PaywithEaseBuzz iOS SDK

The PaywithEaseBuzz iOS SDK makes it quick and easy to build an excellent payment experience in your iOS app. We provide powerful and customizable UI screens and elements that can be used out-of-the-box to collect your users' payment details. We also expose the low-level APIs that power those UIs so that you can build fully custom experiences. See our iOS Integration Guide to get started!

This SDK allows you to integrate payments via PaywithEaseBuzz into your iOS app. It currently supports following modes of payments:

1. Credit / Debit Cards
2. Netbanking
3. Wallets/Cash Cards
4. Debit + ATM Pin
5. UPI
6. Ola-Money

Features

Simplified Security: With fraud detection and prevention mechanism, we provide the most protective layer for each transaction.We are also PCI-DSS compliant.

Native UI: We provide out-of-the-box native screens and elements so that you can get started quickly without having to think about designing the right interfaces.

Requirements

The PaywithEaseBuzz iOS SDK is compatible with apps supporting iOS 11 and above and requires Xcode 10 to build from source.

NOTE: Certain card payments may not be supported on iOS 12.x and below due to a cookie management issue at certain payment switches. For a seamless experience, please use iOS 13.x and above.

Integration

Please see our iOS Integration Guide which explains everything from SDK installation and more.

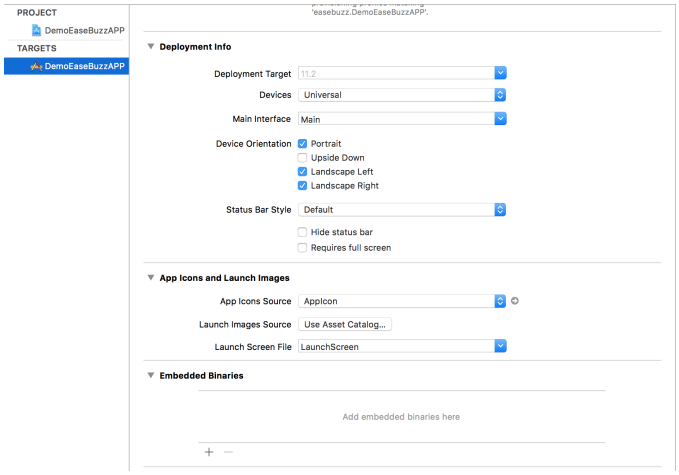
Integration

PaywithEaseBuzz Payment kit integration(iOS)

---

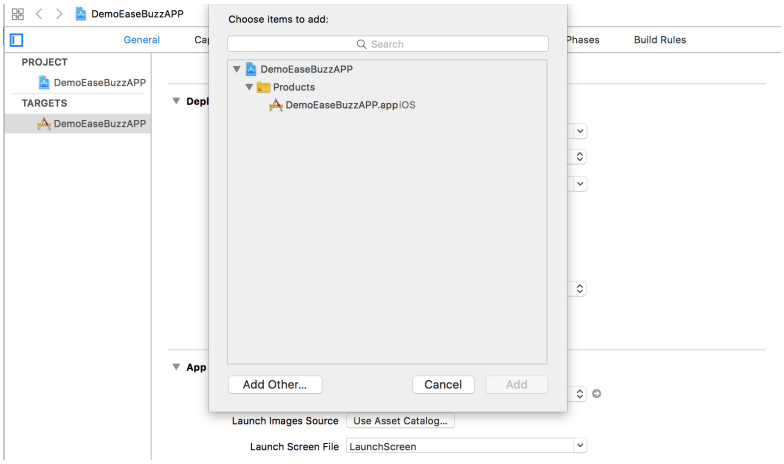
1. Sdk Configuration

1. Copy Easebuzz.xcframework of your application in embedded binaries,Refer Screen 1



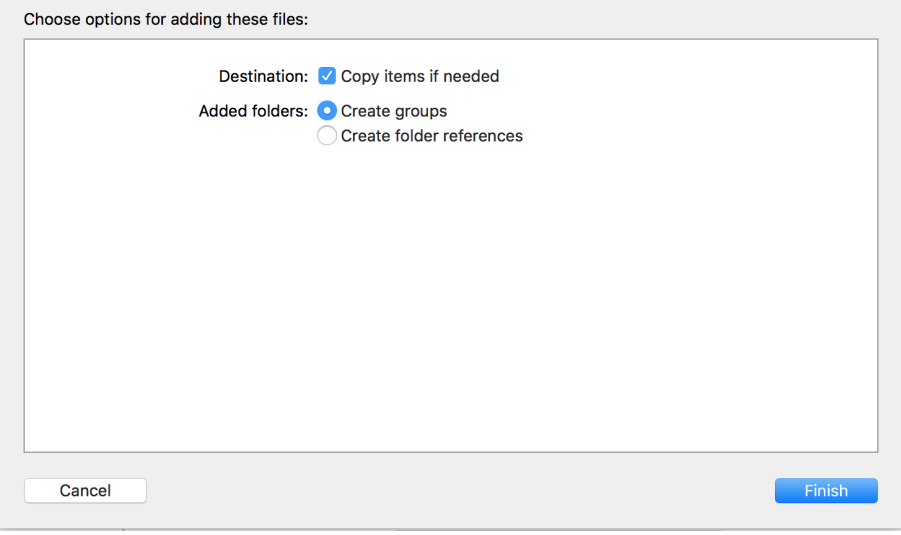
Screen 1: embedded framework

2. Press + and add framework using ‘Add other. ’ button refer Screen 2



Screen 2: Add Other

3. Browse framework: file from your folder and select ‘copy items if needed’. Refer Screen 3.



Screen 3: Copy Items if needed

4. Set Always embed swift standard libraries to YES from project build settings

ALWAYS_EMBED_SWIFT_STANDARD_LIBRARIES = YES

5. To simply disable ATS, you can follow this steps by open **Info.plist**, and add the following lines:

```
<key>NSAppTransportSecurity</key>
```

```
<dict> <key>NSAllowsArbitraryLoads</key>

<true/>

</dict>
```

2. Initiate Payment Request

1. Generate access key using the Initiate Payment API at your backend.
(It is mandatory to integrate the Initiate Payment API at Backend only)
Initiate Payment API Doc : https://docs.easebuzz.in/api/initiate-payment

2. Import Easebuzz module in your ViewController

3. Set Delegate to your ViewController as PayWithEasebuzzCallback and confirm the delegate.

4. On clicking the Pay button from your app, you need to call initiatePaymentAction method. This function included one dictionary with access key and payment mode required parameters. And Pass this dictionary to the Payment class. Before calling the easebuzz kit we need to first generate an access key as mentioned in the first steps. Refer below code for calling payment gateway.

Sample code :

Swift:

```
import Easebuzz

func initiatePaymentAction() {

    let orderDetails = [
        "access_key": "Pass access key generated in step 1",
        "pay_mode": "Pass here 'test' or 'production' mode"
        ] as [String:String]

    let payment = Payment.init(customerData: orderDetails)
    let paymentValid = payment.isValid().validity
    if !paymentValid {
        print("Invalid records")
    }else{
        PayWithEasebuzz.setUp(pebCallback: self )
        PayWithEasebuzz.invokePaymentOptionsView(paymentObj: payment, isFrom: self)
    }
}
```

Objective C:

```
 @import Easebuzz;

@property (nonatomic,retain) Payment *payment;

- (void)initiatePaymentAction {
       NSDictionary *orderDetails =
       @{ @"access_key": @"Pass access key generated in step 1",
            @"pay_mode":@"test"
         };

   _payment = [[Payment alloc]initWithCustomerData:orderDetails];

       BOOL paymentValid = _payment.isValid;
       if (!paymentValid) {
          NSLog(@"Invalid Print");
       }else{
          [PayWithEasebuzz setUpWithPebCallback:self];
          [PayWithEasebuzz invokePaymentOptionsViewWithPaymentObj:_payment isFrom:self];
       }
   }
```

Note : Please do not change the name of the parameter from the orderDetails dictionary mentioned in above code. The datatype of the value can be changed according to the value to be sent.

Description of the values of parameters are given at the end of the document.

3. Handle Payment Response:

You can receive the result and response in the callback delegate method.

Swift:

```
  func PEBCallback(data: [String : AnyObject]) {

    }
```

Objective C:

```
- (void)PEBCallbackWithData:(NSDictionary<NSString *,id> * _Nonnull)data {
}
```

This result indicates the status of the transaction as success or failed the detailed code is described below.

The **result** can be ,

1. "payment_successfull"
2. "payment_failed"
3. “txn_session_timeout”
4. "back_pressed"
5. "user_cancelled"
6. "error_server_error"
7. "error_noretry"
8. "invalid_input_data"
9. "Retry_fail_error"
10. “trxn_not_allowed”

Swift:

```
   func PEBCallback(data: [String : AnyObject]) {  // Handle Payment response_response key, It is a
anyObject
       let payment_response = data["payment_response"]
       print(payment_response ?? "")

       if payment_response as? [String:Any] != nil {
          // payment_response is Json Response
          print("Json response")
       }else{
          print("String response")
          //  payment_response is String
       }

       // Handle result Key : It should be in string
       let result = data["result"] as! String
```

Objective C:

```
- (void)PEBCallbackWithData:(NSDictionary<NSString *,id> * _Nonnull)data {

    NSDictionary *responseDict = [data valueForKey:@"payment_response"];
    NSLog(@"%@",responseDict);

    // Handle Payment response_response key, Its should be anyObject.
    BOOL isValid = [NSJSONSerialization isValidJSONObject:responseDict];
    if (isValid) {
     // payment_response is Json Response
    }else{
        //  payment_response is String
    }

    NSString *strResult = [data valueForKey:@"result"];
```

1. **Payment result values description and equivalent constants.**

1.*StaticDataModel.***TXN_SUCCESS_CODE**
*StaticDataModel.***TXN_SUCCESS_CODE** *is a string constant and its value is "*payment_successfull*"*
result contains this value, if the payment transaction completed successfully.

2.*StaticDataModel.***TXN_TIMEOUT_CODE**
*StaticDataModel.***TXN_TIMEOUT_CODE** *is a string constant and its value is "*txn_session_timeout*"*
result contains this value, if the payment transaction failed because of the transaction time out.

*3.StaticDataModel.***TXN_BACKPRESSED_CODE**
*StaticDataModel.***TXN_BACKPRESSED_CODE** *is a string constant and its value is "*back_pressed*"*
result contains this value, if the user pressed the back button on coupons Activity.

4.*StaticDataModel.***TXN_USERCANCELLED_CODE**
*StaticDataModel.***TXN_USERCANCELLED_CODE** *is a string constant and its value is "*user_cancelled*"*
result contains this value, if the user pressed the cancel button during the payment process.

*5.StaticDataModel.***TXN_ERROR_SERVER_ERROR_CODE**
*StaticDataModel.***TXN_ERROR_SERVER_ERROR_CODE** *is a string constant and its value is "*error_server_error*"*
result contains this value, if the server side error occurred during payment process.

6.*StaticDataModel.***TXN_ERROR_TXN_NOT_ALLOWED_CODE**
*StaticDataModel.***TXN_ERROR_TXN_NOT_ALLOWED_CODE** *is a string constant and its value is "*trxn_not_allowed*"*

7.*StaticDataModel.***TXN_BANK_BACK_PRESSED_CODE**
*StaticDataModel.***TXN_BANK_BACK_PRESSED_CODE** *is a string constant and its value is "bank_back_pressed"*
The result contains this value if the user presses the back button on the bank page.

*8.StaticDataModel.***TXN_INVALID_INPUT_DATA_CODE**
*StaticDataModel.***TXN_INVALID_INPUT_DATA_CODE** *is a string constant and its value is "*invalid_input_data'

result contains this value if payment request input parameters are not valid.

9. *StaticDataModel.***TXN_FAILED_CODE**
   *StaticDataModel.***TXN_FAILED_CODE** *is a string constant and its value is "*payment_failed*" .*
   result contains this value if payment fails from the bank side.

10. *StaticDataModel.***TXN_ERROR_NO_RETRY_CODE**
    *StaticDataModel.***TXN_ERROR_NO_RETRY_CODE** *is a string constant and its value is "*error_noretry*".*
    This result can be considered as a failed payment.

11. *StaticDataModel.***TXN_ERROR_RETRY_FAILED_CODE**
    *StaticDataModel.***TXN_ERROR_NO_RETRY_CODE** *is a string constant and its value is "*retry_fail_error*".*
    This result can be considered as a failed payment.

Your app can get the detailed response as follows :

Success Response :


```
{
txnid : '1001',
firstname : 'John Doe',
email : 'johndoe@gmail.com',
phone : '7767819428',
key : 'DF3252FDSF',
mode : 'DC',
status : 'success',
unmappedstatus : 'failed',
cardCategory : 'domestic',
addedon : '2016-07-22 17:17:08',
payment_source : 'Easebuzz',
PG_TYPE : 'SBIPG',
bank_ref_num : '',
bankcode : 'MAST',
error : 'E600',
error_msg : 'Bank denied transaction on card.',
name_on_card : 'John',
cardnum : '519620XXXXXX7840',
issuing_bank : '',
card_type : '',
easepayid : 'H5T2RYZKW',
amount : '100.00',
net_amount_debit : '100.00',
cash_back_percentage : '50',
deduction_percentage : '2.50',
productinfo : 'Tshirt',
udf10 : '',
udf9 : '',
```

```
  udf8 : '',
  udf7 : '',
  udf6 : '',
  udf5 : '',
  udf4 : '',
  udf3 : '',
  udf2 : '',
  udf1 : '',
  hash :
'ce2d0588f8648c62db86475d343d3433d00b87827502c676a093730f04cec5fea2eb0e8bb4f47fcdea955f61b674171f1
93c883686d2da42300d00e921a217c3'
}
```


Failed Response :


```
{
  txnid : '1001',
  firstname : 'John Doe',
  email : 'johndoe@gmail.com',
  phone : '7767819428',
  key : 'DF3252FDSF',
  mode : 'DC',
  status : 'failure',
  unmappedstatus : 'failed',
  cardCategory : 'domestic',
  addedon : '2016-07-22 17:17:08',
  payment_source : 'Easebuzz',
  PG_TYPE : 'SBIPG',
  bank_ref_num : '',
  bankcode : 'MAST',
  error : 'E600',
  error_msg : 'Bank denied transaction on card.',
  name_on_card : 'John',
  cardnum : '519620XXXXXX7840',
  issuing_bank : '',
  card_type : '',
  easepayid : 'H5T2RYZKW',
  amount : '100.00',
  net_amount_debit : '100.00',
  cash_back_percentage : '50',
  deduction_percentage : '2.50',
  productinfo : 'Tshirt',
  udf10 : '',
  udf9 : '',
```

```
  udf8 : '',
  udf7 : '',
  udf6 : '',
  udf5 : '',
  udf4 : '',
  udf3 : '',
  udf2 : '',
  udf1 : '',
  hash :
'ce2d0588f8648c62db86475d343d3433d00b87827502c676a093730f04cec5fea2eb0e8bb4f47fcdea955f61b674171f1
93c883686d2da42300d00e921a217c3'
}
```

4.Request/Response description:

1.**txnid** : Mandatory parameter. Data type should be String.This is the transaction id.

2.amount : Mandatory parameter. Data type should be String . This is the amount of the transaction.

3.**productinfo** : Mandatory parameter. Data type should be String . This parameter should contain a brief product description

4.**firstname**: Mandatory parameter. Data type should be String. This is the name of the customer who is doing the transaction.

5.**email** : Mandatory parameter. Data type should be String . This is the email id of the customer who is doing the transaction.

6.phone : Optional parameter. Data type should be String. Phone number of the customer.

7.**key** : Mandatory parameter. Data type should be String. This parameter is the unique Merchant Key provided by Easebuzz for your merchant account. The Merchant Key acts as the unique identifier (primary key) to identify a particular Merchant Account for our interpretation. While posting the data to us, you need to send this Merchant Key value.

8. **udf1** : Optional parameter. Data type should be String. User defined field 1 – This parameter has been made for you to keep any information corresponding to the transaction, which may be useful for you to keep in the database. UDF1-UDF5 fields are for this purpose only. It's completely for your usage and you can post any string value in this parameter. udf1-udf5 are optional parameters and you may use them only if needed.

9.**udf2** : Optional parameter. Data type should be String. User defined field 2 – User Defined field.

10.**udf3** : Optional parameter. Data type should be String. User defined field 3 – User Defined field.

11.**udf4** : Optional parameter. Data type should be String. User defined field 4 – User Defined field.

12.udf5 : Optional parameter. Data type should be String. User defined field 5 – User Defined field.

13.address1 : This parameter is an optional parameter that can be used to pass the address. Data type should be String. **Allowed characters :** Alpha-numeric characters, comma, space, dot(period).

14.address2 : This parameter is an optional parameter that can be used to pass the address. Data type should be String. **Allowed characters :** Alpha-numeric-characters, comma, space, dot(period).

15.city : This parameter is an optional parameter that can be used to pass the city of the customer. Data type should be String.

16.state : This parameter is an optional parameter that can be used to pass the state of the customer. Data type should be String.

17.country : This parameter is an optional parameter that can be used to pass the country of the customer. Data type should be String.

18.zipcode : This parameter is an optional parameter that can be used to pass the zipcode of a customer. Data type should be String.

19.hash : Mandatory parameter. Data type should be String. Hash generation details are given below.

20.**unique_id**: Mandatory parameter. This is a customer's unique id.

21.pay_mode : Mandatory parameter. Data type should be String. This parameter is to specify the integration of test sdk kit or live sdk kit. Its value can be either "**test**" or "**production**".

20.**isMobile**: Mandatory parameter. This is always "1" for mobile apps.

21.**show_payment_mode**: This parameter is an optional parameter.Pass the require payment values with comma separated like below

show_payment_mode: 'NB,DC,CC,Debit+ATM Pin,MW,UPI,OM,EMI'

Note : If the value of **pay_mode** is **"test"** then you must use test key and salt. If the value of **pay_mode** is **"production"** then you must use production key and salt.