

NetCore SDK

Push Notification Integration Help for iOS.

Push notifications have become more and more powerful since they were first introduced. Using Push Notifications is way to add real time messaging to your application. Push notifications allow

developers to reach users and perform small tasks even when users aren't actively using an app. NetCore Push SDK enables you to handle push notification in a better way.

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 App Id

You can create a wildcard App ID that matches one or more apps or an explicit App ID that exactly matches your bundle ID. The app services enabled for an App ID serve as a whitelist of the services one or more apps may use. What services an app actually uses is configured in the Xcode project. You can enable app services when you create an App ID or modify these settings later. Game Center and In-App Purchase are enabled by default for an explicit App ID.

To register an 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





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



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.

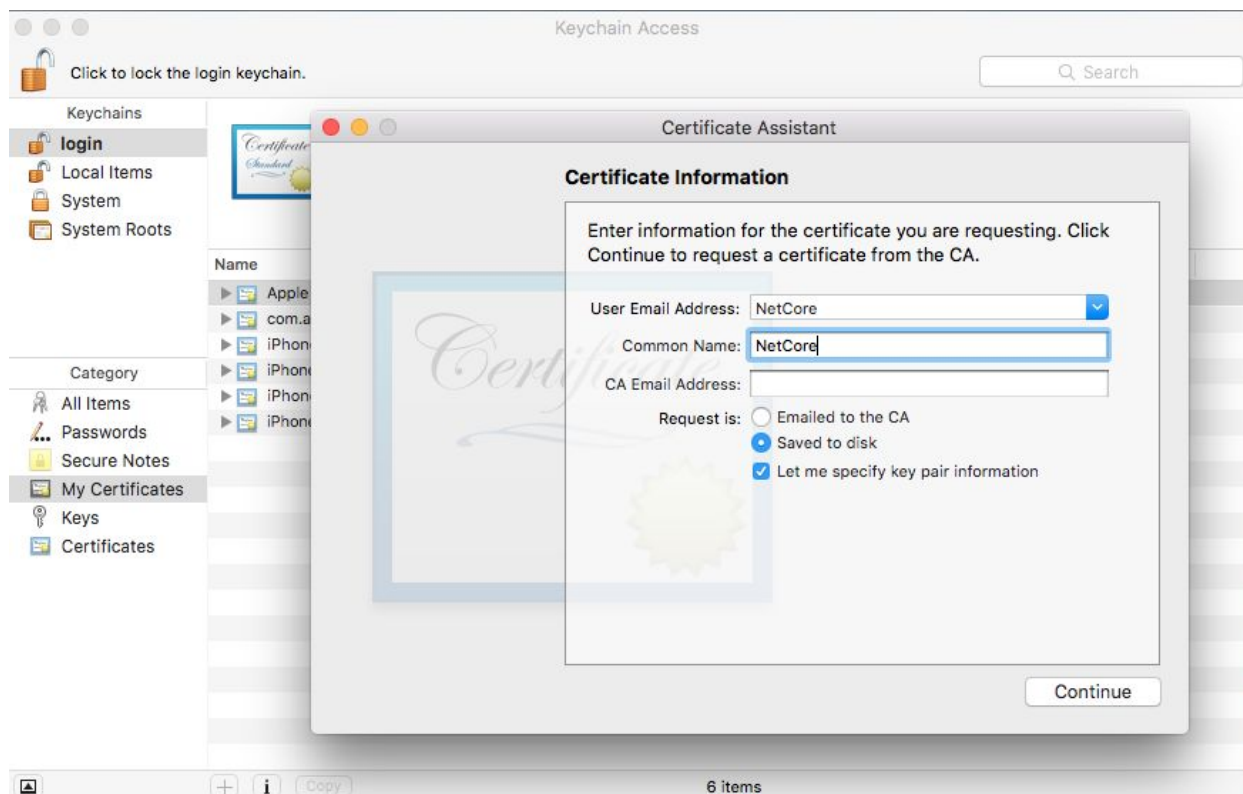
file

Cancel

Back

Continue

Creating CSR
from Keychain



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 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](#).

Creating Development Provisioning Profiles

Before creating a development provisioning profile, verify that you have an [App ID](#), one or more development certificates, and one or more devices. If you want to register your own App ID, read [Registering App IDs](#). (You can also use one of the App IDs that Xcode manages for you.) If you need to create your development certificate, read [Creating Signing Identities](#). If you need to register devices, read [Registering Devices Using Your Developer Account](#).

To create a development 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 the type of provisioning profile you want to create and click Continue.
5. Select the App ID you want to use for development, and click Continue.
6. Select one or more development certificates, and click Continue.
7. Select one or more devices, and click Continue.
8. Enter a profile name, and click Generate.
9. Click Done.

Add iOS Provisioning Profiles


+

Select Type

Configure

Generate

Download



What type of provisioning profile do you need?

Development

☒ **iOS App Development**

Create a provisioning profile to install development apps on test devices.

☐ **tvOS App Development**

Create a provisioning profile to install development apps on tvOS test devices.

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

An ad hoc provisioning profile allows testers to run your app on their device without needing Xcode. To create an ad hoc provisioning profile, you select an [App ID](#), a single distribution certificate, and multiple test devices.

Note: Alternatively, you can create an ad hoc provisioning profile in Xcode by exporting your app, described in [Exporting Your App for Testing \(iOS, tvOS, watchES\)](#).

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.

Creating Store Provisioning Profiles

Before uploading your app to the store, you provision it using a store provisioning profile. (For Mac apps that don’t enable any app services, you can code sign your app using just a distribution certificate.) You don’t select any devices to create a store provisioning profile.

To create a store 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 the distribution method, and click Continue.
5. Choose the App ID you used for development (the App ID that 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 Xcode Wildcard App ID, select it. If the menu contains an Xcode-managed explicit App ID (it begins with “Xcode” and contains your bundle ID), select that App ID. If you created your own App ID, select that one.
7. Select your distribution certificate, and click Continue.
8. A store provisioning profile contains a single distribution certificate.
9. Enter a profile name, and click Continue.
10. Wait while your developer account generates the provisioning profile.
11. At the bottom of the page, Click Done.

Distribution

- ☐ **App Store**
Create a distribution provisioning profile to submit your app to the App Store.
- ☐ **tvOS App Store**
Create a distribution provisioning profile to submit your tvOS app to the App Store.
- ☐ **Ad Hoc**
Create a distribution provisioning profile to install your app on a limited number of registered devices.
- ☐ **tvOS Ad Hoc**
Create a distribution provisioning profile to install your app on a limited number of registered tvOS devices.

Cancel

Continue

[Reference link](#)

[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 PARTY

POSTPAID

DO NOT DELIVER
is Disable

ACTIVE
since 14 Jun 2016

netCORE
Smartech

[Broadcast](#)

[Automation](#)

Manage

[Reports](#)

[Admin](#)

[Logs](#)

What's New

?

[List Dashboard](#)

[Templates](#)

[Assets](#)

Create New App

1 2 3 > >>

About netCORE

Product Demo

Email Marketing

Blog

Terms & Conditions

Anti-Spam Policy

Privacy Policy

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

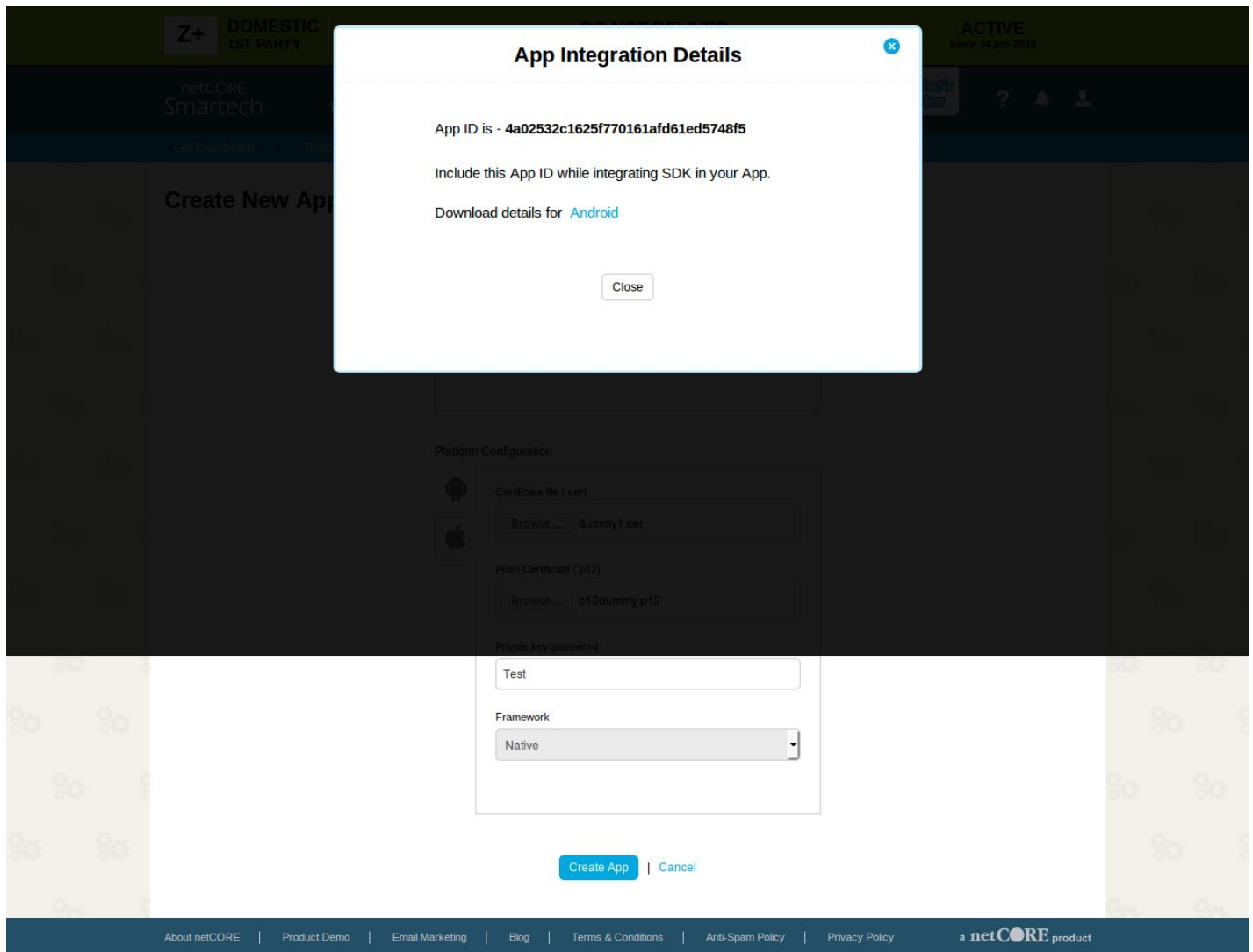
Blog

Terms & Conditions

Anti-Spam Policy

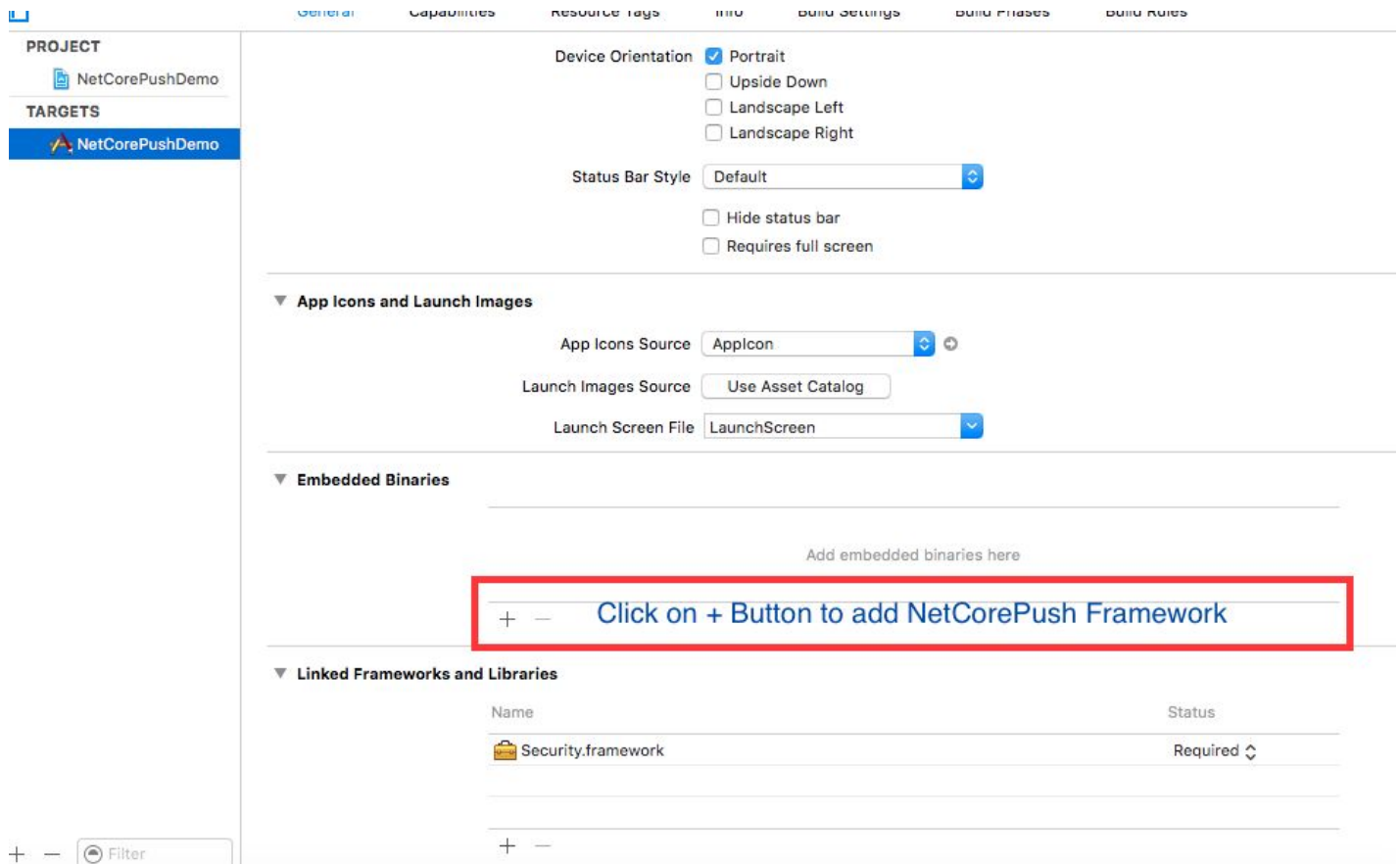
Privacy Policy

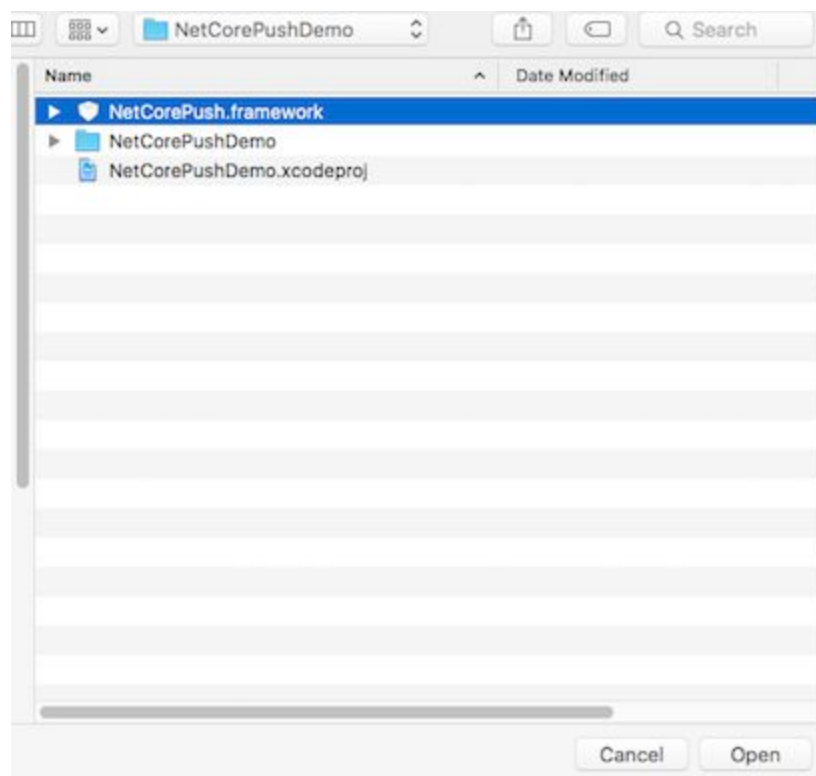
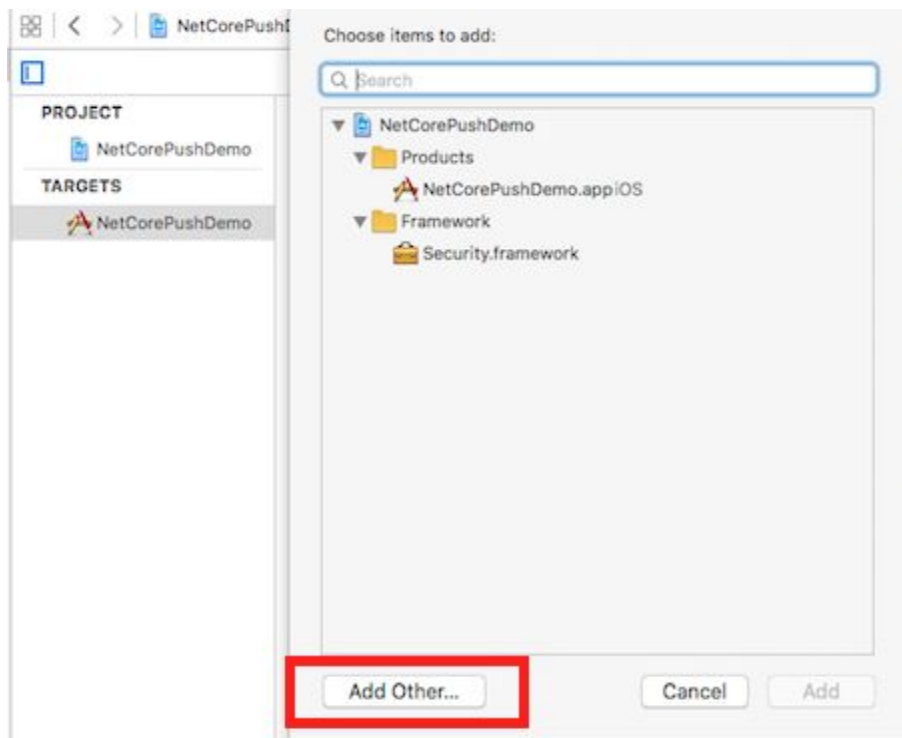
a netCORE product

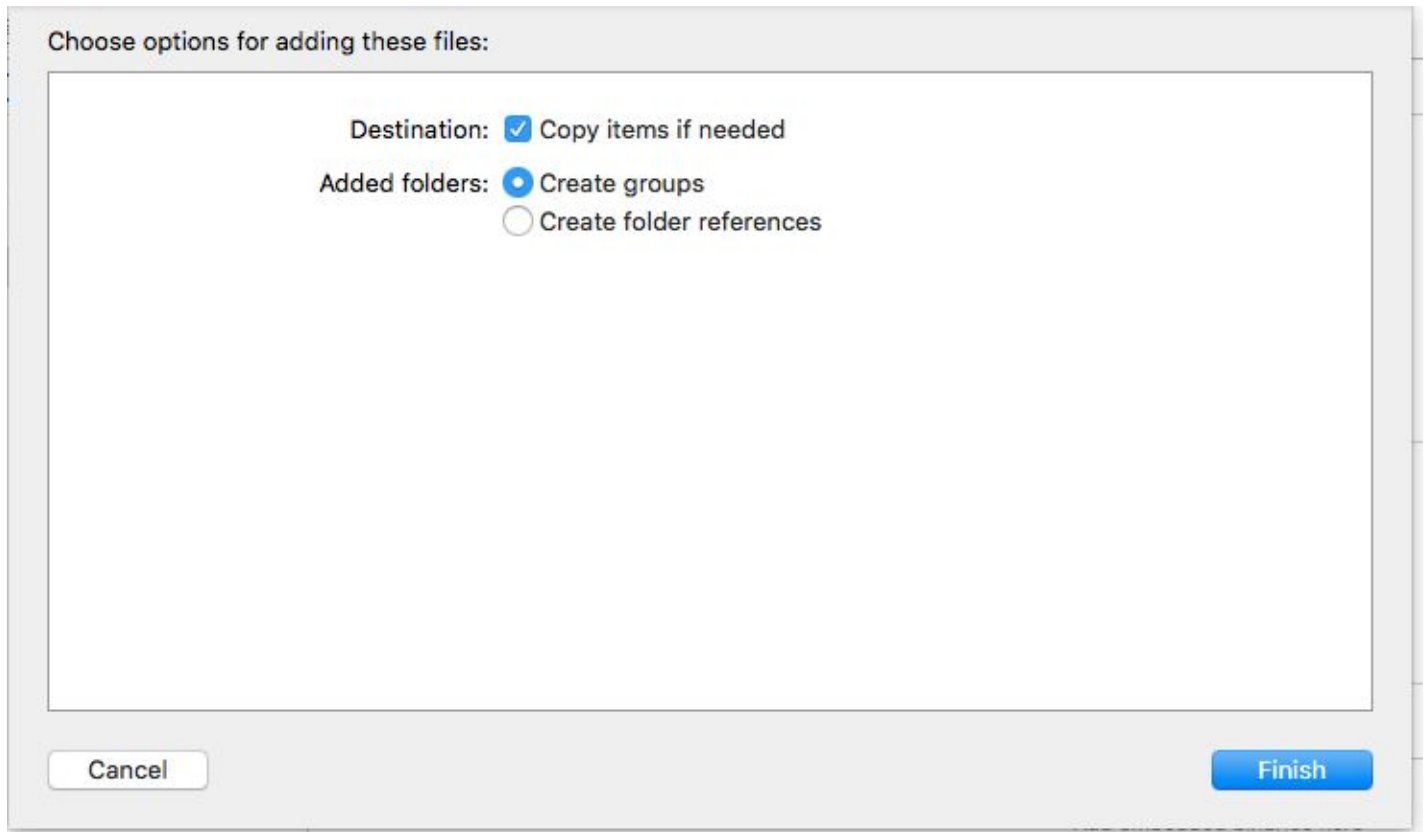


4. Setup the NetCore Push SDK

- 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. Integrate NetCore SDK & Push Methods.](#)

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


```
import NetCorePush
```

2) Import following file in App Delegate File

```
import User Notifications
import UserNotificationsUI
```

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

```
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)

```
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)

```

if (launchOptions != nil){

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

```

5. Register Device With NetCore SDK (AppDelegate file)

```

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)

```

func application(_ application: UIApplication, didReceiveRemoteNotification
userInfo: [AnyHashable: Any]){
    // perform notification received/click action as per third party SDK as per their
    document

    NetCorePushTaskManager.sharedInstance().didReceiveRemoteNotification(userInfo)
}

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) {
            // perform notification received/click action as per third party SDK as per their
            document
        }
    }
}

```

```

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) {

// perform notification received action as per third party SDK as per their document

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

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

```

7)Handle Deep Linking

```

func application(_ application: UIApplication,
                open url: URL,
                sourceApplication: String?,
                annotation: Any) -> Bool{
    if url.absoluteString.lowercased().contains ("your app deep link"){
        // 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
    }
}
}

```

9) Login with NetCore

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

10) Logout

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

11) Events Tracking :

Following is the list of tracking events

```
tracking_PageBrowse = 1,
tracking_AddToCart = 2,
tracking_CheckOut = 3,
tracking_CartExpiry = 4,
tracking_RemoveFromCart = 5,
tracking_FirstLaunch = 20,
tracking_AppLaunch = 21
```

You can use this events following ways

12)Track normal event

```
// for sending application launch event

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

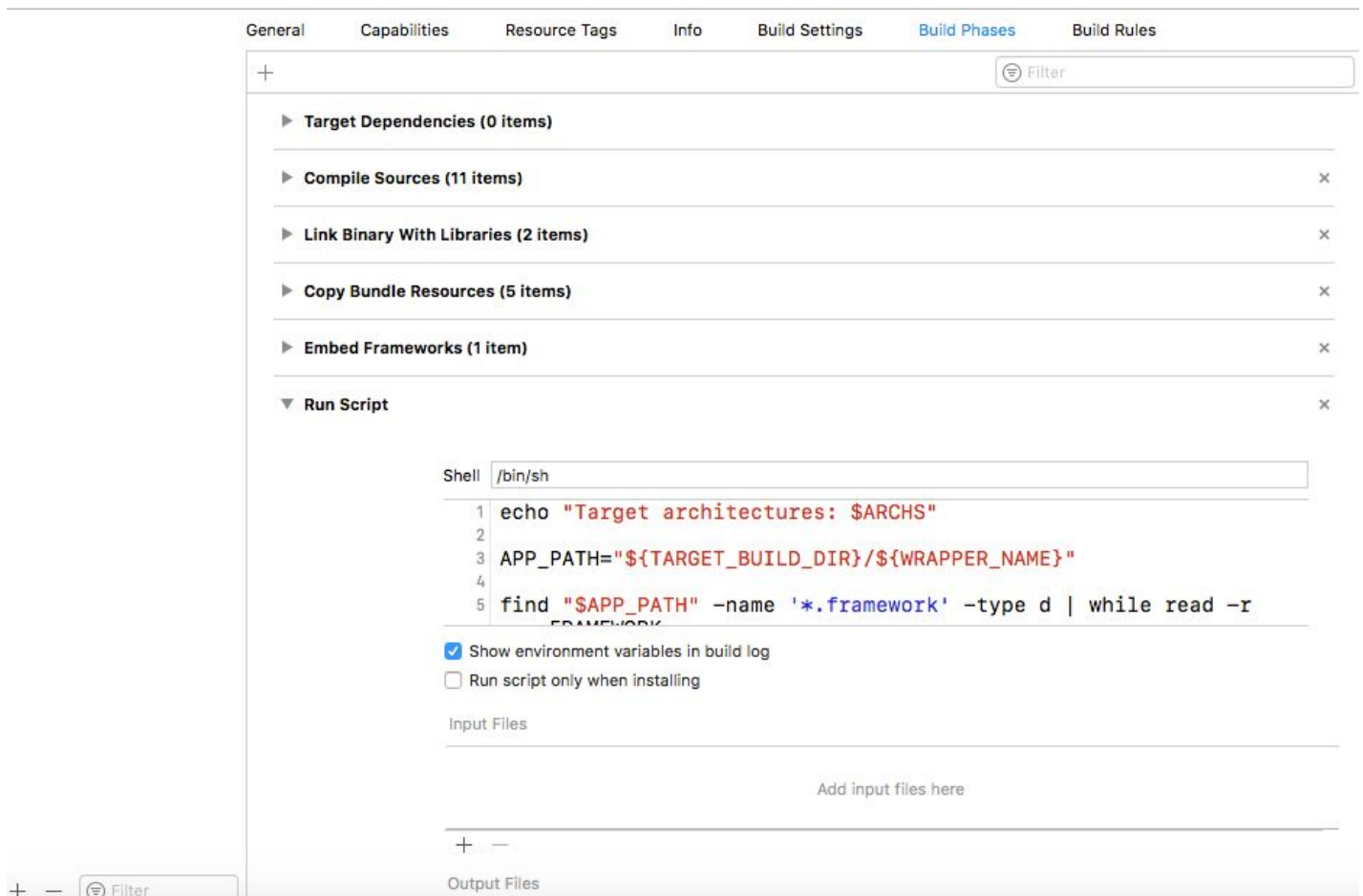
13)Track event with custom payload

```
//add To cart event with custom array of data
NetCoreAppTracking.sharedInstance().sendEvent(withCustomPayload:
```

```
Int(UInt32(tracking_PageBrowse.rawValue)), payload: arrayAddToCart , block: nil)
```

Deployment Over Apple Store

Add Following runscrip in your application target ,when you are deploying application over apple store,this run scrip 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_NAM
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
```