

NetCore IOS SDK

Integration Doc for Cordova/PhoneGap 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.

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 BundleID 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

[Back to 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.

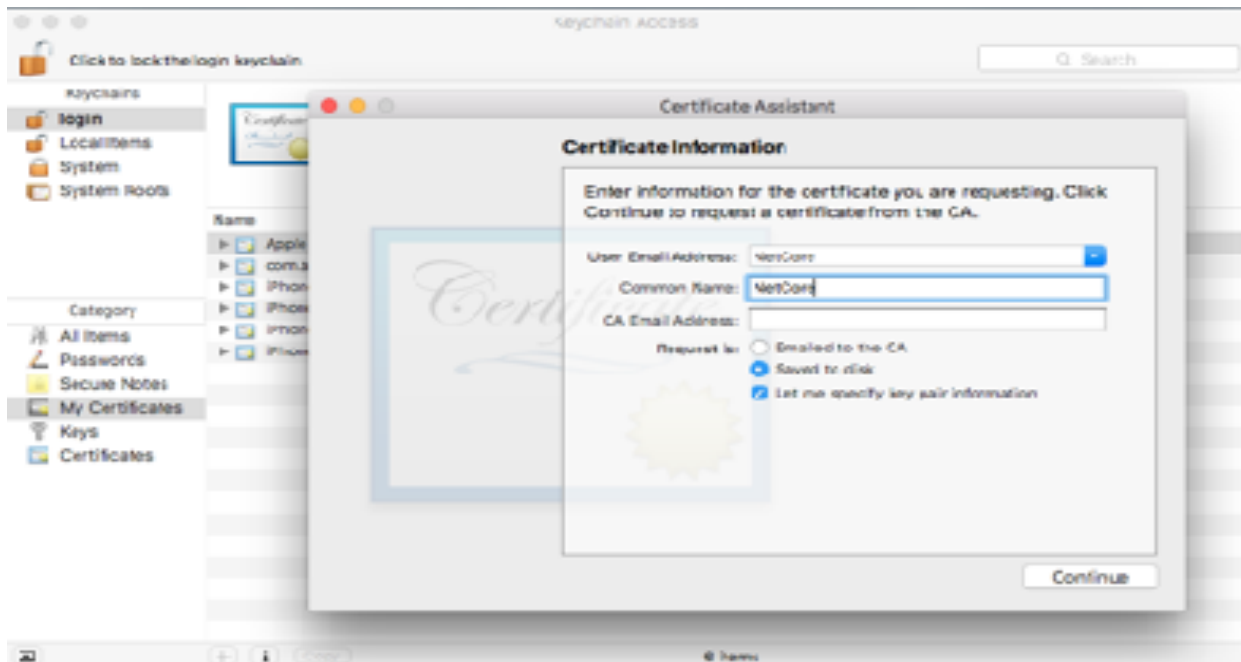
Create a CSR file.

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 to:" 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

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 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, 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 shows the NetCore Smartech App Dashboard. At the top, there's a green header with 'Z+ DOMESTIC 1ST PARTY', 'POSTPAID', 'DO NOT DELIVER In Brazil', and 'ACTIVE since 14 Jan 2018'. Below this is a dark blue navigation bar with 'netCORE Smartech' logo, 'Broadcast', 'Automation', 'Manage' (highlighted), 'Reports', 'Admin', and 'Logs'. A 'What's New' badge is also present. Below the navigation bar is a light blue bar with '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	SUBS/DEVS	PLATFORMS
Testone	4a02530c1a25f70106af01ed37465	0	Android, iOS
Testing	4004830581c02c0c01700178571079	2	Android, iOS
TestQuagria	21348d9fa4b547:58ec8c2768da5e87	250	Android, iOS
Testing_Newit	75a841c018d1340c019c7085ca08a	0	Android, iOS
NewitCGTesting	c8d55d93a8fbd57bce507b70100479c	0	Android, iOS

At the bottom of the table, there are pagination controls: '1 2 3 > 30'.

The footer of the page contains links: 'About netCORE', 'Privacy Policy', 'Email Marketing', 'Ping', 'Terms & Conditions', 'Anti-Spam Policy', 'Privacy Policy', and 'a netCORE product'.



Create New App

App Name *

App Description

Platform Configuration



Certificate file (.cer)

No file selected



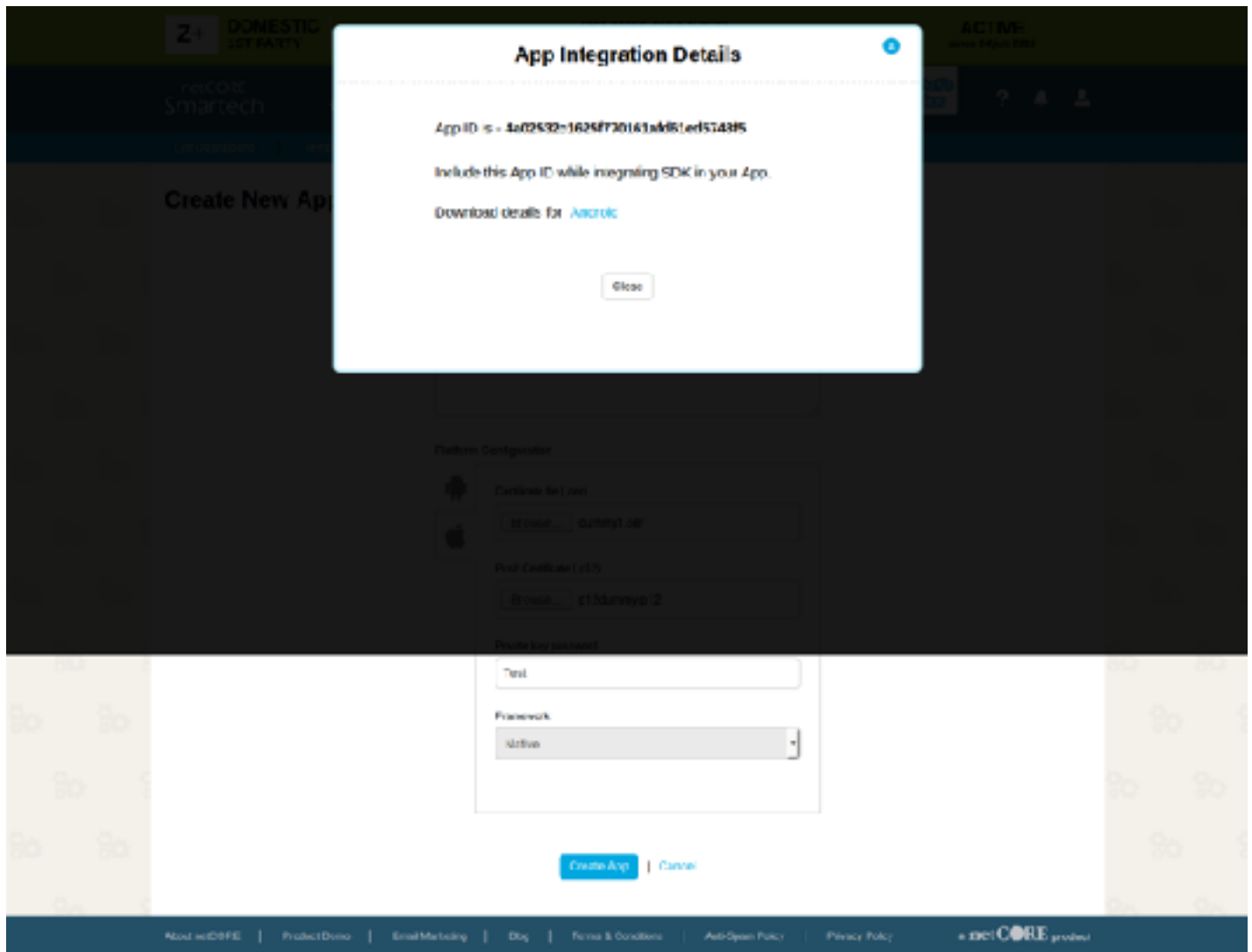
Push Certificate (.p12)

No file selected

Private key password

Framework

Native



[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

PROJECT

NetCorePushDemo

TARGETS

NetCorePushDemo

GENERAL

ADDITIONAL OPTIONS

PROVISIONING PROFILE

IDENTITY

FRAMEWORKS AND LIBRARIES

EMBEDDED BINARYS

LAUNCH IMAGES

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

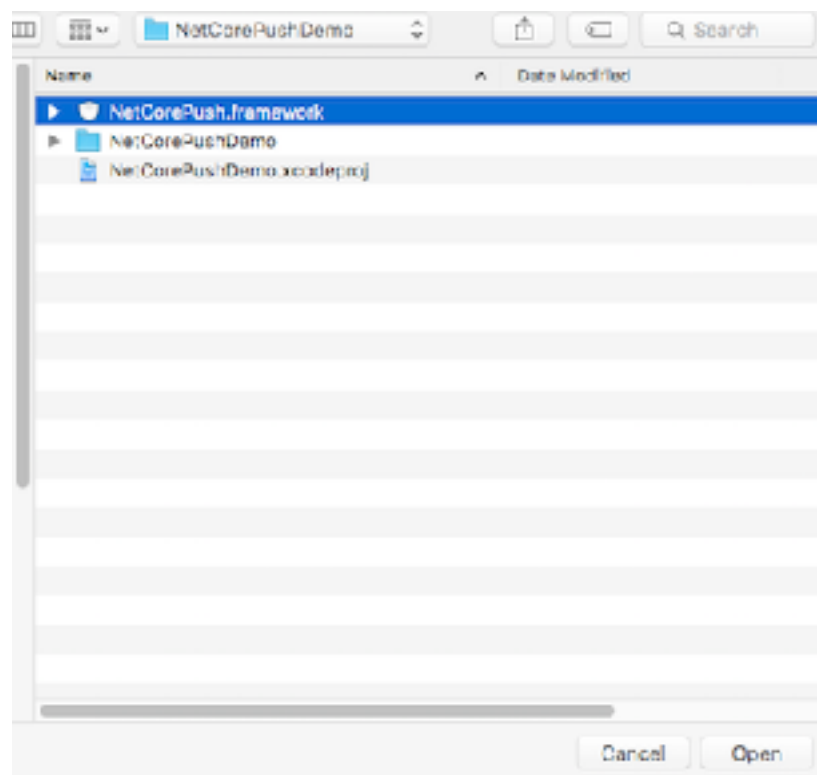
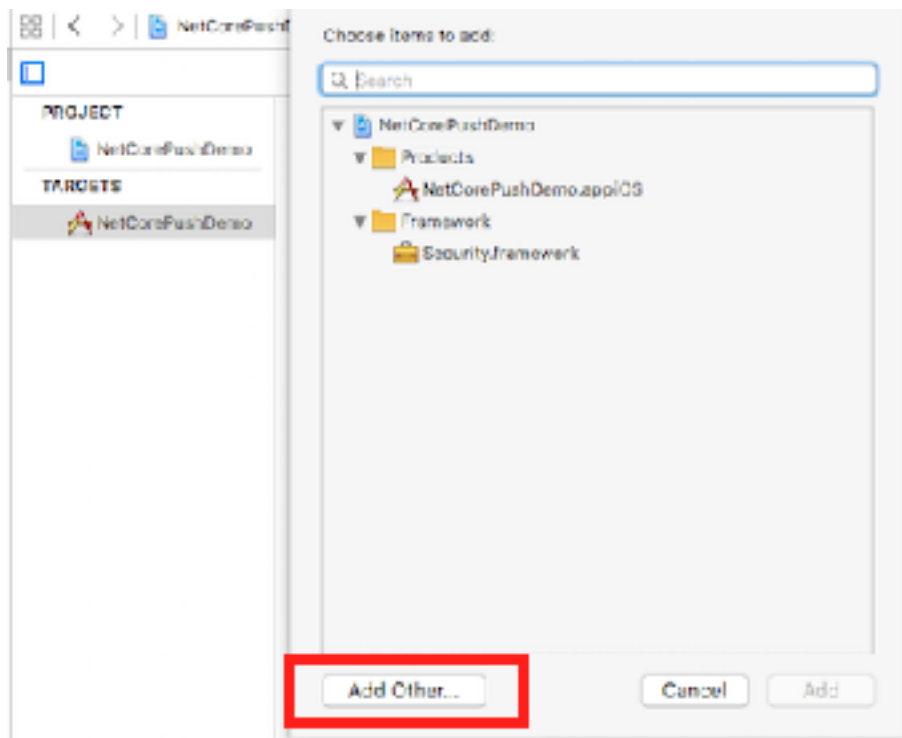
Add embedded binaries here

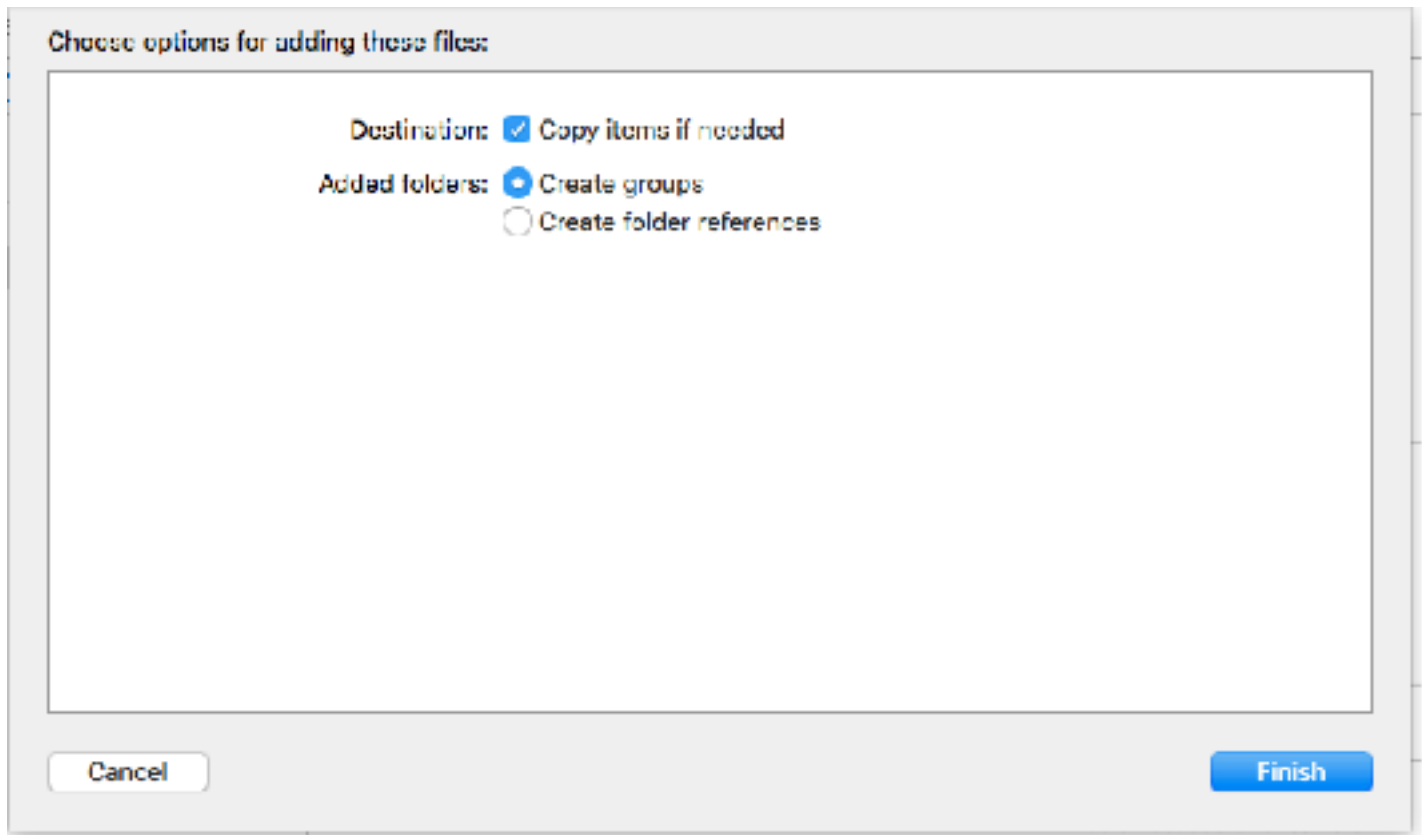
Click on + Button to add NetCorePush Framework

Linked Frameworks and Libraries

Name	Status
 Security.framework	Required 

+ -





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

NetCore SDK Initialization

1) Import following file in AppDelegate File

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

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

```
#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
```

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

```

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

```

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

```

// Handle launch event
if (launchOptions != nil){
    [[NetCorePushTaskManager sharedInstance]handelApplicationLaunchEvent:
launchOptions];
}

```

5. Register Device With NetCore SDK (AppDelegate file)

```

- (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) {
    }];
}

```

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


```

-(void)application:(UIApplication *)application didReceiveLocalNotification:
(UILocalNotification *)notification{
    [[NetCorePushTaskManager
sharedInstance]didReceiveLocalNotification:notification.userInfo];
}
- (void)application:(UIApplication*)application didReceiveRemoteNotification:
(NSDictionary*)userInfo{
    [[NetCorePushTaskManager
sharedInstance]didReceiveRemoteNotification:userInfo];
}
//Called when a notification is delivered to a foreground app.
-(void)userNotificationCenter:(UNUserNotificationCenter* )center
willPresentNotification:(UNNotification* )notification withCompletionHandler:(void
(^)(UNNotificationPresentationOptions options))completionHandler{
    completionHandler(UNAuthorizationOptionSound | UNAuthorizationOptionAlert |
UNAuthorizationOptionBadge);
    [[NetCorePushTaskManager
sharedInstance]userNotificationCenterWillPresentNotification:notification];
}
//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(^)(void))completionHandler{
    [[NetCorePushTaskManager
sharedInstance]userNotificationCenterDidReceiveNotificationResponse:response];
}

```

7)Handle Deep Linking

```

- (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
    }
}
}

```

8) Add NetCore Cordova Plugin files (NetCorePushPlugin.m and NetCorePushPlugin.h files) in Plugins folder.

You can download from <https://github.com/NetcoreSolutions/Smartech-ios-sdk>

9) Configure this plugin to project

add below code to config.xml

```

<platform name="ios">
    <config-file target="config.xml" parent="/*">
        <feature name="NetCorePushPlugin">
            <param name="ios-package" value="NetCorePushPlugin"/>
        </feature>
    </config-file>
    <header-file src="src/ios/NetCorePushPlugin.h" />
    <source-file src="src/ios/NetCorePushPlugin.m" />
</platform>

```

10) Now add below sample events wherever needed in cordova js file (which are inside www/js folder) as per your requirement

- For login event

```
cordova.exec(null, null, "NetCorePushPlugin", "pushLogin", [<yourIdentity>]);
```

- For logout event

```
cordova.exec(null, null, "NetCorePushPlugin", "pushLogout", []);
```

- **For activity tracking**

- a)app launch**

- var launchId = 20**

- cordova.exec(null, null, "NetCorePushPlugin", "sendInAppActivity",[**launchId**]);

- b) add to cart custom event with custom payload**

- var addToCartEventId = 2; // example

- var productList = [{"s^name":'Nexus 6', "i^prid": 1, "s^price": 199.99,"i^prqt":2 },
{"s^name": 'Nexus 5', "i^prid": 2, "s^price": 155.22,"i^prqt":1 }];

- cordova.exec(null, null, "NetCorePushPlugin", "sendInAppActivity",[**addToCartEventId**, productList]);

- **For user profile push**

- var profilePushInfo = {"name":"Tester", "age": "23", "mobile":"32424342"}

- cordova.exec(null, null, "NetCorePushPlugin", "profilePush", [yourIdentity, profilePushInfo]);

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

GeneralCapabilitiesResource TagsInfoBuild SettingsBuild PhasesBuild Rules

Filter

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

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

Filter

```
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"
    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
```

