

NetCore SDK

Push Notification Integration Help for iOS.

Steps are as follows:

1. Creating App ID And Configure Push notification Certificates
2. Creating Provisional Profile.
3. Configure Application in NetCore
4. Setup the NetCore Push SDK into your project.
5. Integrate NetCore SDK & Push Methods.

1.1 Creating Explicit App Id

For Push Notification you need create explicit App id and enable push notification for you app id, following are list of step to create app id.

1. Sign in to developer.apple.com/account, and click Certificates, IDs & Profiles.
2. Under Identifiers, select App IDs.
3. Click the Add button (+) in the upper-right corner.
4. Enter a name or description for the App ID in the description field.
5. To create an explicit App ID, select Explicit App ID and enter the app's bundle ID in the Bundle ID field. An explicit App ID exactly matches the bundle ID of an app you're building. For example , **com.NetCore.PushDemo**. An explicit App ID can't contain an asterisk (*).
6. Select the Push Notification checkbox to enable the app Push Notification service
7. Click Continue.
8. Review the registration information, and click Register.
9. Click Done.

Certificates

All

Pending

Development

Production

Identifiers

App IDs

Pass Type IDs

Website Push IDs

iCloud Containers

App Groups

Merchant IDs

Devices

All

Apple TV

Apple Watch

iPad

iPhone

iPod Touch

Provisioning Profiles

All

Development

ID

Registering an App ID

The App ID string contains two parts separated by a period (.) — an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. [Learn More](#)

App ID Description

Name:

com.NetCore.PushDemo

You cannot use special characters such as @, &, *, ', "

Please enter a valid Name

App ID Prefix

Value:

Y344Y7796A (Team ID)

App ID Suffix

Explicit App ID

1.2 Configuring Push Notification for your application



Apple Push Notification service SSL Certificates

To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

Development SSL Certificate

Create an additional certificate to use for this App ID.

Create Certificate...

Production SSL Certificate

Create an additional certificate to use for this App ID.

Create Certificate...

Creating Push Notification Certificates & CSR File



About Creating a Certificate Signing Request (CSR)

To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac. To create a CSR file, follow the instructions below to create one using Keychain Access.

Create a CSR file.

In the Applications folder on your Mac, open the Utilities folder and launch Keychain Access.

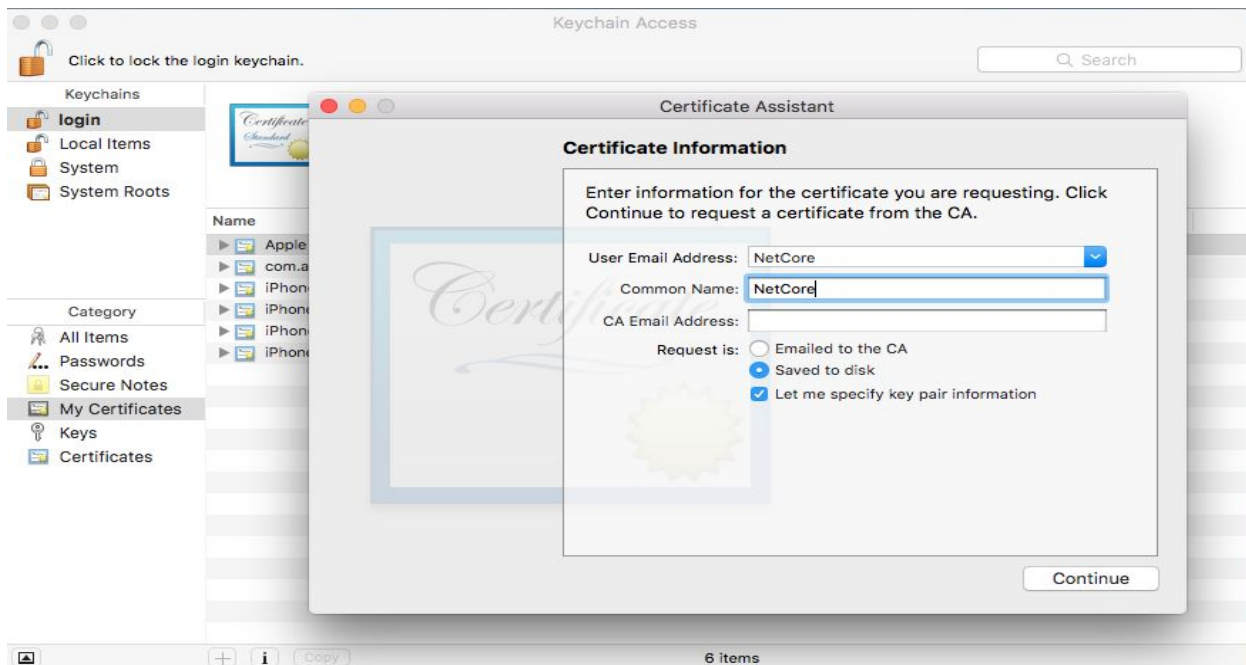
Within the Keychain Access drop down menu, select Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority.

- In the Certificate Information window, enter the following information:
 - In the User Email Address field, enter your email address.
 - In the Common Name field, create a name for your private key (e.g., John Doe Dev Key).
 - The CA Email Address field should be left empty.
 - In the "Request is" group, select the "Saved to disk" option.
- Click Continue within Keychain Access to complete the CSR generating process.

Cancel

Back

Continue



Generating Certificates and Download it



Your certificate is ready.

Download, Install and Backup

Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.



Name: Apple Push Services: com.NetCore.PushDemo
Type: Apple Push Services
Expires: Sep 15, 2017

[Download](#)



Push Notifications

Configurable

Apple Push Notification service SSL Certificates

To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

Development SSL Certificate	
Create an additional certificate to use for this App ID.	Create Certificate...
Production SSL Certificate	
Name: Apple Push Services: com.NetCore.PushDemo	
Type: Apple Push Services	Revoke Download
Expires: Sep 15, 2017	

- Double click on the downloaded SSL certificate to install it in your Keychain.
- In Keychain Access, under "My Certificates", find the certificate you just added.
- Export certificates .p12 file and add "NetCorePush.p12" name

[Reference link for SSL generation](#)

2. Creating Provisioning Profile

A provisioning profiles is use to install your application to iPhone device ,following are type of provisioning profiles

1. Development provisioning profiles :- This profile use install application on limited team members devices.Follow the steps in [Creating Development Provisioning Profiles](#) if you want to create your own development provisioning profile.

2. Distribution provisioning profiles : Distribution profile having 2 types

1) Ad Hoc Distribution :- you can create this profile when you export an app archive and select the ad hoc deployment option, as described in [Exporting Your App for Testing Outside the Store](#). To create an ad hoc provisioning profile directly in your developer account, read [Creating Ad Hoc Provisioning Profiles](#)

2) App Store Distribution

We Create store App Store Distribution file when we want deploy application over app store . To create a store provisioning profile directly in your developer account, read [Creating Store Provisioning Profiles](#).

Note: For push notification you can create Ad Hoc Provisioning Profiles for testing , NetCore is configure production level environment for push notification.

Creating Ad Hoc Provisioning Profiles (iOS, tvOS, watchES)

To create an ad hoc provisioning profile

1. Sign in to developer.apple.com/account, and click Certificates, IDs & Profiles.
2. Under Provisioning Profiles, select All.
3. Click the Add button (+) in the upper-right corner.
4. Select Ad Hoc as the distribution method, and click Continue.
5. Choose the App ID you used for development, which matches your bundle ID, from the App ID pop-up menu, and click Continue.
6. If you used a team provisioning profile during development and the menu contains only the XC Wildcard, select it. If the menu contains another Xcode-managed explicit App ID (it begins with “XC” followed by the bundle ID), select that App ID. If you created your own App ID, select that one.
7. Select the distribution certificate you want to use, and click Continue.

8. If you don't have a distribution certificate, create one using Xcode, as described in [Creating Signing Identities](#), before continuing.
9. Select the devices you want to use for testing, and click Continue.
10. Enter a profile name, and click Continue.
11. Wait while your developer account generates the provisioning profile.
12. At the bottom of the page, Click Done.

[3. Configure Application in NetCore](#)

To Use NetCore push notification you have to first create application inside NetCore website and upload .p12 certificate of your application.

The screenshot displays the NetCore Smarttech App Dashboard. At the top, there's a green header with status indicators: 'Z+ DOMESTIC 1ST PARTY', 'POSTPAID', 'DO NOT DELIVER is Disable', and 'ACTIVE since 14 Jun 2016'. Below this is a dark blue navigation bar with the 'netCORE Smarttech' logo and links for Broadcast, Automation, Manage (active), Reports, Admin, and Logs. A 'What's New' badge is also present. Underneath is a light blue bar with links for List Dashboard, Templates, and Assets.

The main content area is titled 'App Dashboard' and features a 'Create New App' button. It contains a table with the following data:

App Name	App ID	Subscribers	Platforms
Test ios	4a02532c1625f770161afd61ed5748f5	0	Android, Apple
Testing	4de4838587cd5cbce17d6b77b5710ff9	2	Android, Apple
TestQuagnitia	2714dd99a4b547c56ec8c276bd4a5e87	159	Android, Apple
Testing_NewId	75a841cb018d1369cff19f07b85ca88a	0	Android, Apple
NewIOSTesting	c8d65d33abfbda7bde537fe7076067d0	8	Android, Apple

At the bottom of the table, there are pagination controls showing '1 2 3 > >>'. The footer of the dashboard includes links for About netCORE, Product Demo, Email Marketing, Blog, Terms & Conditions, Anti-Spam Policy, and Privacy Policy, along with the text 'a netCORE product'.



DOMESTIC
1ST PARTY

POSTPAID

DO NOT DELIVER
is Disable

ACTIVE
since 14 jun 2016

netCORE
Smartech



Broadcast

Automation

Manage

Reports

Admin

Logs



List Dashboard

Templates

Assets

Create New App

App Name *

Enter App Name

App Description

Platform Configuration



Certificate file (.cer)

Browse...

No file selected.



Push Certificate (.p12)

Browse...

No file selected.

Private key password

Framework

Native

Create App | Cancel

About netCORE

Product Demo

Email Marketing

Blog

Terms & Conditions

Anti-Spam Policy

Privacy Policy

a netCORE product

Z+DOMESTIC
1ST PARTY

netCORE
Smartech

Go Dashboard

ACTIVE
Since 14 Jun 2018

?

App Integration Details

App ID is - 4a02532c1625f770161afd61ed5748f5

Include this App ID while integrating SDK in your App.

Download details for [Android](#)

Close

Platform Configuration

Certificate file (.cer)

Browse... dummy1.cer

Push Certificate (.p12)

Browse... p12dummy.p12

Private key password

Test

Framework

Native

Create App | Cancel

About netCORE | Product Demo | Email Marketing | Blog | Terms & Conditions | Anti-Spam Policy | Privacy Policy

a netCORE product

4. Setup the NetCore Push SDK

NetCore Integration Using Cocoa Pod

1. Install CocoaPods on your computer.
2. Open your project and create pod file using below command

```
pod init
```

3. Add following line in your podfile

```
pod 'Netcore-Smartech-iOS-SDK'
```

4. Run following command in your project directory

```
pod install
```

5. Open App.xcworkspace and build app.

NetCore Manual Integration

Download NetCore iOS SDK from below links

GitHub

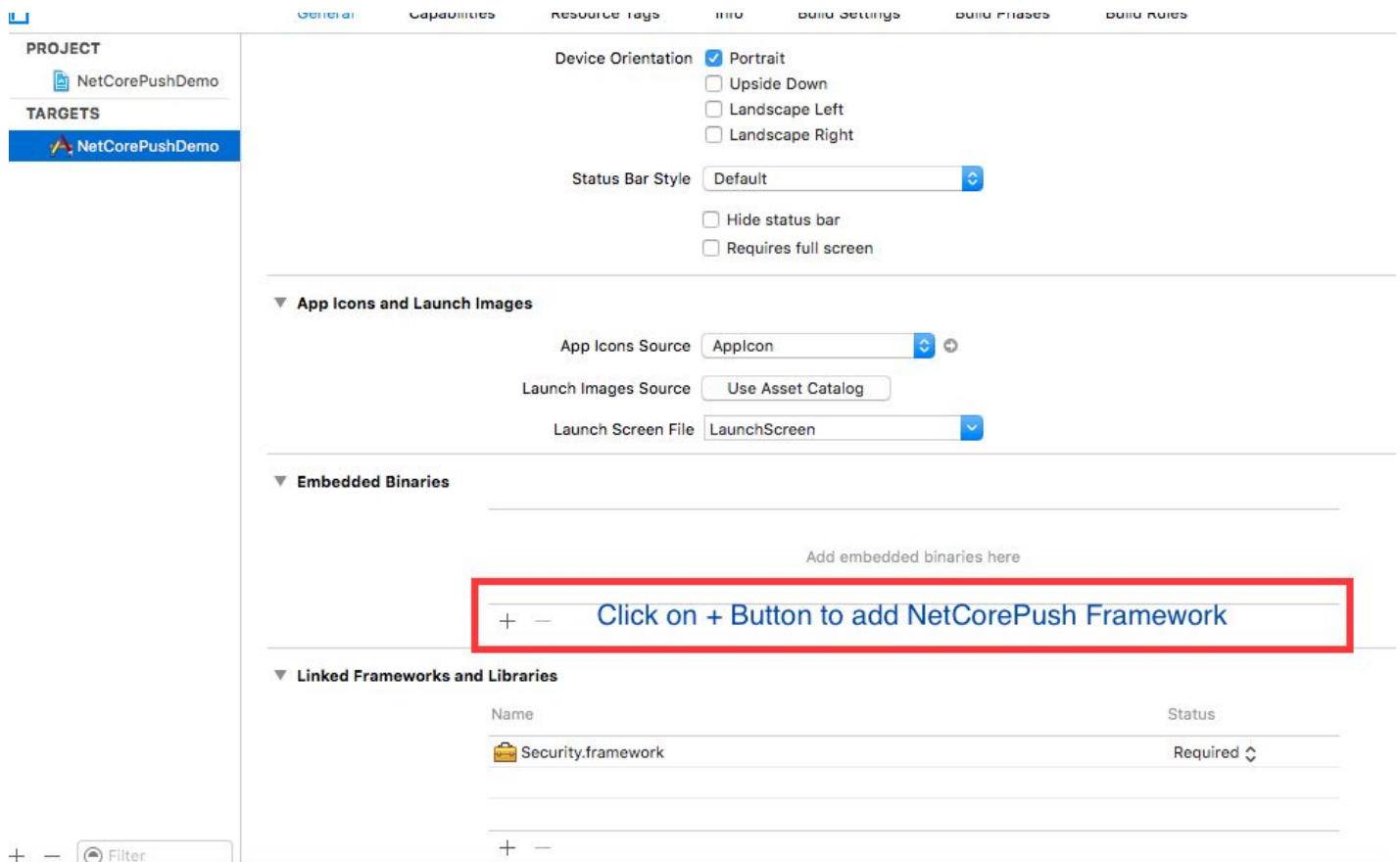
[git clone https://github.com/NetcoreSolutions/Smartech-ios-sdk](https://github.com/NetcoreSolutions/Smartech-ios-sdk)

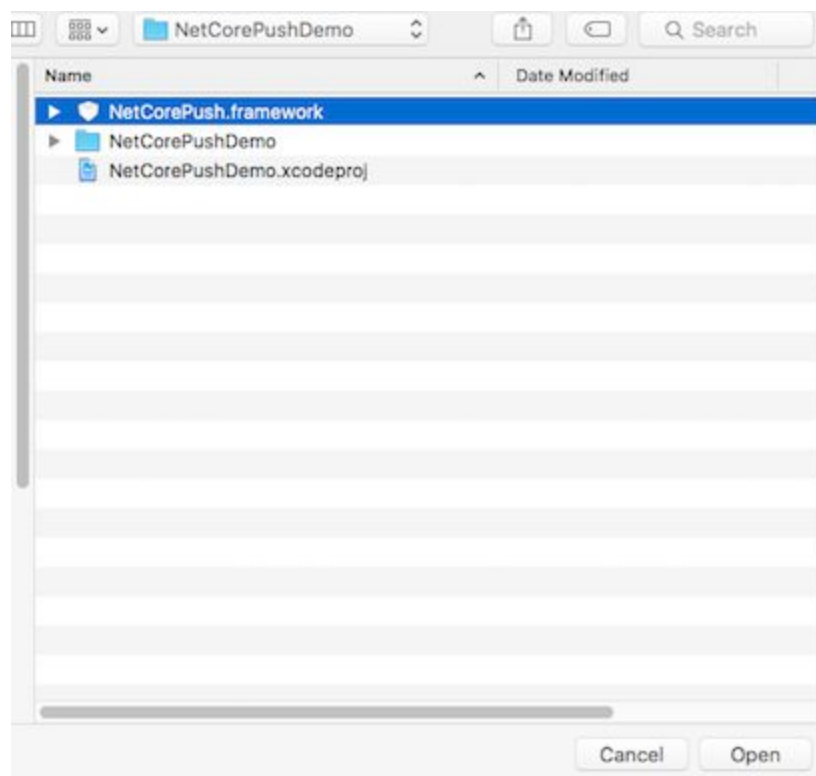
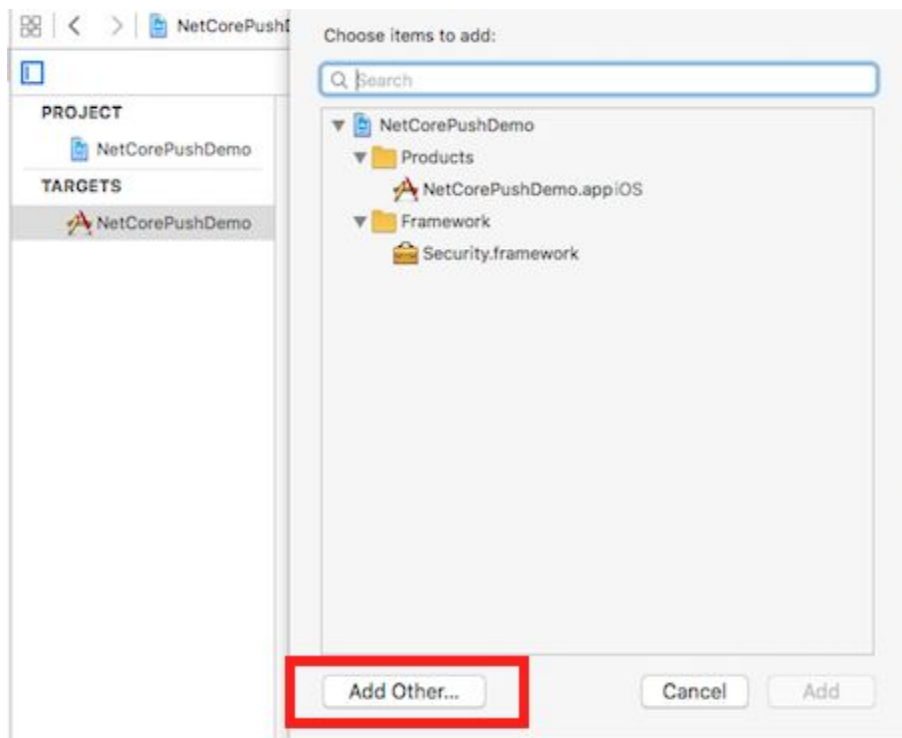
From Browser

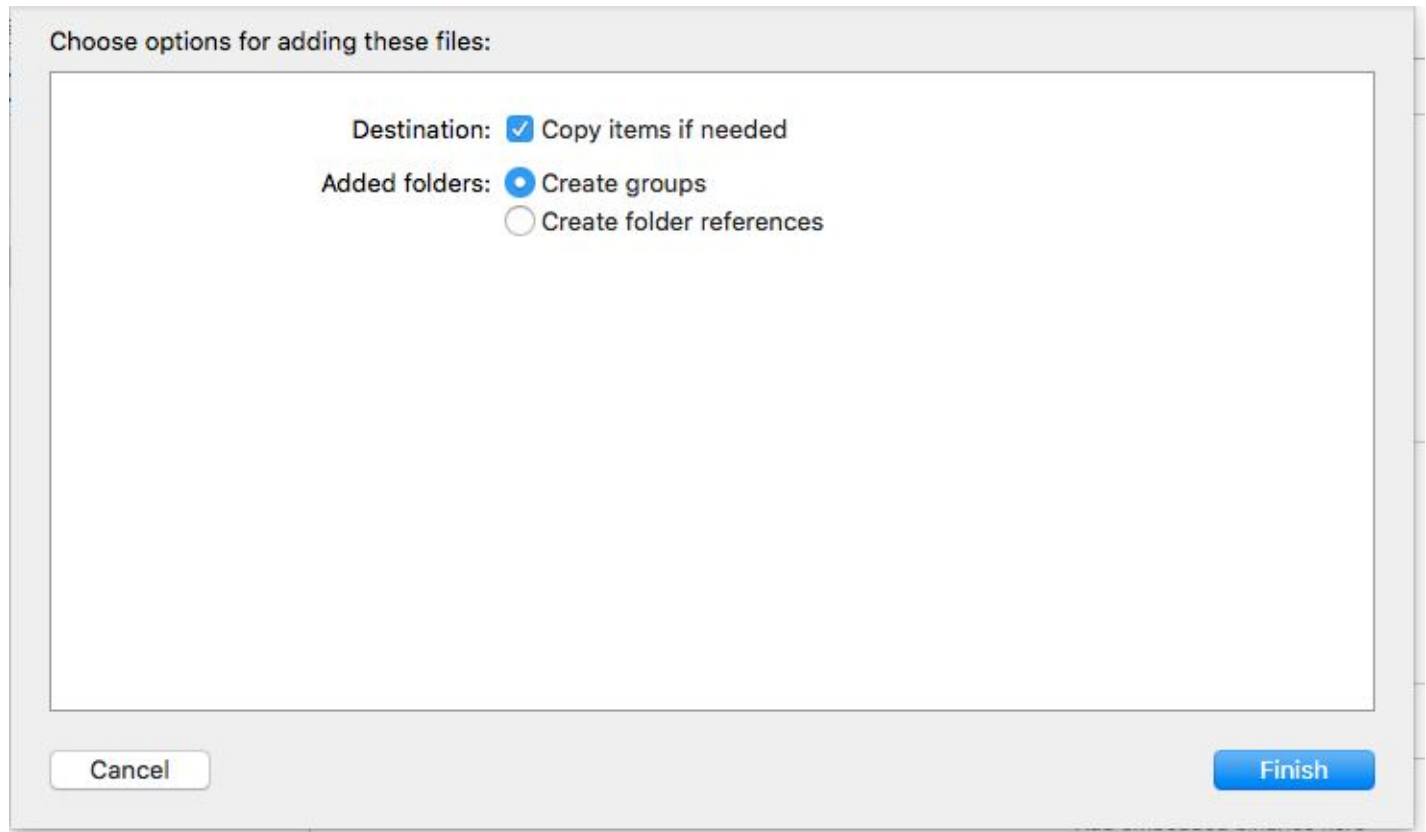
<https://github.com/NetcoreSolutions/Smartech-ios-sdk/archive/master.zip>

1. **Download iOS SDK** and Unzip the file. Open Framework folder - inside it you will see NetCorePush.framework file.

2. Open existing or create a new project in Xcode and drag drop or add framework in Target > Embedded Binaries section







3. Add following frameworks inside your application if required

- 1)Security
- 2)CoreLocation
- 3)SystemConfiguration
- 4)JavaScriptCore

4. Add Following capability inside your application

- 1)Push Notification
- 2)Keychain

5. Swift Bridge file Reference

Create Bridge file in existing swift project if required and add Following code inside file

```
import NetCorePush
```

NetCore SDK Initialization

1. Import following file in AppDelegate File

Objective C

```
#import <NetCorePush/NetCorePush.h>
#import <UserNotifications/UserNotifications.h>
#import <UserNotificationsUI/UserNotificationsUI.h>
```

Swift

```
import User Notifications
import UserNotificationsUI
import NetCorePush
```

2. Add NetCore Application AppID in support in Finish Launching Methods (AppDelegate file)

Objective C

```
#define kAppID @"Your App Id which you get from Netcore smartech admin panel ";

[[NetCoreSharedManager sharedInstance]setUpApplicationId:kAppID];
[NetCorePushTaskManager sharedInstance].delegate = self;
// set up your third party framework initialization process as per their document
```

Swift

```
let netCore_AppID = "your App Id which you get from Netcore smartech admin panel"
// Set up NetCore Application Id-----
NetCoreSharedManager.sharedInstance().setUpApplicationId(netCore_AppID)
//set up push delegate
NetCorePushTaskManager.sharedInstance().delegate = self

// set up your third party framework initialization process as per their document
```

3. Add Push Notification support in Finish Launching Methods (AppDelegate file)

Objective C

```
if ([[UIDevice currentDevice] systemVersion] floatValue] >= 10.0){

    UNUserNotificationCenter *center = [UNUserNotificationCenter
currentNotificationCenter];
    center.delegate = self;
    [center
requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge)
completionHandler:^(BOOL granted, NSError * _Nullable error){
        if(granted){
            [[UIApplication sharedApplication]
registerForRemoteNotifications];
        }
    }];
    [[UIApplication sharedApplication]
registerForRemoteNotifications];
}else{
    UIUserNotificationType userNotificationTypes =
(UIUserNotificationTypeAlert | UIUserNotificationTypeBadge |
UIUserNotificationTypeSound);
    UIUserNotificationSettings *settings =
[UIUserNotificationSettings settingsForTypes:userNotificationTypes
categories:nil];
    [[UIApplication
```

```
registerUserNotificationSettings:settings];  
    [[UIApplication sharedApplication]  
registerForRemoteNotifications];  
}
```

Swift

```
if #available(iOS 10, *) {  
    UNUserNotificationCenter.current().delegate = self  
  
    UNUserNotificationCenter.current().requestAuthorization(options:  
[.alert, .badge, .sound]) { (granted, error) in  
        guard error == nil else {  
            return  
        }  
        if granted {  
            UIApplication.shared.registerForRemoteNotifications()  
        }  
    }  
    UIApplication.shared.registerForRemoteNotifications()  
} else {  
    let settings: UIUserNotificationSettings =  
        UIUserNotificationSettings(types: [.alert, .badge, .sound],  
categories: nil)  
    UIApplication.shared.registerUserNotificationSettings(settings)  
    UIApplication.shared.registerForRemoteNotifications()  
}
```

4. Check Application Launching from Push/Local Notification support in Finish Launching Methods (AppDelegate file)

Objective C

```
// Handle launch event  
if (launchOptions != nil){
```



```
[[NetCorePushTaskManager sharedInstance]handelApplicationLaunchEvent:
launchOptions];
}
```

Swift

```
if (launchOptions != nil){

NetCorePushTaskManager.sharedInstance().handelApplicationLaunchEvent(launchOpt
ions)
}
```

5. Register Device With NetCore SDK (AppDelegate file)

Objective C

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error{
    // manage notification token failure process as per third party SDK as per their
    document
}

- (void)application:(UIApplication*)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken{

    // Register device token with third party SDK as per their document

    // Set up device token inside NetCoreSharedManager
    [[NetCoreSharedManager sharedInstance]setDeviceToken:deviceToken];
    //strEmail = your application identity

    [[NetCoreSharedManager sharedInstance]setUpIdentity:strEmail];

    [[NetCoreInstallation sharedInstance]netCorePushRegistration:strEmail
Block:^(NSInteger statusCode) {
        }];
}
```

Swift

```
func application(_ application: UIApplication,
didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {

    // Register device token with third party SDK as per their document

    NetCoreSharedManager.sharedInstance().setDeviceToken(deviceToken)
    //strEmail = your application identity
    NetCoreSharedManager.sharedInstance().setUpIdentity(strEmail as!
String!)
    // Register User Device with NetCore
    NetCoreInstallation.sharedInstance().netCorePushRegistration(strEmail as!
String!, block: { (code) in})
}

func application(_ application: UIApplication,
didFailToRegisterForRemoteNotificationsWithError error: Error) {

    // manage notification token failure process as per third party SDK as per their
document
}
```

6. Handle Push/Local Notification Delegate Events (AppDelegate file)

Objective C

```
-(void)application:(UIApplication *)application
didReceiveLocalNotification:(UILocalNotification *)notification{
    [[NetCorePushTaskManager
sharedInstance]didReceiveLocalNotification:notification.userInfo];
}
- (void)application:(UIApplication*)application
```

```

didReceiveRemoteNotification:(NSDictionary*)userInfo{
    [[NetCorePushTaskManager
sharedInstance]didReceiveRemoteNotification:userInfo];
}
//Called to let your app know which action was selected by the user for a given
notification.
-(void)userNotificationCenter:(UNUserNotificationCenter* )center
didReceiveNotificationResponse:(UNNotificationResponse* )response
withCompletionHandler:(void(^)())completionHandler{
    [[NetCorePushTaskManager
sharedInstance]userNotificationCenterdidReceiveNotificationResponse:response];
}

```

Swift

```

func application(_ application: UIApplication, didReceiveRemoteNotification
userInfo: [AnyHashable: Any]){

NetCorePushTaskManager.sharedInstance().didReceiveRemoteNotification(userInfo)
}

// MARK: - didReceiveLocalNotification method
func application(_ application: UIApplication, didReceive notification:
UILocalNotification){

NetCorePushTaskManager.sharedInstance().didReceiveLocalNotification(notification.u
serInfo)
}

extension AppDelegate: UNUserNotificationCenterDelegate {

// called when application is open when user click on notification

@objc(userNotificationCenter:didReceiveNotificationResponse:withCompletionHandler
:) @available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive
response: UNNotificationResponse, withCompletionHandler completionHandler:
@escaping () -> Void) {

```

```

NetCorePushTaskManager.sharedInstance().userNotificationdidReceive(response)
}
// This is key callback to present notification while the app is in foreground
@objc(userNotificationCenter:willPresentNotification:withCompletionHandler:)
@available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent
notification: UNNotification, withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void) {

    completionHandler( [.alert,.sound,.badge])

NetCorePushTaskManager.sharedInstance().userNotificationWillPresent(notification)
}
}

```

7. Handle Deep Linking

Objective C

```

- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    //handel deep Link here
    return YES;
}
// PushManagerDelegate Method
-(void)handleNotificationOpenAction:(NSDictionary *)userInfo
DeepLinkType:(NSString *)strType{
    if (strType !=nil ){
        //handel deep Link here
    }
}
}

```

Swift

```

func application(_ application: UIApplication, open url: URL, sourceApplication:
String?, annotation: Any) -> Bool{
    // handle deep link here
    return true
}

```

```

    }

extension AppDelegate : NetCorePushTaskManagerDelegate{
    func handleNotificationOpenAction(_ userInfo: [AnyHashable : Any]!,
    deepLinkType strType: String!) {
        if strType.lowercased().contains ("your app deep link"){
            // handle deep link here
        }
    }
}

```

8. Login with NetCore

Objective C

```

// strEmail = pass your device identity
[[NetCoreInstallation sharedInstance]netCorePushLogin:strEmail Block:^(NSInteger
statusCode) {
    }];

```

Swift

```

// strEmail = pass your device identity
NetCoreInstallation.sharedInstance().netCorePushLogin(strEmail) { (statusCode:Int)
in }

```

9. Logout

Objective C

```

[[NetCoreInstallation sharedInstance]netCorePushLogout:^(NSInteger statusCode) {
    }];

```

Swift

```
// strEmail = pass your device identity
NetCoreInstallation.sharedInstance().netCorePushLogout { (statusCode:Int) in }
```

10. Profile Push

Objective C

```
NSDictionary *info = @{@"name":@"Tester",@"age":
@"23",@"mobile":@"32424342"};
[[NetCoreInstallation sharedInstance]netCoreProfilePush:strEmail payload:info
Block:nil];
```

Swift

```
// strEmail = pass your device identity
let info = ["name":"Tester","age": "23","mobile":"32424342"]
NetCoreInstallation.sharedInstance().netCoreProfilePush(strEmail, payload: info,
block: nil)
```

11. Events Tracking :

Following is the list of tracking events

```
tracking_CheckOut = 3,
tracking_FirstLaunch = 20,
tracking_AppLaunch = 21
```

You can use this events following ways

12. Track normal event

Objective C

```
[[NetCoreAppTracking sharedInstance]sendAppTrackingEvent:tracking_FirstLaunch
Block:^(NSInteger statusCode) {
}];
```

```
[[NetCoreAppTracking sharedInstance]sendAppTrackingEvent:tracking_AppLaunch
Block:^(NSInteger statusCode) {
}];
```

Swift

```
// for sending application launch event
```

```
NetCoreAppTracking.sharedInstance().sendEvent(Int(UInt32(tracking_FirstLaunch.rawValue)), block: nil)
```

```
NetCoreAppTracking.sharedInstance().sendEvent(Int(UInt32(tracking_AppLaunch.rawValue)), block: nil)
```

13. Track event with custom payload

Objective C

```
// eg checkOutArray custom payload format
(
    {
        "i^prid" = 2;
        "i^prqt" = 1;
        "s^name" = "TestProduct";
        "s^price" = 30000;
    },
    {
        "i^prid" = 1;
        "i^prqt" = 2;
        "s^name" = "TestProduct";
        "s^price" = 40000;
    }
)
```

```
[[NetCoreAppTracking sharedInstance]sendAppTrackingEvent:tracking_CheckOut
payload: checkOutArray Block:^(NSInteger statusCode) {
}];
```

Swift

```
NetCoreAppTracking.sharedInstance().sendEvent(withCustomPayload:  
Int(tracking_CheckOut), payload: checkOutArray , block: nil)
```

14. To fetch delivered push notifications

Objective C

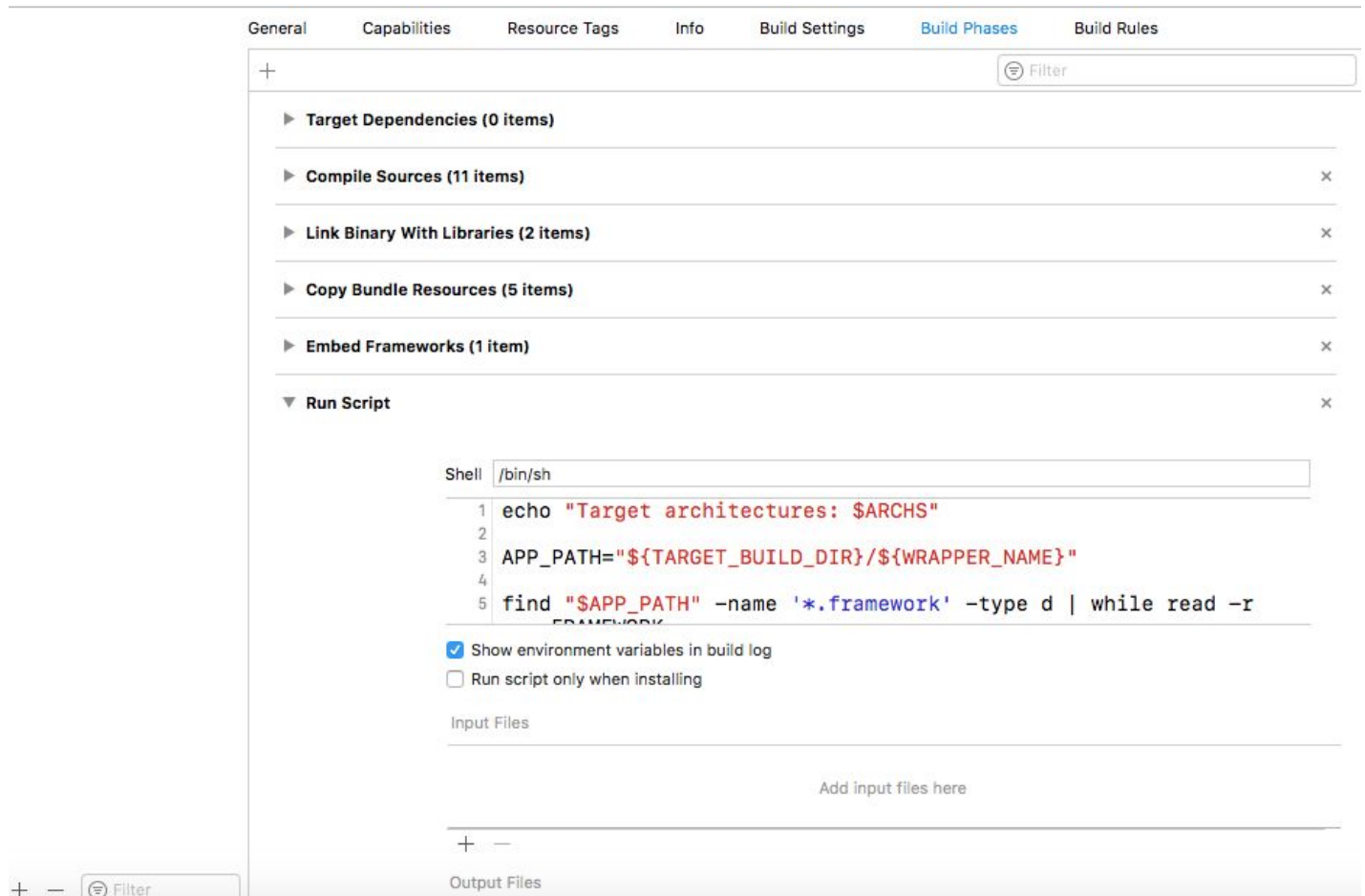
```
NSArray *notificationsArray = [[NetCoreSharedManager sharedInstance]  
getNotifications];
```

Swift

```
let array : Array =  
NetCoreSharedManager.sharedInstance().getNotifications()
```

Deployment Over Apple Store

Add Following runscript in your application target ,when you are deploying application over apple store,this run script use remove unused architecture in release mode



```
echo "Target architectures: $ARCHS"
```

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

```
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
```

```
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
```

```
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
E"
```

```
echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"
```

```
echo $(lipo -info $FRAMEWORK_EXECUTABLE_PATH)
```

```
FRAMEWORK_TMP_PATH="$FRAMEWORK_EXECUTABLE_PATH-tmp"
```

```
# remove simulator's archs if location is not simulator's directory
```

```
case "${TARGET_BUILD_DIR}" in
*"iphonesimulator")
echo "No need to remove archs"
;;
*)
if $(lipo $FRAMEWORK_EXECUTABLE_PATH -verify_arch "i386") ; then
lipo -output $FRAMEWORK_TMP_PATH -remove "i386"
$FRAMEWORK_EXECUTABLE_PATH
echo "i386 architecture removed"
rm $FRAMEWORK_EXECUTABLE_PATH
mv $FRAMEWORK_TMP_PATH $FRAMEWORK_EXECUTABLE_PATH
fi
if $(lipo $FRAMEWORK_EXECUTABLE_PATH -verify_arch "x86_64") ; then
lipo -output $FRAMEWORK_TMP_PATH -remove "x86_64"
$FRAMEWORK_EXECUTABLE_PATH
echo "x86_64 architecture removed"
rm $FRAMEWORK_EXECUTABLE_PATH
mv $FRAMEWORK_TMP_PATH $FRAMEWORK_EXECUTABLE_PATH
fi
;;
esac

echo "Completed for executable $FRAMEWORK_EXECUTABLE_PATH"
echo $(lipo -info $FRAMEWORK_EXECUTABLE_PATH)

done
```