Salesforce Mobile Notifications Implementation Guide

Salesforce, Winter '27

Last updated: October 9, 2026

© Copyright 2000–2026 Salesforce, Inc. All rights reserved. Salesforce is a registered trademark of Salesforce, Inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

Mobile Notifications Overview 1
Push Notification Registration and Flow 3
Steps for Implementing Mobile Notifications 6
Step 1. Developer Registration with Mobile OS Vendors 6
Step 2. Creating an External Client App 7
Create an External Client App for Android 8
Create an External Client App for Apple iOS 9
Testing Your External Client App Push Configuration 11
Step 3. Configure Your Mobile SDK App 12
Enable Push Notifications in a Salesforce Mobile SDK Android App 12
Enable Push Notifications in a Salesforce Mobile SDK iOS App 13
Enable Push Notifications in a Salesforce Mobile SDK Hybrid App 16
Handle Notification Encryption in Salesforce Mobile SDK Apps 17
Build Your Own In-App Notification Feature 18
Step 4. Packaging Components for Customer Organizations 22
Sending Mobile Notifications 23
Using Apex Triggers to Send Push Notifications 23
Custom Notification Actions 26
Using the Connect REST API Push Notification Resource (Optional) 29
PushNotificationPayload Class 30
apple(alert, sound, badgeCount, userData) 30
apple(alertBody, actionLocKey, locKey, locArgs, launchImage, sound, badgeCount, userData) 31
Apex Limits Functions for Push Notifications 33
Push Notification Limits 34
Debug Log Events 35
Notifications Resources 36
CustomNotificationType (Tooling API) 36
Notification Builder Platform Push Payloads 38
Index 39

MOBILE NOTIFICATIONS OVERVIEW

Mobile notifications on Lightning Platform allow mobile app developers to notify their customers when business events occur in customers' orgs. Salesforce provides considerable flexibility for using mobile notifications. You can push notifications to devices or deliver them within an app.

All notifications sent from Salesforce can support both Salesforce apps and custom apps. You can define custom types to fit your own business needs. However, an org can limit the types it allows and their routings.

Lightning Platform provides two server-side notification systems:

Notification Builder: Full-featured, UI-centric platform. Notification Builder is integrated with the core notifications framework that powers Chatter and other Salesforce apps. As a result, you can use familiar Salesforce tools for creating, scheduling, and sending notifications. For example, you can send invocable action notifications through Process Builder, Flow, or REST API. You can also retrieve past notifications for use in custom in-app notification displays. You can send notifications to both Salesforce apps and customer apps.

Apex: Legacy code-based platform for push notifications only. With Apex notifications, Apex triggers capture business events in the customer's org. Apps can then send push notifications with either Apex or Connect REST APIs. Apex push does not retain notification payloads for subsequent use.

Considerations

If you have an existing Apex push implementation, the legacy platform remains fully supported.

Consider which notification features suit your business needs. Types include:

Mobile push notifications: Notifications sent in reaction to an event. For example, push notifications usually arrive on mobile devices while the customer isn't looking at the app. Salesforce supports two types of push: Apex push—traditional "send, fire, and forget" mechanism—and Notification Builder push, which boasts a richer feature set. Notification Builder push notifications require you to subscribe your external client app to the custom notification types you plan to send.

In-app notifications: Notifications sent to customers while they're using either the Salesforce mobile app, another Salesforce app, or any properly configured custom app. In Salesforce mobile app, customers receive these notifications in the Notification Bell. Other client apps can implement their own notification displays using Notification Builder APIs.

Custom notification types: Notifications that you create through Notification Builder in Setup to suit your unique business needs. You can then send notifications of those types through Process Builder, Flow Builder, or the invocable action API.

Configurable delivery settings and preferences: For custom notifications types, you can select delivery channels—desktop, mobile, or both. You can choose to subscribe your external client apps to custom notification types to have them returned by the API to your custom in-app notification tray. If preferred, you can also send your subscribed notifications as push notifications.

Invocable action notifications: You can send custom notifications as invocable actions using the `customNotificationAction` API.

Scheduled notifications: In Process Builder, you can schedule a notification to be sent when triggered by a record change, platform event, or process. In Flow Builder, you can schedule the `customNotificationAction` API to send custom notifications at predefined intervals.

Notification platforms support the following types and features:

Type or Feature	Apex Push	Notification Builder Platform
Push notifications	Supported	Supported
Subscriptions	Not supported	Supported
Scheduled notifications	Not supported	Supported
Invocable action notifications	Not supported	Supported
In-app notifications	Not supported	Supported
Retrievable payloads	Not supported	Supported
Custom notification types	Not supported	Supported
Encryption	Not supported	Supported

Notification systems deliver payloads and the list of recipients to services operated by device OS vendors—Apple, Google—for delivery to customers' mobile devices. Salesforce preserves sent Notification Builder notifications and provides APIs for retrieving them.

To handle events that Apex triggers can't catch—for example, from sources outside of Salesforce—Connect REST API provides an alternative sending mechanism on page 29. You can use Connect REST API to send either Apex push or Notification Builder invocable actions.

All apps that intend to receive mobile notifications require:

A Salesforce external client app configuration on the Salesforce server: All Salesforce mobile notifications use the external client app framework. They support customer apps built in-house; Mobile SDK apps distributed through the App Store or Google Play' and partner or ISV apps installed through managed packages.

Registration with targeted mobile OS vendors—Apple and Google: You must register as a developer of apps that are designed to receive notifications. This registration is separate from the runtime registration that an app performs to receive notifications.

Minimal coding in client apps: Mobile SDK provides boilerplate implementations for app notification registration. You can choose whether to go further with features such as in-app notifications.

Server-side coding: Depending on the mechanism used, sending push notifications can require coding of Apex triggers or REST API calls.

This guide covers these configuration tasks, and also testing and sending notifications. To learn how to use Notification Builder functionality in Salesforce, see Manager Your Notifications with Notification Builder. For notification types, see Salesforce App Notification Types.

These instructions do not apply to Marketing Cloud apps. To implement Marketing Cloud MobilePush notifications, see MobilePush and Journey Builder for Apps SDKs.

See Also

- Create a Custom Notification Type
- Send Custom Notifications
- Send Custom Notifications with Notification Builder Platform
- Send a Custom Notification from a Process
- Flow Core Action: Send Custom Notification

IN THIS SECTION:

Push Notification Registration and Flow
To enable push notifications, you register with several different entities and configure the required settings.

Push Notification Registration and Flow

To enable push notifications, you register with several different entities and configure the required settings.

Entities involved when sending a push notification include:

- The OS vendor that delivers the notification to devices
- The Salesforce organization that sends the notification
- The mobile devices that receive and display the notification

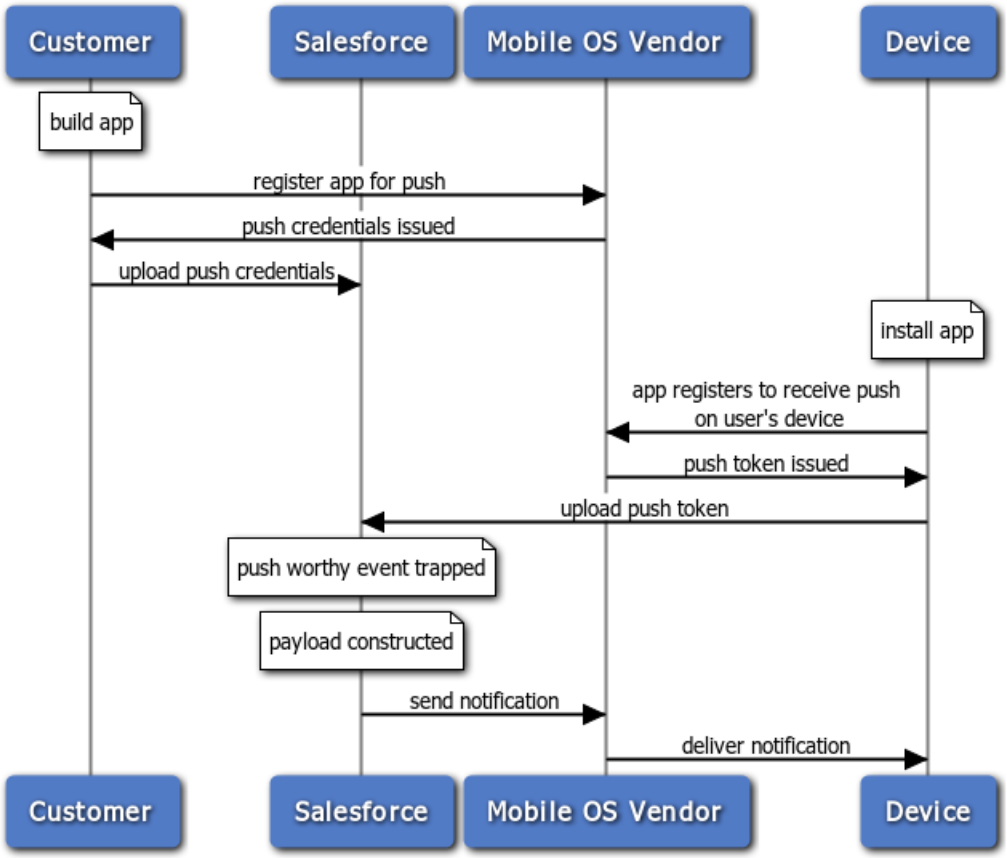
Here is an outline of the registration process for developers.

1. Register with the mobile OS vendor, such as Apple or Google, for push service.
2. Create a external client app in Salesforce to upload the push credentials, such as the iOS `.p12` certificate or the Android token.
3. Enable the mobile client app to handle push notifications using the Salesforce Mobile SDK.
4. Write Apex triggers to send push notifications when certain events occur on Salesforce records.
5. (For partners only) For partners who are developing the push notification triggers for other customer organizations, this additional step should be performed.

   Create a second-generation managed package containing the external client app and Apex triggers. Distribute this package to customer organizations.

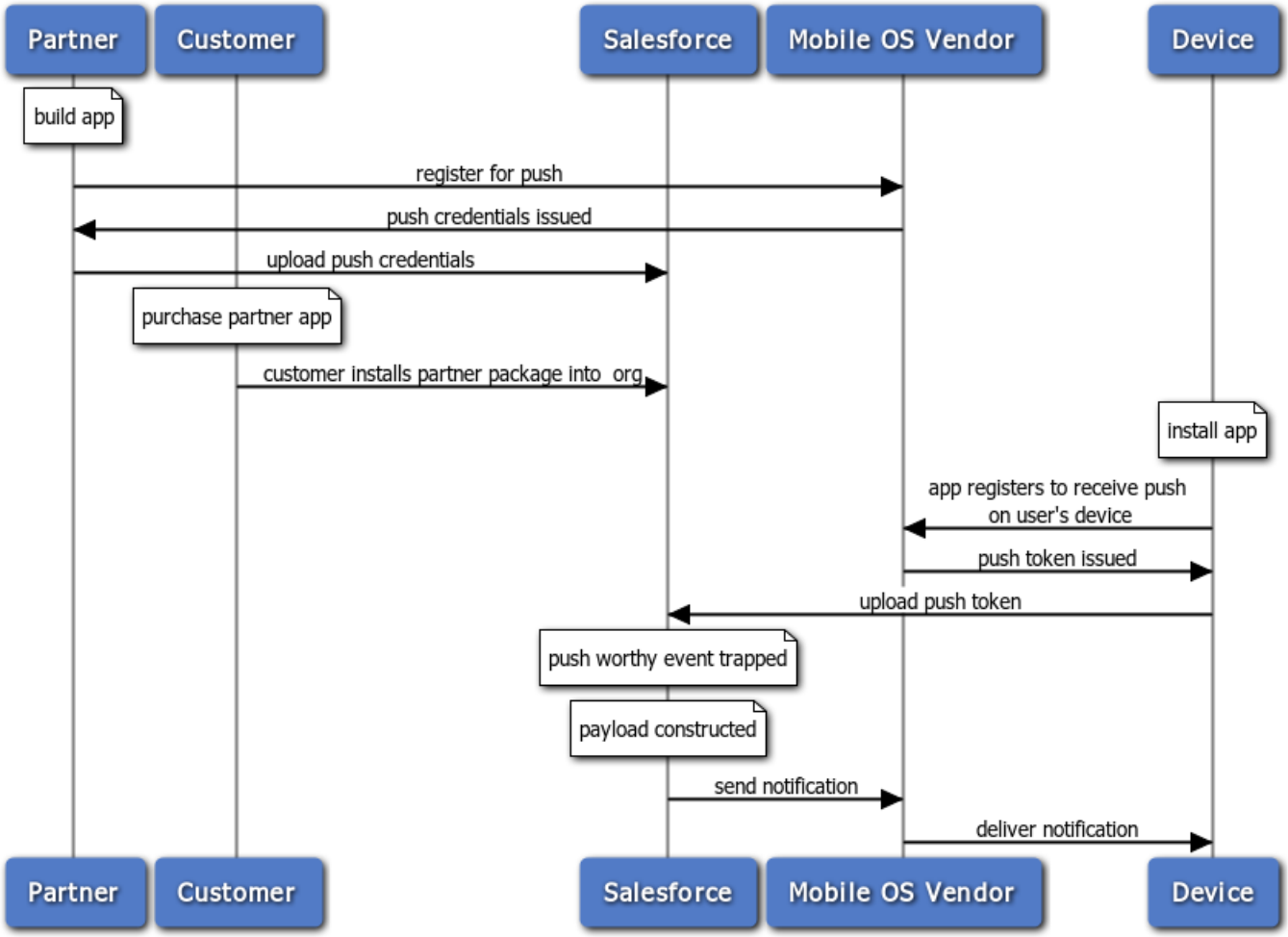
The following figure illustrates the complete push notification flow for customers who develop their own mobile apps and Apex triggers. The flow consists of the following sequence of events:

1. Developer registration with the OS vendor (Apple in this figure)
2. external client app setup in Salesforce
3. Sending the push notification via the trigger from the Salesforce organization
4. Delivery of the push notification to mobile devices by the OS vendor



The Push Notification Service sends the message to the users specified in the `send` call of the Apex `Messaging.PushNotification` class.

This next figure illustrates the complete push notification flow for partners and customers. In addition to the flow represented in the previous diagram, this flow includes steps for customers who have installed the partner's package. It includes purchase of the partner app, and installation of the partner's package to get the external client app and the Apex triggers. The last few steps of sending and delivering the push notification are similar to the previous flow.


Partner
Customer
Salesforce
Mobile OS Vendor
Device
build app
register for push
push credentials issued
upload push credentials
purchase partner app
customer installs partner package into org
install app
app registers to receive push on user's device
push token issued
upload push token
push worthy event trapped
payload constructed
send notification
deliver notification
Partner
Customer
Salesforce
Mobile OS Vendor
Device

STEPS FOR IMPLEMENTING MOBILE NOTIFICATIONS

To implement mobile notifications, you configure each of the participating technologies.

IN THIS SECTION:

Step 1. Developer Registration with Mobile OS Vendors

This step asks the OS vendor to be prepared to handle Salesforce push notifications sent to your app. Developer registration also provides some information you'll need to finish configuring your Salesforce external client app.

Step 2. Creating an External Client App

Once you've registered your mobile app with the OS vendor's push service, the next step is to create an external client app in Salesforce.

Step 3. Configure Your Mobile SDK App

In your Mobile SDK app, implement push notification protocols required by Salesforce and the device OS provider.

Step 4. Packaging Components for Customer Organizations

To distribute the Apex triggers and the external client app to customers, partners can create a managed package.

Step 1. Developer Registration with Mobile OS Vendors

This step asks the OS vendor to be prepared to handle Salesforce push notifications sent to your app. Developer registration also provides some information you'll need to finish configuring your Salesforce external client app.

To register your mobile client app, follow the process for your target mobile OS.

Android Registration

When developing an Android app that supports push notifications, remember these key points:

- You must be a member of the Android Developer Program.
- You can test FCM push services only on an Android device with either the Android Market app or Google Play Services installed. Push notifications don't work on an Android emulator.

Salesforce sends push notifications to Android apps through the Firebase Cloud Messaging for Android (FCM) framework. See https://firebase.google.com/docs/cloud-messaging for an overview of this framework.

To begin, create a Google API project for your app. Your project must have the FCM for Android feature enabled. See https://firebase.google.com/docs/cloud-messaging/android/client for instructions on setting up your project.

The setup process for your Google API project creates a key for your app. Once you've finished the project configuration, you'll need to add the Key for Server Applications (API Key) from FCM to your external client app settings.

Push notification registration occurs at the end of the OAuth login flow. Therefore, an app does not receive push notifications unless and until the user logs into a Salesforce org.

iOS Registration

When developing an iOS app that supports push notifications, remember these key points:

- You must be a member of the iOS Developer Program.
- You can test Apple push services only on an iOS physical device. Push notifications don't work in the iOS simulator.
- There are no guarantees that all push notifications will reach the target device, even if the notification is accepted by Apple.
- Apple Push Notification Services setup requires the use of the OpenSSL command line utility provided in Mac OS X.

Before you can complete registration on the Salesforce side, you need to register with Apple Push Notification Services (APNS).

Registration with APNS requires the following items.

Certificate Signing Request (CSR) File

Generate this request using the Keychain Access feature in Mac OS X. You'll also use OpenSSL to export the CSR private key to a file for later use.

App ID from iOS Developer Program

In the iOS Developer Member Center, create an ID for your app, then use the CSR file to generate a certificate. Next, use OpenSSL to combine this certificate with the private key file to create an `appkey.p12` file. You'll need this file later to configure your external client app.

iOS Provisioning Profile

From the iOS Developer Member Center, create a new provisioning profile using your iOS app ID and developer certificate. You then select the devices to include in the profile and download to create the provisioning profile. You can then add the profile to Xcode. Install the profile on your test device using Xcode's Organizer.

When you've completed the configuration, sign and build your app in Xcode. Check the build logs to verify that the app is using the correct provisioning profile. To view the content of your provisioning profile, run the following command at the Terminal window:
`security cms -D -i <your profile>.mobileprovision`

For detailed instructions see:

- Local and Push Notification Programming Guide at https://developer.apple.com/library/mac
- http://www.raywenderlich.com/32960/

Step 2. Creating an External Client App

USER PERMISSIONS

To read, create, update, or delete connected apps:	Customize Application AND either Modify All Data OR Create, edit, and delete External Client Apps
To update all fields except Profiles, Permission Sets, and Service Provider SAML Attributes:	Customize Application AND either Modify All Data OR Create, edit, and delete External Client Apps
To update Profiles, Permission Sets, and Service Provider SAML Attributes:	Customize Application AND Modify All Data AND Manage Profiles and Permission Sets
To rotate the consumer key and consumer secret:	Allow consumer key and secret rotation

EDITIONS

Available in: Lightning Experience

External Client Apps can be created in: **Group**, **Professional**, **Enterprise**, **Essentials**, **Performance**, **Unlimited**, and **Developer** Editions

External Client Apps can be installed in: **All** Editions

To install and uninstall connected apps:	Customize Application AND either Modify All Data OR Create, edit, and delete External Client Apps
To install and uninstall packaged connected apps:	Customize Application AND either Modify All Data OR Create, edit, and delete External Client Apps AND Download AppExchange Packages

Once you've registered your mobile app with the OS vendor's push service, the next step is to create an external client app in Salesforce.

Note: Connected apps creation is restricted as of Spring '26. You can continue to use existing connected apps during and after Spring '26. However, we recommend using external client apps instead. If you must continue creating connected apps, contact Salesforce Support.

See New connected apps can no longer be created in Spring '26 for more details.

Use the Salesforce external client app to provide the registration certificate or token of your mobile client app so that Salesforce can send push notifications. Also, the external client app provides the unique API name that identifies the mobile app to send notifications to.

IN THIS SECTION:

Create an External Client App for Android

Create an External Client App for Apple iOS

Testing Your External Client App Push Configuration

To run a quick test of your push notification setup, use the Send Test Notification page. Troubleshoot round-trip push notifications in a synchronous mechanism without having to configure custom notification types, Apex calls, or REST calls. You can also gain insights into what's going on behind the scenes in the asynchronous environment of real-world push notifications.

Create an External Client App for Android

Note: Connected apps creation is restricted as of Spring '26. You can continue to use existing connected apps during and after Spring '26. However, we recommend using external client apps instead. If you must continue creating connected apps, contact Salesforce Support.

See New connected apps can no longer be created in Spring '26 for more details.

To create an external client app, log in to your Salesforce Developer Edition org and complete these steps.

1. From Setup, in the `Quick Find` box, enter *`External Client`*, and then select **External Client Apps Manager**.
2. Either click **Edit Settings** in the action menu for an existing external client app, or click **New External Client App** to create a new one.
3. If you're creating a new external client app:
   a. Enter a unique name for your connected app. The API name field is auto-filled.
   b. Optionally, fill in other fields, such as a description for your connected app.
   c. Enter a contact email address.

d. In the OAuth Settings section, select **Enable OAuth Settings**.

e. Enter a callback URL.

f. Select OAuth scopes. At a minimum, select **Access and manage your data (api)** and **Perform requests on your behalf at any time (refresh_token)**.

4. Under Push Notification Settings, select **Enable Push**.

5. For Mobile Platform, select **Android**.

1. To configure Android push notifications using HTTP v1, complete these steps.

a. For **Firebase Admin SDK Private Key**, upload the JSON file that contains the private key that you generated for your Firebase service account.

b. For **Firebase Project ID**, enter the project ID that you collected from the Firebase project.

6. Save your work.
After saving the external client app, you get a consumer key. Mobile apps use this key as their connection token.

Create an External Client App for Apple iOS

Note: Connected apps creation is restricted as of Spring '26. You can continue to use existing connected apps during and after Spring '26. However, we recommend using external client apps instead. If you must continue creating connected apps, contact Salesforce Support.

See New connected apps can no longer be created in Spring '26 for more details.

To create a connected app, log in to your Salesforce Developer Edition org and perform these steps.

1. From Setup, in the `Quick Find` box, enter *`External Client`*, and then select **External Client Apps Manager**.

2. Either click **Edit Settings** in the action menu for an existing external client app, or click **New External Client App** to create a new one.

3. If you're creating a new external client app:

a. Enter a unique name for your connected app. The API name field is auto-filled.

b. Optionally, fill in other fields, such as a description for your connected app.

c. Enter a contact email address.

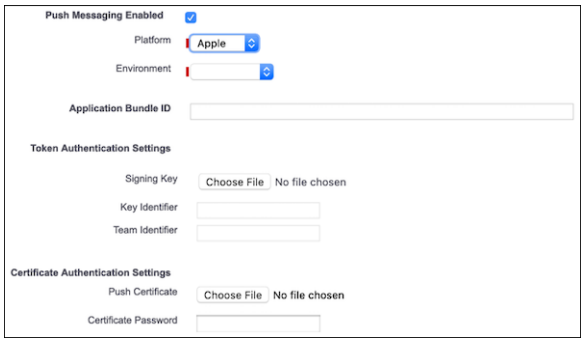
d. In the OAuth Settings section, select **Enable OAuth Settings**.

e. Enter a callback URL.

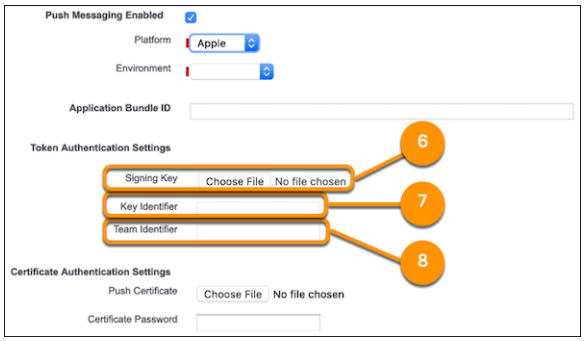
f. Select OAuth scopes. At a minimum, select **Access and manage your data (api)** and **Perform requests on your behalf at any time (refresh_token)**.

4. Under Push Notification Settings, select **Enable Push**.

5. For Mobile Platform, select **iOS**.
When you select the **iOS** option, the dialog expands to show more settings.



6. Upload the Signing Key from your Apple developer account.
7. Enter the Key Identifier from your Apple developer account.
8. Enter the Team ID from your Apple developer account.



Note: You can use authentication tokens, push certificates, or both. If a token is provided, Salesforce always uses the token. We also use Application Bundle ID when it's provided. We recommend using Application Bundle ID to avoid problems with your push notification settings.

Testing Your External Client App Push Configuration

To run a quick test of your push notification setup, use the Send Test Notification page. Troubleshoot round-trip push notifications in a synchronous mechanism without having to configure custom notification types, Apex calls, or REST calls. You can also gain insights into what's going on behind the scenes in the asynchronous environment of real-world push notifications.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

External client apps can be created in: **Group**, **Professional**, **Enterprise**, **Essentials**, **Performance**, **Unlimited**, and **Developer** Editions

Connected Apps can be installed in: **All** Editions

USER PERMISSIONS

To send a push notification from the Test Push Notifications page:

- Author Apex

  AND

  Create, edit, and delete External Client Apps

About the Send Test Notification Page

The Send Test Notification page uses information from your Apple Push Notification Service (APNS) or Firebase Cloud Messaging (FCM) for Android setup to configure a synchronous push mechanism. You select a device to receive the push notification by entering a connection token string. If you don't know the token string, you can use the Search tool to select from the list of devices that are registered for your external client app. The Search tool automatically displays the five most recently registered devices. You can also enter a user's name to search for devices that are registered to that user.

For Android FCM push notifications, you can select the Dry Run option to test your FCM setup. This option sends the notification to the FCM server but doesn't forward it to a device.

You can use any JSON-formatted payload for a dry-run test. For example, you can start with something as simple as `{"alert": "test"}`.

Each push attempt returns a status message that indicates success or failure.

To reach the test page:

1. In Setup, enter *External Client App* in the Quick Find box, then select **External Client App Manager**.
2. Click **Edit Settings** in the action menu for your external client app.
3. Click **Send test notification** next to Supported Push Platform. This link appears only if you've configured your external client app to support mobile push notifications.

Important: Successful test push notifications apply against the push notification daily limits for your organization.

IN THIS SECTION:

Check a User's Mobile Push Registrations

From a User page in your organization, an administrator can easily check which of the user's devices are currently registered for push notifications. Checking the registrations may help you troubleshoot push notification failures.

Check a User's Mobile Push Registrations

From a User page in your organization, an administrator can easily check which of the user's devices are currently registered for push notifications. Checking the registrations may help you troubleshoot push notification failures.

1. In Setup, enter *Users* in the Quick Find box, then select **Users**.
2. Select a user's name from the Full Name column.
3. Under User Detail, click **View** next to Mobile Push Registrations.

Note: If a device you expected doesn't appear in the list, it doesn't necessarily mean that the device isn't properly configured. Mobile push registrations are volatile and depend on the device state as well as the mobile app state.

Step 3. Configure Your Mobile SDK App

In your Mobile SDK app, implement push notification protocols required by Salesforce and the device OS provider.

Follow the instructions for your target mobile platform (Android, iOS, or hybrid).

IN THIS SECTION:

Enable Push Notifications in a Salesforce Mobile SDK Android App

Enable Push Notifications in a Salesforce Mobile SDK iOS App

Enable Push Notifications in a Salesforce Mobile SDK Hybrid App

Handle Notification Encryption in Salesforce Mobile SDK Apps

Mobile notifications travel from Salesforce, through third-party messaging services to mobile carriers, then on to mobile devices. Such an indirect path only increases the opportunities for malicious attacks. To guard against snooping or hacking, Salesforce encrypts Notification Builder notifications. Your app is responsible for decrypting these notifications.

Build Your Own In-App Notification Feature

If you use the Salesforce mobile app, you're familiar with the Salesforce Bell. The Bell provides an in-app notification tray that lists the current user's most recent notifications. With Salesforce Notification APIs, you can design and build your own tray.

Enable Push Notifications in a Salesforce Mobile SDK Android App

1. Add an entry for `androidPushNotificationClientId`.
   a. In `res/values/bootconfig.xml` (for native apps):

   ```
   <string name="androidPushNotificationClientId">1234567890</string>
   ```

   b. In `assets/www/bootconfig.json` (for hybrid apps):

   ```
   "androidPushNotificationClientId": "1234567890"
   ```

   This example value represents the project number of the Google project that is authorized to send push notifications to Android devices.

   Note: Behind the scenes, Mobile SDK automatically reads this value and uses it to register the device against the Salesforce external client app. Each time the user re-opens the app, Mobile SDK automatically renews the registration. This validation allows Salesforce to send notifications to the external client app.

   A device becomes unregistered if:

   - The app goes unused for an extended period of time

     OR

   - The user logs out of the app

2. Create a class in your app that implements `PushNotificationInterface`. `PushNotificationInterface` is a Mobile SDK Android interface for handling push notifications. `PushNotificationInterface` has a single method, `onPushMessageReceived(Bundle message)`.

```
public interface PushNotificationInterface {
    public void onPushMessageReceived(Bundle message);
}
```

   In this method you implement your custom functionality for displaying, or otherwise disposing of, push notifications.

3. In the `onCreate()` method of your `Application` subclass, call the `SalesforceSDKManager.setPushNotificationReceiver()` method, passing in your implementation of `PushNotificationInterface`. Call this method immediately after the `SalesforceSDKManager.initNative()` call.

```
@Override
public void onCreate() {
    super.onCreate();
    SalesforceSDKManager.initNative(getApplicationContext(),
        new KeyImpl(), MainActivity.class);
    SalesforceSDKManager.getInstance().
        setPushNotificationReceiver(myPushNotificationInterface);
 }
```

Enable Push Notifications in a Salesforce Mobile SDK iOS App

Salesforce Mobile SDK for iOS provides the `SFPushNotificationManager` class to handle push registration. To use it, import `<SalesforceSDKCore/SFPushNotificationManager>`. The `SFPushNotificationManager` class is available as a runtime singleton:

Swift

```
PushNotificationManager.sharedInstance()
```

Objective-C

```
[SFPushNotificationManager sharedInstance]
```

This class implements four registration methods:

Swift

```
func registerForRemoteNotifications()

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

func registerSalesforceNotifications(completionBlock: (() → Void)?,
    fail: (() → Void)?)

func unregisterSalesforceNotifications(withCompletionBlock: UserAccount,
    completionBlock: (() → Void)?) // for internal use
```

Objective-C

```
- (void)registerForRemoteNotifications;
- (void)didRegisterForRemoteNotificationsWithDeviceToken:
    (NSData*)deviceTokenData;
- (BOOL)registerForSalesforceNotifications; // for internal use
- (BOOL)unregisterSalesforceNotifications; // for internal use
```

Mobile SDK calls `registerForSalesforceNotifications` after login and `unregisterSalesforceNotifications` at logout. You call the other two methods from your `AppDelegate` class.

Mobile SDK 8.2 updates the push notification implementation to use the iOS `UNUserNotificationCenter` class. This class authorizes the calling app to receive push notifications from APNS. Mobile SDK iOS template apps request authorization for alerts, sounds, and badges.

To decode encrypted notifications, you extend the iOS `UNNotificationServiceExtension` class. Mobile SDK provides a special app template that includes the boilerplate extension code for you to copy. See Handle Notification Encryption in Salesforce Mobile SDK Apps.

1. In your `AppDelegate` class, create an instance method named `- (void) registerForRemotePushNotifications`.
   a. In this method, request authorization to receive push notifications.

```
- (void)registerForRemotePushNotifications {
    [[UNUserNotificationCenter currentNotificationCenter]
        requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
                                         UNAuthorizationOptionAlert |
                                         UNAuthorizationOptionBadge)
                      completionHandler:^(BOOL granted,
                                          NSError * _Nullable error) {
        if (granted) {

        } else {

        }

        if (error) {

        }
    }];
}
```

   b. If authorization is granted, call the `SFPushNotificationManager registerForRemoteNotifications` method. If authorization isn't granted or the operation returned an error, log the appropriate error messages.

```
- (void)registerForRemotePushNotifications {
    [[UNUserNotificationCenter currentNotificationCenter]
        requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
                                         UNAuthorizationOptionAlert |
                                         UNAuthorizationOptionBadge)
                      completionHandler:^(BOOL granted,
                                          NSError * _Nullable error) {
        if (granted) {
            // All good!
            dispatch_async(dispatch_get_main_queue(), ^{
               [[SFPushNotificationManager sharedInstance]
                   registerForRemoteNotifications];
```

```
            });
        } else {
            // Authorization not granted
            [SFLogger d:[self class]
                format:@"Push notification authorization denied"];
        }

        if (error) {
            // Operation failed with error message
            [SFLogger e:[self class]
                 format:@"Push notification authorization error: %@",
                        error];
        }
    }];
}
```

2. Call your `registerForRemotePushNotifications` in the `application:didFinishLaunchingWithOptions:` method.

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc]
        initWithFrame:[UIScreen mainScreen].bounds];
    [self initializeAppViewState];

    [self registerForRemotePushNotifications];

    [[SFAuthenticationManager sharedManager]
        loginWithCompletion:self.initialLoginSuccessBlock
                    failure:self.initialLoginFailureBlock];

    return YES;
}
```

If registration succeeds, Apple passes a device token to the `application:didRegisterForRemoteNotificationsWithDeviceToken:` method of your `AppDelegate` class.

3. In the `application:didRegisterForRemoteNotificationsWithDeviceToken:` method, register the device token from Apple with Mobile SDK's push notification manager.

   a. Call the `SFPushNotificationManager didRegisterForRemoteNotificationsWithDeviceToken:` method.

```
- (void)application:(UIApplication*)application
    didRegisterForRemoteNotificationsWithDeviceToken:
        (NSData*)deviceToken
{
    [[SFPushNotificationManager sharedInstance]
        didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];


}
```

b. If the current user's access token exists, call the `SFPushNotificationManager registerSalesforceNotificationsWithCompletionBlock:failBlock:` method.

```
- (void)application:(UIApplication*)application
    didRegisterForRemoteNotificationsWithDeviceToken:
        (NSData*)deviceToken
{
    [[SFPushNotificationManager sharedInstance]
        didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];

    if ([SFUserAccountManager sharedInstance].
            currentUser.credentials.accessToken != nil) {
        [[SFPushNotificationManager sharedInstance]
            registerSalesforceNotificationsWithCompletionBlock:nil
                                                     failBlock:nil];
    }
}
```

4. To log an error if registration with Apple fails, implement the `application:didFailToRegisterForRemoteNotificationsWithError:` method.

```
- (void)application:(UIApplication*)application
    didFailToRegisterForRemoteNotificationsWithError:
        (NSError*)error
{
    // Example of an implementation
    [SFLogger e:[self class] format:@"Push notification authorization error: %@", error];
}
```

Enable Push Notifications in a Salesforce Mobile SDK Hybrid App

1. (Android only) If your target platform is Android:

a. Add an entry for `androidPushNotificationClientId`. In `assets/www/bootconfig.json`:

```
"androidPushNotificationClientId": "33333344444"
```

This value is the project number of the Google project that is authorized to send push notifications to an Android device.

2. In your callback for `cordova.require("com.salesforce.plugin.oauth").getAuthCredentials()`, add the following code:

```
cordova.require("com.salesforce.util.push").registerPushNotificationHandler(
    function(message) {
        // add code to handle notifications
    },
    function(error) {
        // add code to handle errors
    }
);
```

Example: This code demonstrates how you might handle messages. The server delivers the payload in `message["payload"]`.

```
function(message) {
    var payload = message["payload"];
    if (message["foreground"]) {
        // Notification is received while the app is in
        // the foreground
        // Do something appropriate with payload
    }
    if (!message["foreground"]) {
        // Notification was received while the app was in
        // the background, and the notification was clicked,
        // bringing the app to the foreground
        // Do something appropriate with payload
    }
}
```

Handle Notification Encryption in Salesforce Mobile SDK Apps

Mobile notifications travel from Salesforce, through third-party messaging services to mobile carriers, then on to mobile devices. Such an indirect path only increases the opportunities for malicious attacks. To guard against snooping or hacking, Salesforce encrypts Notification Builder notifications. Your app is responsible for decrypting these notifications.

Setting up encryption for notifications requires no configuration on the server. For client-side decryption, Salesforce Mobile SDK does most of the work for you. Mobile SDK negotiates the encryption policy with the Salesforce server and internally handles the exchange of public and private keys.

Your tasks for notification decryption on client apps depends on the mobile operating system.

iOS: Apps that are registered for Salesforce notifications must extend the `UNNotificationServiceExtension` class. A specialized Mobile SDK template app provides a boilerplate extension that you can bring into your iOS app, provided you're using Mobile SDK 8.2 or later. Earlier versions of Mobile SDK do not fully support notification encryption.

Android: Mobile SDK for Android handles decryption internally and requires no client-side code changes.

Implementing a Decryption Extension (Swift)

In Mobile SDK 8.2 and later, the forceios iOSNativeSwiftTemplate app requests notification authorization through the iOS `UNUserNotificationCenter` object. A specialized version of `iOSNativeSwiftTemplate`, iOSNativeSwiftEncryptedNotificationTemplate, extends the `UNNotificationServiceExtension` iOS class to handle notification decryption. This extension class, `NotificationService`, provides boilerplate decryption code that Mobile SDK apps can use without changes. To support encrypted notifications, you must be using Mobile SDK 8.2 or later, and your app must include this extension.

To create a Swift project that supports Notification Builder encrypted notifications, you can use the `iOSNativeSwiftEncryptedNotificationTemplate` template with forceios. Even if you're updating an existing Mobile SDK project, it's easiest to start fresh with a new forceios template project. If you'd rather update a Swift project manually, skip to "Example: Add Push Registration Manually (Swift)".

1. Install the latest forceios version from node.js:

```
[sudo] npm install -g forceios
```

2. Call the forceios `createWithTemplate` command:

```
forceios createWithTemplate
```

3. At the first prompt, enter *`iOSNativeSwiftEncryptedNotificationTemplate`*:

```
forceios createWithTemplate
Enter URI of repo containing template application or a Mobile SDK template name:
    iOSNativeSwiftEncryptedNotificationTemplate
```

4. In the remaining prompts, enter your company and project information. If your information is accepted, forceios creates a project that is ready for encrypted notifications.
5. If you're updating an older Mobile SDK project, copy your app-specific resources from your old project into the new project.

See Also

- UNNotificationServiceExtension at developer.apple.com

Build Your Own In-App Notification Feature

If you use the Salesforce mobile app, you're familiar with the Salesforce Bell. The Bell provides an in-app notification tray that lists the current user's most recent notifications. With Salesforce Notification APIs, you can design and build your own tray.

Queries calculate results from stored Notification Builder notifications. Salesforce provides REST APIs for the following Notification queries.

Notification: Get a specific notification for the context user.

Notifications: Get notifications for the context user.

Notifications Status: Get the "read" and "seen" properties a range of notifications.

Notification Update: Update the "read" and "seen" properties of a specific notification.

Notifications Update: Update the "read" and "seen" properties of a range of notifications.

For example, using these APIs, an app can

- Display a list of notifications (Notifications)
- When the customer taps a list entry, display the details of a single notification (Notification)
- Set the state of a widget that indicates whether a notification has been read or seen (Notification)
- Report how many notifications the customer hasn't read or seen (Notifications Status)
- Update the "read" or "seen" properties on one or more notifications after the customer visits your tray (Notification Update, Notifications Update)

Custom In-App Notification Tray Template App

If you're looking for a head start in Swift, Mobile SDK provides an example of a custom notifications tray in the MobileSyncExplorerSwift template app. You can create your own app based on this template with the `forceios createwithtemplate` CLI command:

1. Install the latest forceios version from `node.js`:

```
[sudo] npm install -g forceios
```

2. Call the forceios `createWithTemplate` command:

```
forceios createWithTemplate
```

3. At the first prompt, enter *`MobileSyncExplorerSwift`*:

```
forceios createWithTemplate
Enter URI of repo containing template application or a Mobile SDK template name:
    MobileSyncExplorerSwift
```

4. In the remaining prompts, enter your company and project information. If your information is accepted, forceios creates a project with a notifications tray that you can customize.
5. If you're updating an older Mobile SDK project, copy your app-specific resources from your old project into the new project.

See the Reference section for more information on Notification APIs.

Mobile SDK iOS Wrappers for Notification APIs

On iOS, Mobile SDK defines convenience methods for Notification APIs in the following classes:

Swift

```
RestClient
```

Objective-C

```
SFRestApi
```

The following methods are currently available for native apps only.

Notification

Get a notification.

Delegate Method

Swift

```
RestClient.shared.request(forNotification:apiVersion:)
```

Objective-C

```
- (SFRestRequest *)requestForNotification:(NSString *)notificationId apiVersion:(NSString
 *)apiVersion;
```

Block Method

Not available.

Notifications Status

Get the status of a range of notifications, including unread and unseen count.

Delegate Method

Swift

```
RestClient.shared.request(forNotificationsStatus:)
```

Objective-C

```
- (SFRestRequest *)requestForNotificationsStatus:(NSString *)apiVersion;
```

Block Method

Not available.

Notifications

Get the given number (maximum 20) of archived Notification Builder notifications based on the given "before" or "after" date. In Mobile SDK, use the Swift `FetchNotificationsRequestBuilder` object or the Objective-C `SFSDKFetchNotificationsRequestBuilder` to create GET requests for notifications.

Delegate Method

Swift

```
let builder = FetchNotificationsRequestBuilder.init()
builder.setSize(10)
builder.setBefore(Date.init())
let request = builder.buildFetchNotificationsRequest(SFRestDefaultAPIVersion)
```

Objective-C

```
SFSDKFetchNotificationsRequestBuilder *builder =
    [[SFSDKFetchNotificationsRequestBuilder alloc] init];
[builder setBefore: [NSDate date]];
[builder setSize:10];
SFRestRequest *fetchRequest =
    [builder buildFetchNotificationsRequest:kSFRestDefaultAPIVersion];
```

Block Method

Not available.

Notifications Update

Update the "read" and "seen" statuses of a given set of Notification Builder notifications. In Mobile SDK, use the Swift `UpdateNotificationsRequestBuilder` object or the Objective-C `SFSDKUpdateNotificationsRequestBuilder` object to create update requests.

To update a single notification, set the `notificationId` property. To update a range of notifications, set either the `notificationIds` or the `before` property. These properties—`notificationId`, `notificationIds`, and `before`—are mutually exclusive.

Delegate Method

Swift

```
let builder = UpdateNotificationsRequestBuilder.init()
builder.setBefore(Date.init())
builder.setRead(true)
builder.setSeen(true)
let request = builder.buildUpdateNotificationsRequest(SFRestDefaultAPIVersion)
```

Objective-C

```
SFSDKUpdateNotificationsRequestBuilder *builder =
    [[SFSDKUpdateNotificationsRequestBuilder alloc] init];
[builder setRead:true];
[builder setSeen:true];
[builder setBefore: [NSDate date]];
SFRestRequest *updateRequest = [builder
buildUpdateNotificationsRequest:kSFRestDefaultAPIVersion];
```

Block Method

Not available.

Mobile SDK Android Wrappers for Notification APIs

On Android, the `RestRequest` class defines the following convenience wrappers for Notification APIs.

Notification

Get a notification.

```
public static RestRequest getRequestForNotification(String apiVersion,
    String notificationId)
```

Notification Update

Update the "read" and "seen" statuses of a given Notification Builder notification.

```
public static RestRequest getRequestForNotificationUpdate(String apiVersion,
    String notificationId, Boolean read, Boolean seen)
```

Notifications

Get the given number (maximum 20) of archived Notification Builder notifications based on the given "before" or "after" date.

```
public static RestRequest getRequestForNotifications(String apiVersion,
    Integer size, Date before, Date after)
```

Notifications Status

Get the status of a range of notifications, including unread and unseen count.

```
public static RestRequest getRequestForNotificationsStatus(String apiVersion)
```

Notifications Update

Update the "read" and "seen" statuses of a given set of Notification Builder notifications.

```
* public static RestRequest getRequestForNotificationsUpdate(String apiVersion,
    List<String> notificationIds, Date before, Boolean read, Boolean seen)
```

Step 4. Packaging Components for Customer Organizations

To distribute the Apex triggers and the external client app to customers, partners can create a managed package.

When customers install your managed package, they will be able to send push notifications to the external client app from their organizations when certain events occur on Salesforce records, such as updating records.

Customers who install a managed package containing a external client app provided by a partner can't write their own Apex triggers to send push notifications to the external client app. This is to safeguard the partner's registration with the mobile OS vendor from abuse.

SENDING MOBILE NOTIFICATIONS

To send mobile notifications, Salesforce offers several options: Apex, REST API, and, in some cases, declarative feature-based tools. The type of notification, the delivery channel, and your skill comfort levels guide your choice.

Send Apex push notifications with the `Messaging.PushNotification` methods in Apex triggers, or through the `PushNotifications` API.

You can send custom notifications through the `customNotificationAction` API, Process Builder, or Flow.

See Also

- Send Mobile Push Notifications
- Considerations for Notification Builder
- Send Custom Notifications with Notification Builder Platform
- Other Ways to Send a Custom Notification

IN THIS SECTION:

Using Apex Triggers to Send Push Notifications

After registering with the mobile OS vendor for push notification service and creating a external client app, you can send push notifications to a mobile client app using Apex triggers.

Custom Notification Actions

Send custom notifications to recipients via desktop or mobile channels.

Using Apex Triggers to Send Push Notifications

After registering with the mobile OS vendor for push notification service and creating a external client app, you can send push notifications to a mobile client app using Apex triggers.

Note: Legacy Cloud Messaging API users are required to migrate to the new Firebase Cloud Messaging API (HTTP v1) by June 2024. We recommend that you test your app's compatibility with HTTP v1 before migrating.

Push notification triggers use methods in the Apex `Messaging.PushNotification` and `Messaging.PushNotificationPayload` classes. The external client app in the Salesforce organization represents the mobile client app that will receive the notifications.

To send push notifications to a external client app, either of the following conditions must be met:

- Apex triggers are added in the *same organization* in which the external client app is created.
- Apex triggers are installed as part of a partner-provided managed package, along with the external client app.

Sample Apex Trigger

This sample Apex trigger sends push notifications to the external client app named *Test_App*, which corresponds to a mobile app on iOS mobile clients. The trigger fires after cases have been updated and sends the push notification to two users: the case owner and the user who last modified the case.

```
trigger caseAlert on Case (after update) {

    for(Case cs : Trigger.New)
    {
        // Instantiating a notification
        Messaging.PushNotification msg =
            new Messaging.PushNotification();

        // Assembling the necessary payload parameters for Apple.
        // Apple params are:
        // (<alert text>,<alert sound>,<badge count>,
        // <free-form data>)
        // This example doesn't use badge count or free-form data.
        // The number of notifications that haven't been acted
        // upon by the intended recipient is best calculated
        // at the time of the push. This timing helps
        // ensure accuracy across multiple target devices.
        Map<String, Object> payload =
            Messaging.PushNotificationPayload.apple(
                'Case ' + cs.CaseNumber + ' status changed to: '
                + cs.Status, '', null, null);

        // Adding the assembled payload to the notification
        msg.setPayload(payload);

        // Getting recipient users
        String userId1 = cs.OwnerId;
        String userId2 = cs.LastModifiedById;

        // Adding recipient users to list
        Set<String> users = new Set<String>();
        users.add(userId1);
        users.add(userId2);

        // Sending the notification to the specified app and users.
        // Here we specify the API name of the external client app.
        msg.send('Test_App', users);
    }
}
```

Sample Android Payload

The trigger sample uses `Messaging.PushNotificationPayload` to create a payload for an iOS notification. Unlike iOS, Android doesn't have special attributes or requirements for the payload; it just needs to be in JSON format. In Apex, you create the

Android payload as a MAP<String,ANY> object. The `Messaging.PushNotification` class handles conversion to JSON. Here's a sample Android payload.

```
Map<String, Object> androidPayload = new Map<String, Object>();
androidPayload.put('number', '1');
androidPayload.put('name', 'test');
```

PushNotification Class, Methods, and Constructors

- PushNotification Class
- PushNotification Constructors
- PushNotification Methods

Push Notifications Resource

Send a mobile push notification to client apps on users' devices. This resource is only accessible when the session is established with a client app that is developed in the same org or installed in the same package as the recipient's client app.

Resource:

```
/connect/notifications/push
```

Available version: 31.0

HTTP methods: POST

Request body for POST:

Root XML tag: `<pushNotification>`

JSON example:

```
{    "appName" : "TestApp",
     "namespace" : "abc",
     "userIds" : ["005x00000013dPK", "005x00000013dPL"],
     "payload" : "{'aps':{'alert':'test', 'badge':0, 'sound':'default'}}"
}
```

Properties

Name	Type	Description	Required or Optional	Available Version
`appName`	String	The API name of the client app that the push notification is sent to.	Required	31.0
`namespace`	String	The namespace of the client app that the push notification is sent to.	Required if the namespace is set	31.0
`payload`	String	The push notification payload in JSON format.	Required	31.0
`userIds`	String[]	The push notification recipient user ids.	Required	31.0

Request parameters for POST

Parameter Name	Type	Description	Required or Optional	Available Version
`appName`	String	The API name of the client app that the push notification is sent to.	Required	31.0
`namespace`	String	The namespace of the client app that the push notification is sent to.	Required if the namespace is set	31.0
`payload`	String	The push notification payload in JSON format.	Required	31.0
`userIds`	String[]	The push notification recipient user ids.	Required	31.0

Custom Notification Actions

Send custom notifications to recipients via desktop or mobile channels.

Before you send a custom notification, you must first create a notification type.

Important: In orgs created in Winter '21 or later, the Send Custom Notifications user permission is required to trigger the Send Custom Notification action in flows that run in user context, REST API calls, and Apex callouts.

The Send Custom Notifications user permission isn't required to trigger the Send Custom Notification action in processes or flows that run in system context.

This object is available in API version 46.0 and later.

Supported REST HTTP Methods

URI: `/services/data/v`***`XX.X`***`/actions/standard/customNotificationAction`

Formats: JSON, XML

HTTP Methods: GET, HEAD, POST

Authentication: `Authorization: Bearer` ***`token`***

Inputs

Input	Type	Description
`actionGroup`	string	Optional. Action group for custom notification type. This makes the mobile push notifications actionable.
`customNotifTypeId`	reference	Required. The ID of the Custom Notification Type being used for the notification.
`recipientIds`	reference	Required. The ID of the recipient or recipient type. Valid values are: `UserId` (sent to this user if active), `AccountId` (sent to all active Account Team members; requires account teams enabled), `OpportunityId` (sent to all active Opportunity Team members; requires team selling enabled), `GroupId` (sent to all active group members), or `QueueId` (sent to all active queue members). Values can be combined in a list up to 500 values.

Input	Type	Description
`senderId`	reference	Optional. The User ID of the sender of the notification.
`title`	string	Required. The title of the notification, as seen by recipients. Maximum characters: 250. The content of mobile push notifications depends on the content push notification setting.
`body`	string	Required. The body of the notification, as seen by recipients. Maximum characters: 750. The content of mobile push notifications depends on the content push notification setting.
`targetId`	reference	Optional. The Record ID for the target record of the notification. You must specify either a `targetID` or a `targetPageRef`.
`targetPageRef`	string	Optional. The `PageReference` for the navigation target of the notification. To see how to specify the target using JSON, see pageReference. You must specify either a `targetID` or a `targetPageRef`.

Usage

GET

The following example shows how to get information about the custom notification action type:

```
curl --include --request GET \
--header "Authorization: Authorization: Bearer 00DR...xyz" \
--header "Content-Type: application/json" \
"https://instance.salesforce.com/services/data/v46.0/actions/standard/customNotificationAction"
```

POST

The following example shows how to create a custom notification action:

```
curl --include --request POST \
--header "Authorization: Authorization: Bearer 00DR...xyz" \
--header "Content-Type: application/json" \
--data '{ "inputs" :
  [
  {
    "customNotifTypeId" : "0MLR0000000008eOAA",
    "recipientIds" : ["005R0000000LSqtIAG"],
    "title" : "Custom Notification",
    "body" : "This is a custom notification.",
    "targetId" : "001R0000003fSUDIA2"
  }
  ]
}' \
"https://instance.salesforce.com/services/data/v46.0/actions/standard/customNotificationAction"
```

The response is:

```
[
   {
      "actionName" : "customNotificationAction",
```

```
      "errors" : null,
      "isSuccess" : true,
      "outputValues" : {
         "SuccessMessage" : "Your custom notification is processed successfully."
      }
   }
]
```

USING THE CONNECT REST API PUSH NOTIFICATION RESOURCE (OPTIONAL)

If you prefer to send push notifications without looping through the Apex engine—in other words, if you just want to push a simple message from an external client app on one mobile device to another—use the Connect REST API push notification resource. This resource is useful for sending notifications of events that occur outside of Salesforce and can be handled entirely within the mobile external client app. You can use this resource in any type of custom app: a native Mobile SDK app, a hybrid Mobile SDK app, or an HTML5 app.

The Connect REST API resource establishes an HTTP session to send POST requests from the sending device to one or more receiving devices. This means that:

- The sending app and the receiving apps must either be developed in the same organization or be installed from the same package.
- Only the receiving apps are required to be registered for push notifications with Apple or Google.
- Only the receiving apps are required to implement the Salesforce Mobile SDK push notification protocols.
- Each app requires an external client app, but only the receiving external client apps must be configured for push notifications.

PUSHNOTIFICATIONPAYLOAD CLASS

Contains methods to create the notification message payload for an Apple device.

Namespace

Messaging

Usage

Apple has specific requirements for the notification payload. and this class has helper methods to create the payload. For more information on Apple's payload parameters, search for "Apple Push Notification Service" at https://developer.apple.com/library/mac/documentation/.

Example

See the Push Notification Example .

The following are the methods for `PushNotificationPayload`. All are global static methods.

apple(alert, sound, badgeCount, userData)

Helper method that creates a valid Apple payload from the specified arguments.

Signature

```
public static Map<String,Object> apple(String alert, String sound, Integer badgeCount,
Map<String,Object> userData)
```

Parameters

- **alert**:

  Type: String

  Notification message to be sent to the mobile client.

- **sound**:

  Type: String

  Name of a sound file to be played as an alert. This sound file should be in the mobile application bundle.

- **badgeCount**:

  Type: Integer

  Number to display as the badge of the application icon.

- **userData**:

  Type: Map<String, Object>

  Map of key-value pairs that contains any additional data used to provide context for the notification. For example, it can contain IDs of the records that caused the notification to be sent. The mobile client app can use these IDs to display these records.

Return Value

Type: Map<String, Object>

Returns a formatted payload that includes all of the specified arguments.

Usage

To generate a valid payload, you must provide a value for at least one of the following parameters: `alert`, `sound`, `badgeCount`.

Example

See the Push Notification Example.

apple(alertBody, actionLocKey, locKey, locArgs, launchImage, sound, badgeCount, userData)

Helper method that creates a valid Apple payload from the specified arguments.

Signature

```
public static Map<String,Object> apple(String alertBody, String actionLocKey, String locKey, String[] locArgs, String launchImage, String sound, Integer badgeCount, Map<String,Object> userData)
```

Parameters

- **alertBody**:

  Type: String

  Text of the alert message.

- **actionLocKey**:

  Type: String

  If a value is specified for the actionLocKey argument, an alert with two buttons is displayed. The value is a key to get a localized string in a `Localizable.strings` file to use for the right button's title.

- **locKey**:

  Type: String

  Key to an alert-message string in a `Localizable.strings` file for the current localization.

- **locArgs**:

  Type: List <String>

  Variable string values to appear in place of the format specifiers in locKey.

- **launchImage**:

  Type: String

  File name of an image file in the application bundle.

- **sound**:

  Type: String

  Name of a sound file to be played as an alert. This sound file should be in the mobile application bundle.

- **badgeCount**:

  Type: Integer

  Number to display as the badge of the application icon.

- **userData**:

  Type: Map <String, Object>

  Map of key-value pairs that contains any additional data used to provide context for the notification. For example, it can contain IDs of the records that caused the notification to be sent. The mobile client app can use these IDs to display these records.

Return Value

Type: Map <String, Object>

Returns a formatted payload that includes all of the specified arguments.

Usage

To generate a valid payload, you must provide a value for at least one of the following parameters: `alert`, `sound`, `badgeCount`.

APEX LIMITS FUNCTIONS FOR PUSH NOTIFICATIONS

Use the Apex limits functions for push notifications to get usage information.

System.Limits Class

`getLimitMobilePushApexCalls()`: Returns the total number of Apex calls that are allowed per transaction for mobile push notifications.

`getMobilePushApexCalls()`: Returns the number of Apex calls that have been used by mobile push notifications during the current metering interval.

PUSH NOTIFICATION LIMITS

Push notification limits apply to test push notifications as well as to production notifications.

An org can send up to 20,000 iOS and 10,000 Android push notifications per hour (for example, 4:00 to 4:59 UTC).Only *deliverable* notifications count toward this limit. For example, a notification is sent to 1,000 employees in your company, but 100 employees haven't installed the mobile app yet. Only the notifications sent to the 900 employees who have installed the mobile app count toward this limit.Each test push notification that is generated through the Test Push Notification page is limited to a single recipient. Test push notifications count toward an org's hourly push notification limit.When an org's hourly push notification limit is met, any additional notifications are still created for in-app display and retrieval via REST API.

Note:

- For Flow Builder notification limits, see https://help.salesforce.com/articleView?id=flow_ref_elements_actions_sendcustomnotification.htm
- The limits described here are for core Salesforce push notifications. They do not apply to Marketing Cloud MobilePush notifications. For further information on Marketing Cloud usage, see developer.salesforce.com/docs/atlas.en-us.mc-sdks.meta/mc-sdks/mobile-push-sdk.htm.

DEBUG LOG EVENTS

The mobile push notification service logs events that are useful to understand when you are monitoring an Apex debug log.

Events

The following events are supported:

- `PUSH_NOTIFICATION_INVALID_APP`
- `PUSH_NOTIFICATION_INVALID_CERTIFICATE`
- `PUSH_NOTIFICATION_INVALID_NOTIFICATION`
- `PUSH_NOTIFICATION_NO_DEVICES`
- `PUSH_NOTIFICATION_NOT_ENABLED`
- `PUSH_NOTIFICATION_SENT`

For more information on these events, log levels and categories, and how to use the Developer Console or monitor a debug log, see the Apex Code Developer's Guide.

NOTIFICATIONS RESOURCES

Get or update notifications. Execute an action for a notification. Get the status of notifications. Get supported notification type details and actions.

For more information about notifications resources, refer to the Connect REST API Developer Guide.

IN THIS SECTION:

CustomNotificationType
Stores information about custom notification types. This object is available in API version 46.0 and later.

CustomNotificationType

Stores information about custom notification types. This object is available in API version 46.0 and later.

Supported Calls

`create()`, `delete()`, `describeSObjects()`, `query()`, `retrieve()`, `update()`, `upsert()`

Fields

Field	Type	Properties	Description
`CustomNotifTypeName`	string	Create, Filter, Group, idLookup, Sort, Unique, Update	Specifies a notification type name. The notification type name is unique within your organization. Maximum number of characters: 80.
`Description`	textarea	Create, Filter, Group, Nillable, Sort, Update	Specifies a general description of the notification type, which is displayed with the notification type name. Maximum number of characters: 255.
`Desktop`	boolean	Create, Defaulted on create, Filter, Group, Sort, Update	Indicates whether the desktop delivery channel is enabled (`true`) or not (`false`). The default value is `false`.
`DeveloperName`	string	Create, Filter, Group, Sort, Update	Specifies the API name of the notification type.
`IsSlack`	boolean	Reserved for future use.	Reserved for future use.
`Language`	picklist	Create, Defaulted on create, Filter, Group, Nillable, Restricted picklist, Sort, Update	Specifies the language of the custom notification type. The value for this field is the language value of the org.

Field	Type	Properties	Description
`ManageableState`	ManageableState enumerated list	Filter, Group, Nillable, Restricted picklist, Sort	Indicates the manageable state of the specified component that is contained in a package. Values are `beta`, `deleted`, `deprecated`, `deprecatedEditable`, `installed`, `installedEditable`, `released`, or `unmanaged`.
`MasterLabel`	string	Create, Filter, Group, Sort, Update	Specifies the notification type label.
`Mobile`	boolean	Create, Defaulted on create, Filter, Group, Sort, Update	Indicates whether the mobile delivery channel is enabled (`true`) or not (`false`). The default value is `false`.
`NamespacePrefix`	string	Filter, Group, Nillable, Sort	Specifies the namespace of the notification type, if installed with a managed package.

CustomNotificationType is exposed in Tooling API to user profiles with the View Setup and Configuration permission. To create and edit notification types, the Customize Application permission is required.

NOTIFICATION BUILDER PLATFORM PUSH PAYLOADS

Learn about how we handle push payload content for custom notifications.

Notification Builder Platform provides mobile push notifications for iOS apps through the Apple Push Notification Service (APNs) and for Android apps through the Firebase Cloud Messaging (FCM).

Push payloads are sent with encrypted content when the mobile app supplies an RSA public encryption key upon push registration with the Salesforce push notification service. When a payload is sent from a customer org to a user's device, the mobile app processes and decrypts the payload. For more details about key registration and payload decryption, see the Mobile SDK Development Guide.

Push payloads are sent with unencrypted content if the mobile app doesn't provide an encryption key.

INDEX

C

Connect REST API push resource 29

M

Mobile Push Notification Service
    check user device registrations 11