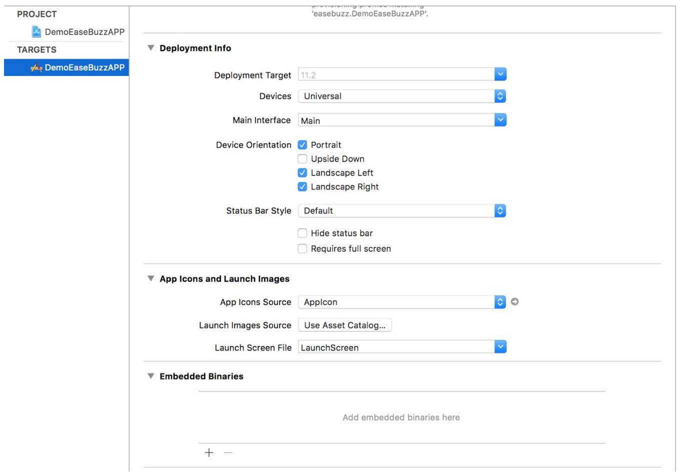
Detailed procedure

PaywithEaseBuzz Payment kit integration(iOS)

---

1. Browse Framework:

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



PROJECT
DemoEaseBuzzAPP
TARGETS
DemoEaseBuzzAPP

'easebuzz.DemoEaseBuzzAPP'.

Deployment Info

Deployment Target: 11.2
Devices: Universal
Main Interface: Main
Device Orientation: ✓ Portrait, ☐ Upside Down, ✓ Landscape Left, ✓ Landscape Right
Status Bar Style: Default
☐ Hide status bar
☐ Requires full screen

App Icons and Launch Images

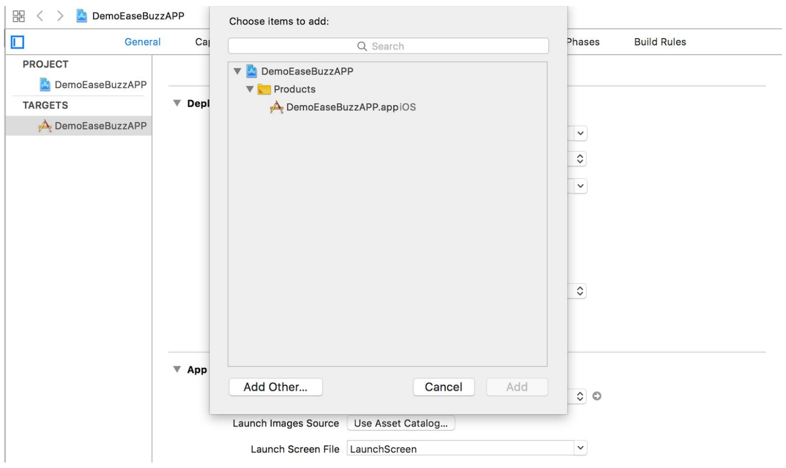
App Icons Source: AppIcon
Launch Images Source: Use Asset Catalog...
Launch Screen File: LaunchScreen

Embedded Binaries

Add embedded binaries here

Screen 1: embedded framework

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



DemoEaseBuzzAPP
General
Phases
Build Rules
PROJECT
DemoEaseBuzzAPP
TARGETS
DemoEaseBuzzAPP

Choose items to add:

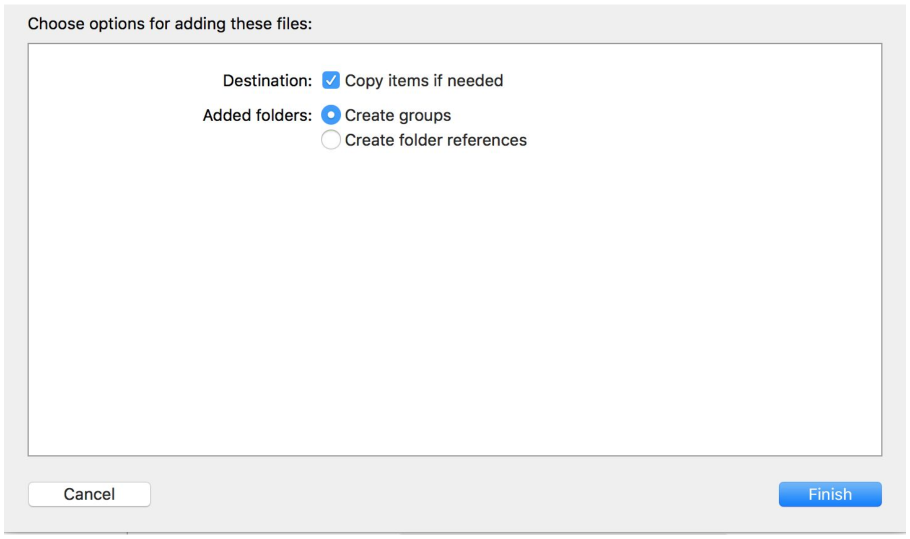
Search
DemoEaseBuzzAPP
Products
DemoEaseBuzzAPP.app iOS

Add Other... | Cancel | Add

Launch Images Source: Use Asset Catalog...
Launch Screen File: LaunchScreen

Screen 2: Add Other

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


Choose options for adding these files:

Destination: [✓ Copy items if needed]

Added folders: [✓ Create groups] Create folder references

Screen 3: Copy Items if needed

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

```
//:configuration = Debug

ALWAYS_EMBED_SWIFT_STANDARD_LIBRARIES = YES

//:configuration = Release

ALWAYS_EMBED_SWIFT_STANDARD_LIBRARIES = YES

//:completeSettings = some

ALWAYS_EMBED_SWIFT_STANDARD_LIBRARIES
```

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. Integration.

Your Easebuzz Kit configuration is done. Follow on for steps to integrate.

1. Import Easebuzz module in your ViewController.

Swift:	import Easebuzz
Objective C:	@import Easebuzz;

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

Swift:

```
class ViewController: UIViewController,PayWithEasebuzzCallback {
   func PEBCallback(data: [String : AnyObject]) {

   }  }
```

Objective C:

```
@interface ViewController ()<PayWithEasebuzzCallback>
{
}

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

@end
```

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.

Swift:

```
   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",
         "salt": "Pass salt here provided by easebuzz team",
         "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:

```
@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",
          @"salt": @"Salt",
          @"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 each parameter :

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.salt : Mandatory parameter. Data type should be String. This parameter is provided by Easebuzz for your merchant account.

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.

3. Handling the response of the payment process.

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 :

let payment_response = data["payment_response"]

This response is a json string and can be parsed to get the details about the ongoing transaction.

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'
}
```

OnCallback Detailed code and description:

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

       switch result {
       case StaticDataModel.TXN_SUCCESS_CODE:
          break
          //StaticDataModel.TXN_SUCCESS_CODE is a string constant and its value is
"payment_successfull"
          //Code here will execute if the payment transaction completed successfully.
          // here merchant can show the payment success message.
       case StaticDataModel.TXN_TIMEOUT_CODE:
          break
          //StaticDataModel.TXN_TIMEOUT_CODE is a string constant and its value is
"txn_session_timeout"
          //Code here will execute if the payment transaction failed because of the transaction time out.
          // here merchant can show the payment failed message.
       case StaticDataModel.TXN_BACKPRESSED_CODE:
          break
          //StaticDataModel.TXN_BACKPRESSED_CODE is a string constant and its value is
"back_pressed"
          //Code here will execute if the user pressed the back button on coupons Activity.
          // here merchant can show the payment failed message.
```

```
        case StaticDataModel.TXN_USERCANCELLED_CODE:
            break
            //StaticDataModel.TXN_USERCANCELLED_CODE is a string constant and its value is
"user_cancelled"
            //Code here will execute if the the user pressed the cancel button during the  payment process.
            // here merchant can show the payment failed message.
        case StaticDataModel.TXN_ERROR_SERVER_ERROR_CODE:
            break
            //StaticDataModel.TXN_ERROR_SERVER_ERROR_CODE is a string constant and its value is
"error_server_error"
            //Code here will execute if the server side error occured during payment process.
            // here merchant can show the payment failed message.

        case StaticDataModel.TXN_ERROR_TXN_NOT_ALLOWED_CODE:
            break
            //StaticDataModel.TXN_ERROR_TXN_NOT_ALLOWED_CODE is a string constant and its value
is "trxn_not_allowed"
            //Code here will execute if the the transaction is not allowed.
            // here merchant can show the payment failed message.

        case StaticDataModel.TXN_BANK_BACK_PRESSED_CODE:
            break
            //StaticDataModel.TXN_BANK_BACK_PRESSED_CODE is a string constant and its value is
"bank_back_pressed"
            //Code here will execute if the the customer press the back button on bank screen.
            // here merchant can show the payment failed message.

        default: break
            // Here the value of result is "payment_failed" or "error_noretry" or "retry_fail_error"
            //Code here will execute if payment is failed some other reasons.
            // here merchant can show the payment failed message.
        }
    }
```

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"];

    if ([StaticDataModel.TXN_BACKPRESSED_CODE isEqualToString:strResult]) {
        //StaticDataModel.TXN_BACKPRESSED_CODE is a string constant and its value is “back_pressed”
        //Code here will execute if the user pressed the back button on coupons Activity.
        // here merchant can show the payment failed message.
    }else if ([StaticDataModel.TXN_SUCCESS_CODE isEqualToString:strResult]) {
        NSLog(@"Payment Success.");
        //StaticDataModel.TXN_SUCCESS_CODE is a string constant and its value is
“payment_successfull”
        //Code here will execute if the payment transaction completed successfully.
        // here merchant can show the payment success message.
    }else if ([StaticDataModel.TXN_TIMEOUT_CODE isEqualToString:strResult]) {
        //StaticDataModel.TXN_TIMEOUT_CODE is a string constant and its value is “txn_session_timeout”
        //Code here will execute if the payment transaction failed because of the transaction time out.
        // here merchant can show the payment failed message.
    }else if ([StaticDataModel.TXN_USERCANCELLED_CODE isEqualToString:strResult]) {
        //StaticDataModel.TXN_USERCANCELLED_CODE is a string constant and its value is
“user_cancelled”
        //Code here will execute if the the user pressed the cancel button during the  payment process.
        // here merchant can show the payment failed message.
    }else if ([StaticDataModel.TXN_ERROR_SERVER_ERROR_CODE isEqualToString:strResult]) {
        //StaticDataModel.TXN_ERROR_SERVER_ERROR_CODE is a string constant and its value is
“error_server_error”
        //Code here will execute if the server side error occured during payment process.
        // here merchant can show the payment failed message.
    }else if ([StaticDataModel.TXN_ERROR_TXN_NOT_ALLOWED_CODE isEqualToString:strResult]) {
        //StaticDataModel.TXN_ERROR_TXN_NOT_ALLOWED_CODE is a string constant and its value is
“trxn_not_allowed”
        //Code here will execute if the the transaction is not allowed.
        // here merchant can show the payment failed message.
    }else if ([StaticDataModel.TXN_BANK_BACK_PRESSED_CODE isEqualToString:strResult]) {
        //StaticDataModel.TXN_BANK_BACK_PRESSED_CODE is a string constant and its value is
“bank_back_pressed”
        //Code here will execute if the the customer press the back button on bank screen.
        // here merchant can show the payment failed message.
    }
    else {
        // Here the value of result is “payment_failed” or “error_noretry” or “retry_fail_error”
        //Code here will execute if payment is failed some other reasons.
        // here merchant can show the payment failed message.
    }
}
```

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.

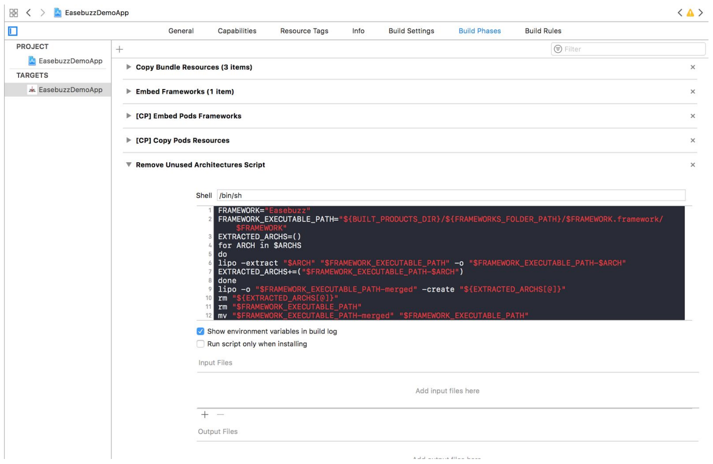
Select the Project, Choose Target → Project Name → Select Build Phases → Press “+” → New Run Script Phase → Name the Script as “Remove Unused Architectures Script”. Refer Screen4.

Always this script should be placed below “Embed Frameworks”.

To achieve this, I wrote Custom Run Scripts,

```
FRAMEWORK="Easebuzz"
FRAMEWORK_EXECUTABLE_PATH="${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH
}/$FRAMEWORK.framework/$FRAMEWORK"
EXTRACTED_ARCHS=()
for ARCH in $ARCHS
do
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o
"$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
rm "${EXTRACTED_ARCHS[@]}"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"
```

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



Shell /bin/sh

```
FRAMEWORK="Easebuzz"
FRAMEWORK_EXECUTABLE_PATH="${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH}/$FRAMEWORK.framework/
    $FRAMEWORK"
EXTRACTED_ARCHS=()
for ARCH in $ARCHS
do
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
rm "${EXTRACTED_ARCHS[@]}"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"
```

- [✓] Show environment variables in build log
- [ ] Run script only when installing

Screen4: RunScript