

## **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](https://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:   
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

☒  **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

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.

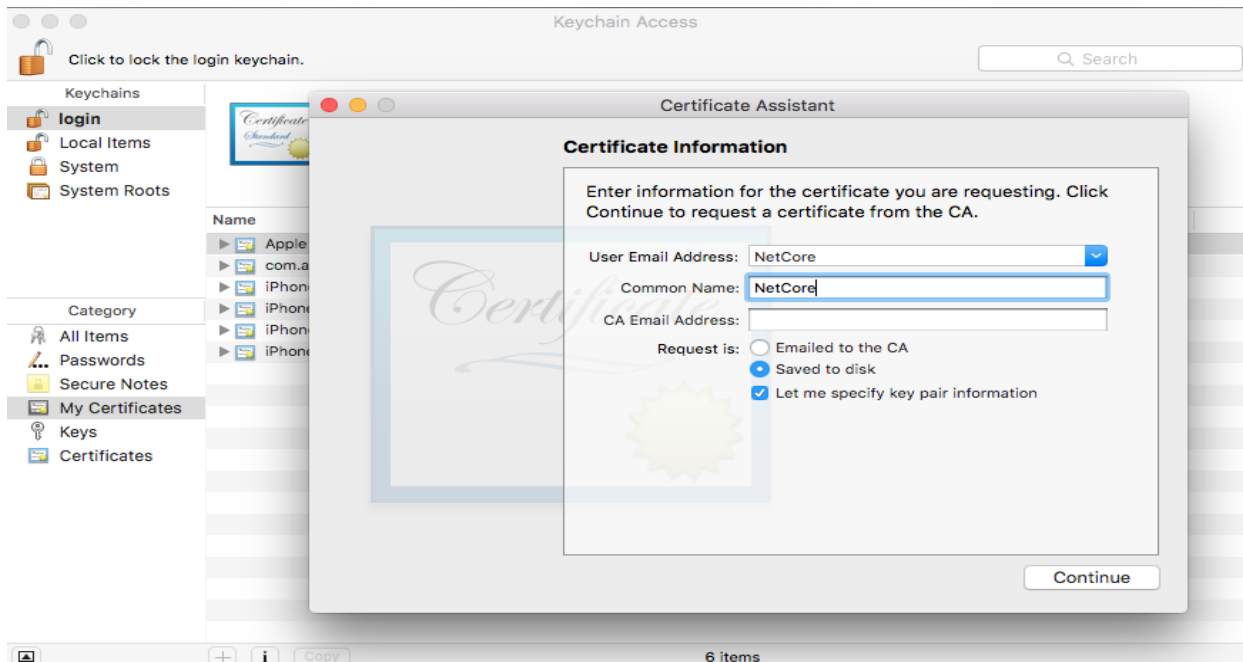
### Create a CSR file.

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  
Expires: Sep 15, 2017

Revoke Download

- 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 profile is used to install your application on an iPhone device, following are the types of provisioning profiles

**1)Development provisioning profiles** :- This profile is used to install applications 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 a store App Store Distribution file when we want to deploy an application over the 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 configured for 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](https://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.











Z+DOMESTIC 1ST PARTYPOSTPAIDDO NOT DELIVER is DisableACTIVE since 14 jun 2016

netCORE SmartechBroadcastAutomationManageReportsAdminLogsWhat's New?🔔👤

List DashboardTemplatesAssets

### App Dashboard

Create New App

App Name	App ID	Subscribers	Platforms
Test ios	4a02532c1625f770161afd61ed5748f5	0	 
Testing	4de4838587cd5cbce17d6b77b5710f9	2	 
TestQuagnitia	2714dd99a4b547c56ec8c276bd4a5e87	159	 
Testing_NewId	75a841cb018d1369cff19f07b85ca88a	0	 
NewIOSTesting	c8d65d33abfbda7bde537fe7076067d0	8	 

123>>

About netCOREProduct DemoEmail MarketingBlogTerms & ConditionsAnti-Spam PolicyPrivacy Policya netCORE product



## 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



### 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

## 4. Setup the NetCore Push SDK

### NetCore Integration Using Cocoa Pod

- 1) Install CocoaPods on your computer.
- 2) open your project add 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**

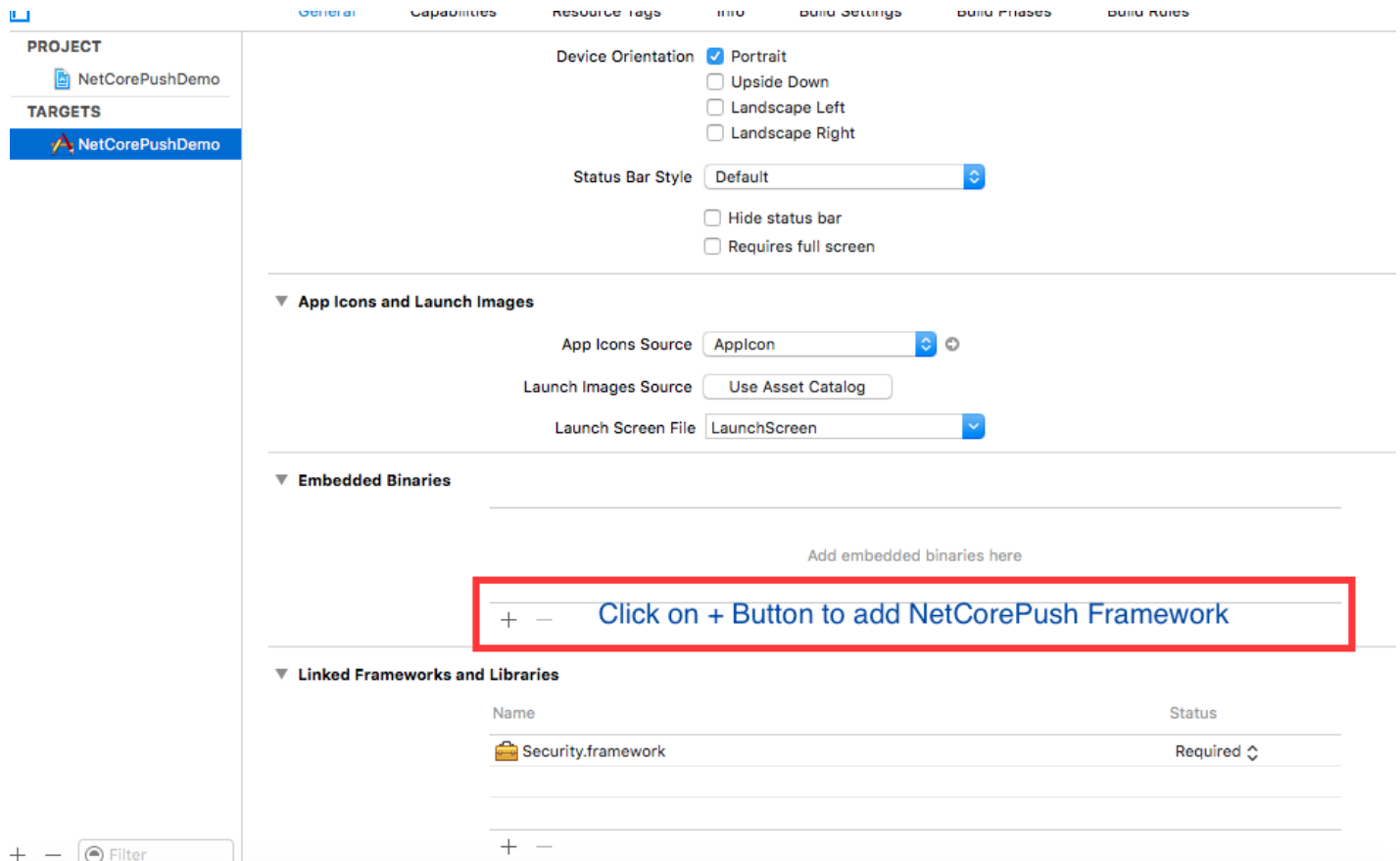
[git clone https://github.com/NetcoreSolutions/Smartech-ios-sdk](https://github.com/NetcoreSolutions/Smartech-ios-sdk)

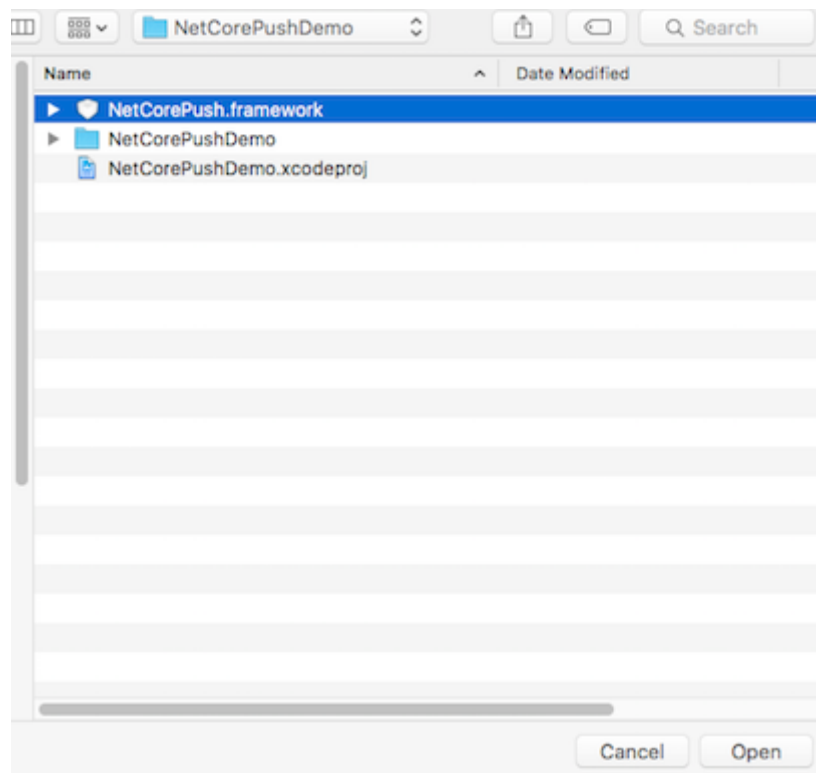
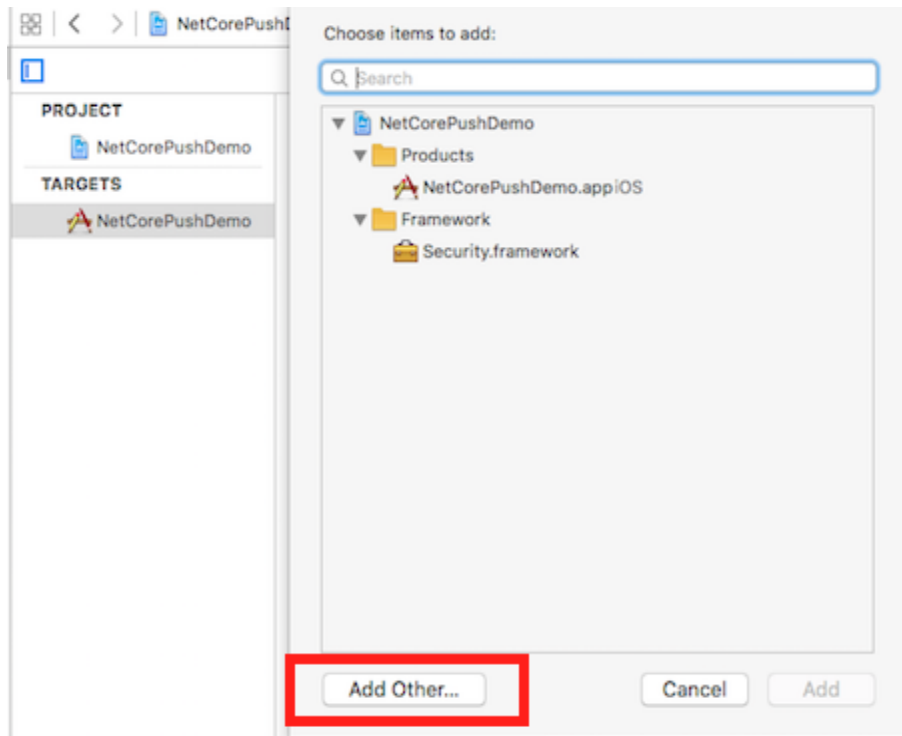
#### **From Browser**

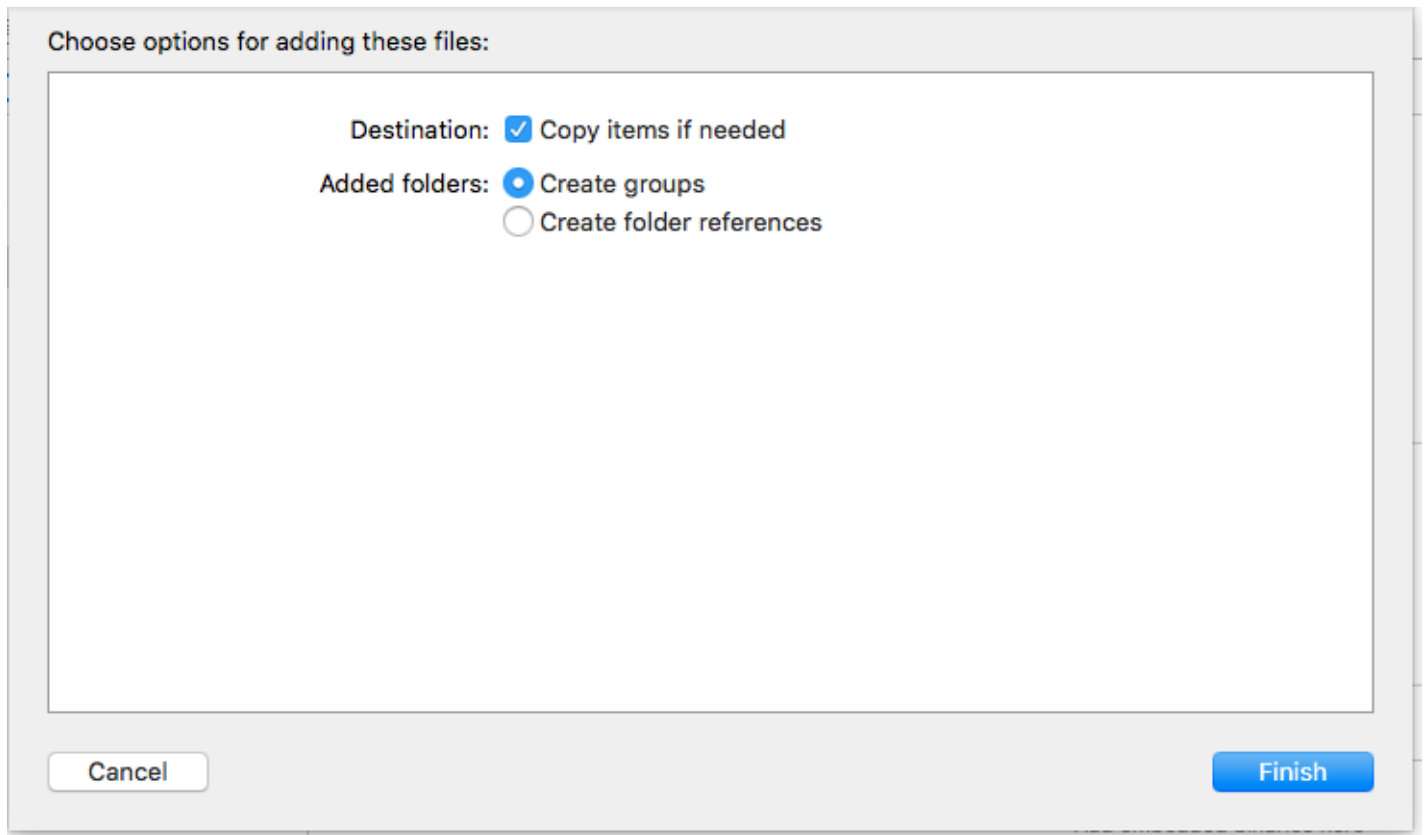
<https://github.com/NetcoreSolutions/Smartech-ios-sdk/archive/master.zip>

- 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
- 3) Background Mode -> Remote Notification
- 4) App Groups -> Create new group with name "group.com.Smartech.com"

## 5) Swift Bridge file Reference

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

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

## 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 UserNotifications
import UserNotificationsUI
import NetCorePush
```

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

#### Objective C

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

// Set up NetCore Application Id
[[NetCoreSharedManager sharedInstance]
handleApplicationLaunchEvent:launchOptions forApplicationId: netCore_AppID];

//set up push delegate
[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().handleApplicationLaunchEvent(launchOptions, forApplicationId: netCore_AppID)

//set up push delegate
NetCorePushTaskManager.sharedInstance().delegate = self

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

### 3. Register Device With NetCore SDK (AppDelegate file)

## Objective C

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

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

    // Identity must be ""(blank) or as per Primary key which defined on Smartech Panel
    [[NetCoreInstallation sharedInstance] netCorePushRegistration: Identity
withDeviceToken:deviceToken Block:^(NSInteger statusCode) {
        }];
}
```

## Swift

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

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

    //Identity must be ""(blank) or as per Primary key which defined on Smartech Panel
    NetCoreInstallation.sharedInstance().netCorePushRegistration(Identity,
withDeviceToken: deviceToken) { (status) in
        }
}
```

## 4. Handle Remote/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]
userNotificationdidReceiveNotificationResponse: 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.us
erInfo)
}

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)
}
}

```

## 5. Handle Deep Linking

### Objective C

```

- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    //handle URL Link here
    return YES;
}

// PushManagerDelegate Method
-(void)handleNotificationOpenAction:(NSDictionary *)userInfo DeepLinkType:
(NSString *)strType{
    if ([strType containsString:@"your app deep link"]){
        //handle deep Link here
    }
}

```

## Swift

```

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

```

## 6. Handle Interactive buttons

### Objective C

//Method for Interactive Button

```
-(void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier forRemoteNotification:(NSDictionary *)userInfo withResponseInfo:(NSDictionary *)responseInfo completionHandler:(void (^)(void))completionHandler{

    [[NetCorePushTaskManager sharedInstance] handleActionWithIdentifier:identifier forRemoteNotification:userInfo withResponseInfo:responseInfo];

    completionHandler();
}
```

## Swift

```
func application(_ application: UIApplication, handleActionWithIdentifier identifier: String?, forRemoteNotification userInfo: [AnyHashable : Any], withResponseInfo responseInfo: [AnyHashable : Any], completionHandler: @escaping () -> Void) {

    NetCorePushTaskManager.sharedInstance().handleAction(withIdentifier: identifier, forRemoteNotification: userInfo, withResponseInfo: responseInfo)

    completionHandler()
}
```

## 7. Login

### Objective C

```
// Identity must be ""(blank) or as per Primary key which defined on Smartech Panel
[[NetCoreInstallation sharedInstance]netCorePushLogin: Identity Block:^(NSInteger statusCode) {
    }];
```

## Swift

```
// Identity must be ""(blank) or as per Primary key which defined on Smartech Panel
NetCoreInstallation.sharedInstance().netCorePushLogin(Identity) { (statusCode:Int)
in }
```

## 8. Logout

### Objective C

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

### Swift

```
NetCoreInstallation.sharedInstance().netCorePushLogout { (statusCode:Int) in }
```

## 9. Profile Push

### Objective C

```
// Identity must be ""(blank) or as per Primary key which defined on Smartech Panel

NSDictionary *info = @{@"NAME":@"Tester",@"AGE":
@"23",@"MOBILE":@"32424342"};

[[NetCoreInstallation sharedInstance]netCoreProfilePush:Identity payload:info
Block:nil];
```

### Swift

```
// Identity must be ""(blank) or as per Primary key which defined on Smartech Panel
let info = ["NAME":"Tester","AGE": "23","MOBILE":"32424342"]

NetCoreInstallation.sharedInstance().netCoreProfilePush(Identity, payload: info,
block: nil)
```

## 10. Track custom event

// Activity tracking code can be generated from Smartech panel

### Objective C

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

eg.

```
NSMutableDictionary * dict = [NSMutableDictionary new];
[dict setObject:@"Nexus" forKey:@"s^brand"];
[dict setObject:@11 forKey:@"i^price"];
[dict setObject:@2 forKey:@"i^prid"];
```

```
NSMutableArray *payloadArray = [[NSMutableArray alloc] init];
[payloadArray addObject:dict];
```

```
[[NetCoreAppTracking sharedInstance] sendAppTrackingEventWithCustomPayload:2
Payload:payloadArray Block:^(NSInteger statusCode) {} ];
```

### Swift

```
NetCoreAppTracking.sharedInstance().sendEvent(withCustomPayload: Int(Event_Id),
payload: payloadArray , block: nil)
```

eg.

```
var dict = [String: Any]()
dict["s^brand"] = "Nexus"
dict["i^price"] = 11
dict["i^prid"] = 2
```

```
var payloadArray = [AnyHashable]()
payloadArray.append(dict)
```

```
NetCoreAppTracking.sharedInstance().sendEvent(withCustomPayload:2, payload:
payloadArray , block: nil)
```

## 11. To fetch delivered push notifications

## Objective C

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

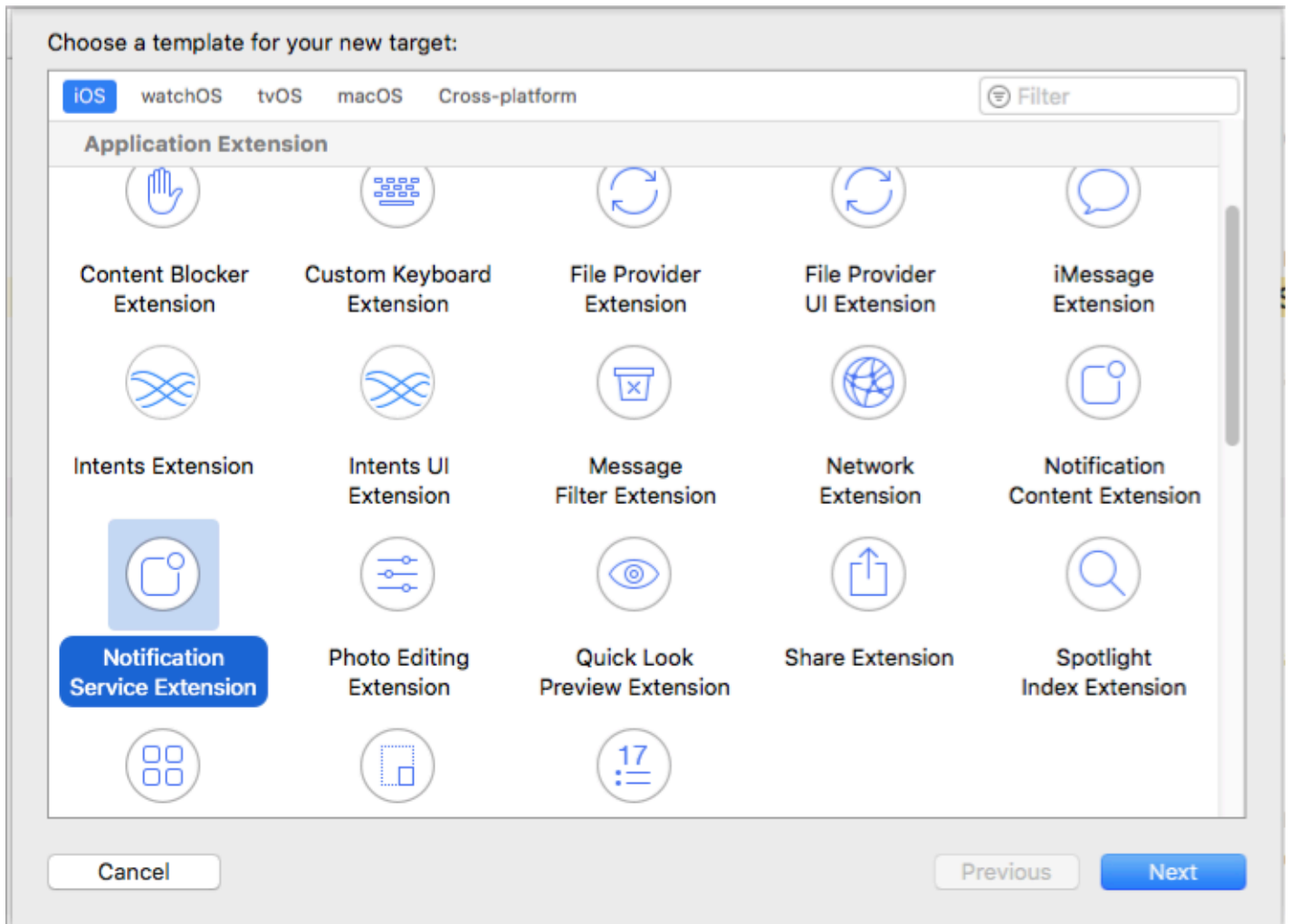
## Swift

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

## 12. For Rich Push Notifications

### Configuration Changes

**1)** Add “[Notification Service Extension](#)” to your app. [File->New->Target- >Notification Service Extension](#).



2) Click Next and when asked to "Activate", Click Activate.

3) Add **"App Groups"** to your apps Capabilities(Add one group with name **"group.com.Smartechnology.com"**).

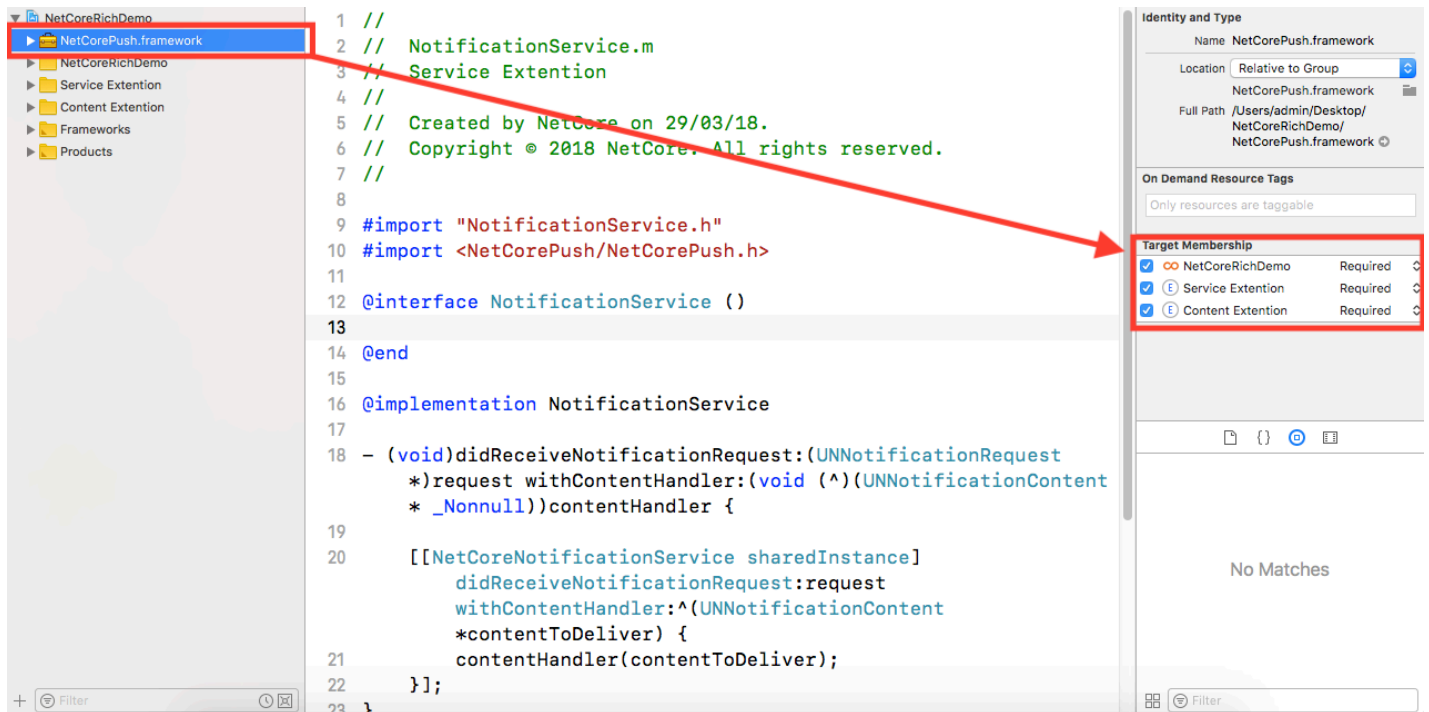
4) Enable **App groups in Service Extension** too and select group with name **"group.com.Smartechnology.com"**.

5) If App group is not activated on the provisioning profile you are using, then

1. Enable App groups in your provisioning profile from your Apple Developer's account and replace the profile with the new one. **Or,**
2. In your app's, **Target->General-> Signing**, Select **"Automatically manage signing"** and enable App groups by going to **Target->Capabilities->App group**. This will automatically add app groups capability to your provisioning profile.



6) Make sure to add NetCore SDK to your Extension's too.



## Implementation Changes

### Objective C

Remove all the code written in **"NotificationService"** implementation part.

- 1) Import NetCore Framework into Extension

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

- 2) Handle Notification Request

```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request  
withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {  
  
[[NetCoreNotificationService sharedInstance] didReceiveNotificationRequest:request  
withContentHandler:^(UNNotificationContent *contentToDeliver) {  
  
contentHandler(contentToDeliver); }];  
  
}
```

### 3) Handle Notification Service Time Expire

```
- (void)serviceExtensionTimeWillExpire {  
  
[[NetCoreNotificationService sharedInstance] serviceExtensionTimeWillExpire];  
  
}
```

4) After all the above code, Your “**Notification Service**” should look like the image below.

```

8
9 #import "NotificationService.h"
10 #import <NetCorePush/NetCorePush.h>
11
12 @interface NotificationService ()
13
14 @end
15
16 @implementation NotificationService
17
18 - (void)didReceiveNotificationRequest:(UNNotificationRequest *)request
    withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {
19
20     [[NetCoreNotificationService sharedInstance] didReceiveNotificationRequest:request
        withContentHandler:^(UNNotificationContent *contentToDeliver) {
21         contentHandler(contentToDeliver);
22     }];
23 }
24
25 - (void)serviceExtensionTimeWillExpire {
26
27     [[NetCoreNotificationService sharedInstance] serviceExtensionTimeWillExpire];
28 }
29
30 @end
31
32

```

## Swift

Remove all the code written in "**NotificationService**" class part .

1) Import NetCore Framework into Extension

```
import NetCorePush
```

2) Handle Notification Request

```
override func didReceive(_ request: UNNotificationRequest, withContentHandler  
contentHandler: @escaping (UNNotificationContent) -> Void) {  
  
    NetCoreNotificationService.sharedInstance().didReceive(request)  
    { (contentToDeliver:UNNotificationContent) in  
  
        contentHandler(contentToDeliver) }}}
```

### 3) Handle Notification Service Time Expire

```
override func serviceExtensionTimeWillExpire() {  
  
    NetCoreNotificationService.sharedInstance().serviceExtensionTimeWillExpire()  
  
}
```

4) After all the above code, Your **"Notification Service"** class should look like the image below.

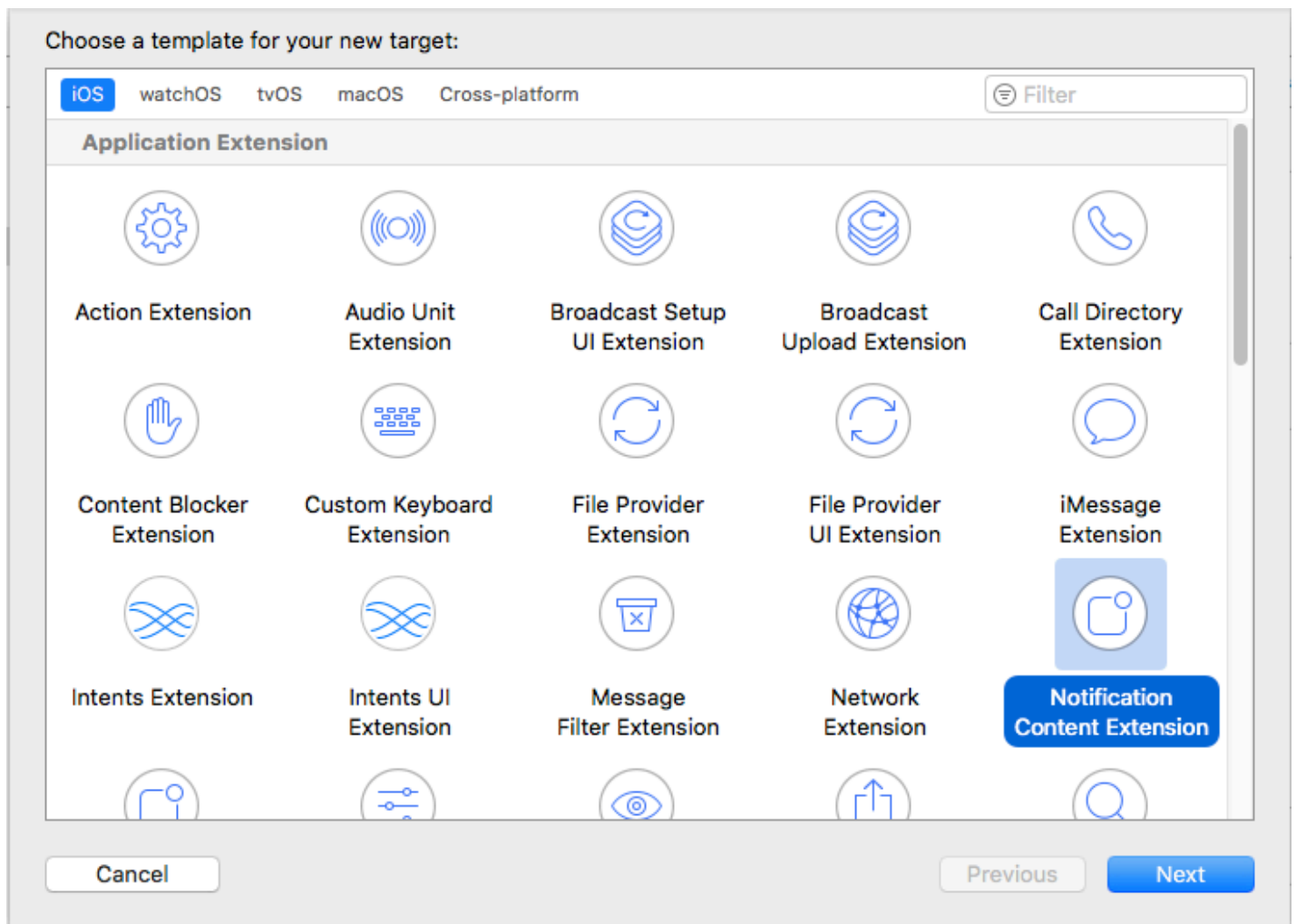
```

RichDemoSwift > Service Extension > NotificationService.swift > didReceive(_:withContentHandler:)
1 //
2 // NotificationService.swift
3 // Service Extension
4 //
5 // Created by Admin on 03/05/18.
6 // Copyright © 2018 Manish Kumar. All rights reserved.
7 //
8
9 import UserNotifications
10 import NetCorePush
11
12 class NotificationService: UNNotificationServiceExtension {
13
14     override func didReceive(_ request: UNNotificationRequest, withContentHandler
        contentHandler: @escaping (UNNotificationContent) -> Void) {
15
16         NetCoreNotificationService.sharedInstance().didReceive(request)
17         { (contentToDeliver:UNNotificationContent) in
18             contentHandler(contentToDeliver)
19         }
20     }
21
22     override func serviceExtensionTimeWillExpire() {
23         NetCoreNotificationService.sharedInstance().serviceExtensionTimeWillExpire()
24     }
25 }
26

```

### 13. For Carousel Push Notifications

1) Add **"Notification Content Extension"** to your app. [File->New->Target->Notification Content Extension](#).



- 2) Click Next and when asked to **"Activate"**, Click yes.
- 3) Add **"App Groups"** to your apps Capabilities(Add one group with name **"group.com.Smartech.com"**).
- 4) Enable **App groups in Service Extension** too and select group with name **"group.com.Smartech.com"**.
- 5) Replace **"MainInterface.storyboard"** of Content Extension with the provided by us.
- 6) In **"Info.plist"** file of Content Extension, replace **"UNNotificationExtensionCategory"** value with **"SmartechPushCategory"**

7) In **“Info.plist”** file of Content Extension, add **“UNNotificationExtensionDefaultContentHidden”** Boolean value with **“NO”**.

CarouselTestSwift > Content Extension > Info.plist > No Selection

Key	Type	Value
▼ Information Property List	Dictionary	(10 items)
Localization native development re...	String	\$(DEVELOPMENT_LANGUAGE)
Bundle display name	String	Content Extension
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	XPC!
Bundle versions string, short	String	1.0
Bundle version	String	1
▼ NSExtension	Dictionary	(3 items)
▼ NSExtensionAttributes	Dictionary	(3 items)
UNNotificationExtensionCategory	String	smartechPushCategory
UNNotificationExtensionInitialC...	Number	1
UNNotificationExtensionDefault...	Boolean	NO
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.usernotifications.content-extension

8) Replace **“NotificationViewController”** class files from the Rich Push folder into your project.

## Deployment Over Apple Store

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

- |  |   |
|--|---|
| ▶ Target Dependencies (0 items)        |   |
| ▶ Compile Sources (11 items)           | × |
| ▶ Link Binary With Libraries (2 items) | × |
| ▶ Copy Bundle Resources (5 items)      | × |
| ▶ Embed Frameworks (1 item)            | × |
| ▼ Run Script                           | × |

Shell /bin/sh

```
1 echo "Target architectures: $ARCHS"
2
3 APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"
4
5 find "$APP_PATH" -name '*.framework' -type d | while read -r
```

- ☒ Show environment variables in build log
- ☐ Run script only when installing

Input Files

Add input files here

## Output Files



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

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
    FRAMEWORK_EXECUTABLE_NAME=$(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
```

