life-time of only one hour if no new request from the same device is sent.

5.1 Mobile SDK token URLs

These are the URLs you need to use when you request a mobile SDK token for your iOS app:

Test Environment URL

https://sbpaymentservices.payfort.com/FortAPI/paymentApi

Production Environment URL

https://paymentservices.payfort.com/FortAPI/paymentApi

5.2 Submitting token request parameters

You need to submit parameters as a REST POST request using JSON. Below we list the range of parameters for the iOS mobile SDK token request.

5.2.1.1 iOS Mobile SDK Token Request Parameters

When you send your request for an iOS mobile SDK token you must send the following parameters to Amazon Payment Services:

Request Parameters							
Parameter Name	**Type**	**Mandatory**	**Description**	**Length**	**Allowed Special Characters**	**Possible/ Expected Values**	**Example**
service_command	Alpha	Yes	Command	20	_	SDK_TOKEN	
access_code	Alphanumeric	Yes	Access code	20			zx0IPmPy5jp1vAz8Kpg7
merchant_identifier	Alphanumeric	Yes	Your merchant ID	20			CycHZxVj
language	Alpha	Yes	The checkout page and messages language.	2		- en - ar	
device_id	Alphanumeric	Yes	A unique device identifier.	100	-		ffffffff-a9fa0b44-7b2729e70033c587
signature	Alphanumeric	Yes	A string hashed using the Secure Hash Algorithm.	200			7cad05f0212ed933c9a5d5dffa31661acf2c827a

NOTE: device_id - This value to be generated from the UIDevice Class Reference, and you can generate this parameter by using the following command: [payFort getUDID]

5.2.1.2 iOS Mobile SDK Token Response Parameters

These parameters will be returned in the Amazon Payment Services response:

Response Parameters						
Parameter Name	**Type**	**Mandatory**	**Description**	**Length**	**Possible/ Expected Values**	**Example**
service_command	Alpha	Yes	Command	20	SDK_TOKEN	
access_code	Alphanumeric	Yes	Access code	20		zx0IPmPy5j p1vAz8Kpg 7
merchant_identifier	Alphanumeric	Yes	Your merchant ID	20		CycHZxVj
language	Alpha	Yes	The checkout page and messages language.	2	- en - ar	
device_id	Alphanumeric	Yes	A unique device identifier.	100		ffffffff-a9fa0b44-7b2729e70033c587
sdk_token	Alphanumeric	Yes	An SDK authentication token to enable using the iOS Mobile SDK.	100		Dwp78q3
signature	Alphanumeric	Yes	A string hashed using the Secure Hash Algorithm.	200		7cad05f0212ed933c9a5d5dffa31661acf2c827a
status	Numeric	No	A two-digit numeric value that indicates the status of the transaction.	2		

NOTE: Every parameter the merchant sends in the request should be received by the merchant in the response - even the optional parameters.

6 Processing transactions with the iOS SDK

In this section we outline how you process a transaction using the iOS mobile SDK.

6.1 Two routes for payment processing

As a merchant you have two ways in which you can process payments using the Amazon Payment Services iOS mobile SDK.

1. **Standard payment screen.** You can use the standard Amazon Payment Services iOS SDK interface to display a standard payment screen.

   This standard payment view is customizable in three ways. You can hide the loading screen, and you can change the presentation style from full screen to OS default. You can also customize some of the UI elements. We address customizing the standard payment screen in Section 7.

2. **Custom integration**. You can choose to build your own payment processing screen by coding your own payment processing screen. With this mobile SDK feature, we allow merchants to integrate a native app checkout experience without displaying our standard payment screen – while still using SDK features for rapid go-live.

   With this integration route your customers can securely pay using a custom merchant checkout interface. The SDK provides card input fields and a pay button that merchants can encapsulate inside their checkout interface to match their own inline customer experience.

   The Amazon Payment Services SDK will securely transmit the completed card details to the Amazon Payment Services API for processing in order to complete the transaction. We discuss the custom-coded payment UI in Section 5.

6.2 Steps for standard checkout

These are the steps you need to follow to perform a checkout using our standard UI. See the next section for building your own customized payment UI.

1. **Import the framework into your app**

   Start by importing the Amazon Payment Service iOS SDK library. You do so by using the following code:

   Objective-C

```
#import <PayFortSDK/PayFortSDK-Swift.h>
```

   Swift

```
import PayFortSDK
```

2. **Initialize the controllers**

   Initialize **PayFortController** at class level and only once within the targeted environment. You set the target environment by setting one of the two ENUM, either **PayFortEnviromentSandBox** or **PayFortEnviromentProduction.**

   Objective-C

```
PayFortController *payFort = [[PayFortController alloc]initWithEnviroment:
PayFortEnviromentSandBox];
```

   Swift

```
let payFort = PayFortController.init(enviroment: .sandBox)
```


3. **Preparing Request Parameters**

Set a dictionary that contains all keys and values for the SDK

Objective-C

```
NSMutableDictionary *request = [[NSMutableDictionary alloc]init];
  [request setValue:@"10000" forKey:@"amount"];
  [request setValue:@"AUTHORIZATION" forKey:@"command"];
  [request setValue:@"USD" forKey:@"currency"];
  [request setValue:@ "email@domain.com" forKey:@"customer_email"]
  [request setValue:@"en" forKey:@"language"];
  [request setValue:@"112233682686" forKey:@"merchant_reference"]
  [request setValue:@"SDK TOKEN GOES HERE" forKey:@"sdk_token"];
  [request setValue:@"" forKey:@"payment_option"];
  [request setValue:@"gr66zzwW9" forKey:@"token_name"];
```

Swift

```
let request = ["amount" : "1000",
    "command" : "AUTHORIZATION",
    "currency" : "AED",
    "customer_email" : "rzghebrah@payfort.com",
    "installments" : "",
    "language" : "en",
    "sdk_token" : "token"]
```

4 . **Response callback function**

Amazon Payment Services allows you retrieve and receive the response parameters after processing a transaction once the transaction is completed. It only happens during the installation process. This is the code you need to use:

Objective-C

```
[payFort callPayFortWithRequest:request currentViewController:self
                        success:^(NSDictionary *requestDic, NSDictionary
*responeDic) { NSLog(@"Success");
        NSLog(@"responeDic=%@",responeDic);
    }
                       canceled:^(NSDictionary *requestDic, NSDictionary
*responeDic) { NSLog(@"Canceled");
        NSLog(@"responeDic=%@",responeDic);
    }
                          faild:^(NSDictionary *requestDic, NSDictionary
*responeDic, NSString *message) {
        NSLog(@"Faild");
        NSLog(@"responeDic=%@",responeDic);
    }];
```


Swift

```
payFort.callPayFort(withRequest: request, currentViewController: self, success: {
(requestDic, responeDic) in    print("success")
        },
        canceled: { (requestDic, responeDic) in
            print("canceled")
        },    faild: { (requestDic, responeDic, message) in print("faild")
})
```


6.3 iOS mobile SDK operations

6.3.1 Request parameters

This is a list of the parameters you need to send when you send a request to the iOS SDK.

Request Parameters							
Parameter Name	**Type**	**Mandatory**	**Description**	**Length**	**Allowed special characters**	**Possible or Expected Values**	**Example**
command	Alpha	Yes	Command	20		- AUTHORIZATION -PURCHASE	
merchant_reference	Alphanumeric	Yes	Your unique order ID	40	- _		XYZ9239yu898
amount	Numeric	Yes	*Each currency has predefined allowed decimal points that should be taken into consideration when sending the amount.	10			10000
currency	Alpha	Yes	The currency of the transaction's amount in ISO code 3.	3			AED
language	Alpha	Yes	Language used on the checkout page and for messages	2		en ar	
customer_email	Alphanumeric	Yes	Your customer's email address	254	_ - . @ +		customer@domain.com
sdk_token	Alphanumeric	Yes	The SDK token you generate for	100			Dwp78q3

			every SDK transaction				
token_name	Alphanumeric	No	The token received from the tokenization process	100	. @ - _		Op9Vmp
payment_option	Alpha	No	Payment option	10		VISA MASTERCARD AMEX MADA (for Purchase operations and eci Ecommerce only).	
eci	Alpha	No	E-commerce indicator	16		ECOMMERCE	
order_description	Alphanumeric	No	It holds the description of the order	150	# ' / . _ - : $ Space		iPhone 12 Pro
customer_ip	Alphanumeric	No	It holds the customer 's IP address. *It's Mandatory, if the fraud service is active.	45			192.178.1.10
customer_ name	Alpha	No	The customer’s name	40	_ \ / - . '		John Smith
phone_number	Alphanumeric	No	The customer’s phone number	19	+ - ( ) Space		00962797219966
settlement_reference	Alphanumeric	No	The Merchant submits this value to Amazon Payment Services. The value is then passed to the Acquiring bank and displayed to the merchant	34	- _ .		XYZ9239yu898

			in the Acquirer settlement file.				
merchant_ extra	Alphanumeric	No	Extra data sent by merchant . Will be received and sent back as received. Will not be displayed in any report.	999	; / _ - , ' @		JohnSmith
merchant_ extra1	Alphanumeric	No	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250	; / _ - , ' @		JohnSmith
merchant_ extra2	Alphanumeric	No	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250	; / _ - , ' @		JohnSmith
merchant_ extra3	Alphanumeric	No	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250	; / _ - , ' @		JohnSmith
merchant_ extra4	Alphanumeric	No	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250	; / _ - , ' @		JohnSmith
merchant_ extra5	Alphanumeric	No	Extra data sent by merchant. Will be received and	250	; / _ - ,		JohnSmith

			sent back as received. Will not be displayed in any report.		' @		

NOTE: Before sending the transaction value you must multiply the value by a factor that matches the ISO 4217 specification for that currency. Multiplication is necessary to accommodate decimal values. Each currency's 3-digit ISO code will have a specification for the number of digits after the decimal separator.

For example: If the transaction value is 500 AED; according to ISO 4217, you should multiply the value with 100 (to accommodate 2 decimal points). You will therefore send an AED 500 purchase amount as a value of 50000.

Another example: If the amount value was 100 JOD; according to ISO 4217, you should multiply the value with 1000 (to accommodate 3 decimal points). You therefore send a JOD 100 purchase amount as a value of 100000.

6.3.2 Response parameters

Request Parameters					
Parameter Name	**Type**	**Description**	**Length**	**Possible or Expected Values**	**Example**
command	Alpha	Command	20	- AUTHORIZATION -PURCHASE	
merchant_reference	Alphanumeric	Your unique order ID	40		XYZ9239y u898
amount	Numeric	*Each currency has predefined allowed decimal points that should be taken into consideration when sending the amount.	10		10000
currency	Alpha	The currency of the transaction's amount in ISO code 3.	3		AED
language	Alpha	Language used on the checkout page and for messages	2	en ar	
customer_email	Alphanumeric	Your customer's email address	254		customer @domain. com
fort_id	Numeric	The order's unique reference returned by our system.	20		14437968668 48
sdk_token	Alphanumeric	The SDK token you generate for every SDK transaction	100		Dwp78q3

token_name	Alphanumeric	The token received from the tokenization process	100		Op9Vmp
payment_option	Alpha	Payment option	10	VISA MASTERCARD AMEX MADA (for Purchase operations and eci Ecommerce only).	
eci	Alpha	E-commerce indicator	16	ECOMMERCE	
authorization_code	Alphanumeric	The authorization code returned from the 3rd party.	100	-	P10000000000000372136
order_description	Alphanumeric	It holds the description of the order	150		iPhone 12 Pro
response_message	Alphanumeric	Message description of the response code. It returns according to the request language.	150	-	Insufficient funds
response_code	Numeric	Response Code carries the value of our system's response. The code is made up of five digits.	5		
status	Numeric	A two-digit numeric value that indicates the status of the transaction.	2		
customer_ip	Alphanumeric	It holds the customer 's IP address. *It's Mandatory, if the fraud service is active.	45		192.178.1.10

expiry_date	Numeric	The card's expiry date.			
card_number	Numeric	The masked credit card number. Only the MEEZA payment option takes 19 digits card number. *AMEX payment option takes 15 digits card number. *Otherwise, they take 16 digits card number.	16		400555******0001
customer_ name	Alpha	The customer’s name	40		John Smith
phone_number	Alphanumeric	The customer’s phone number	19		00962797219966
settlement_reference	Alphanumeric	The Merchant submits this value to Amazon Payment Services. The value is then passed to the Acquiring bank and displayed to the merchant in the Acquirer settlement file.	34		XYZ9239yu898
merchant_ extra	Alphanumeric	Extra data sent by merchant . Will be received and sent back as received. Will not be displayed in any report.	999		JohnSmith
merchant_ extra1	Alphanumeric	Extra data sent by merchant. Will be	250		JohnSmith

		received and sent back as received. Will not be displayed in any report.			
merchant_ extra2	Alphan umeric	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250		JohnSmith
merchant_ extra3	Alphan umeric	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250		JohnSmith
merchant_ extra4	Alphan umeric	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250		JohnSmith
merchant_ extra5	Alphan umeric	Extra data sent by merchant. Will be received and sent back as received. Will not be displayed in any report.	250		JohnSmith

NOTE: Every parameter the Merchant sends in the Request should be received by the Merchant in the Response - even the optional ones.

7 Complete sample code for standard UI checkout:

The following sample code shows you how to process a payment using the standard view. The code sample illustrates how you send a request operation in the mobile SDK.

Objective-C

```
- (void)viewDidLoad {
    [super viewDidLoad];
    payfort = [[PayFortController alloc]
initWithEnviroment:PayFortEnviromentSandBox];
}

NSMutableDictionary *requestDictionary = [[NSMutableDictionary alloc]init];
[requestDictionary setValue:@"10000" forKey:@"amount"];
[requestDictionary setValue:@"AUTHORIZATION" forKey:@"command"];
[requestDictionary setValue:@"USD" forKey:@"currency"];
[requestDictionary setValue:@"email@domain.com" forKey:@"customer_email"];
[requestDictionary setValue:@"en" forKey:@"language"];
[requestDictionary setValue:@"112233682686" forKey:@"merchant_reference"];
[requestDictionary setValue:@"" forKey:@"payment_option"];
[requestDictionary setValue:@"gr66zzwW9" forKey:@"token_name"];

    [payFort callPayFortWithRequest:requestDictionary currentViewController:self
success:^(NSDictionary *requestDic, NSDictionary *responeDic) {

    } canceled:^(NSDictionary *requestDic, NSDictionary *responeDic) {
    } faild:^(NSDictionary *requestDic, NSDictionary *responeDic, NSString
*message) {

    }];
```

Swift

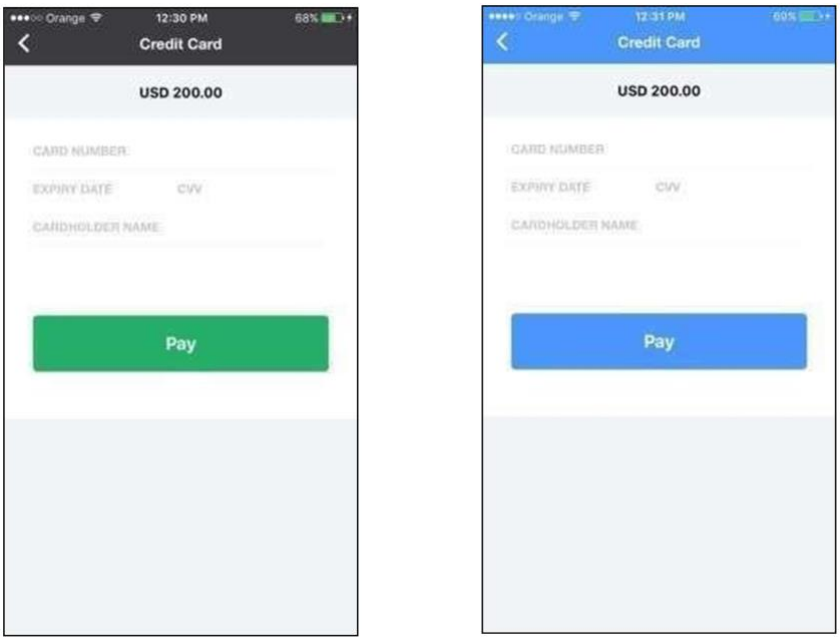
```
let request = ["amount" : "1000",
                            "command" : "AUTHORIZATION",
                            "currency" : "AED",
                            "customer_email" : "rzghebrah@payfort.com",
                            "installments" : "",
                            "language" : "en",
                            "sdk_token" : "token"]

payFort.callPayFort(withRequest: request, currentViewController: self,
                                        success: { (requestDic, responeDic) in
                                          print("success")
                                          print("responeDic=\(responeDic)")
                                          print("responeDic=\(responeDic)")

                                        },
                                        canceled: { (requestDic, responeDic) in
                                          print("canceled")
                                          print("responeDic=\(responeDic)")
                                          print("responeDic=\(responeDic)")

                                        },
                                        faild: { (requestDic, responeDic, message)
in
                                          print("faild")
                                          print("responeDic=\(responeDic)")
                                          print("responeDic=\(responeDic)")
                                      })
```


8 Customizing the standard payment UI

When you use the standard payment UI you can customize the payment UI presented by our iOS SDK in a number of ways to better reflect your business. We outline your options below.

8.1 Customizing the standard payment UI

You can customize the standard payment user interface in your iOS app. This is an example of a customized payment UI:



Standard vs. Customized Mobile SDK Payment Page

Follow these steps to configure a customized payment UI:

1. Create your nibFile .xib and set the name of Arabic xib same name with English one with suffix -ar.

2. Link the xib with PayFortView and bind all the IBOutlets in interface section IBOutlet UILabel *titleLbl;

```
IBOutlet UIButton *BackBtn;
IBOutlet UILabel *PriceLbl;
IBOutlet JVFloatLabeledTextField *CardNameTxt;
IBOutlet JVFloatLabeledTextField *CardNumberTxt;
IBOutlet JVFloatLabeledTextField *CVCNumberTxt;
IBOutlet JVFloatLabeledTextField *ExpDateTxt;
IBOutlet UILabel *cardNumberErrorlbl;
```



```
IBOutlet UILabel *cVCNumberErrorlbl;
IBOutlet UILabel *expDateErrorlbl;
IBOutlet UISwitch *savedCardSwitch;
IBOutlet UIButton *paymentBtn;
IBOutlet UILabel *saveCardLbl;
IBOutlet UIImageView *imageCard;
```

3. Assign new created xib file to Amazon Payment Services controller.

```
[payFort setPayFortCustomViewNib:@"PayFortView2"];
```

NOTE. If you call Arabic view and the Arabic view not existed the application will crash. Don't forget to set the custom view field in the identity inspector

8.2 Hiding the Amazon Payment Service loading prompt

There is an option to hide the loading prompt when the iOS SDK initializes the connection request. You can disable the loading prompt by using following option:

Objective-C

```
payFort.hideLoading = YES;
```

Swift

```
payFort.hideLoading = true
```


8.3 Changing the presentation style

It’s easy to change the presentation style from full screen to default by using the following property:

Objective-C

```
payFort.presentAsDefault = YES;
```

Swift

```
payFort.presentAsDefault = true
```

NOTE: The default type is full screen when you set the value to false, when set to true it will appear according to the OS default.

8.4 iOS SDK response

There is an option to show the response view more directly in an elegant view that shows the response results either as success or failed. You do so by activating the following option:

Objective-C

```
payFort.isShowResponsePage = YES;
```

Swift

```
payFort.isShowResponsePage= true
```


9 Using a custom payment processing UI

In this section we outline the key information you need to create your own payment processing screen using the tools in the iOS SDK.

9.1 Stage 1: Generate an SDK token

You need to generate an SDK token before you can start processing payments using your custom payment processing UI. Refer to the SDK token section earlier in this document for instructions on creating an SDK token.

9.2 Stage 2: Create the card components

You create your custom payment screen by using the following five components included in the Amazon Payment Services iOS SDK:

- CardNumberView
- CVCNumberView
- ExpiryDateView
- CardHolderNameView
- PayButton

Attributes for custom payment UI		
Attributes	Type	Description
textColor	UIColor	The input text filed text color
fontStyle	UIFont	The input text filed font style
backgroundColor	UIColor	The input text filed background color
errorText	String	The Error Label text
errorFontStyle	UIFont	The Error Label text Font Style
errorTextColor	UIColor	The Error Label text Text Color
titleText	String	The Title Label text
titleTextColor	UIColor	The Title Label text Color
titleErrorTextColor	UIColor	The Error Title Label text Color
titleFontStyle	UIFont	The Title Label Font style

Components Views

Item property, all these properties are available for each component.

Swift

```
let property = Property()
        property.textColor
        property.fontStyle
        property.backgroundColor
        property.errorFontStyle
        property.errorTextColor
        property.titleTextColor
        property.titleErrorTextColor
        property.titleFontStyle
```

Objective-c

```
Property *property = [[Property alloc] init]
        property.textColor
        property.fontStyle
        property.backgroundColor
        property.errorFontStyle
        property.errorTextColor
        property.titleTextColor
        property.titleErrorTextColor
        property.titleFontStyle
```

1. CardNumberView:

The **CardNumberView** inheritance from UIView, **CardNumberView is** used to validate the card number, card brand and card number length.

Swift

```
@IBOutlet private weak var cardNumberView: CardNumberView!
cardNumberView.property = property
```

Objective-c

```
@property (nonatomic, weak) IBOutlet CardNumberView *cardNumberView;
cardNumberView.property = property
```


2. **ExpiryDateView**:

   The **ExpiryDateView** inheritance from UIView, **ExpiryDateView** is used to check the expiry date for the credit card.

   Swift

```
@IBOutlet private weak var expiryDateView: ExpiryDateView!
     expiryDateView.property = property
```

   Objective-c

```
@property (nonatomic, weak) IBOutlet ExpiryDateView *expiryDateView;
     expiryDateView.property = property
```

3. **CVCNumberView**

   The **CVCNumberView** inheritance from UIView, **CVCNumberView** used to check if `cvc` matches `cardBrad`.

   Swift

```
@IBOutlet private weak var cvcNumberView: CVCNumberView!
   cvcNumberView.property = property
```

   Objective-c

```
@property (nonatomic, weak) IBOutlet CVCNumberView *cvcNumberView;
     cvcNumberView.property = property
```

4. **HolderNameView**

   The **HolderNameView** inheritance from UIView, **HolderNameView** is used to fill the card holder name.

   Swift

```
@IBOutlet private weak var holderNameView: HolderNameView!
     holderNameView.property = property
```

   Objective-c

```
@property (nonatomic, weak) IBOutlet holderNameView *holderNameView;
     holderNameView.property = property
```

5. **ErrorLabel**

   It will show any error message for owner card view, you can set your custom UILabel, Example:

   Swift

```
@IBOutlet private weak var cardNumberErrorLabel: ErrorLabel!
   cardNumberView.errorLabel = cardNumberErrorLabel
```

   Objective-c

```
@property (nonatomic, weak) IBOutlet ErrorLabel *cardNumberErrorLabel;
   cardNumberView.errorLabel = cardNumberErrorLabel
```

Example of components views in Objective-C and Swift

This is one example of how to customize the component:

Swift

```
let property = Property()
    property.textColor = .yellow
    property.backgroundColor = .green
    property.errorTextColor = .green
    property.titleTextColor = .red
    cardNumberView.property = property
```

Objective-c

```
Property * property = [[Property alloc] init];
    property.textColor = UIColor.yellowColor;
    property.backgroundColor = UIColor.greenColor;
    property.errorTextColor = UIColor.greenColor;
    property.titleTextColor = UIColor.redColor;
    cardNumberView.property = property;
```

9.3 Stage 3: Initiate the payment

PayButton

Used to collect the card data from card components above and to submit successful payment. With a few simple steps it also has the capability to perform Direct Pay without the need for the card component, see the next section.

PayfortPayButton methods

```
/**
     Update Request After doing Setup
     - Parameter request: a new request dictionary
     */
    public func updateRequest(request: [String: String])

 /**
     Responsible for Save token or not
     - Parameter enabled: a new bool value
     */
    public func isRememberEnabled(_ enabled: Bool)
```

9.4 Sample code

In this section we illustrate how you use the PayButton using sample code for Swift and Objective-C.

Swift

```
@IBOutlet weak var payButton: PayButton!

    let builder = PayComponents(cardNumberView: cardNumberView, expiryDateView:
expiryDateView, cvcNumberView: cvcNumberView, holderNameView: holderNameView,
rememberMe: saveCardSwitch.isOn)

    payButton.setup(with: request, enviroment: enviroment, payComponents:
builder, viewController: self) {
    // Process started
    } success: { (requestDic, responeDic) in
    // Process success
    } faild: { (requestDic, responeDic, message) in
    // Process faild
    }
```

Objective-C

```
@property (nonatomic, weak) IBOutlet PayButton *payButton;

    PayComponents *builder = [[PayComponents alloc]
initWithCardNumberView:cardNumberView expiryDateView: expiryDateView
cvcNumberView: cvcNumberView, holderNameView: holderNameView rememberMe:
saveCardSwitch.on];

    [payButton setupWithRequest: request
                      enviroment: enviroment
                  payComponents: builder
          currentViewController: self
                        Success:^(NSDictionary *requestDic, NSDictionary
*responeDic) {
    } Canceled:^(NSDictionary *requestDic, NSDictionary *responeDic) {
    } Faild:^(NSDictionary *requestDic, NSDictionary *responeDic, NSString
*message) {
    }];
```

10 Using Direct Pay

The Direct Pay feature enables Amazon Payment Services merchants to securely process e-commerce transactions using tokenized payment card details.

For customers that already supplied their payment card details in a previous transaction and where a card token was generated, customers need to just provide the card_security_code to complete their purchase.

The card token and provided card security code can be sent to the Amazon Payment Services iOS mobile SDK to complete the customer purchase through Direct Pay operation to complete the order placement using eCommerce channel.

Note you can use Direct Pay both with the standard payment UI or with a customized payment UI.

10.1 Direct Pay sample code

Swift

```
@IBOutlet weak var directPayButton: PayButton!

    // request should has all mandatory params and also you need to send
card_security_code, token_name

    let request = ["amount" : "1000",
                   "command" : "AUTHORIZATION",
                   "currency" : "AED",
                   "customer_email" : "test@payfort.com",
                   "language" : "en",
                   "card_security_code" : "123",
                   "token_name" : "payfort",
                   "merchant_reference" : "merchant reference",
                   "sdk_token" : "token"]

    directPayButton.setup(with: request, enviroment: enviroment, viewController:
self) {
    // Process started
    } success: { (requestDic, responeDic) in
    // Process success
    } faild: { (requestDic, responeDic, message) in
    // Process faild
    }
```

Objective-c

```
@property (nonatomic, weak) IBOutlet PayButton *directPayButton;

    NSMutableDictionary *requestDictionary = [[NSMutableDictionary alloc]init];
    [requestDictionary setValue:@"10000" forKey:@"amount"];
    [requestDictionary setValue:@"AUTHORIZATION" forKey:@"command"];
    [requestDictionary setValue:@"USD" forKey:@"currency"];
    [requestDictionary setValue:@"email@domain.com" forKey:@"customer_email"];
    [requestDictionary setValue:@"en" forKey:@"language"];
    [requestDictionary setValue:@"112233682686" forKey:@"merchant_reference"];
    [requestDictionary setValue:@"payfort" forKey:@"token_name"];
    [requestDictionary setValue:@"123" forKey:@"card_security_code"];
    [requestDictionary setValue:@"token" forKey:@"sdk_token"];

    [directPayButton setupWithRequest: request
                           enviroment: enviroment
                 currentViewController: self
                              Success:^(NSDictionary *requestDic, NSDictionary
*responeDic) {
    } Canceled:^(NSDictionary *requestDic, NSDictionary *responeDic) {
    } Faild:^(NSDictionary *requestDic, NSDictionary *responeDic, NSString
*message) {
    }];
```


11 Amazon Payment Services iOS SDK transaction feedback

While a transaction is processed, we will send a response directly to your direct transaction feedback URL. In theory, direct response feedback cannot be interrupted unless the URL you provided for responses is not functional at the time of the response.

We do this so that your server receives a response even if your customer does not successfully redirect to the return URL on your website, which may happen if your customer's browser or connection fails during the transaction.

11.1 Receiving transaction feedback

There are two ways in which you receive transaction feedback:

Direct transaction feedback. Amazon Payment Services sends an immediate payment processing response whenever a transaction is completed.

You can rely on this response for transaction feedback even where your user closed the browser before getting redirected successfully to the redirection URL or where your user was not redirected due to a drop in the internet connection.

Notification feedback. Were we need to provide you with the status of a transaction once it is received. In other words, we send notification feedback to alert you to any changes in the status of a transaction.

For example, if the transaction was pending due to the unavailability of any party to the transaction, the final update will be pushed to your notification feedback endpoint.

Notification feedback deals with a wide range of scenarios and it is critical that your website is configured to receive notification feedback correctly. For example, transaction feedback can alert you to any transactions that were stuck in "uncertain" status, but which have recovered to final status.

Direct feedback allows you to subscribe to transaction updates for uncertain transactions whenever you process a payment. It is a method for receiving the transaction response automatically once the transaction had completed or if there was an update.

11.2 Registering Transaction Feedback URLs

To receive transaction feedback, you must register with Amazon Payment Services the transaction feedback URLs you set up on your server. Follow these steps:

1. Log in to your back-office account.
2. Select the active channel under **Integration Settings** > **Technical Settings**.
3. Enter your **Direct Transaction Feedback URL** and **Notification Transaction Feedback URL**.
4. Click the “Save Changes” button.

11.3 Transaction Feedback Implementation

We will send the response via HTTP POST request in POST form format to your webhook URL. The submission type can be changed from POST form to JSON or XML in your APS account settings. We only permit configuring URLs in HTTPS for direct feedback and for notification feedback.

To acknowledge receiving the direct feedback and notification successfully, the webhook URL must return a 2xx or 302 HTTP status. In case the URL returns different HTTP responses, our server will keep retrying up to 10 times until we receive a success response. We wait ten seconds between retries.

You can change and configure the retry mechanism attributes to increase or decrease the number of retries, time between retries and the grace period before we send the notifications.

You must create endpoints that accept direct transaction feedback and notification feedback from us via a webhook. You can customize how transaction feedback works -- for example, by including the grace period before a direct feedback notification is sent, and the time elapsed between retrying feedback submission.

To customize transaction feedback, email integration-ps@amazon.com. You can request to change the submission type to JSON or XML. You can also change the grace period or the time interval between the retries – please contact us.

NOTE: You can check the direct and notification feedback logs in your back office account to check the details related to the submission like the Transaction Feedback URL which was triggered, The response which our system pushed, the response Code and Status retuned from your Transaction Feedback URL.

NOTE: The specifics of the data will differ based upon the financial operation that has been processed. Your code must be able to accommodate different data.

12 Validate API

Amazon Payment Services offers an API to request validation of the payment card details without performing an actual transaction.

ValidateAPI can help you to initiate an API request in order to validate the input parameters values, this will reduce the possibility of encountered API errors due to wrong user inputs before processing the actual payment card transaction.

Swift

```
let request = ["amount" : "1000",
               "command" : "AUTHORIZATION",
               "currency" : "AED",
               "customer_email" : "test@payfort.com",
               "language" : "en",
               "sdk_token" : "token"]

   payFortController.callValidateAPI(with: request) {
                // Process started
               } success: {
                // Process success
               } faild: { (requestDic, responeDic, message) in
                // Process faild
               })
```

Objective-c

```
NSMutableDictionary *request = [[NSMutableDictionary alloc]init];
    [requestDictionary setValue:@"10000" forKey:@"amount"];
    [requestDictionary setValue:@"AUTHORIZATION" forKey:@"command"];
    [requestDictionary setValue:@"USD" forKey:@"currency"];
    [requestDictionary setValue:@"email@domain.com" forKey:@"customer_email"];
    [requestDictionary setValue:@"en" forKey:@"language"];
    [requestDictionary setValue:@"112233682686" forKey:@"merchant_reference"];
    [requestDictionary setValue:@"token" forKey:@"sdk_token"];


    [payFortController callValidateAPIWithRequest: request
                                       ShowLoader:^() {
    } Success:^(NSDictionary *requestDic, NSDictionary *responeDic) {
    } Faild:^(NSDictionary *requestDic, NSDictionary *responeDic, NSString
*message) {
    }];
```

NOTE: When you make use of the Validate API call you still need to generate a mobile SDK token even though you are not processing a transaction.

13 Migrating from version 2.3 of the SDK

In this section we outline the steps you need to take to migrate from version 2.3 of the Amazon Payment Services iOS SDK to the latest release. To complete the migration follow these steps:

1. Remove PayFortSDK.framework and PayFortSDK.bundle

2. Use CocoaPods or manual integration – by dragging the PayFortSDK.xcframework into your project

3. Import PayFortSDK.xcframework:

   Objective-C

   The replace import is shown below

```
//From
#import <PayFortSDK/PayFortSDK>
//To
#import <PayFortSDK/PayFortSDK-Swift.h>
```

   Swift import

   If you're using Swift you can import PayFortSDK directly on your controller, there is no need to use Bridging-Header any more

```
import PayFortSDK
```


4. Change callPayFortWithRequest method as below:

Objective-C

```
//From
[payFort callPayFortWithRequest:myRequest currentViewController:self
                Success:^(NSDictionary *requestDic, NSDictionary *responeDic) {

                } Canceled:^(NSDictionary *requestDic, NSDictionary *responeDic)
{

                } Faild:^(NSDictionary *requestDic, NSDictionary *responeDic,
                }];

//TO
[payFort callPayFortWithRequest:myRequest currentViewController:self
            success:^(NSDictionary *requestDic, NSDictionary *responeDic) {

            } canceled:^(NSDictionary *requestDic, NSDictionary *responeDic) {

            } faild:^(NSDictionary *requestDic, NSDictionary *responeDic,
            }];
```

Swift

Chane the callPayFortWithRequest request param from `NSMutableDictionary` to `Dictionary`

5. Change callPayFortForApplePayWithRequest method as below:

Objective-C

```
//From
[payFort callPayFortForApplePayWithRequest:myRequest
                    applePayPayment:payment
              currentViewController:self
                            Success:^(NSDictionary *requestDic, NSDictionary
*responeDic) {

                            }
                              Faild:^(NSDictionary *requestDic, NSDictionary
*responeDic, NSString *message) {

completion(PKPaymentAuthorizationStatusFailure);

                              }];

//To

[payFort callPayFortForApplePayWithRequest:myRequest
                    applePayPayment:payment
              currentViewController:self
                            success:^(NSDictionary *requestDic, NSDictionary
*responeDic) {

                            }
                              faild:^(NSDictionary *requestDic, NSDictionary
*responeDic, NSString *message) {

completion(PKPaymentAuthorizationStatusFailure);

                              }];
```

Swift

Change **callPayFortForApplePay** request param from **NSMutableDictionary** to **Dictionary**

6. Change callPayFortForApplePay method as below

Objective-C

```
//From
paycontroller.callPayFortForApplePay(withRequest: ["":""],
                                                applePay: payment,
                                                currentViewController: self) {
(requestDic, responeDic) in
                                                } faild: { (requestDic, responeDic,
message) in
        }

//To
paycontroller.callPayFortForApplePay(withRequest: ["":""],
                                                applePayPayment: payment,
                                                currentViewController: self) {
(requestDic, responeDic) in
                                                } faild: { (requestDic, responeDic,
message) in
        }
```

Swift

No changes necessary

7. Change environment as follows:

Objective-C

From **KPayFortEnviromentSandBox** to **PayFortEnviromentSandBox**

From **KPayFortEnviromentProduction** to **PayFortEnviromentProduction**

Swift

From **KPayFortEnviromentSandBox** to **.sandbox**

From **KPayFortEnviromentProduction** to **.production**

8. Modify the IsShowResponsePage property:

   Objective-C:

   From IsShowResponsePage to isShowResponsePage

   Swift

   No changes needed

9. HideLoading property:

   Objective-C:

   From HideLoading to hideLoading

   Swift

   No changes needed.

14 Appendix: About the Software

14.1 About this software

The Amazon Payment Services iOS mobile SDK allows merchants to securely integrate payment functionality into a native iOS app. It allows merchants to easily accept in-app payments.

Instead of the traditional, time-consuming, and complex way of being redirected to the mobile browser to complete the payment, in-app payments can easily be offered thanks to the iOS Mobile SDK.

In turn, this gives the merchants' customers a smooth, pleasing user experience thanks to in-app payment functions through the native applications.

14.2 Supported Platforms

The Amazon Payment Services iOS SDK supports IOS 12.2+ and Xcode 11.0 and above.

14.3 Localization

You can use both English and Arabic when you implement the iOS SDK.

14.4 Screen Orientation

Currently, portrait is the only orientation supported within the Amazon Payment Services mobile SDK – unless you build a customized payment integration.

14.5 Supported Payment Methods

Using the iOS SDK the merchant can process debit or credit card transactions only.

14.6 Supported Payment Options

The supported credit card payment options are VISA, MASTERCARD, American Express (AMEX), MADA and MEEZA.