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 9 and above and requires Xcode 9 to build from source.

Integration

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

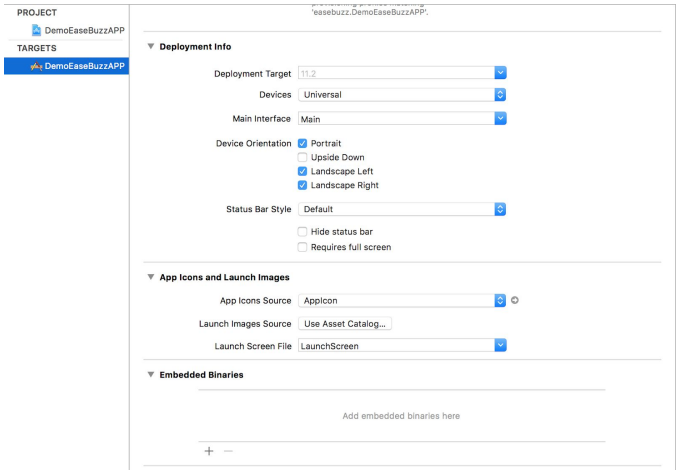
Integration

PaywithEaseBuzz Payment kit integration(iOS)

---

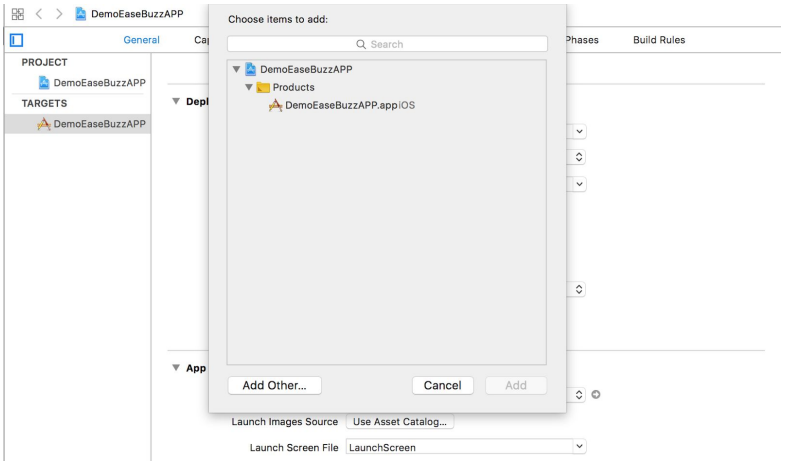
1. Sdk Configuration

1. Copy easebuzz.framework of your application in embedded binaries,Refer Screen 1



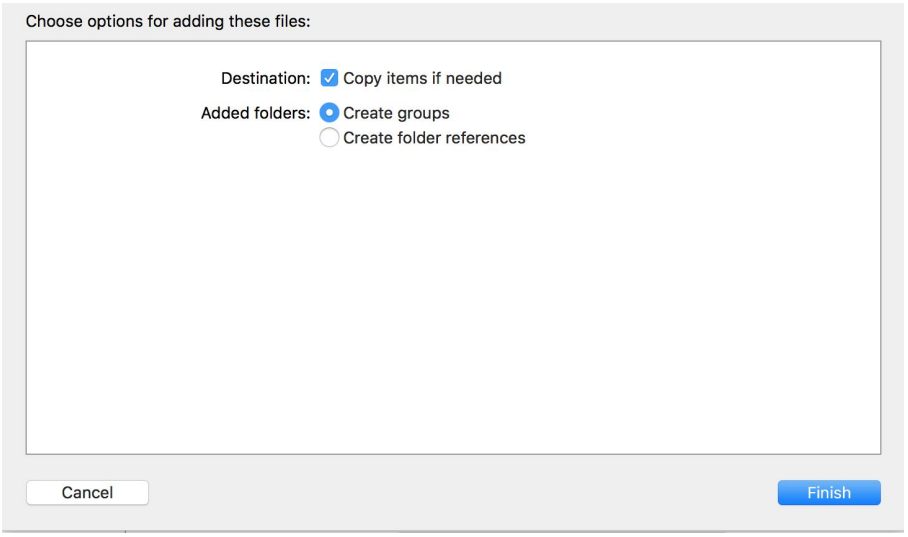
Screen 1: embedded framework

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. Import Easebuzz module in your ViewController

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

3. On click Pay button from your app, you need to call initiatePaymentAction method. This function included one dictionary with all parameters required for Easebuzz kit. And Pass this dictionary to the Payment class. Before call easebuzz kit we first check parameter validation. If required parameters are valid then call the easebuzz payment gateway kit using invokePaymentOptionsView methods. Refer below code for calling payment gateway.

Sample code :

Swift:

```
import Easebuzz

 func initiatePaymentAction() {

    let orderDetails = [
        "txnid": "Pass transaction Id here",
        "amount":"Payment Amount",
        "productinfo":"Your Product Information",
        "firstname":"testUserName",
        "email":"testEmail.com",
        "phone":"1234567899",
        "key":"Pass key here provided by easebuzz team",
        "udf1":"udf1",
        "udf2":"udf2",
        "udf3":"udf3",
        "udf4":"udf4",
        "udf5":"udf5",
        "address1":"",
        "address2":"",
        "city":"",
        "state":"",
        "country":"",
```

```
        "zipcode":"",
        "isMobile":"1",
        "unique_id":  "customer unique id",
        "hash": "Create hash as per below procedure in parameter details",
        "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 =
    @{ @"txnid": @"txasd",
       @"amount":@"Amount",
       @"productinfo": @"Product info",
       @"firstname": @"Name",
       @"email": @"testEmail.com",
       @"phone": @"32424242442",
       @"key": @"key",
       @"udf1": @"udf1",
       @"udf2": @"udf2",
       @"udf3": @"udf3",
       @"udf4": @"udf4",
       @"udf5": @"udf5",
       @"address1": @"Adrress Here",
       @"address2": @"Adress here",
       @"city": @"",
       @"state": @"",
       @"country": @"",
       @"zipcode": @"",
       @"isMobile": @"1",
       @"unique_id":  @"Unique Id",
       @"hash": @"Create hash as per below procedure given in parameter details",
       @"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 change according to the value to be sent.

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

3. Handle Payment Response:

You can recieved the result and response in the call back delegate method.

Swift:

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

    }
```

Objective C:

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

This result indicates the status of 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”

Your app can get the detailed response as follows :

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"];
```

4. Remove Unused Architectures:

Easebuzz is custom universal framework and for in on production, we we need to remove unused architectures. Because Apple doesn't allow the application with unused architectures to the App Store.

Select the Project, Choose Target → Project Name → Select Build Phases → Press "+" → New Run Script Phase → Name the Script as "Run Script". Refer Screen4.

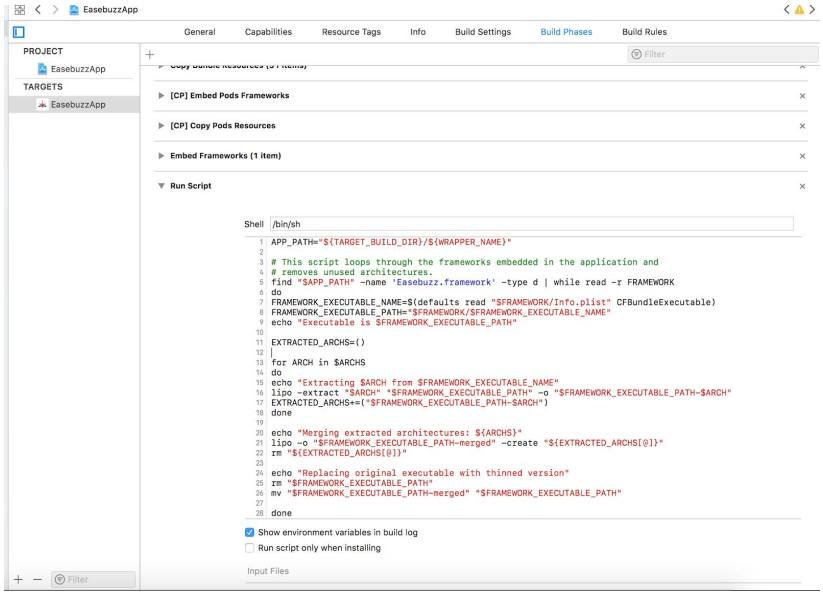
Always this script should be placed below "Embed Frameworks".

Always build the project for both simulator and generic device build before start the archives.

To achieve this, I wrote Custom Run Scripts,

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name 'Easebuzz.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutable)
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

EXTRACTED_ARCHS=()

for ARCH in $ARCHS
do
echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

echo "Merging extracted architectures: ${ARCHS}"
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
rm "${EXTRACTED_ARCHS[@]}"

echo "Replacing original executable with thinned version"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```

This run script removes the unused Simulator architectures only while pushing the Application to the App Store.



Screen4: RunScript

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 name of the customer who is doing the transaction.

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

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

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

18.**zipcode** : This parameter is an optional parameter that can be used to pass the zipcode of 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 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.

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

Hash generation (sha512):

Hash is a mandatory parameter – used specifically to avoid any tampering during the transaction.
It is **sha512** encrypted string. And hash sequence is mentioned below.

Hash sequence:

key|txnid|trxn_amount|productinfo|firstname|email_id|udf1|udf2|udf3|udf4|udf5|||||salt|key

Generate the sha512 of above hash sequence. and pass as a **hash** parameter.

Note:

1. Make sure the parameters that you are passing to Easebuzz SDK object (dictionary) should be exactly the same which has been used to generate the hash.
For example.
1. If you used demo@gmail.com to generate the hash and Demo@gmail.com is passed to SDK intent, Then It will throw an error.
2. If you appended space to any parameter while generating a hash and passed the space appended parameter to SDK intent then It will throw an error.
3. If you are using amount 1.00, Then It will throw an error. Please use amount like 1.0 (Complete number's amount). The amount like 1.12 will work fine.
4. If you are generating hash on mobile side,then you can use easebuzz sdk global function for generating hash. No need to add separate library in your project.

"hash": strHash.sha512()

Suggestion: It would be more secure if the hash generation process is done at back end (Server Side)

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 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 occured 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" result contains this value if user press the back button on 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 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 failed payment.*

2. Response detail.

1.Success response json:


```
{
 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 : 'ce2d0588f8648c62db86475d343d3433d00b87827502c676a093730f04cec5fea2eb0e8bb'
    }
```


2. Failed response json:


```
    {
        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 : 'T5T2RYZKW',
```

```
        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 :'ce2d0588f8648c62db86475d300b87827502c676a093730f04cec5fea2ebb4f47fcdea955f61b6'
    }
```