

WHAT IS PUSH NOTIFICATION?

PUSH Notification is a modern way thanks to which application publishers can notify their users about some kind of action at any time.

Users do not need to have the application opened to receive notifications – which gives us (developers) an incredible tool to keep our user-base active.



ISSUES WITH MULTI-PLATFORM PUSH

Each platform has it is own concept of "Remote Notification Server". Android uses GCM/FCM, iOS – APNS, Windows Mobile – WNS.
Therefore implementation on each platform is completely different. This brings us some thoughts, can we at least partially:

- share code related to registering to PUSH Notification Provider?
- share code related to presenting and handling user-action response of PUSH Notification?

Unfortunately, with "classic" approach presented in official Xamarin/Microsoft samples it is not possible, which in the end results in thousands of duplicated lines of code across different Mobile Projects.





IN'SANELAB GIFT INSANE NOTIFICATIONS LIBRARY

Most of the apps we develop at In'saneLab have PUSH Notifications feature and we could not stand wasting time on writing almost the same code over and over again.

Therefore, we decided to create multiplatform Xamarin library to simplify this process – InsaneNotifcations.





INSANE NOTIFICATIONS FEATURES

- Works for Xamarin.Android, Xamarin.iOS, UWP
- Remote Notifications support for GCM (FCM is on roadmap), APNS and WNS
- Notification Presenter concept based on MvvmCross View Presenters small, modular classes that simplify handling and displaying received PUSH Notification on each platform
- Handling edge cases for example: automatic re-registration to background service when app is updated/device is rebooted on UWP
- All features have corresponding package with support for MvvmCross Framework
- Local Notifications are not supported yet however you can use "Notification Presenter" to speed up PUSH handling development





INSANE NOTIFICATIONS FEATURES WITH BACKEND-BASED PUSH

- Sample works by connecting to REST API that uses Azure Notification Hub.
- Sample C# WebAPI Backend project (used in current sample) is available here:

https://github.com/thefex/Insane.Notifications/tree/master/Insane.Notifications/Samples/PUSH/Insane.PushSample/Insane.PushSample.Backend





IMPLEMENTATION STEPS - PORTABLE

- 1. Implement IRemotePushRegistrationService Backend based implementation should call your REST API method that registers/unregister your device from PUSH Service.

 Sample implementation link
- 2. Implement IPushTagsProvider it should return tags (Notification Hub concept) for which user is/should be currently connected. In case you are not using any Notification Hub tags your implementation can return an empty list.



IMPLEMENTATION STEPS - PORTABLE

- 3. Implement model class/classes that represents the model of data you want to PUSH
- Create those services as a singleton (one instance across application) – perhaps by using your Dependency Injection Container <u>MvvmCross sample link</u>
- 5. Use INotificationsService and call Subscribe/Unsubscribe methods when appropriate (after user logs in/logs out?). It will automatically call a registration process on each platform.

 Code sample link





- Add assembly attribute
 [assembly: UsesPermission(Name = "@PACKAGE_NAME@.permission.C2D_MESSAGE")]
- 2. Add Android Service which inherits from InsaneGcmService You should implement GetPushJsonData method it should extract json data which models your PUSH Notification.

 Code sample link
- 3. Add BroadcastReceiver class which inherits from **GcmBroadcastReceiver** (copy-paste from sample) Code sample link





3. Implement IRemoteNotificationIdProvider
This interface is used to determine the type name of your PUSH Notification.

We can send notifications of different types, the example:

- You have a new Facebook connection (someone accepted your friend request),
- Someone liked your photo on Facebook

We have to somehow select a "Notification Type" from data we have in PUSH Notification. The easiest way is to embed push type in notification json content. Code sample link



- 4. As in Portable, register your services as singletons.
- INotificationsService to GcmRemotePushNotificationService,
- IRemoteNotificationIdProvider to implementation you have created in step 3
- IRemoteNotificationsPresenter to RemoteNotificationsPresenter





5. Implement RemoteNotificationHandler<TypeOfNotificationModelClass> for each PUSH type.

If you want to show a standard Android PUSH – inherit from: **DroidRemoteNotificationHandler<TypeOfNotificationModelClass>**

Implementation will take care of handling/displaying PUSH Notification.

Each handler class should have attribute:

[RemoteNotificationHandlerAttribute("push_type_returned_in_remote_notification_id_provider_implementation")]





6. Your activities should have two additional lines of code:

"Intent?.Extras?.HandlePushDataIfExists();" – in the end of OnCreate(Bundle) method "intent?.Extras?.HandlePushDataIfExists();" – in the end of OnNewIntent(Intent intent) method

Those lines take care of background/foreground PUSH activation.





- As in Android you have to implement
 IRemoteNotificationIdProvider.
 Sample implementation which is adjusted to Apple Push Notification json data template can be found here.
- Override two methods in AppDelegate "void RegisteredForRemoteNotifications", "void FailedToRegisterForRemoteNotifications" like in Code Sample.



3. Register your services as singletons as you would do usually. After that – call

PushiOSNotificationsSetup.Initialize(IIOSRemoteNotificationsPresenter instance); - and you are done ☺

RemotePushNotificationService to iOSRemotePushNotificationService

INotificationsService to RemotePushNotificationService

IRemoteNotificationIdProvider to your implementation

IIOSRemoteNotificationsPresenter to iOSRemoteNotificationPresenter





4. Implement "Notification handler" for each Notification type. (like in Android sample)

APNS supports four types of notifications:

- Toast Notification (standard toast shown automatically) to support this kind of notification your handler should implement IIOSAlertNotificationHandler interface Code sample link
- Badge Notification (badge on app icon) your handler should implement IIOSBadgeNotificationHandler interface





- Sound Notification (sound played) your handler should implement
 IIOSSoundNotificationHandler interface
- RAW Notification (does not have any automatic action, implementation is totally up to you) – your handler should implement IRemoteNotificationHandler interface

Make sure that your Handler class has a "[RemoteNotificationHandler("notificationIdWhichShouldMatchWithYourRemoteNotificationsIdProvider")] attribute.

Additionally - if you want to handle user actions - your handler should inherit from **iOSRemoteNotificationTapAction**.





- 1. WNS (Windows Notification Service) supports four kinds of notifications templates:
 - a) Badge Notification (badge on app tile) xml-based notification template
 - b) Tile Notification (update application tile) xml based notification template
 - c) Toast Notification (standard application toast) xml based notification template
 - d) Raw Notification (your custom logic) notification template can be an json



As we have four different notifications content – we should implement four interfaces.

- IRemoteNotificationIdProvider as in Android/iOS returns Notification ID for RAW Notification
- IToastRemoteNotificationHandlerIdProvider returns notification
 ID for Toast Notification, code sample
- ITileRemoteNotificationHandlerIdProvider returns notification ID for Tile Notification
- IBadgeRemoteNotificationHandlerIdProvider returns notification
 ID for Badge Notification





- 2. As usual you should implement Notification Handler for each Notification Type you plan to push.
- a) Toast Notification Handler should inherit from ToastRemoteNotificationHandler and it should have [ToastRemoteNotificationHandler("notificationId")] attribute, code sample
- b) Badge Notification Handler should implement

 IBadgeRemoteNotificationHandler and it should have

 [BadgeRemoteNotificationHandlerAttribute("notificationid")]

 attribute





- c) Tile notification handler should implement ITileRemoteNotificationHandler and have [TileRemoteNotificationHandler("notificationId")] attribute
- d) Raw Notification should implement same interface as iOS/Android IRemoteNotificationHandler + it should have [RemoteNotificationHandler("notificationId")] attribute

If you want to handle "Tap on push action" on your RAW notification you should implement IRemoteNotificationTapAction as well. If you are going to display "Toast" notification – make sure you provide "LaunchArgument" using

LaunchArgumentSerializator.SerializeLaunchArgument(...) helper.





- 3. Add necessary methods in your Application class (App.xaml.cs).
- On the end of "OnLaunched" method you should call PushServicesExtensions.SetupNotificationsIfNeeded(..) method
- On the end of "OnActivated" method you should call PushServicesExtensions.HandlePushActivation(..) method
- On the end of "OnBackgroundActviated" method you should call PushServicesExtensions.HandlePushRelatedBackgroundActivation(..) method

Check code sample.

4. Register your PUSH related services as singletons.

Code sample is available here.





- 1. Is FCM (Firebase Cloud Messenging) supported?
 Unfortunately not but it is in the library roadmap. Please, use GCM until that.
- 2. Can I use Azure Notification Hubs without Backend? Sure. InsaneNotifications Remote Notifications have been developed to support different providers. (as long as they use WNS/GCM/APNS) Simply use custom IRemotePushRegistrationService – instead of using "REST API" as in sample app use
 - Microsoft.Azure.NotificationHubs library.



3. How can I configure Azure Notification Hub to work with my apps? Please check official Microsoft Documentation,

<u>https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-android-push-notification-google-gcm-get-started</u> - **GCM**

https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-ios-apple-push-notification-apns-get-started - APNS

<a href="https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-windows-store-dotnet-get-started-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification-hubs-wns-push-notification





- 4. How should sent PUSH template look like?
- a) APNS template is described <u>here</u>. It is json data wrapped into "aps" key.

- c) WNS default templates are:
 - Toasts XML Documentation
 - Badge template <badge value="numericValueVisibleByApp">
 - Tile XML Documentation
 - RAW Notification pure text, if you are using it with InsaneNotification please use JSON data format.





5. I am using standard UWP Toast notifications but my Tap Action handler does not work when the application is closed.

It is caused due to the complex nature of UWP Notifications. There are two solutions to the issue:

a) Pass a launch argument at the top of your Toast PUSH XML. It should be in such format:

"YourNotificationHandlerId\$INSANELAB_NOTIFICATIONS\$base 64ofYourJsonDataParameters"





b) Use RAW notifications all the time as it is easier to handle. In RAW Notification Handler display Toast using Microsoft.Toolkit.Uwp.Notifications helper library. Your toast "LaunchArguments" property should be set to: LaunchArgumentSerializator. SerializeLaunchArgument(your notification handler id, your push data model object which will be passed to Tap Action handler)

It's pretty complicated but that's the simplest workaround for the complex nature of **UWP** Background Notification handling.



DZIĘKUJĘ ZA UWAGĘ!

ZAPRASZAMY NA KOLEJNE SPOTKANIA! PRZEMYSŁAW RACIBORSKI



