Notification Android and iOS using Flutter

medium.com/@greg.perry/notifications-in-flutter-2dc4ee6ee6aa

August 20, 2020

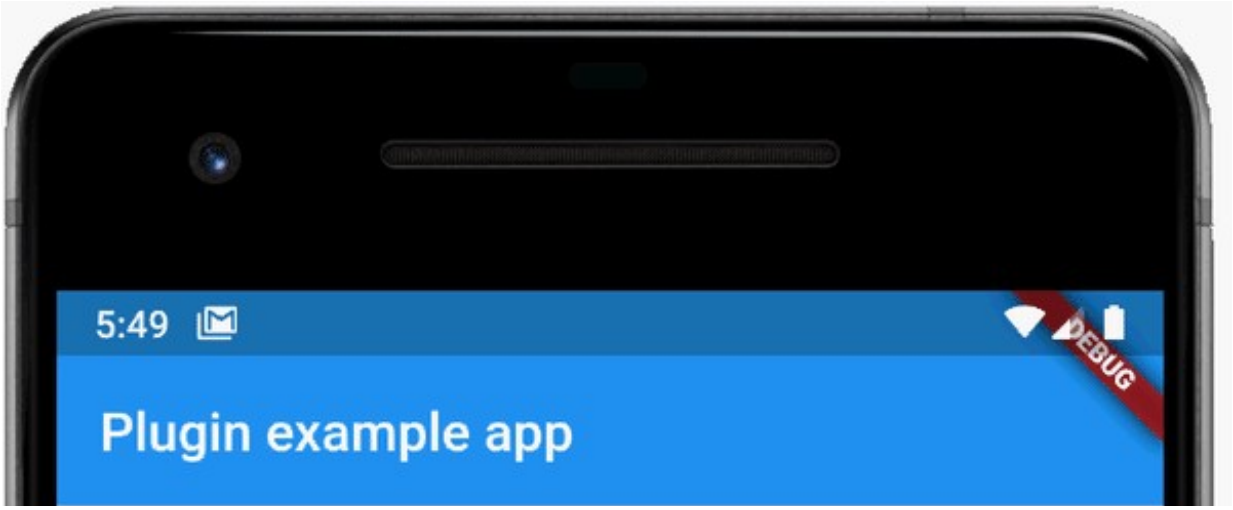
Responses

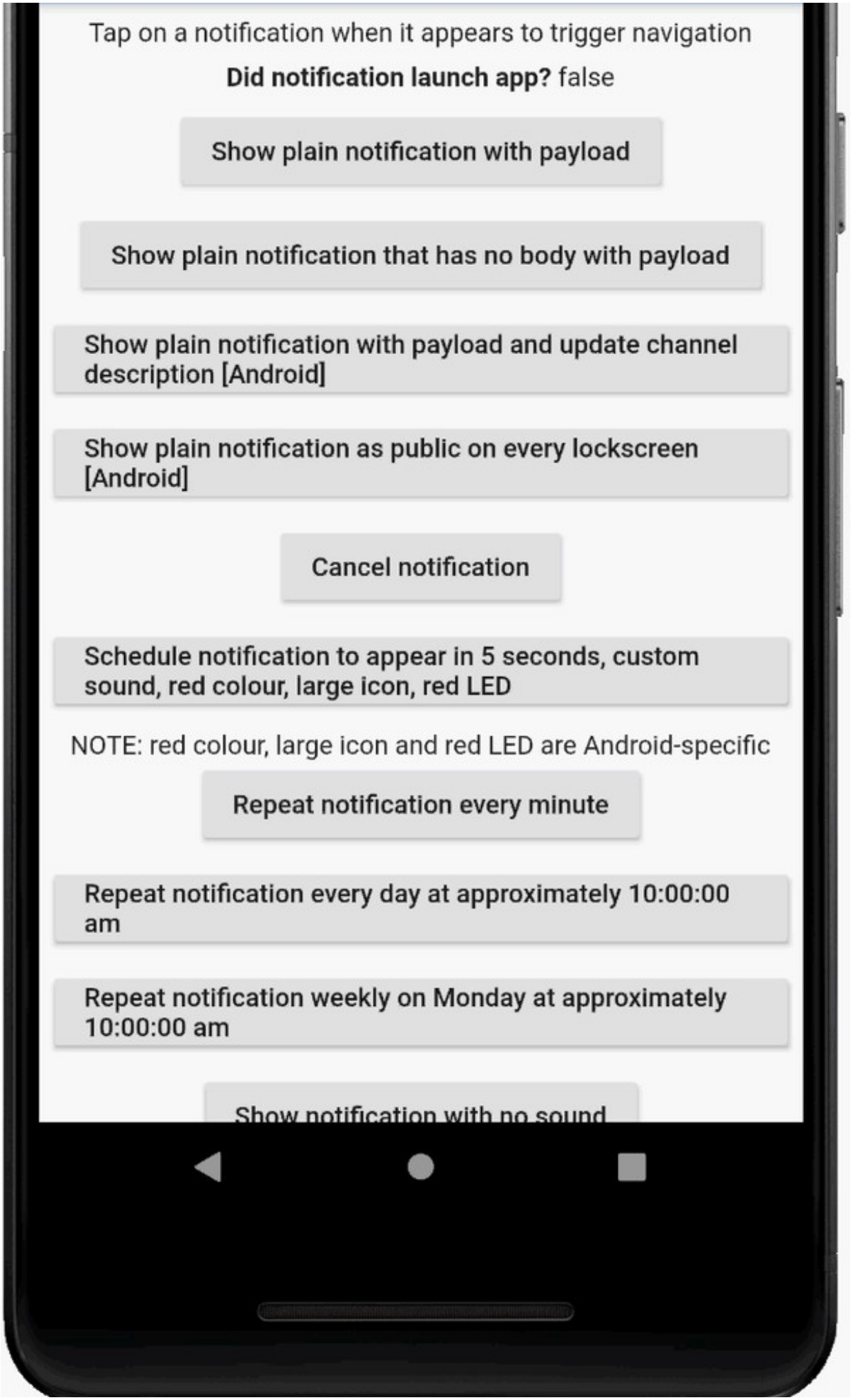
Notifications in Flutter

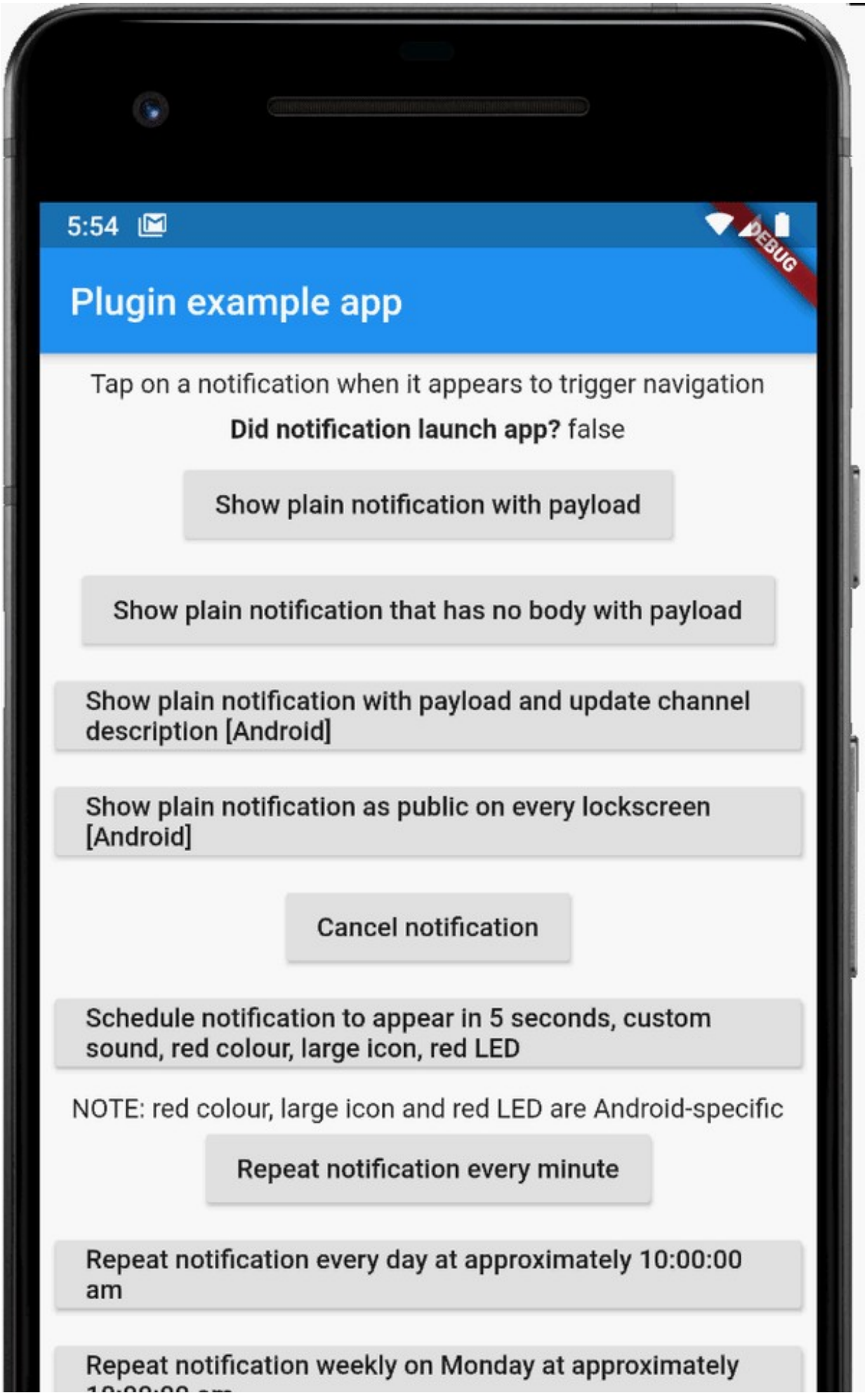
Android and iOS Notifications in one codebase

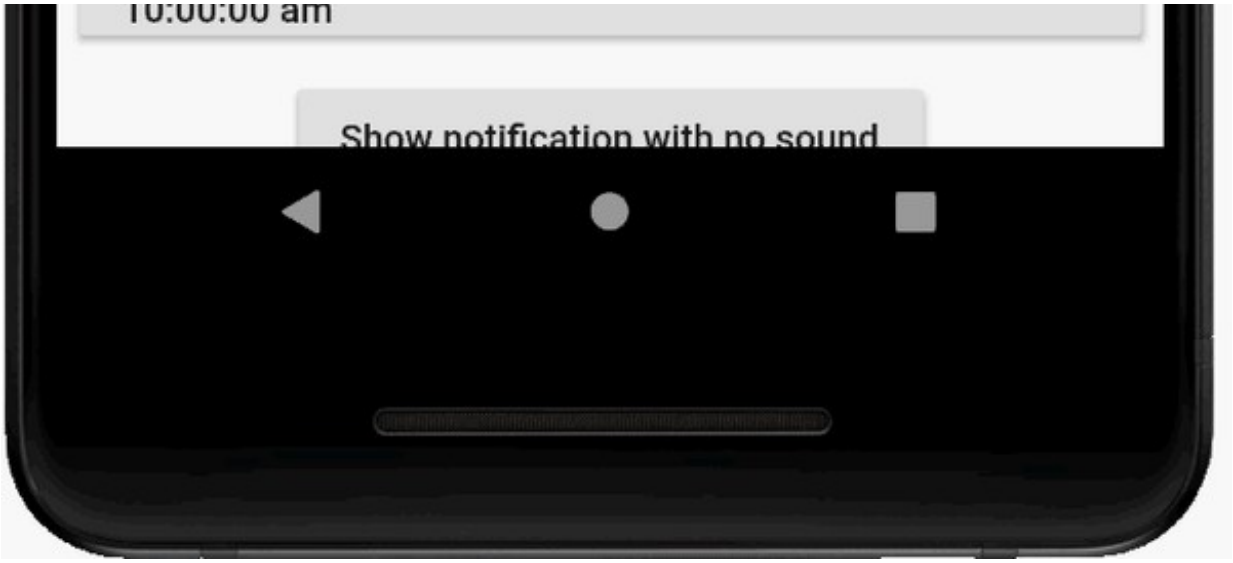
On an Android phone, when you have a new message, email or missed call, you will be able to swipe down from the top of the screen to access the **Notification Panel** and see at a glance what the notification details are. When there are notifications sitting in the Notification panel, you will also see an icon at the very top of the screen, as well as a badge on the application itself.

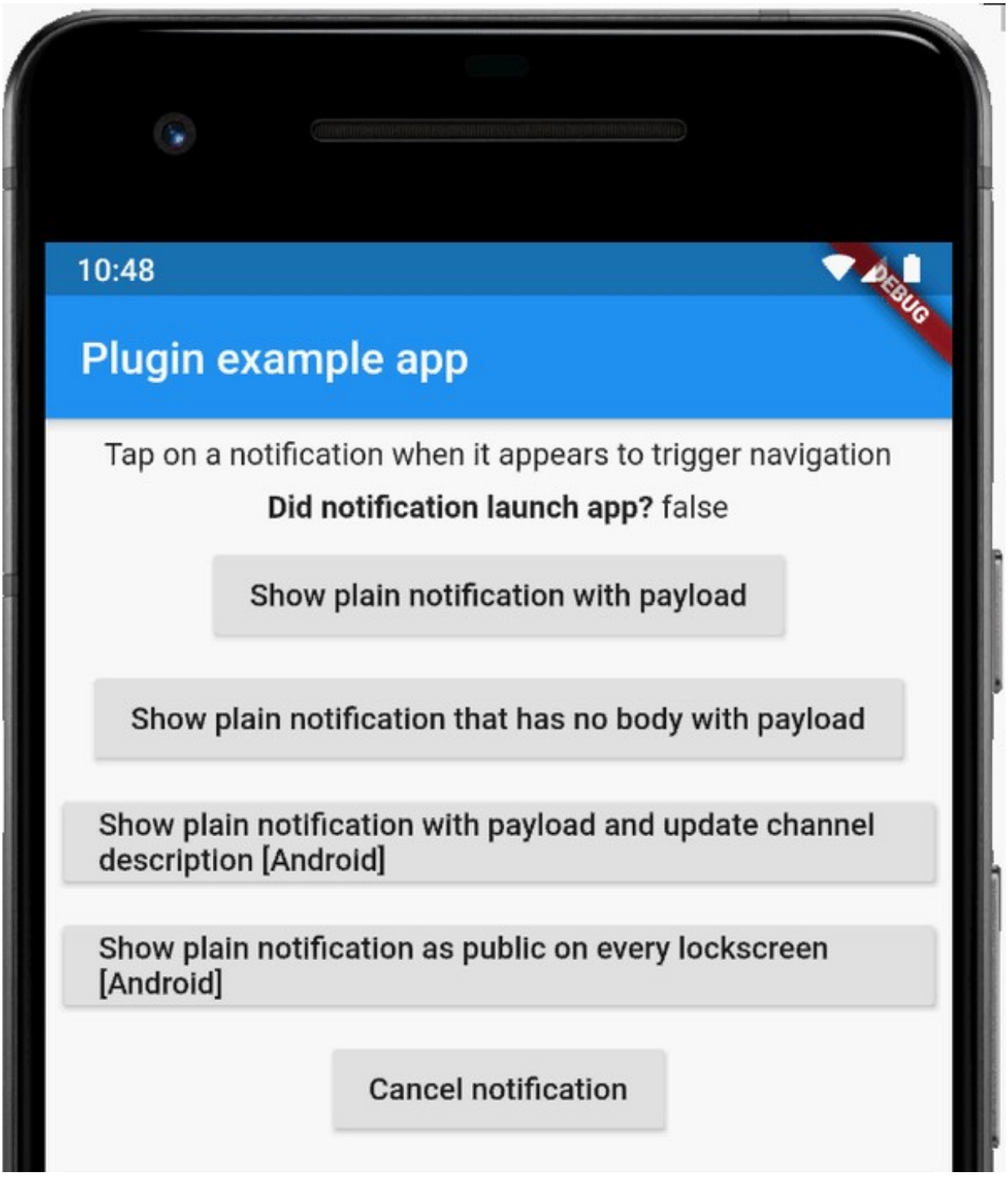
Michael Bui wrote a wonderful plugin, flutter_local_notifications, that provides such notifications not only in Android but also in iOS. You have to consider both platforms when using this plugin, and as it happens, when using the utility class I wrote to work with this plugin. In this article, I'm going to present this class called, ScheduleNotifications.

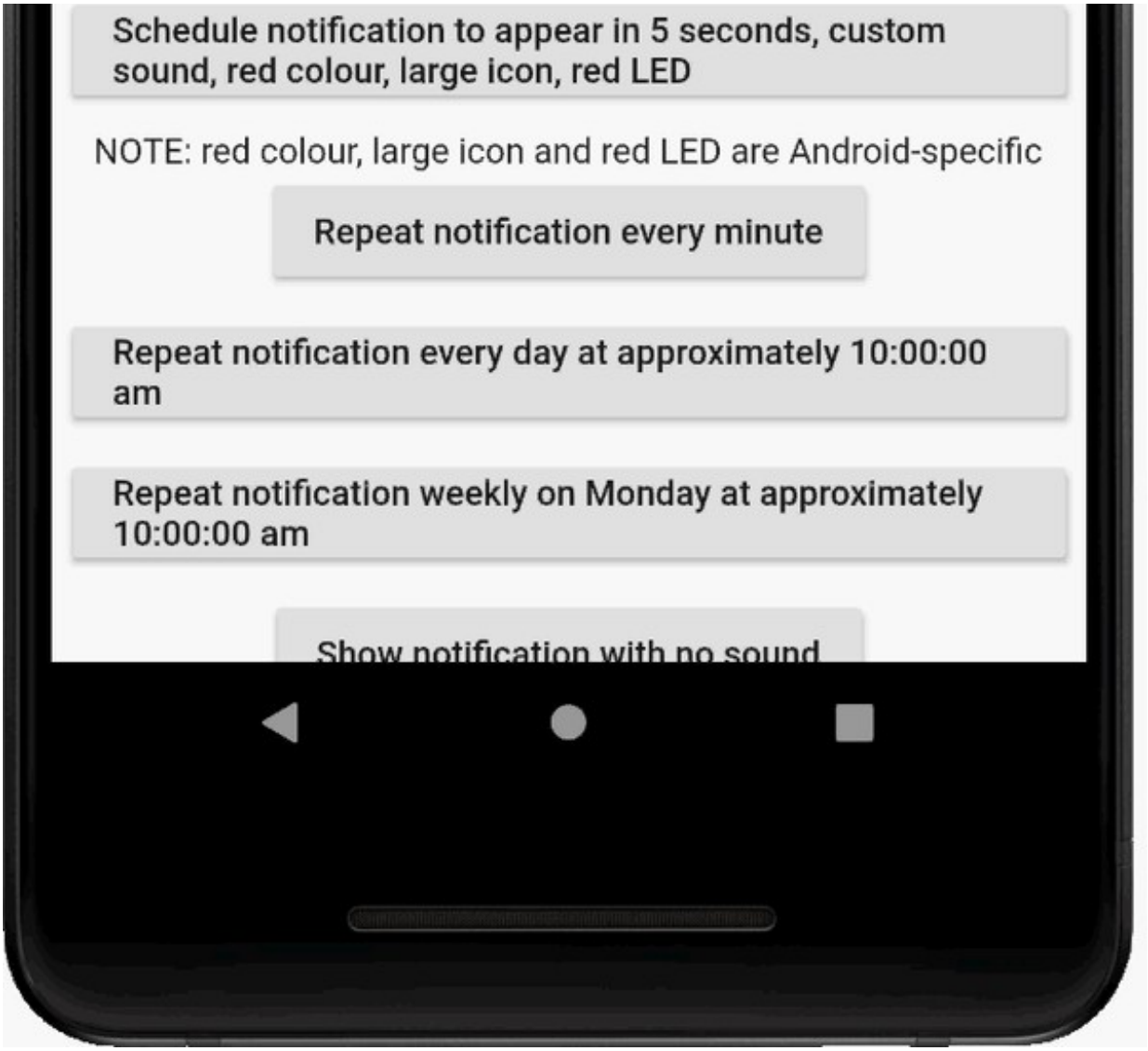


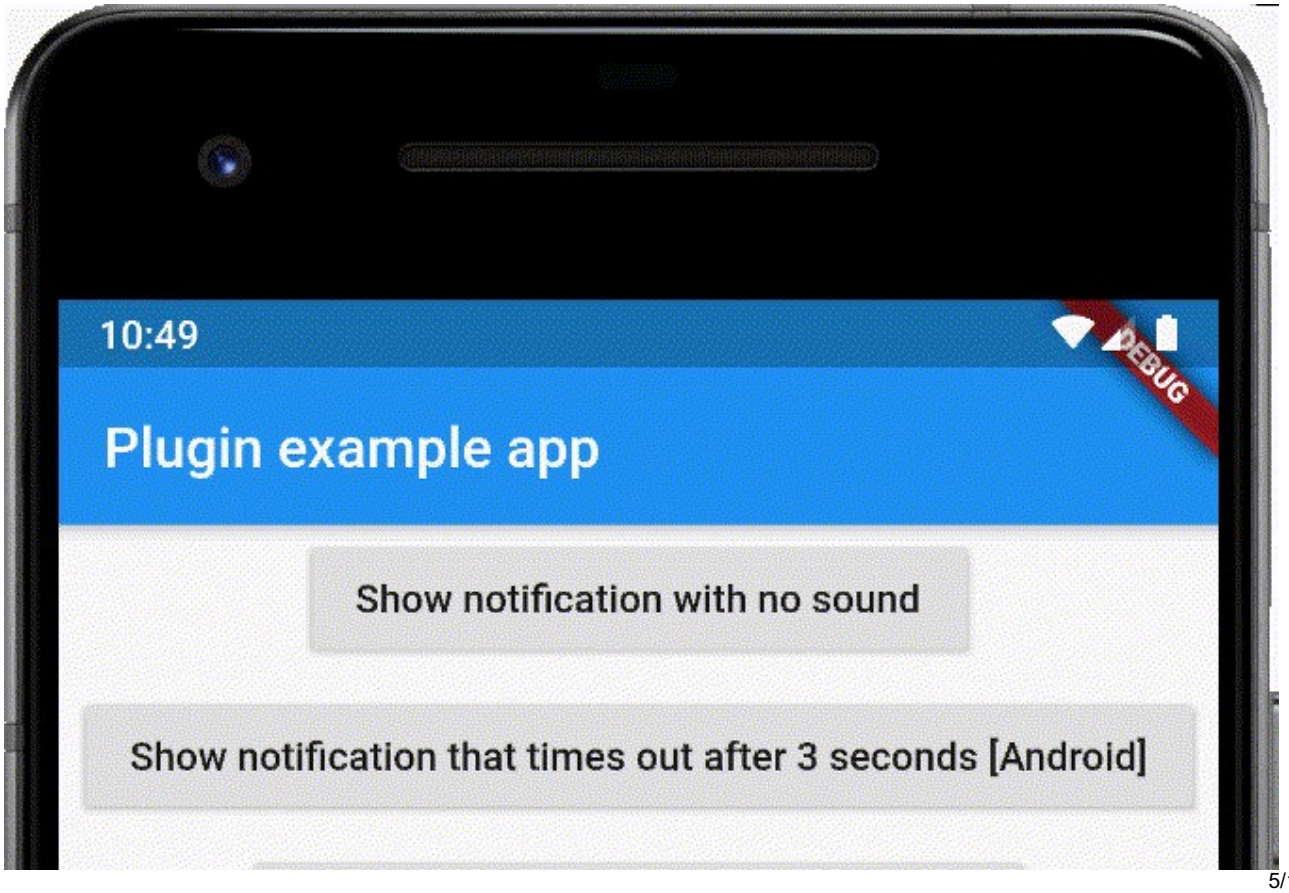

Tap on a notification when it appears to trigger navigation
Did notification launch app? false
Show plain notification with payload
Show plain notification that has no body with payload
Show plain notification with payload and update channel description [Android]
Show plain notification as public on every lockscreen [Android]
Cancel notification
Schedule notification to appear in 5 seconds, custom sound, red colour, large icon, red LED
NOTE: red colour, large icon and red LED are Android-specific
Repeat notification every minute
Repeat notification every day at approximately 10:00:00 am
Repeat notification weekly on Monday at approximately 10:00:00 am
Show notification with no sound

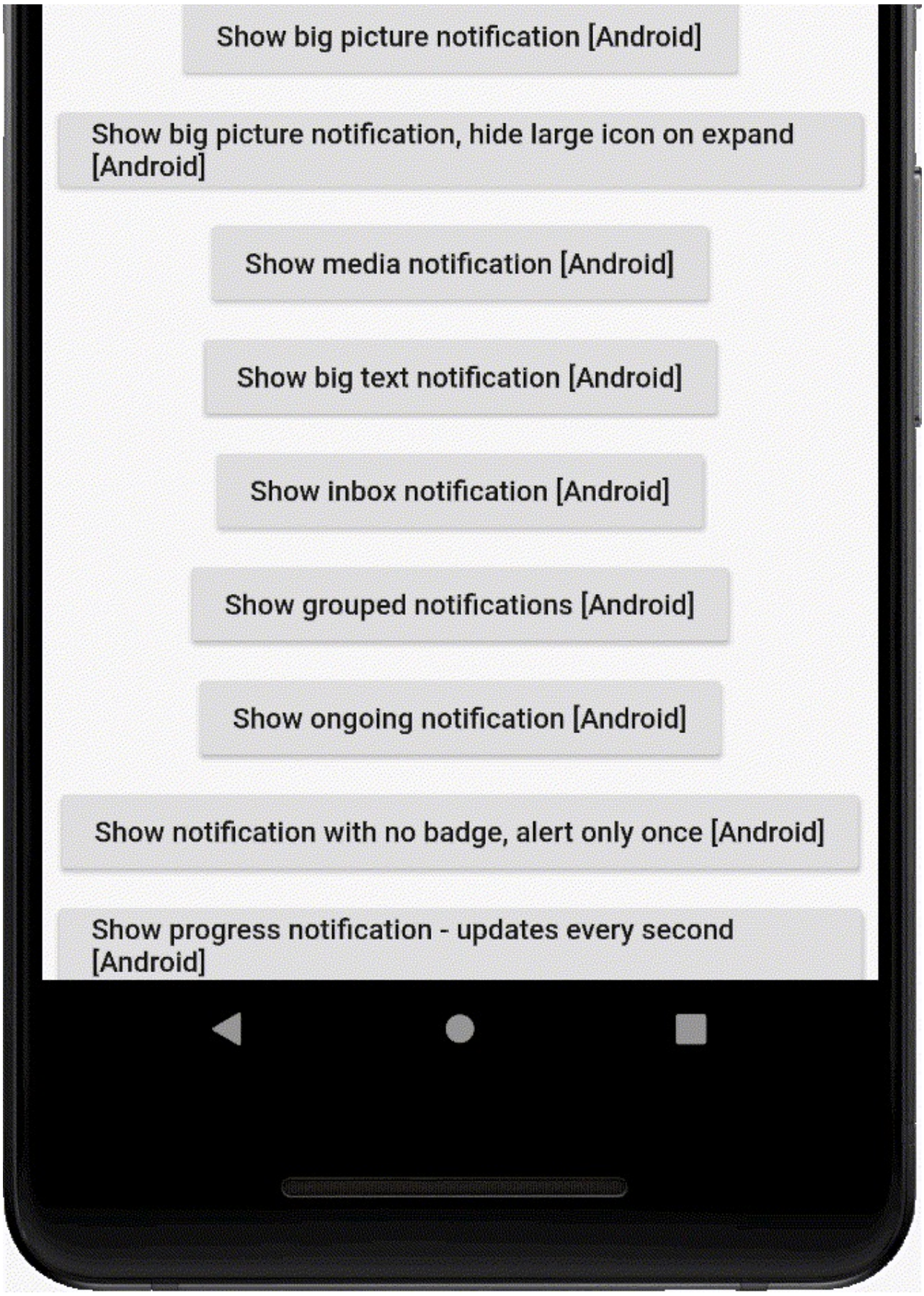

5:54
DEBUG
Plugin example app
Tap on a notification when it appears to trigger navigation
Did notification launch app? false
Show plain notification with payload
Show plain notification that has no body with payload
Show plain notification with payload and update channel description [Android]
Show plain notification as public on every lockscreen [Android]
Cancel notification
Schedule notification to appear in 5 seconds, custom sound, red colour, large icon, red LED
NOTE: red colour, large icon and red LED are Android-specific
Repeat notification every minute
Repeat notification every day at approximately 10:00:00 am
Repeat notification weekly on Monday at approximately

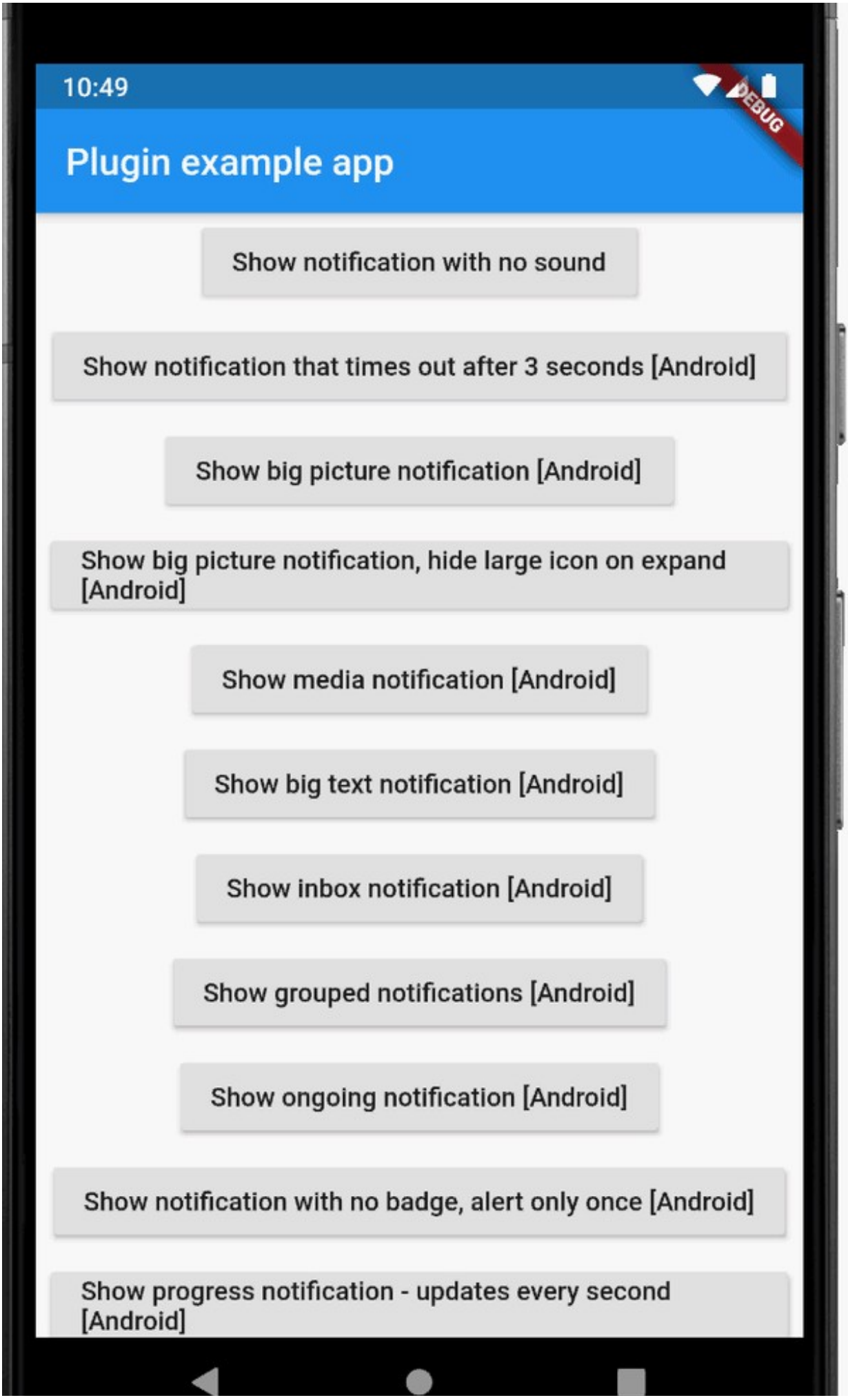

10:00:00 am
Show notification with no sound

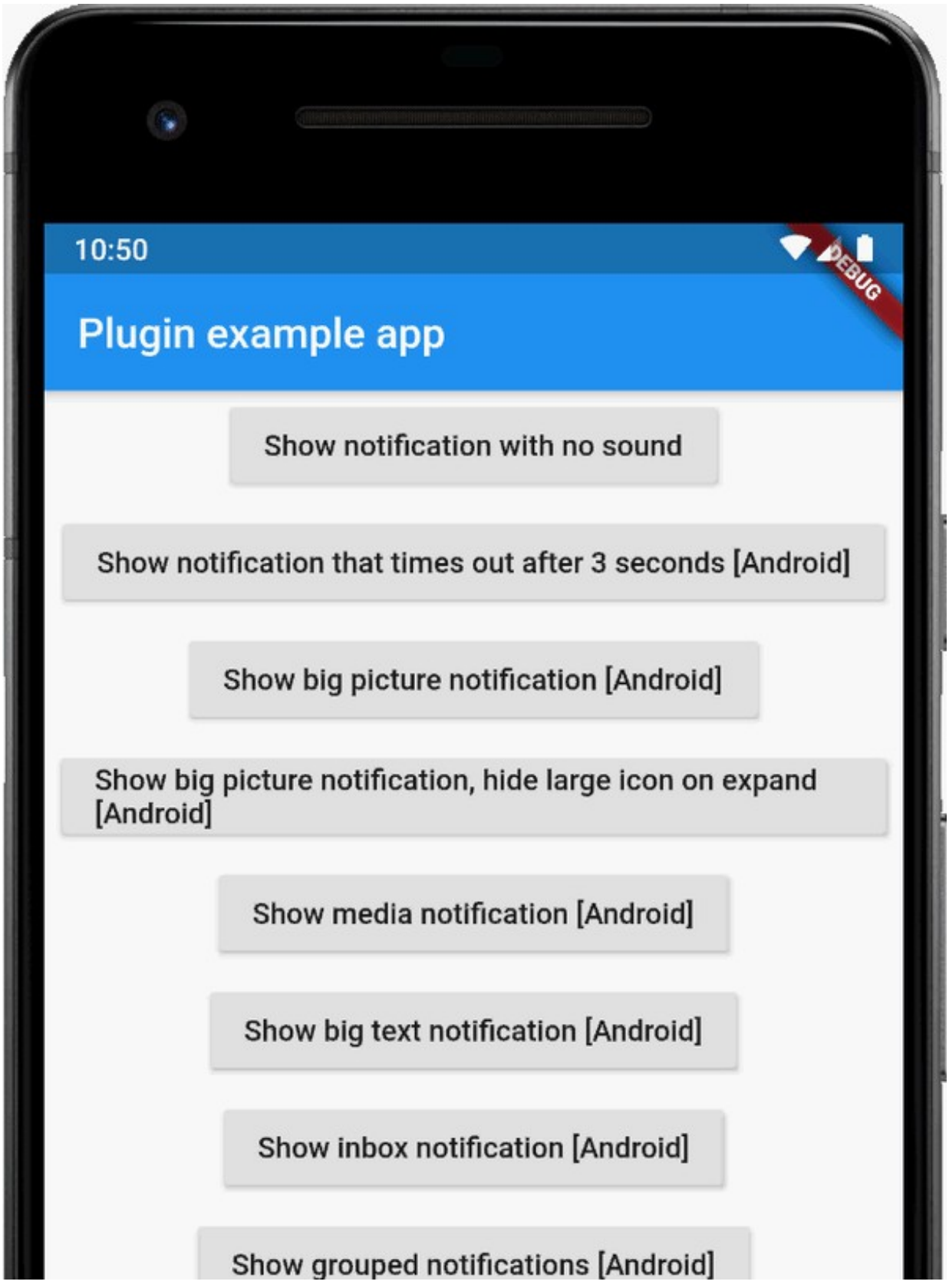

10:48
DEBUG
Plugin example app
Tap on a notification when it appears to trigger navigation
Did notification launch app? false
Show plain notification with payload
Show plain notification that has no body with payload
Show plain notification with payload and update channel description [Android]
Show plain notification as public on every lockscreen [Android]
Cancel notification

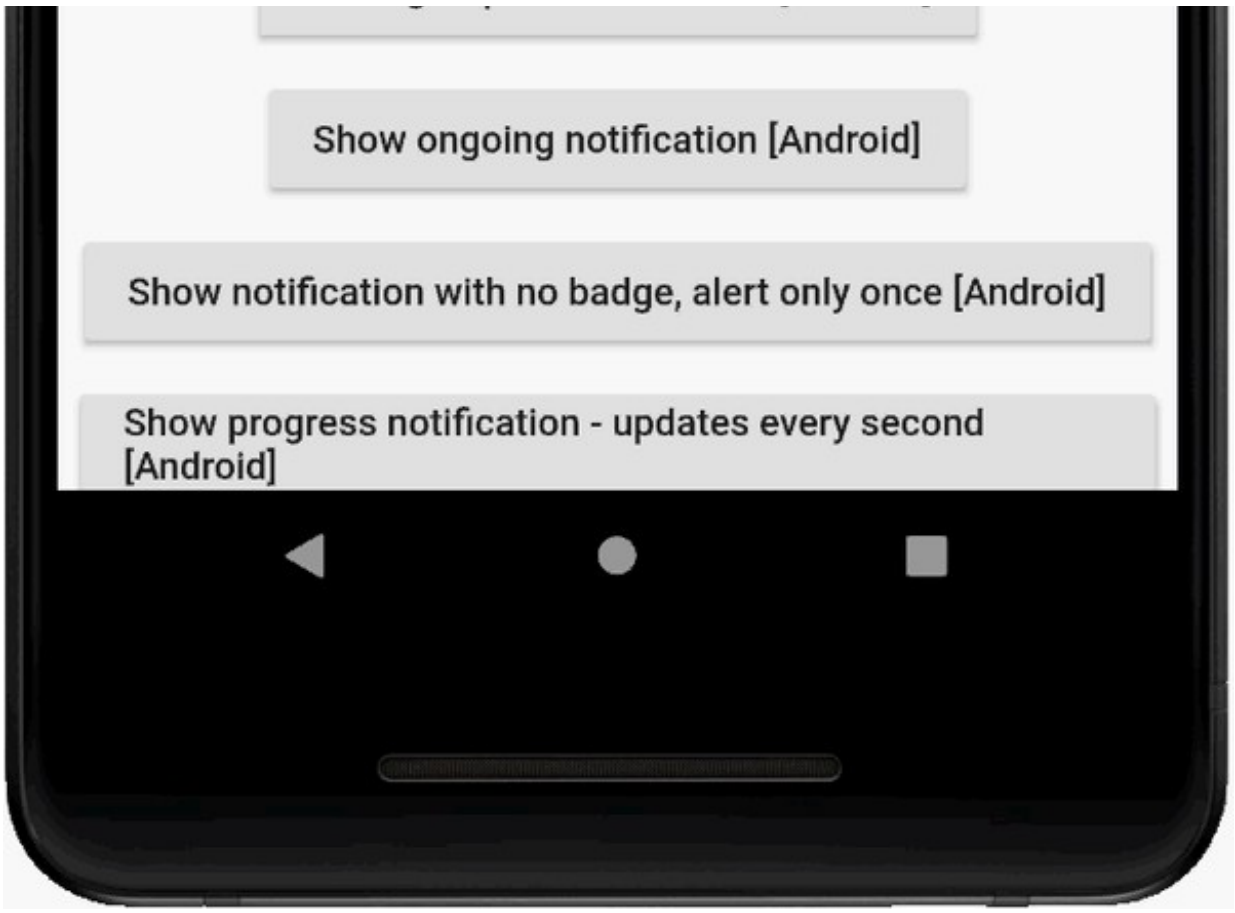

Schedule notification to appear in 5 seconds, custom sound, red colour, large icon, red LED
NOTE: red colour, large icon and red LED are Android-specific
Repeat notification every minute
Repeat notification every day at approximately 10:00:00 am
Repeat notification weekly on Monday at approximately 10:00:00 am
Show notification with no sound

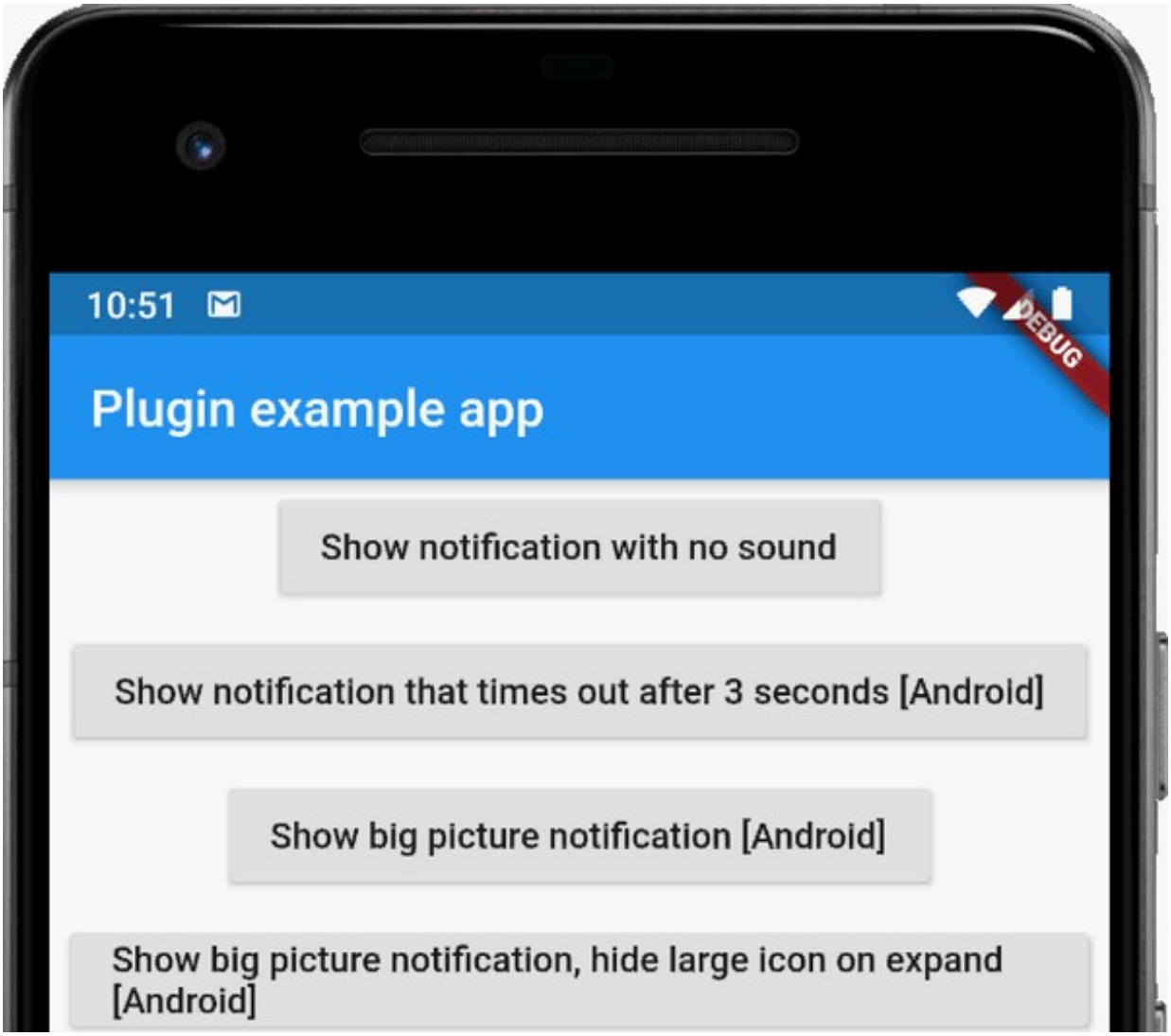

10:49
DEBUG
Plugin example app
Show notification with no sound
Show notification that times out after 3 seconds [Android]

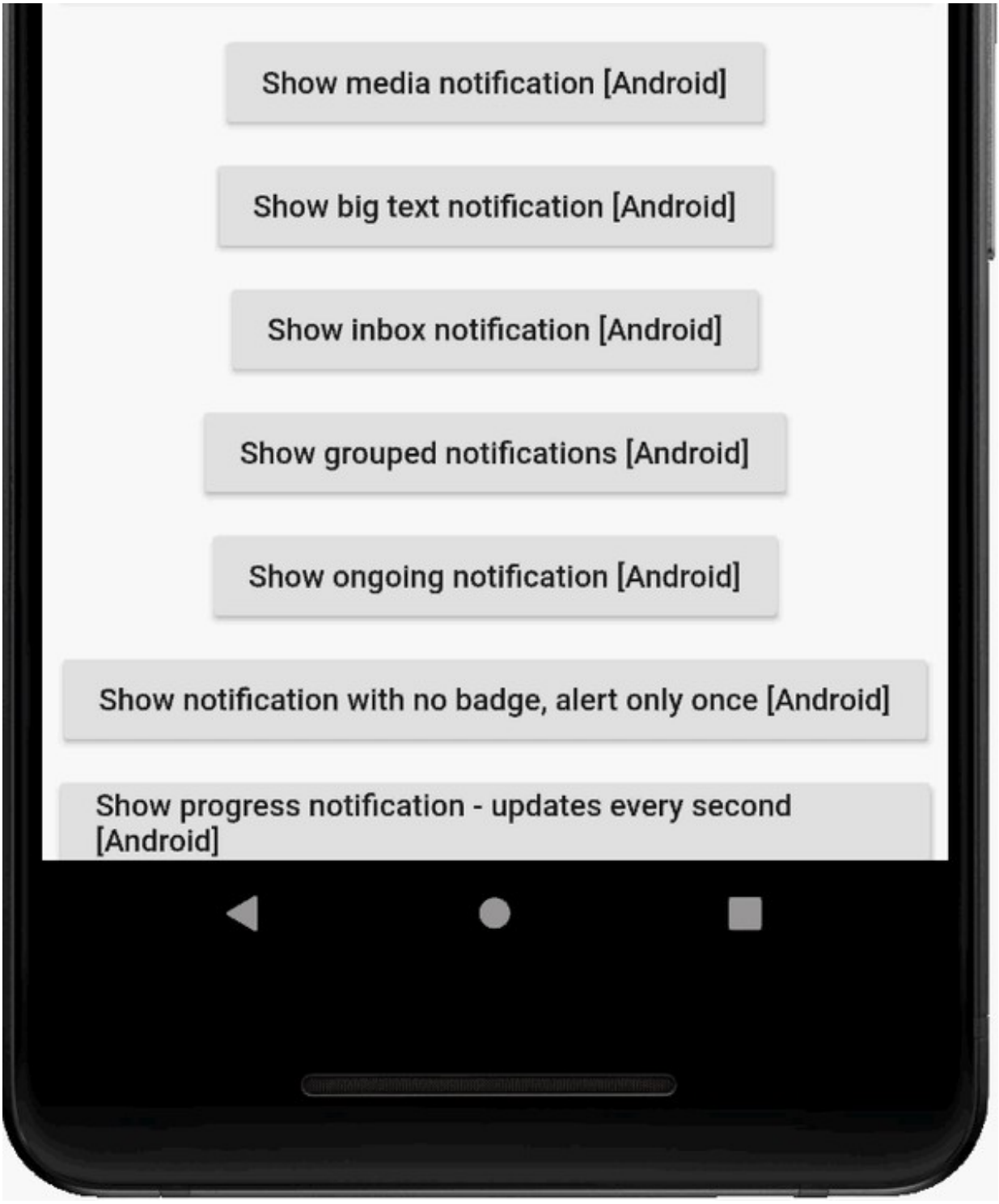

Show big picture notification [Android]
Show big picture notification, hide large icon on expand [Android]
Show media notification [Android]
Show big text notification [Android]
Show inbox notification [Android]
Show grouped notifications [Android]
Show ongoing notification [Android]
Show notification with no badge, alert only once [Android]
Show progress notification - updates every second [Android]

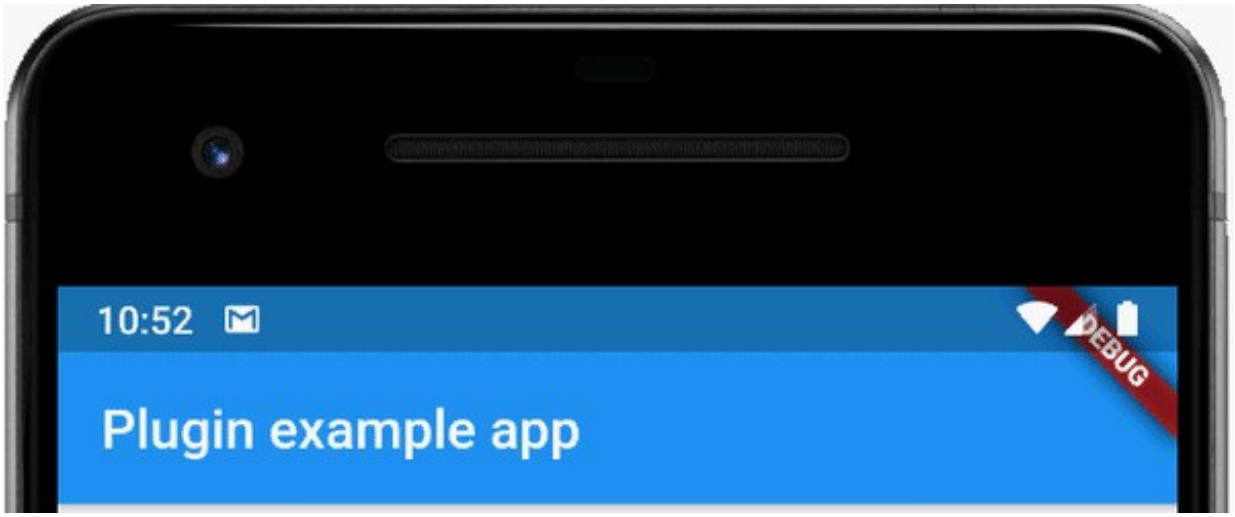

10:49
DEBUG
Plugin example app
Show notification with no sound
Show notification that times out after 3 seconds [Android]
Show big picture notification [Android]
Show big picture notification, hide large icon on expand [Android]
Show media notification [Android]
Show big text notification [Android]
Show inbox notification [Android]
Show grouped notifications [Android]
Show ongoing notification [Android]
Show notification with no badge, alert only once [Android]
Show progress notification - updates every second [Android]

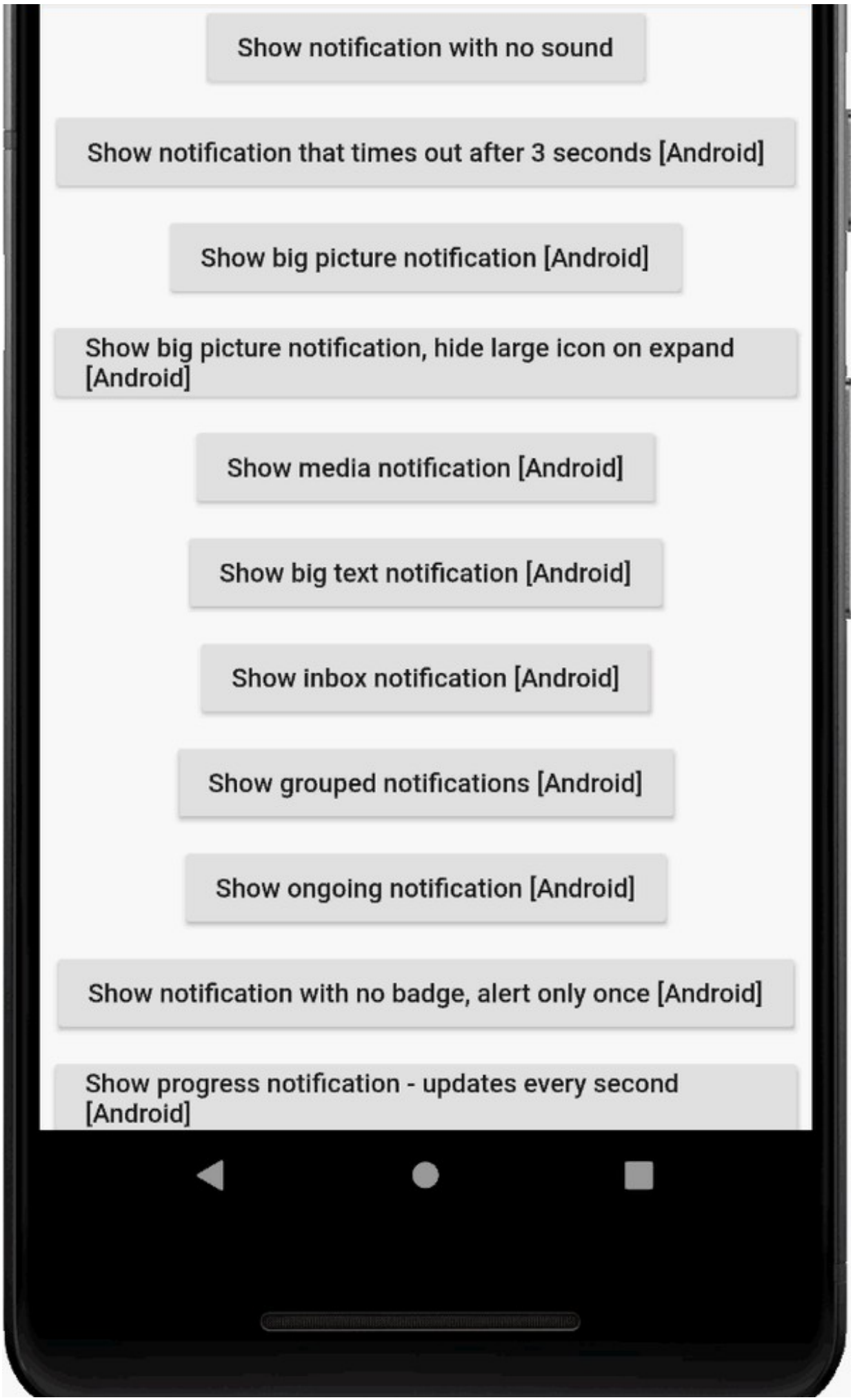

10:50
DEBUG
Plugin example app
Show notification with no sound
Show notification that times out after 3 seconds [Android]
Show big picture notification [Android]
Show big picture notification, hide large icon on expand [Android]
Show media notification [Android]
Show big text notification [Android]
Show inbox notification [Android]
Show grouped notifications [Android]

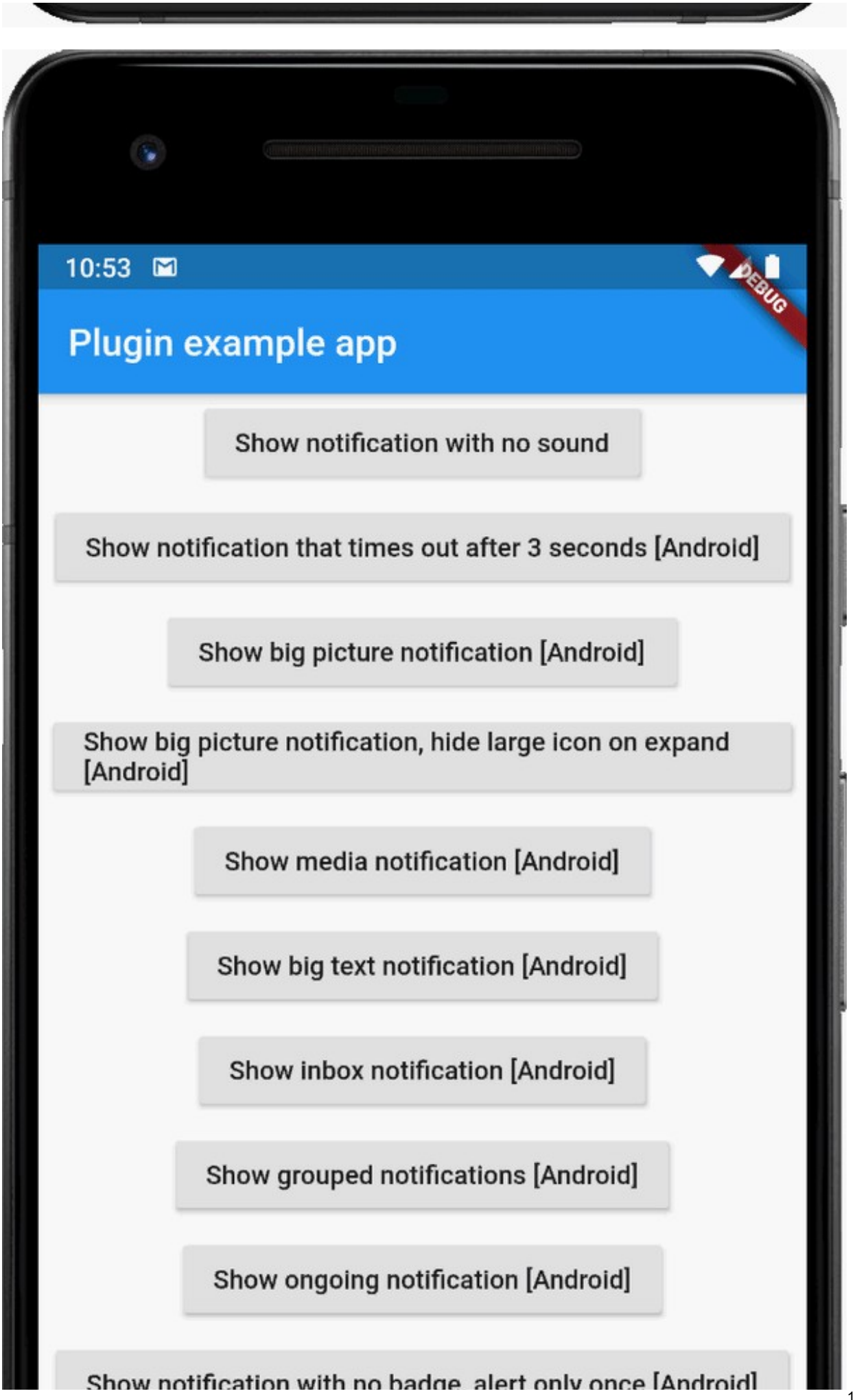

Show ongoing notification [Android]
Show notification with no badge, alert only once [Android]
Show progress notification - updates every second [Android]

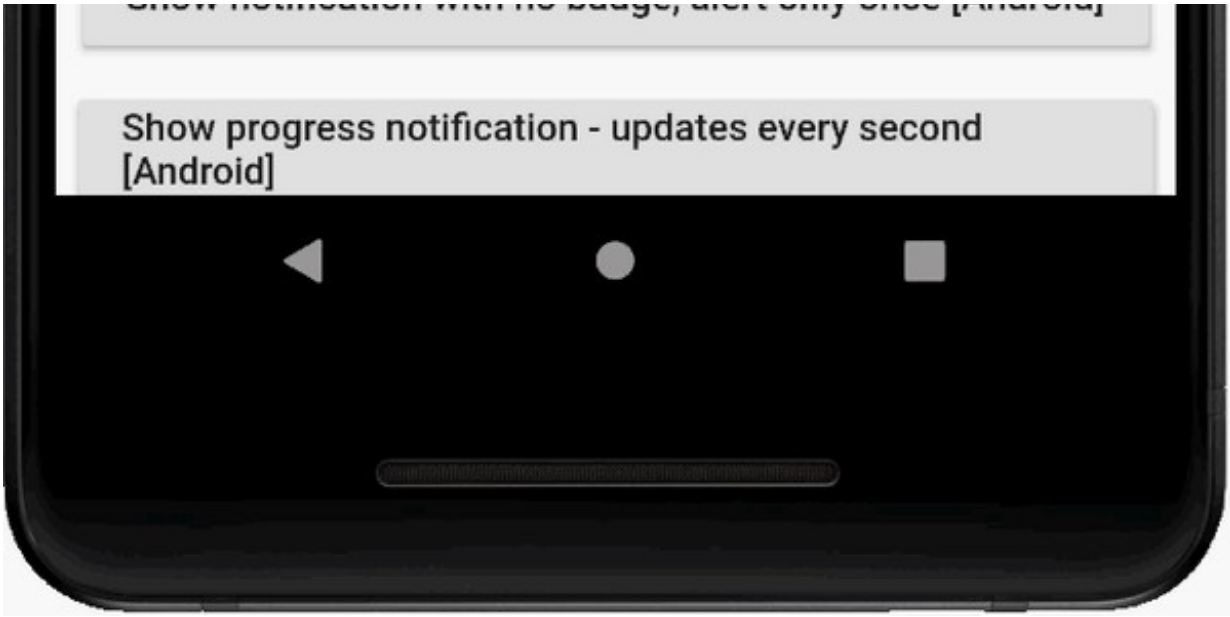

10:51
DEBUG
Plugin example app
Show notification with no sound
Show notification that times out after 3 seconds [Android]
Show big picture notification [Android]
Show big picture notification, hide large icon on expand [Android]


Show media notification [Android]
Show big text notification [Android]
Show inbox notification [Android]
Show grouped notifications [Android]
Show ongoing notification [Android]
Show notification with no badge, alert only once [Android]
Show progress notification - updates every second [Android]


10:52
DEBUG
Plugin example app


Show notification with no sound
Show notification that times out after 3 seconds [Android]
Show big picture notification [Android]
Show big picture notification, hide large icon on expand [Android]
Show media notification [Android]
Show big text notification [Android]
Show inbox notification [Android]
Show grouped notifications [Android]
Show ongoing notification [Android]
Show notification with no badge, alert only once [Android]
Show progress notification - updates every second [Android]


10:53
DEBUG
Plugin example app
Show notification with no sound
Show notification that times out after 3 seconds [Android]
Show big picture notification [Android]
Show big picture notification, hide large icon on expand [Android]
Show media notification [Android]
Show big text notification [Android]
Show inbox notification [Android]
Show grouped notifications [Android]
Show ongoing notification [Android]
Show notification with no badge, alert only once [Android]



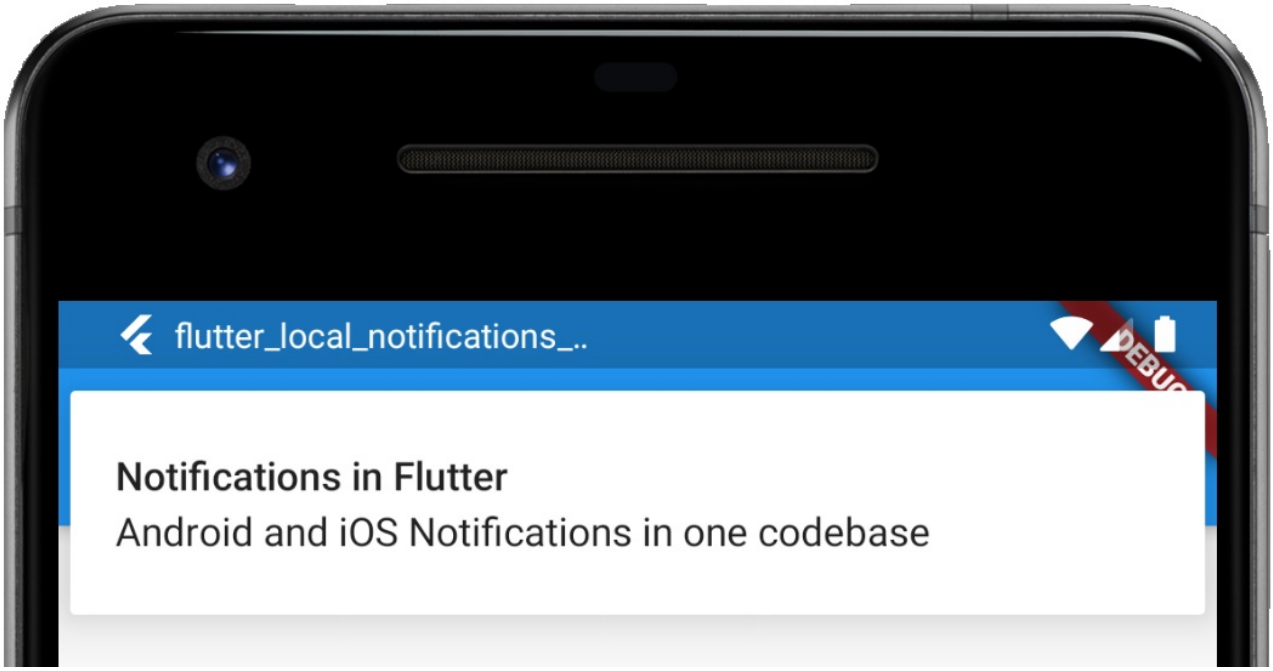
I Like Screenshots. Click For Gists.

As always, I prefer using screenshots over gists to show concepts rather than just show code in my articles. I find them easier to work with frankly. However, you can click or tap on these screenshots to see the code they represent in a gist or in Github. Ironically, it's better to read this article about mobile development on your computer than on your phone. Besides, we program on our computers — not on our phones. For now.

No Moving Pictures, No Social Media

There are a number of *gif* files in this article demonstrating aspects of the topic at hand. However, it's said viewing such *gif* files is not possible when reading this article on platforms like Instagram, Facebook, etc. Please, be aware of this and maybe read this article on medium.com

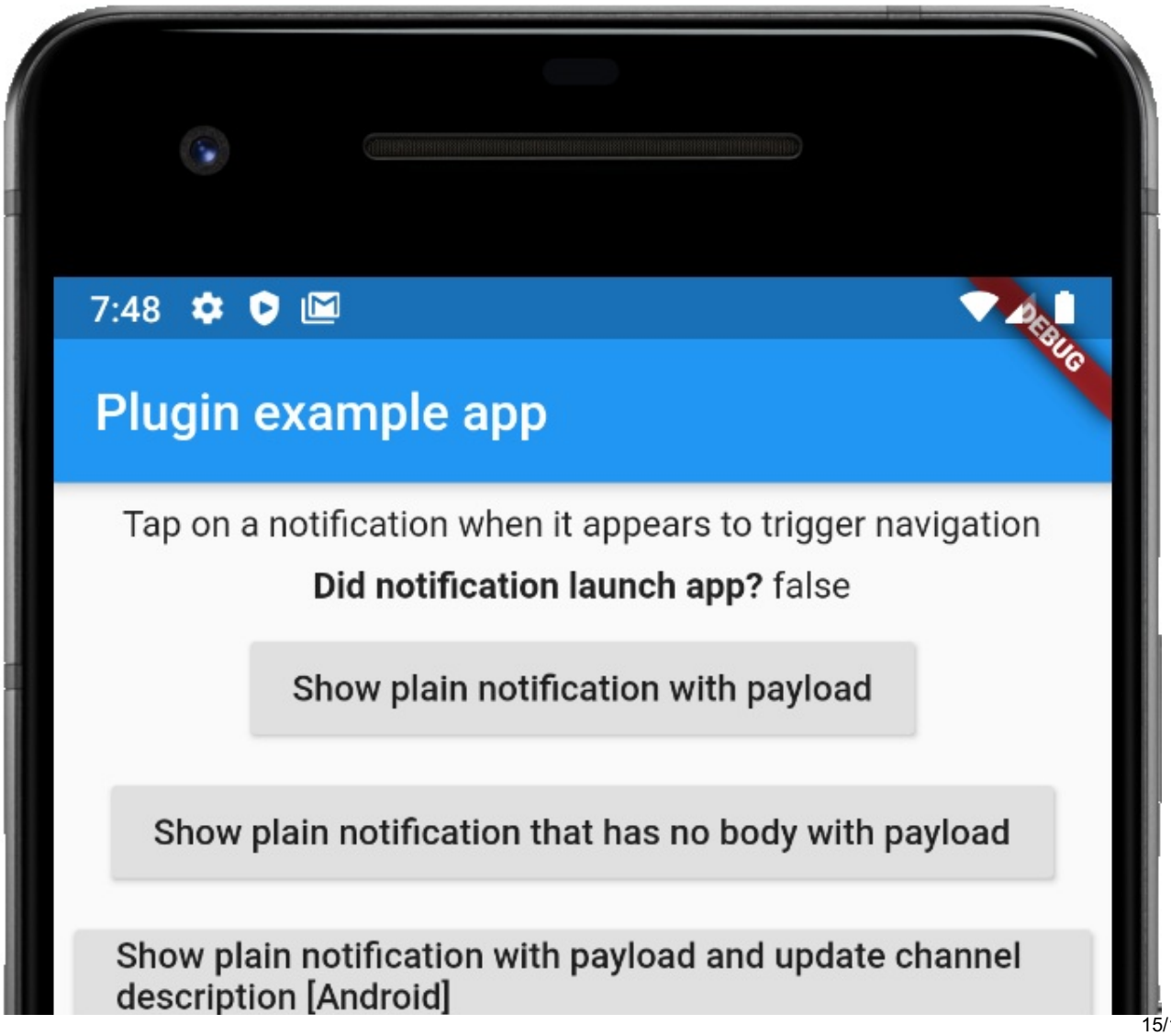
Let's begin.

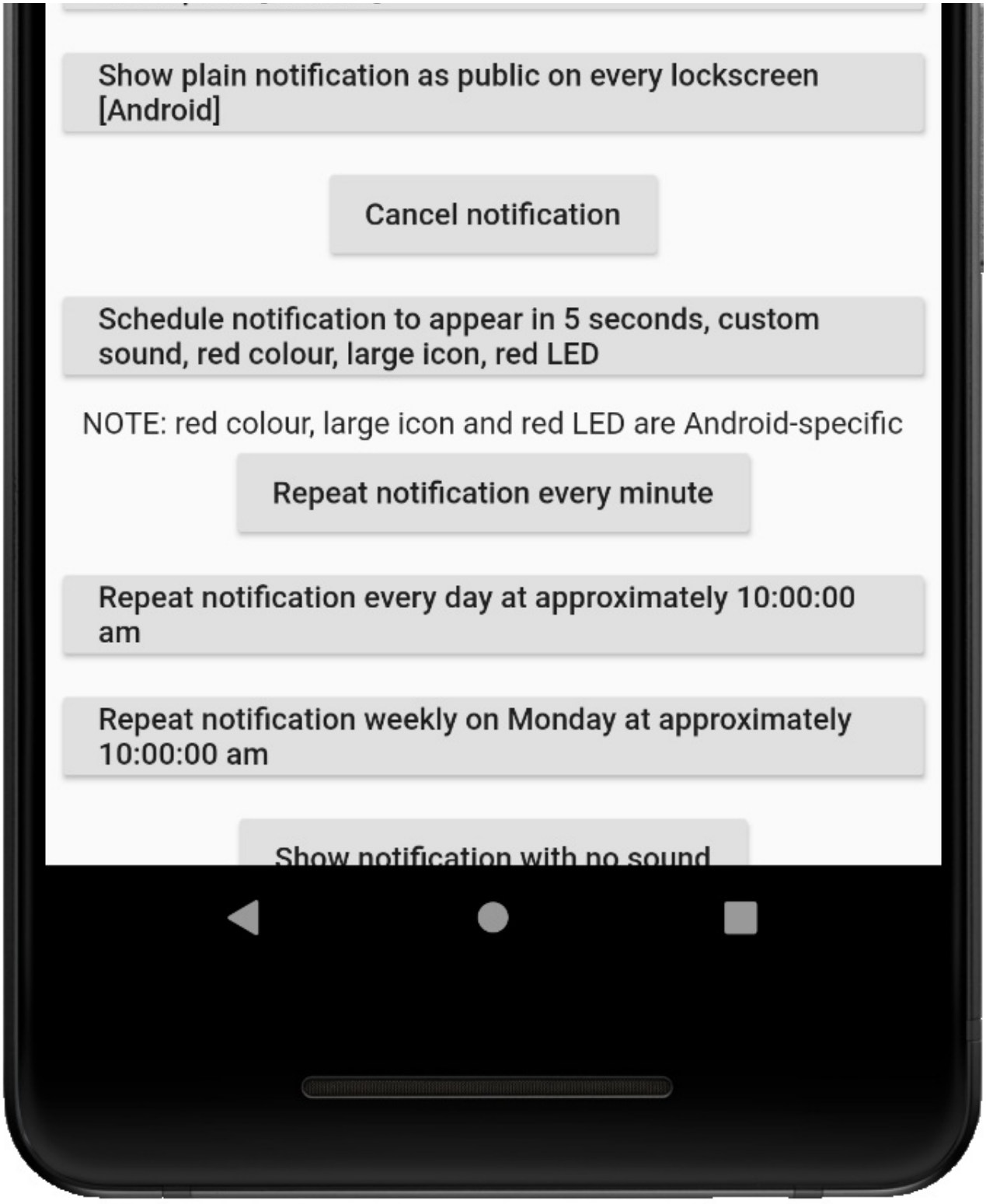


Other Stories by Greg Perry

Of course, I invite you to use this class which, in turn, will use the wonderful plugin by Micheal Bui to display notifications in your app. As you may know, I've supplied such a utility class before working with useful plugins. I've then presented articles on medium.com to demonstrate their use. With each article, I would take advantage of some steadfast examples to assist in the demonstration. Here, there's no exception. In this case, I'll use the very same example Micheal uses to demonstrate his plugin. The *gif* files displayed at the beginning of this article are from his example. Instead of importing his plugin, however, I've imported and used the utility class found in the library file, *schedule_notfications.dart*. Let's go through the example code now and see how it work.

```dart
import 'dart:async' show Future;

import 'dart:io' show File;

import 'dart:typed_data' show Int64List;

import 'dart:ui' show Color, FontWeight, Radius, VoidCallback;

import 'package:http/http.dart' as http;

import 'package:path_provider/path_provider.dart'
    show getApplicationDocumentsDirectory;

import 'package:flutter/cupertino.dart';

import 'package:flutter/material.dart';

import 'schedule_notifications.dart';

NotificationAppLaunchDetails notificationAppLaunchDetails;
```



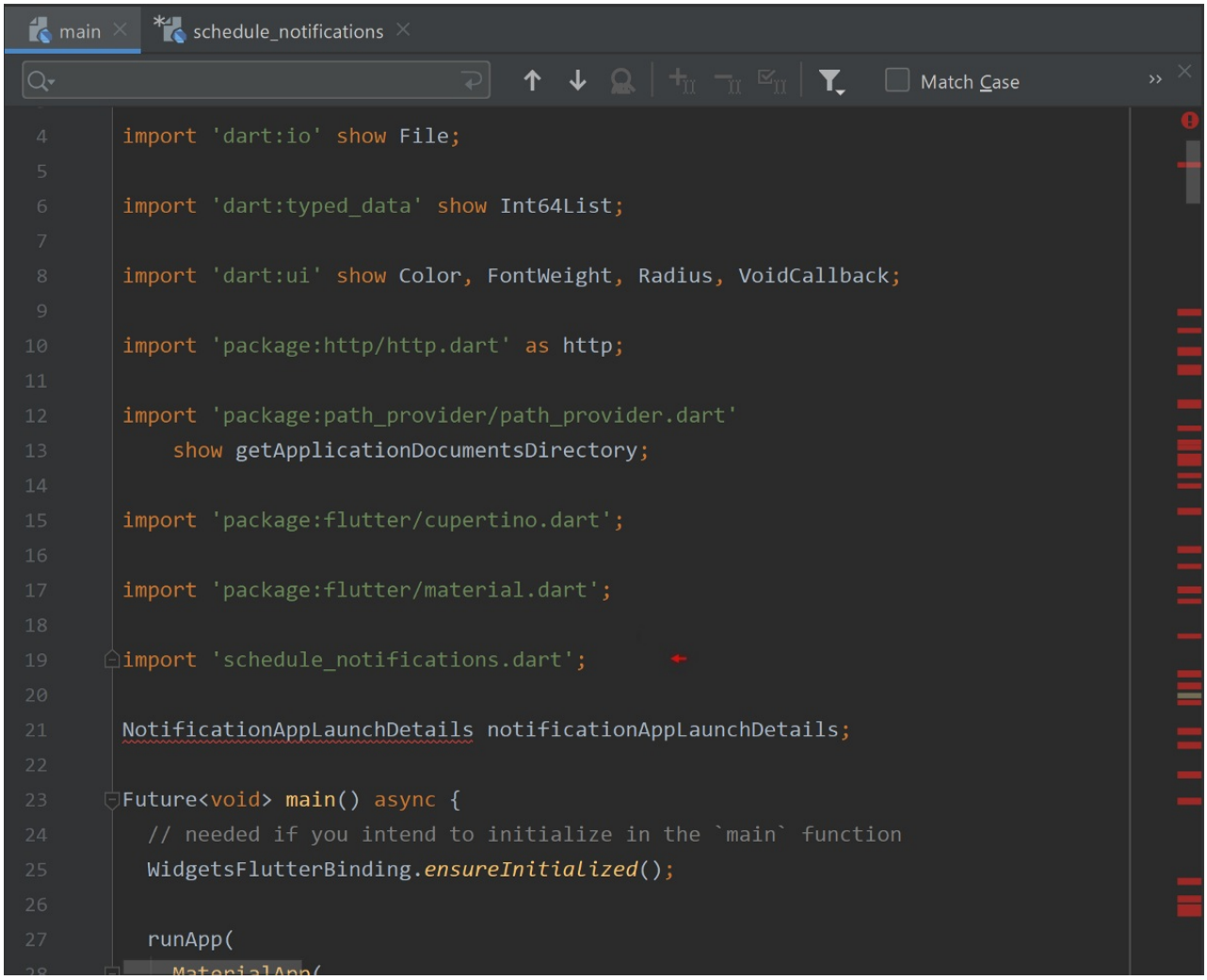


flutter_notifications_example.dart

Export What You Need

Now, instead of being concern with importing the appropriate Dart file (see below) for Micheal's plugin, simply import the file, ***'schedule_notifications.dart';***. As an example, in the screenshot below you can see the example code we're using is suddenly inundated with a bunch of 'undefined errors' everywhere?! What suddenly happened to cause this?

package:flutter_local_notifications/flutter_local_notifications.dart



'undefined errors' in flutter_notifications_example.dart

Those errors came about because I 'commented out' the export command you see in the screenshot below. This export command is found in the Dart file, *schedule_notifications.dart*. It's a little trick I found useful in my projects.

For example, a developer must include the plugin in their *pubspec.yaml* file (see below), but beyond that, there's no need now to import the plugin's Dart file. The file, *schedule_notifications.dart,* containing this utility class will ensure that all you need to work with (i.e. the plugin's secondary classes, functions and such) will also be available to you. You need only to import the utility class and not the file, *flutter_local_notifications.dart*.

flutter_local_notifications: ^1.4.0

By design, you are to just work with and be concerned with the utility class, and so just import that file. The utility class will import the plugin itself and worry about working with that plugin. Hence, you see the only import command needed is the one highlighted by a little red arrow above.

```dart
import 'dart:async' show Future;
import 'dart:math' show Random;
import 'dart:typed_data' show Int64List;
import 'dart:ui' show Color;

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
//
import 'package:flutter_local_notifications/flutter_local_notifications.dart';


/// Export those classes the user is likely to use and pass in.
export 'package:flutter_local_notifications/flutter_local_notifications.dart'
    show
        AndroidNotificationDetails,
        AndroidNotificationChannelAction,
        AndroidNotificationSound,
        AndroidBitmap,
        BigPictureStyleInformation,
        BigTextStyleInformation,
        Day,
        DefaultStyleInformation,
        DidReceiveLocalNotificationCallback,
        DrawableResourceAndroidBitmap,
        FilePathAndroidBitmap,
        GroupAlertBehavior,
        Importance,
        InitializationSettings,
        InboxStyleInformation,
        IOSNotificationDetails,
        IOSNotificationAttachment,
        MediaStyleInformation,
        NotificationAppLaunchDetails,
        NotificationDetails,
        NotificationVisibility,
        Priority,
        RawResourceAndroidNotificationSound,
        RepeatInterval,
        SelectNotificationCallback,
        Time;

export 'dart:typed_data' show Int64List;

export 'package:flutter/material.dart' show Color;
```

schedule_notifications.dart

By Example

Let's get back to the example code Micheal uses to demonstrate his plugin. It continues on as screenshots below. Again, I've removed every reference to his plugin in the example code and replaced it with the utility class called, *ScheduleNotifications*. I've cut up the example code into separate screenshots, but it's all there from top to bottom for your review. Of course, you're free to download the whole code yourself from the available gist, flutter_notifications_example.dart.

```
Future<void> main() async {
  // needed if you intend to initialize in the `main` function
  WidgetsFlutterBinding.ensureInitialized();

  runApp(
    MaterialApp(
      home: HomePage(),
    ), // MaterialApp
  );
}

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  @override
  void initState() {
    super.initState();

    notifications = ScheduleNotifications(
      'your channel id',
      'your other channel name',
```

flutter_notifications_example.dart

And so, in the State object's **initState**() function, we see the utility class being instantiated. The 'Android half' of the plugin requires that any and all Notifications you're going to display in your app be assigned one unique id, a name, and a description — all in Strings. I've chosen to keep the three original choices from the example code. Each pretty self-explanatory.

Init Your Notifications

After it's instantiated, the class object calls its **init**() function (see below). It's this function that, in fact, initializes the underlying plugin as well. You can see there's a

named parameter called, *onSelectNotification*, taking in an anonymous function. This is the function that will fire if and when a user actually taps on the displayed notification. In this case, when a notification is tapped, a second page will be displayed. Of course, all the named parameters used in this class mirror the ones used by the plugin itself — a common attribute for a utility class. In other words, what named parameters you find here in this class, you'd also find if you were using the plugin directly. Now, you may ask yourself. Why don't I just use the plugin directly?! Why indeed. It's hoped by the end of this article, you'll see why this utility class is at least a good alternative. Let's press on.

```
notifications = ScheduleNotifications(
  'your channel id',
  'your other channel name',
  'your other channel description',
);

notifications.init(onSelectNotification: (String payload) async {
  if (payload == null || payload.trim().isEmpty) return null;
  await Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => SecondScreen(payload)),
  );
  return;
});

notifications.getNotificationAppLaunchDetails().then((details){
  notificationAppLaunchDetails = details;
});
```

flutter_notifications_example.dart

Note, this class also involves asynchronous operations. In the screenshot above, the function, **getNotificationAppLaunchDetails**(), is called to assign its details to an instance variable. Such details describe if a notification was responsible for launching the very app the plugin is running in for example. Further details include the data displayed when the notification is tapped on by the user for example. Such data is called the notification's payload.

The next screenshot shows the beginning of the long **build**() function used in the example code to list the many push buttons down the centre of the screen. Besides some segments of this code, I'll display the corresponding gif file that depicts the sort of notification being codified there.

```dart
ScheduleNotifications notifications;

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Plugin example app'),
      ), // AppBar
      body: SingleChildScrollView(
        scrollDirection: Axis.vertical,
        child: Padding(
          padding: EdgeInsets.all(8.0),
          child: Center(
            child: Column(
              children: <Widget>[
                Padding(
                  padding: EdgeInsets.fromLTRB(0.0, 0.0, 0.0, 8.0),
                  child: Text(
                      'Tap on a notification when it appears to trigger naviga
                ), // Padding
                Padding(
                  padding: EdgeInsets.fromLTRB(0.0, 0.0, 0.0, 8.0),
                  child: Text.rich(
```

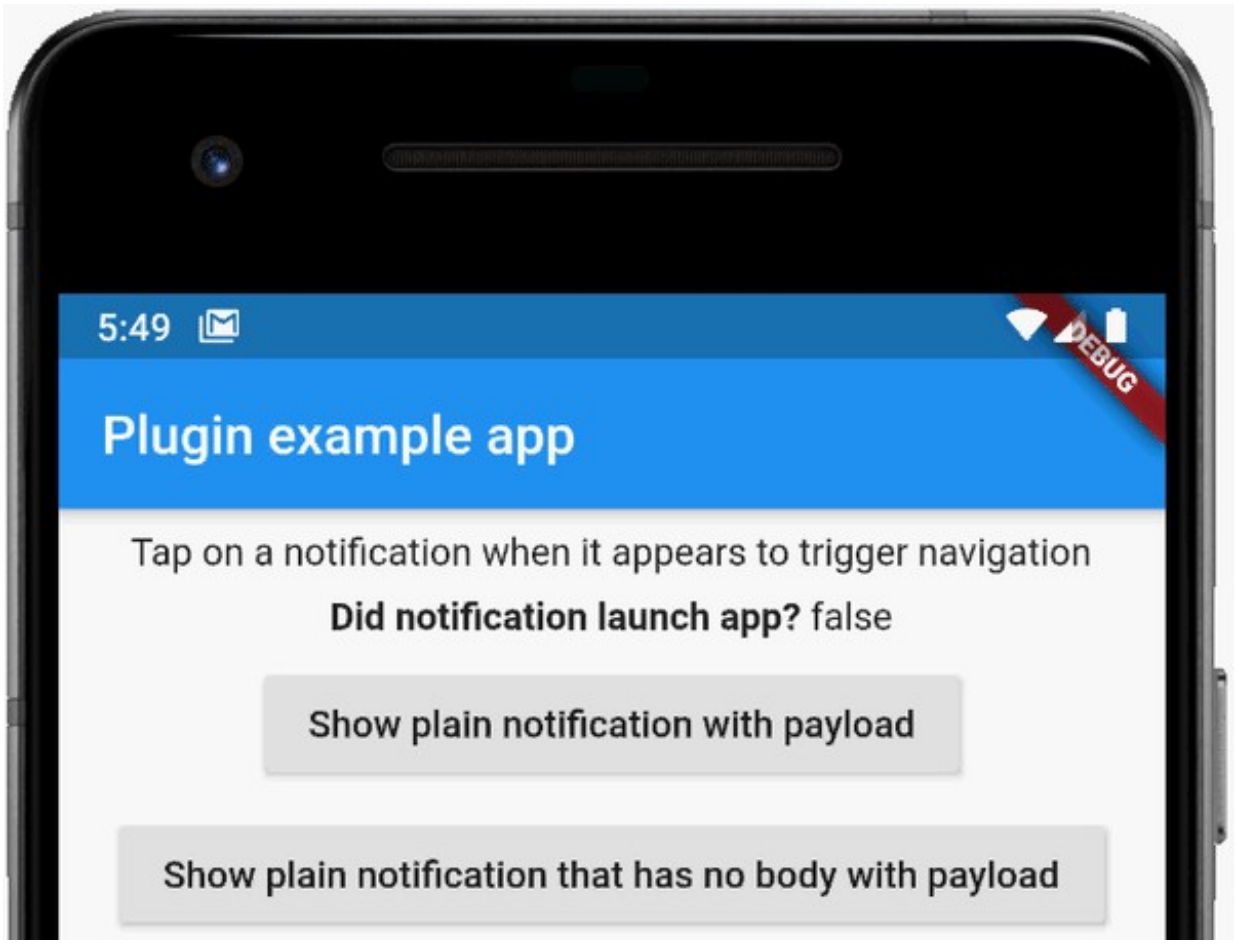
flutter_notifications_example.dart

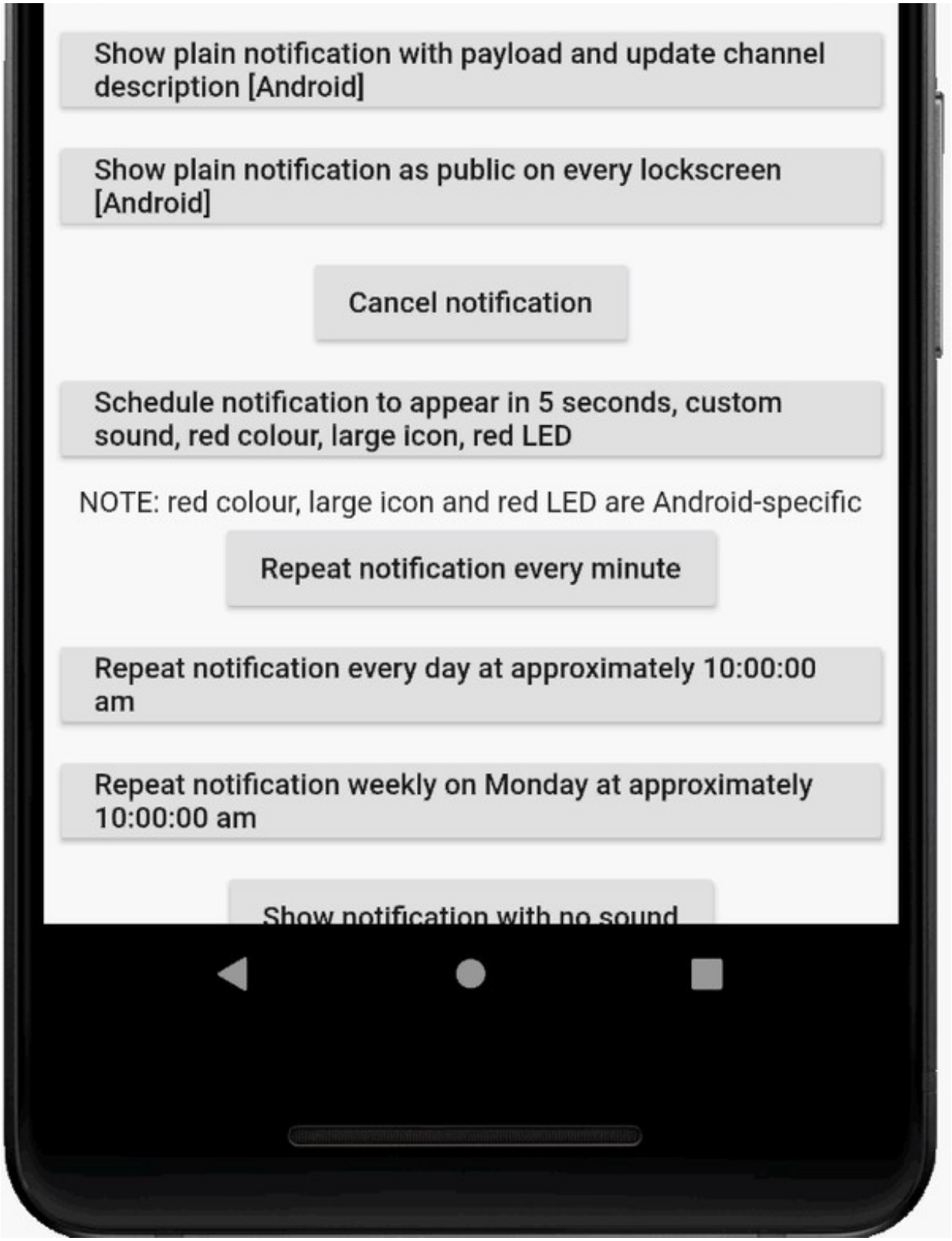
```
            TextSpan(
              children: [
                TextSpan(
                  text: 'Did notification launch app? ',
                  style: TextStyle(fontWeight: FontWeight.bold),
                ), // TextSpan
                TextSpan(
                  text:
                      '${notificationAppLaunchDetails?.didNotification
                ) // TextSpan
              ],
            ), // TextSpan
          ), // Text.rich
        ), // Padding
        if (notificationAppLaunchDetails?.didNotificationLaunchApp ??
            false)
          Padding(
            padding: EdgeInsets.fromLTRB(0.0, 0.0, 0.0, 8.0),
            child: Text.rich(
              TextSpan(
                children: [
                  TextSpan(
                    text: 'Launch notification payload: ',
                    style: TextStyle(fontWeight: FontWeight.bold),
```

flutter_notifications_example.dart

The first button listed is labelled, '*Show plain notification with payload*.' When you press that button, the **show**() function highlighted below is executed. With its 'importance' and 'priority' settings, a small white window pops up near the top of the mobile screen. You see the title and body text displayed in that white window. Then, when that white window is tapped on, a second page is displayed with the payload, 'item x', in the title bar.

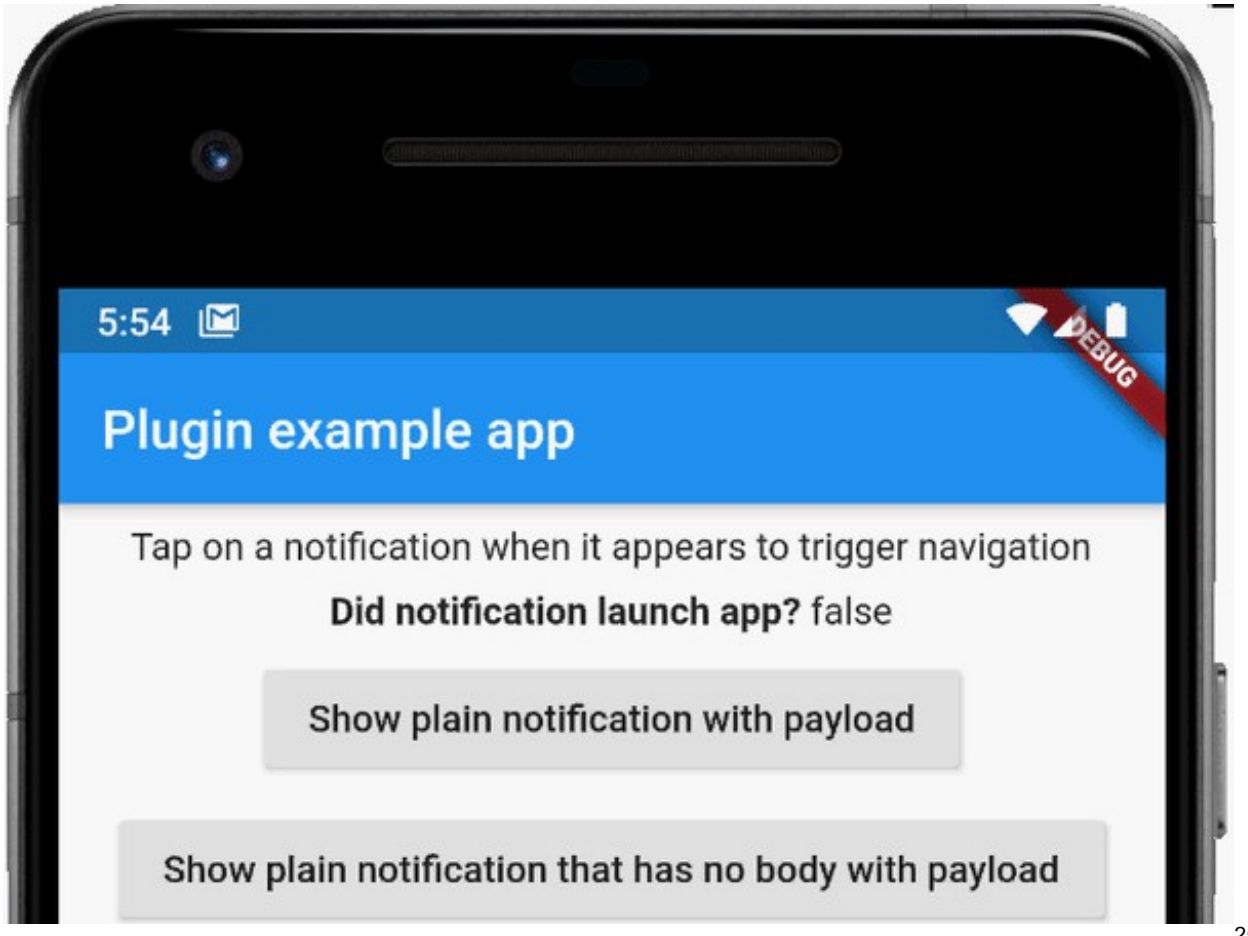
```
                ), // TextSpan
                TextSpan(
                  text: notificationAppLaunchDetails.payload,
                ) // TextSpan
              ],
            ), // TextSpan
          ), // Text.rich
        ), // Padding
        _PaddedRaisedButton(
          buttonText: 'Show plain notification with payload',
          onPressed: () => notifications.show(
            id: 0,
            importance: Importance.Max,
            priority: Priority.High,
            ticker: 'ticker',
            title: 'plain title',
            body: 'plain body',
            payload: 'item x',
          ),
        ), // _PaddedRaisedButton
        _PaddedRaisedButton(
          buttonText:
              'Show plain notification that has no body with payload',
          onPressed: () => notifications.show(
```

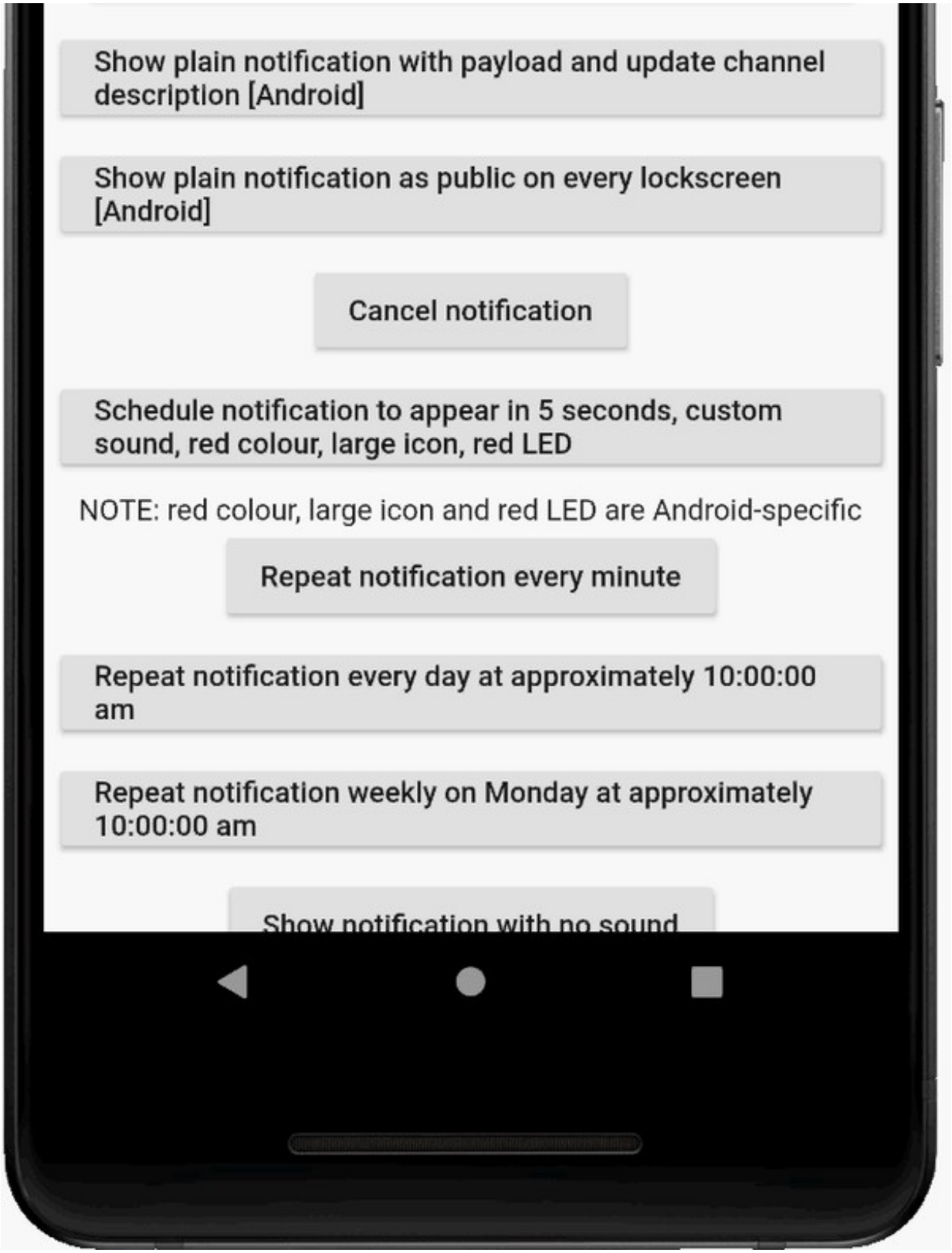




flutter_notifications_example.dart

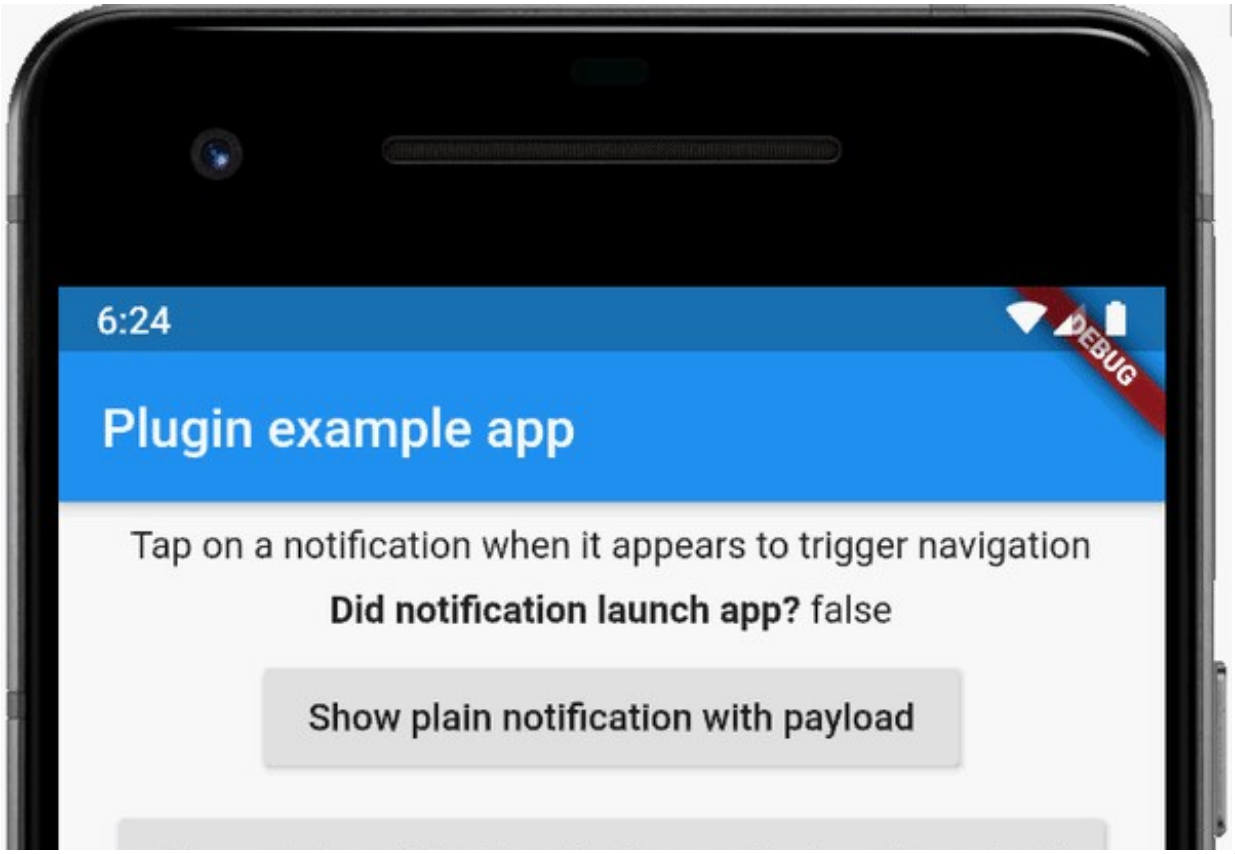
```
      onPressed: () => notifications.show(
        id: 0,
        importance: Importance.Max,
        priority: Priority.High,
        ticker: 'ticker',
        title: 'no body title',
        payload: 'item x',
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
          'Show plain notification with payload and update channel
      onPressed: () => notifications.show(
        id: 0,
        title: 'updated notification channel',
        body: 'check settings to see updated channel description',
        payload: 'item x',
        importance: Importance.Max,
        priority: Priority.High,
        channelAction: AndroidNotificationChannelAction.Update,
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
```

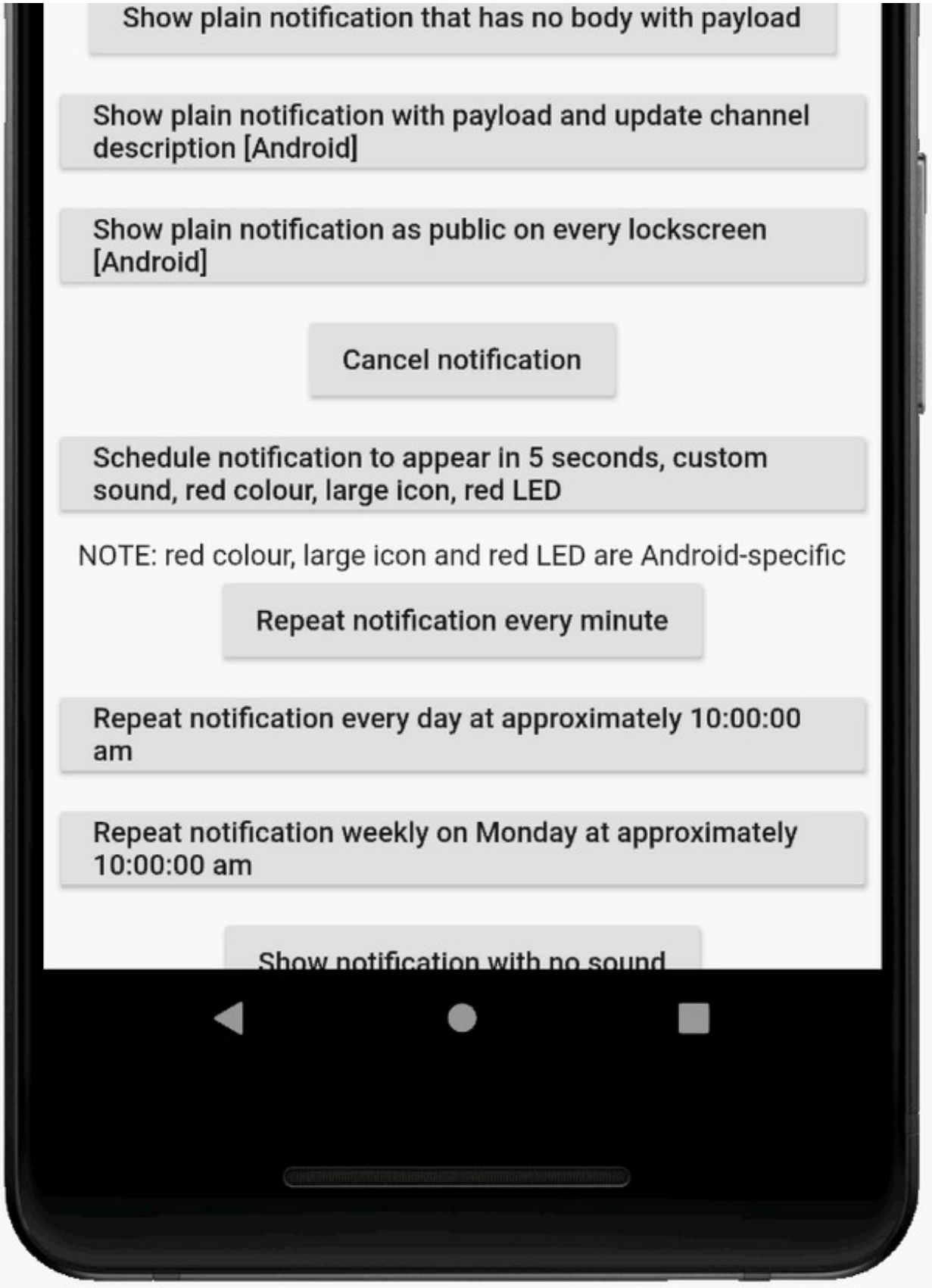




flutter_notifications_example.dart

```
_PaddedRaisedButton(
  buttonText:
      'Show plain notification as public on every lockscreen [
  onPressed: () => notifications.show(
    id: 0,
    title: 'public notification title',
    body: 'public notification body',
    payload: 'item x',
    importance: Importance.Max,
    priority: Priority.High,
    ticker: 'ticker',
    visibility: NotificationVisibility.Public,
  ),
), // _PaddedRaisedButton
_PaddedRaisedButton(
  buttonText: 'Cancel notification',
  onPressed: () => notifications.cancel(0),
), // _PaddedRaisedButton
_PaddedRaisedButton(
  buttonText:
      'Schedule notification to appear in 5 seconds, custom so
  onPressed: () async {
    //
    var vibrationPattern = Int64List(4);
    vibrationPattern[0] = 0;
    vibrationPattern[1] = 1000;
```

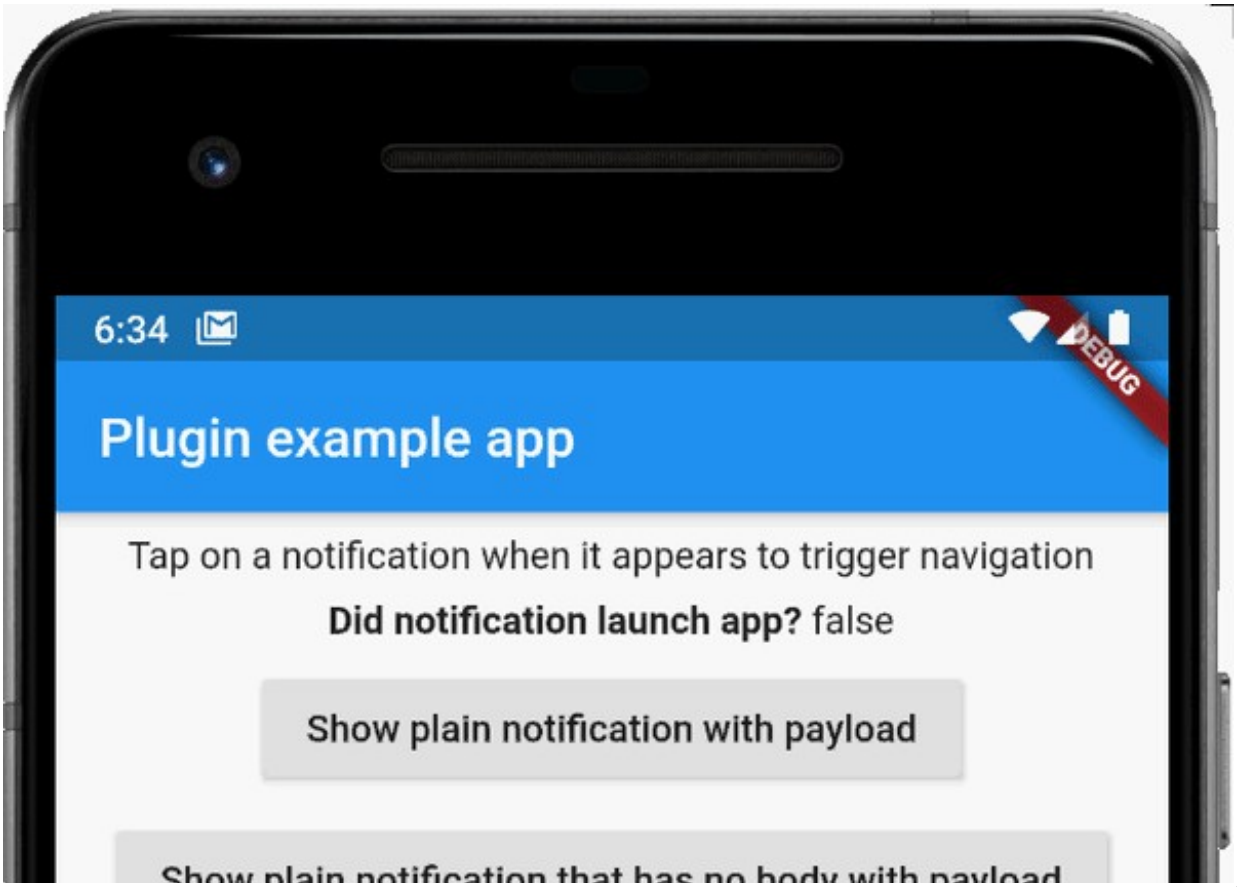


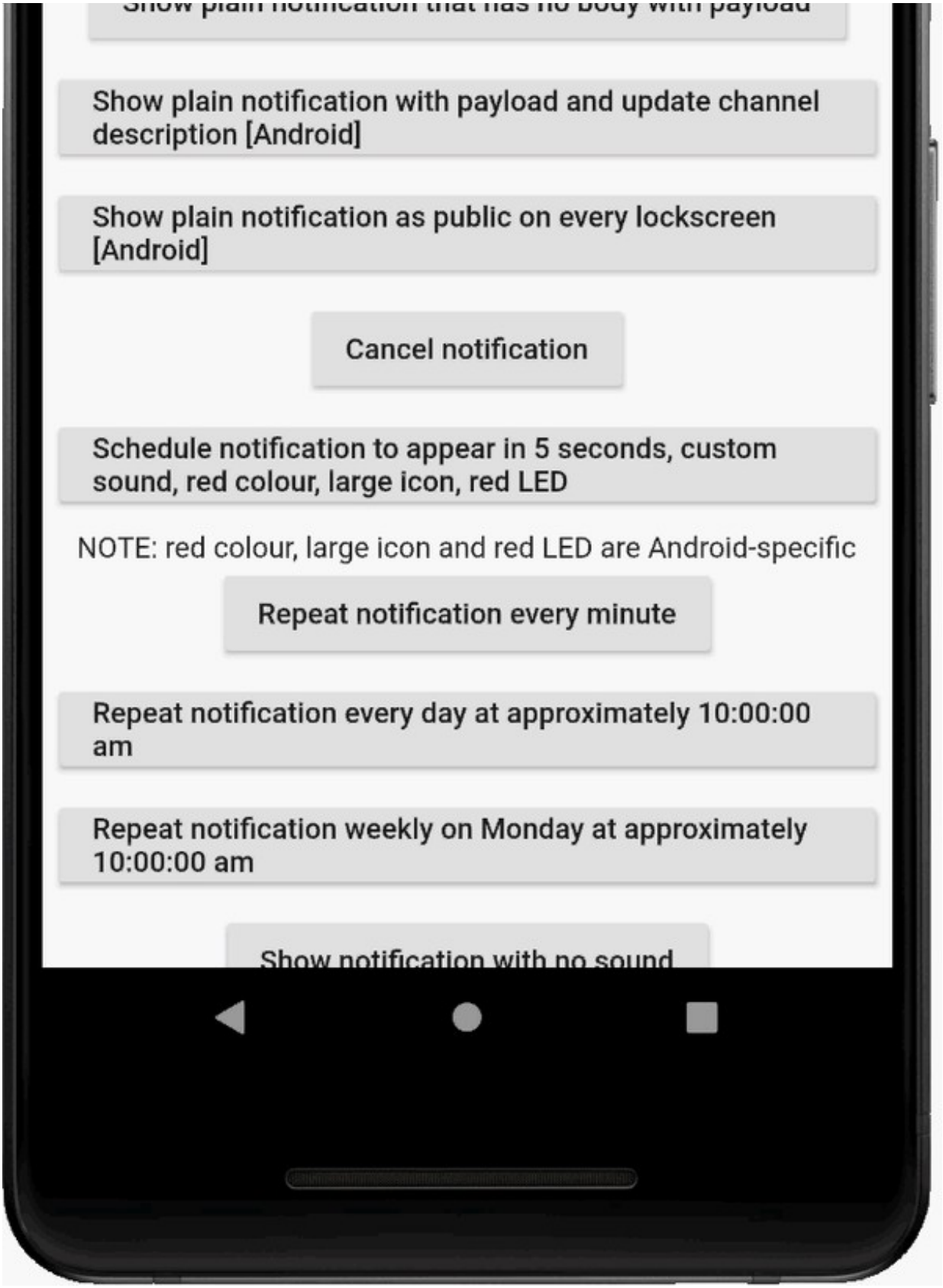


flutter_notifications_example.dart

The next feature supplied by the plugin and consequently this class allows you to schedule a Notification to appear sometime in the future. To do so, you would supply a DateTime object to the function, **schedule**().

```
        vibrationPattern[1] = 1000;
        vibrationPattern[2] = 5000;
        vibrationPattern[3] = 2000;

        var id = notifications.schedule(
          DateTime.now().add(Duration(seconds: 5)),
          title: 'scheduled title',
          body: 'scheduled body',
          sound: RawResourceAndroidNotificationSound(
              'slow_spring_board'), // RawResourceAndroidNotificat
          largeIcon:
              DrawableResourceAndroidBitmap('sample_large_icon'),
          vibrationPattern: vibrationPattern,
          enableLights: true,
          color: const Color.fromARGB(255, 255, 0, 0),
          ledColor: const Color.fromARGB(255, 255, 0, 0),
          ledOnMs: 1000,
          ledOffMs: 500,
          soundFile: 'slow_spring_board.aiff',
        );
        return id;
      },
    ), // _PaddedRaisedButton
    Text(
        'NOTE: red colour, large icon and red LED are Android-spec
```





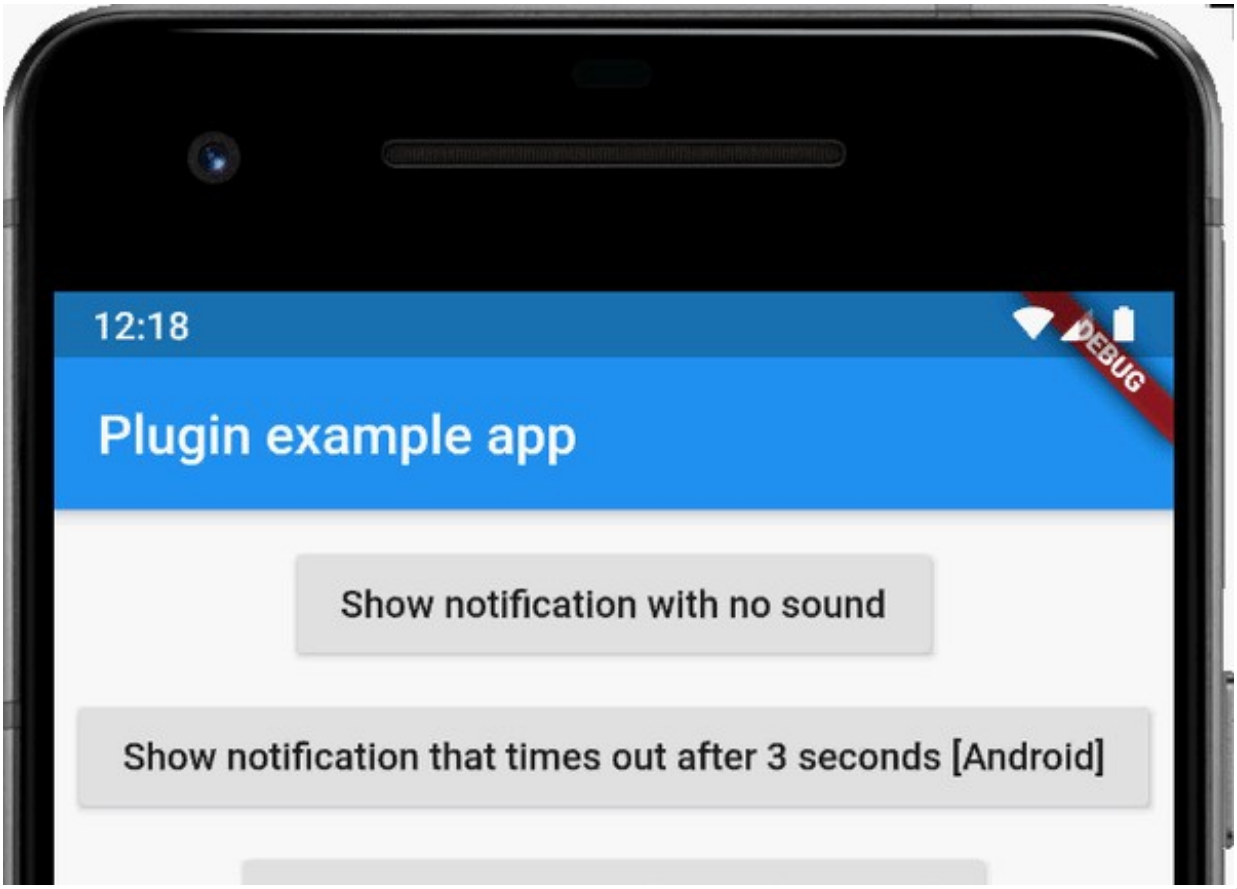
flutter_notifications_example.dart

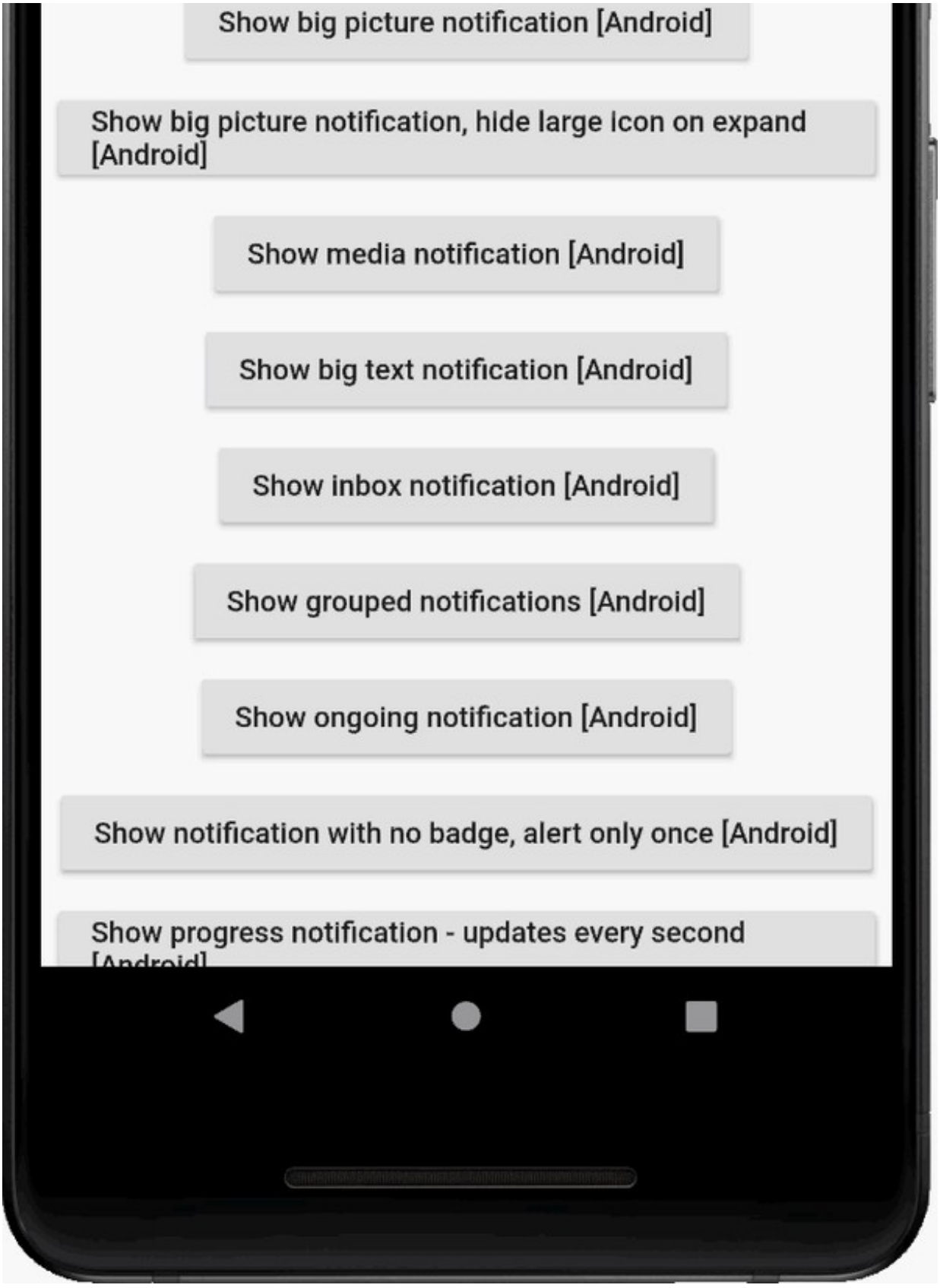
There may be circumstances you wish to present a notification at equal time intervals. The code below displays the notification every minute using the function, **periodicallyShow**().

```
        'NOTE: red colour, large icon and red LED are Android-spec
    _PaddedRaisedButton(
      buttonText: 'Repeat notification every minute',
      onPressed: () => notifications.periodicallyShow(
        RepeatInterval.EveryMinute,
        id: 0,
        title: 'repeating title',
        body: 'repeating body',
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
          'Repeat notification every day at approximately 10:00:00
      onPressed: () => notifications.showDailyAtTime(
        Time(15, 0, 0),
        id: 0,
        title: 'show daily title',
        body:
            'Daily notification shown at approximately ${Time(15,
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
          'Repeat notification weekly on Monday at approximately 1
      onPressed: () => notifications.showWeeklyAtDayAndTime(
        Day.Monday,
```

flutter_notifications_example.dart

```
      onPressed: () => notifications.showWeeklyAtDayAndTime(
        Day.Monday,
        Time(10, 0, 0),
        id: 0,
        title: 'show weekly title',
        body:
            'Weekly notification shown on Monday at approximately
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText: 'Show notification with no sound',
      onPressed: () => notifications.show(
        title: '<b>silent</b> title',
        body: '<b>silent</b> body',
        playSound: false,
        styleInformation: DefaultStyleInformation(true, true),
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
          'Show notification that times out after 3 seconds [Andro
      onPressed: () => notifications.show(
        timeoutAfter: 3000,
        styleInformation: DefaultStyleInformation(true, true),
        title: 'timeout notification',
```

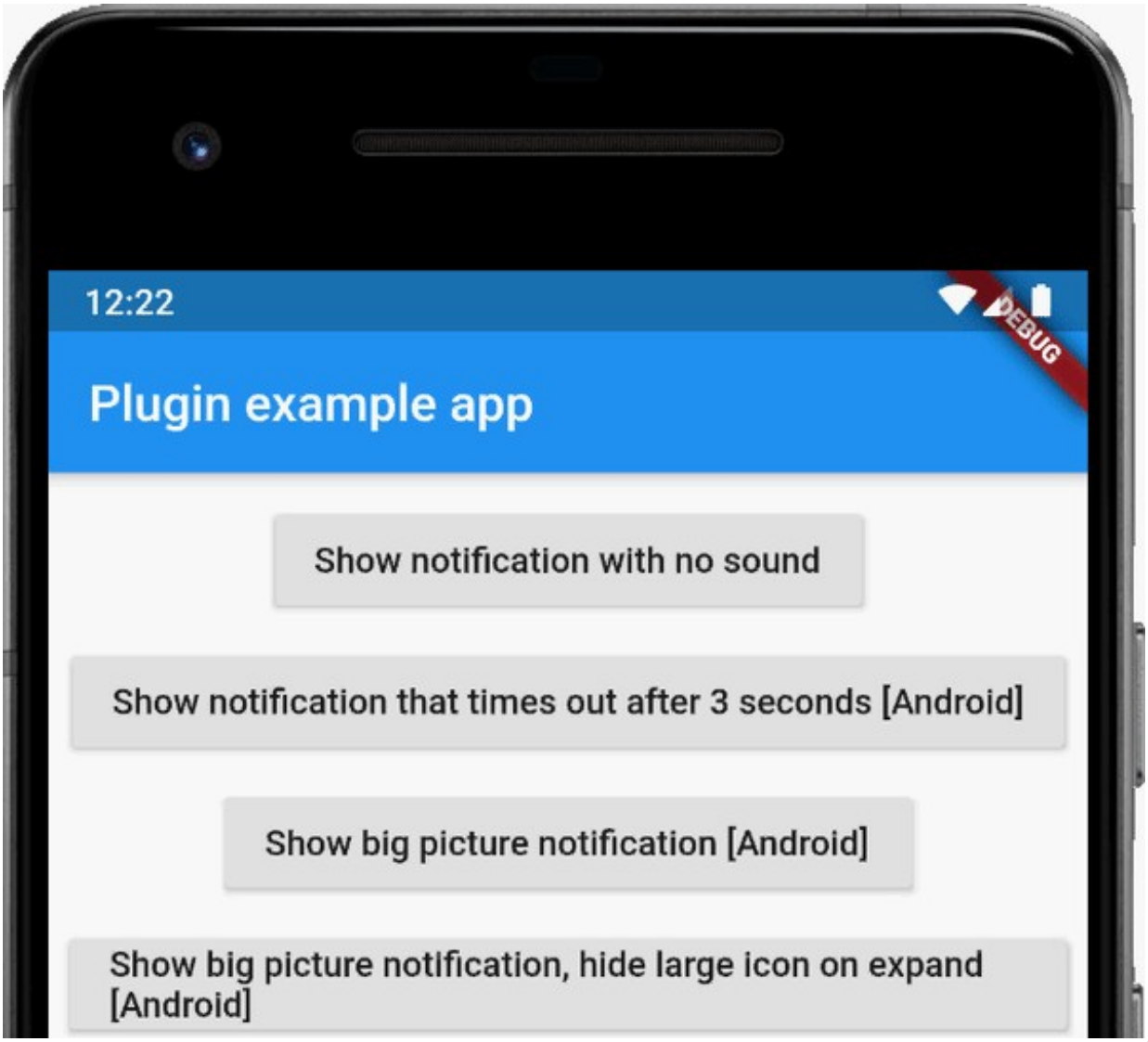


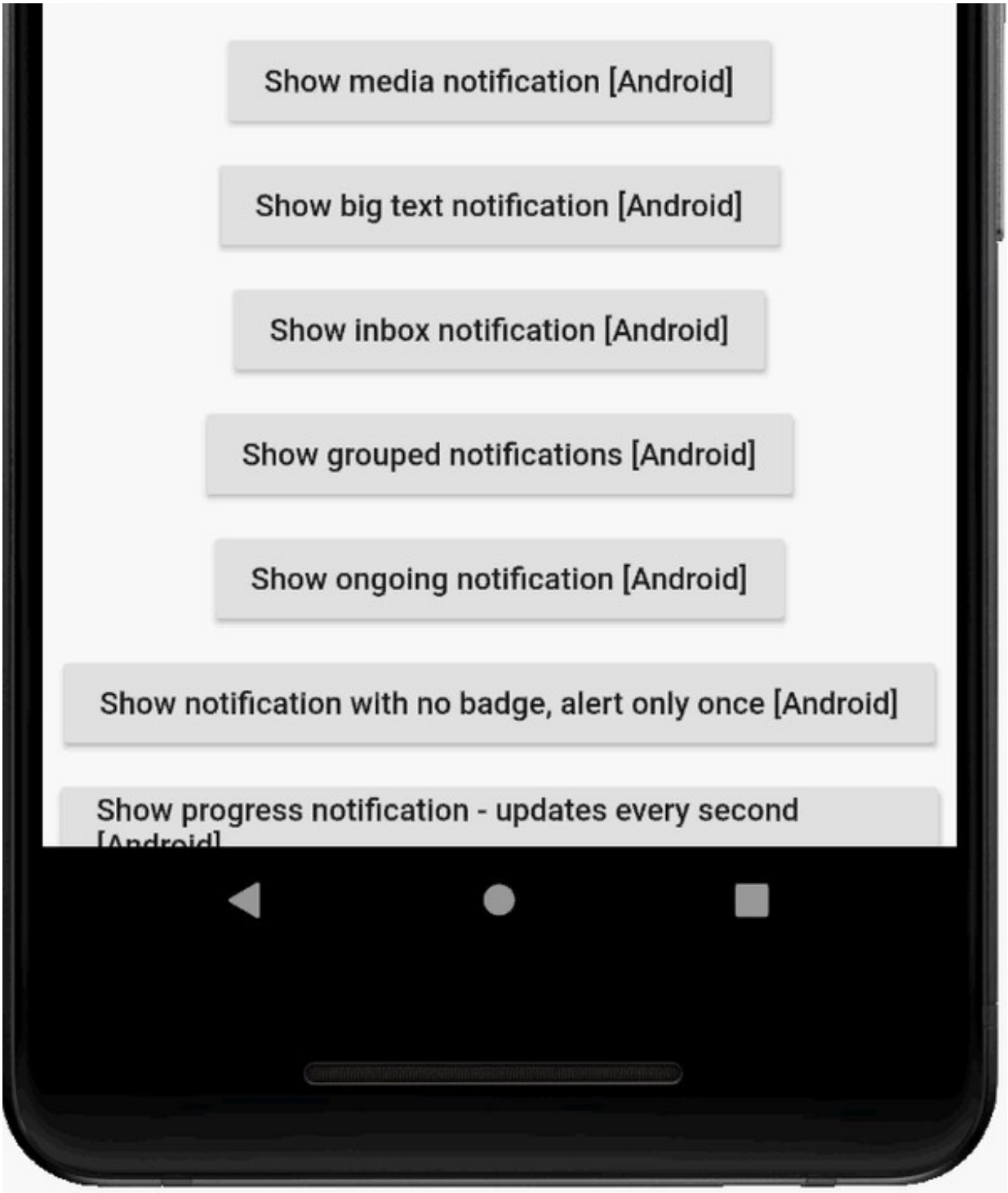


flutter_notifications_example.dart

```
      body: 'Times out after 3 seconds',
      payload: 'item x',
    ),
  ), // _PaddedRaisedButton
  _PaddedRaisedButton(
    buttonText: 'Show big picture notification [Android]'
```

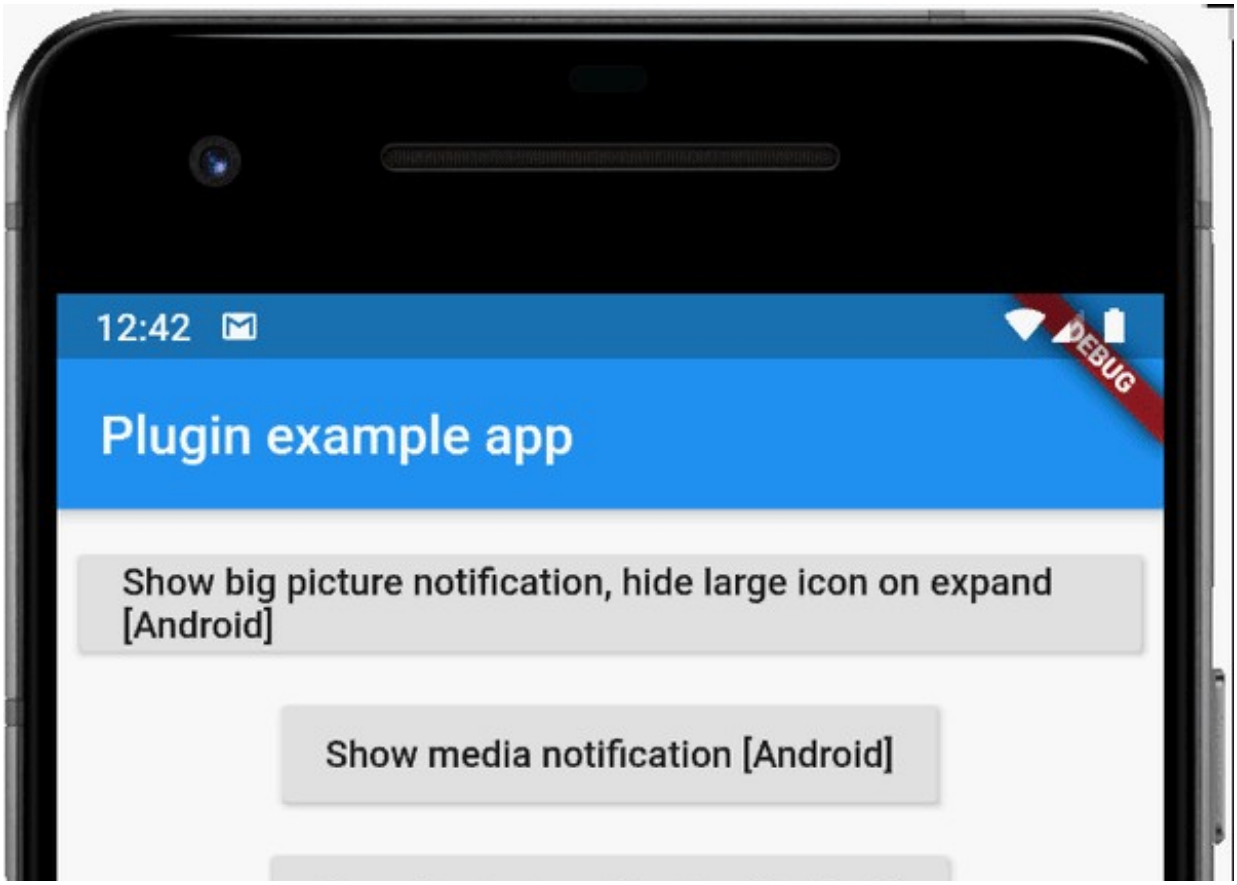
```
onPressed: () async {
  //
  var largeIconPath = await _downloadAndSaveFile(
      'http://via.placeholder.com/48x48', 'largeIcon');

  var bigPicturePath = await _downloadAndSaveFile(
      'http://via.placeholder.com/400x800', 'bigPicture');

  var bigPictureStyleInformation =
      BigPictureStyleInformation(
          FilePathAndroidBitmap(bigPicturePath),
          largeIcon: FilePathAndroidBitmap(largeIconPath),
          contentTitle:
              'overridden <b>big</b> content title',
          htmlFormatContentTitle: true,
          summaryText: 'summary <i>text</i>',
          htmlFormatSummaryText: true); // BigPictureStyleIn

  notifications.show(
```

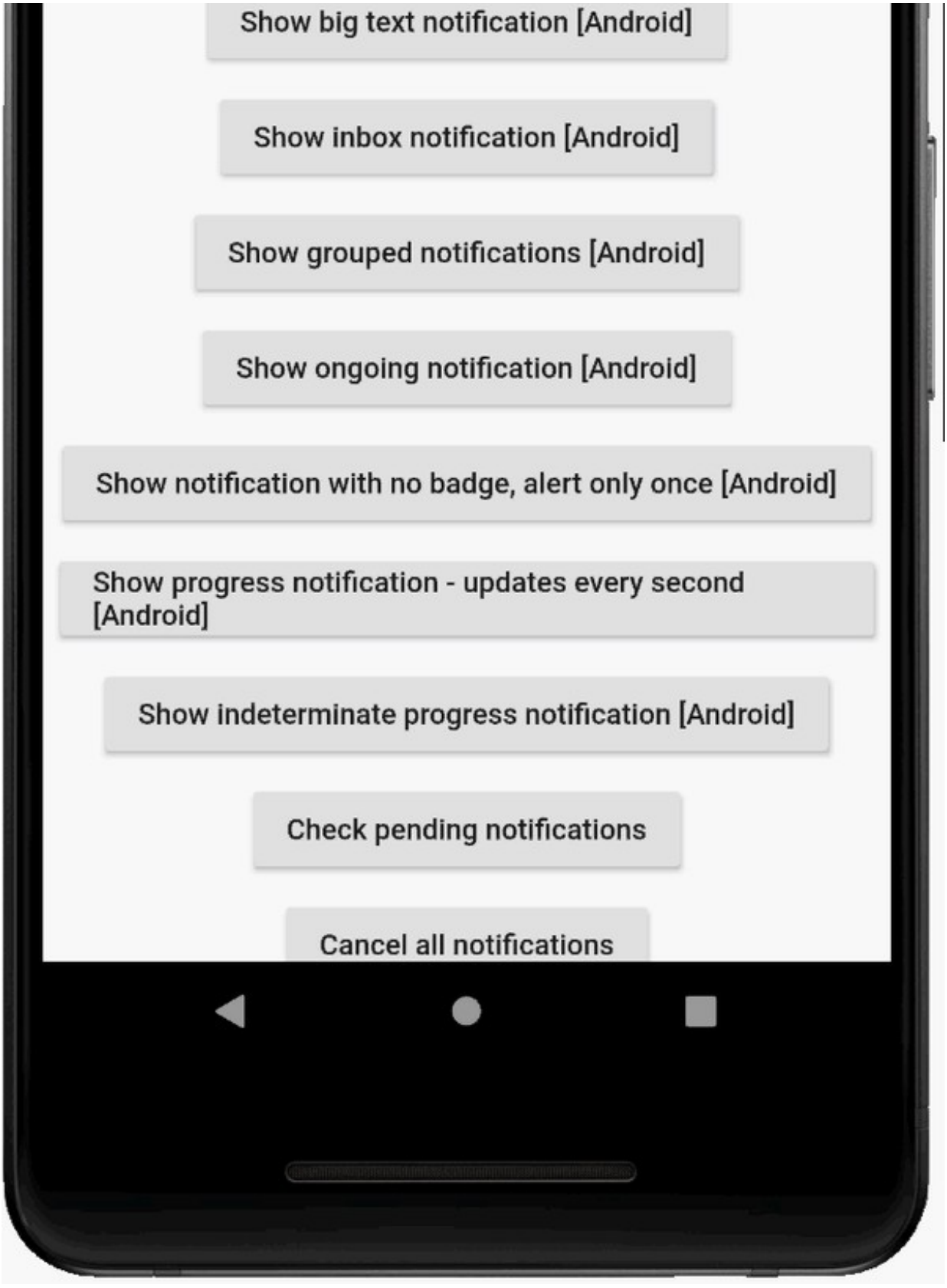




flutter_notifications_example.dart

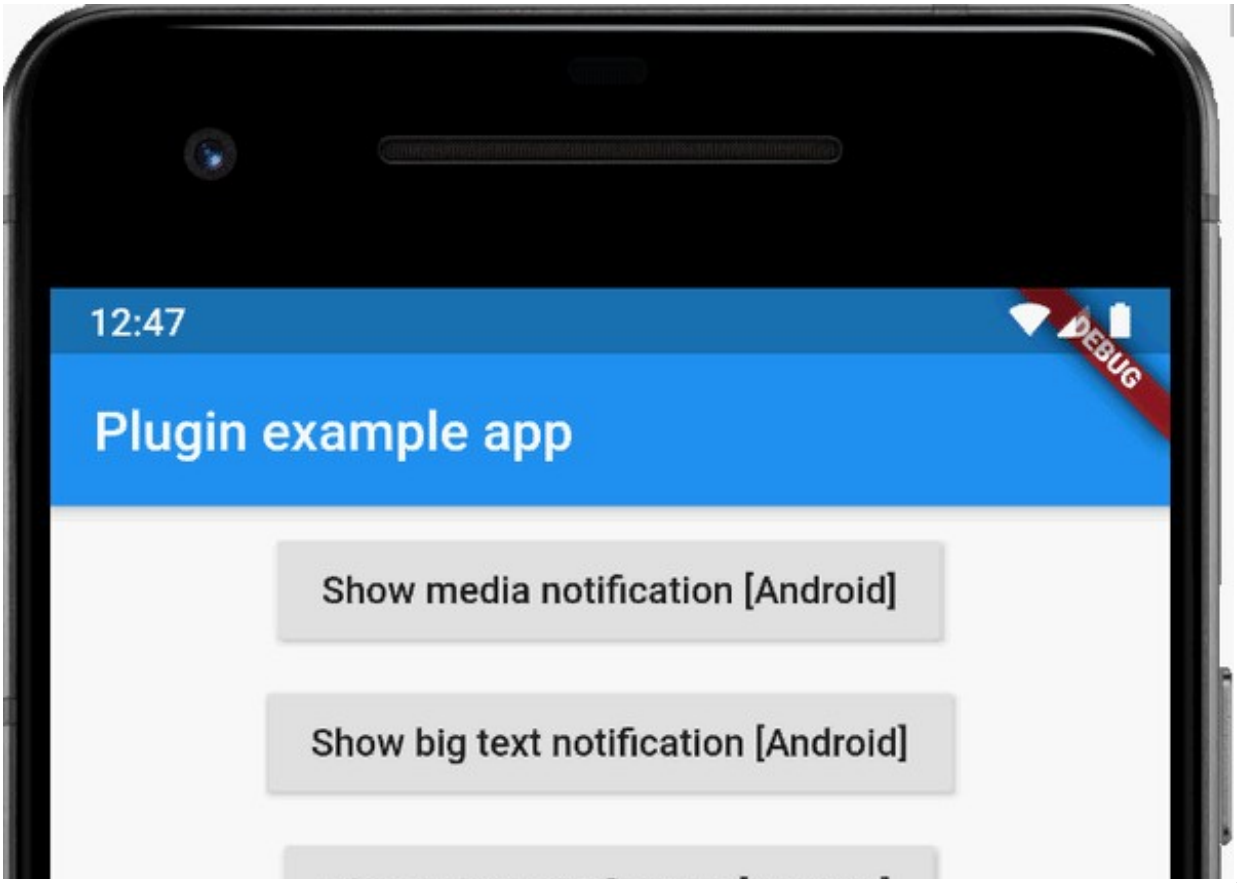
```
      notifications.show(
        id: 0,
        title: 'big text title',
        body: 'silent body',
        styleInformation: bigPictureStyleInformation,
      );
    },
  ), // _PaddedRaisedButton
  _PaddedRaisedButton(
    buttonText:
        'Show big picture notification, hide large icon on expar
    onPressed: () async {
      //
      var largeIconPath = await _downloadAndSaveFile(
          'http://via.placeholder.com/48x48', 'largeIcon');

      var bigPicturePath = await _downloadAndSaveFile(
          'http://via.placeholder.com/400x800', 'bigPicture');

      var bigPictureStyleInformation =
          BigPictureStyleInformation(
              FilePathAndroidBitmap(bigPicturePath),
              hideExpandedLargeIcon: true,
              contentTitle:
                  'overridden <b>big</b> content title',
```

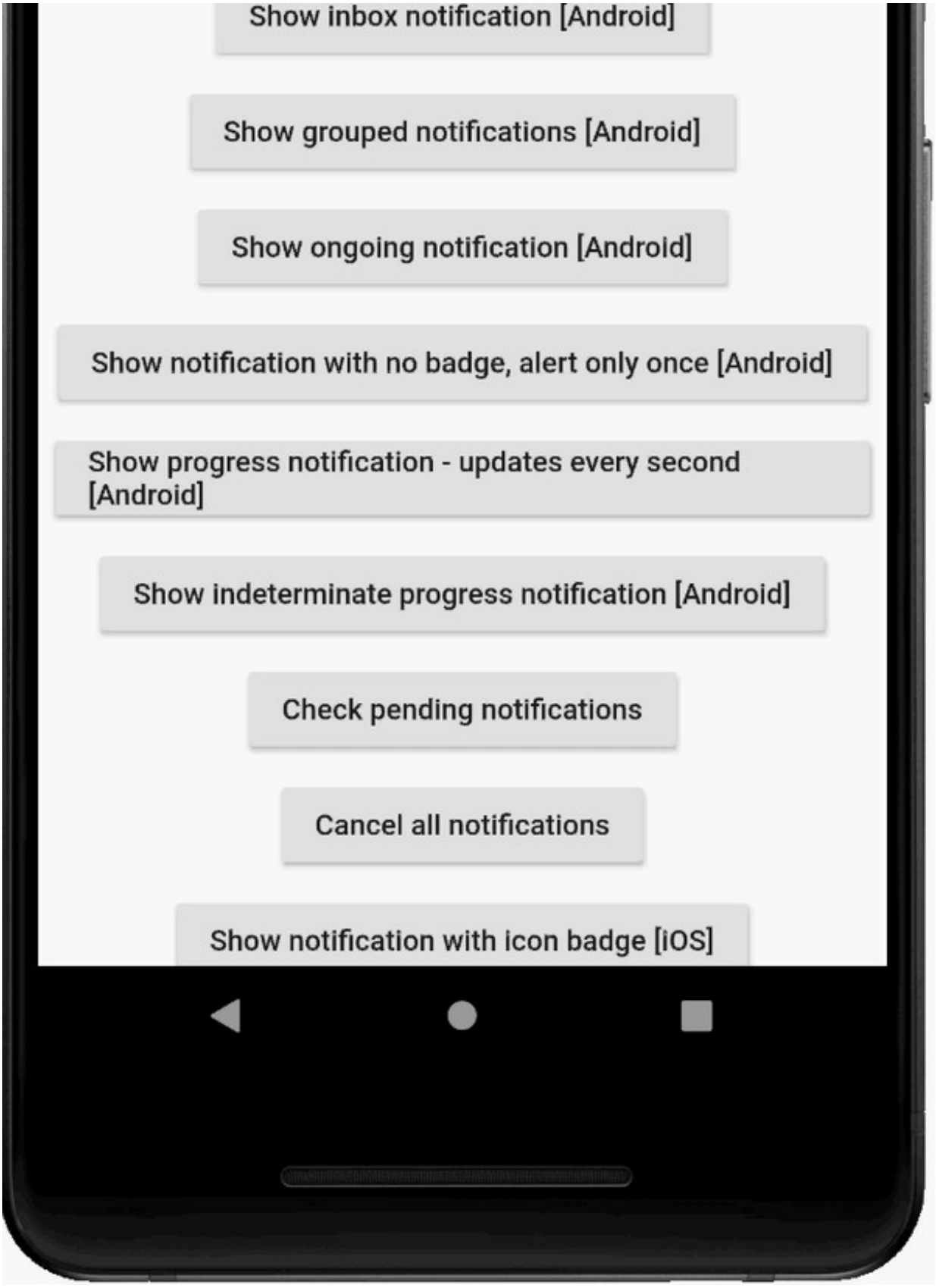




flutter_notifications_example.dart

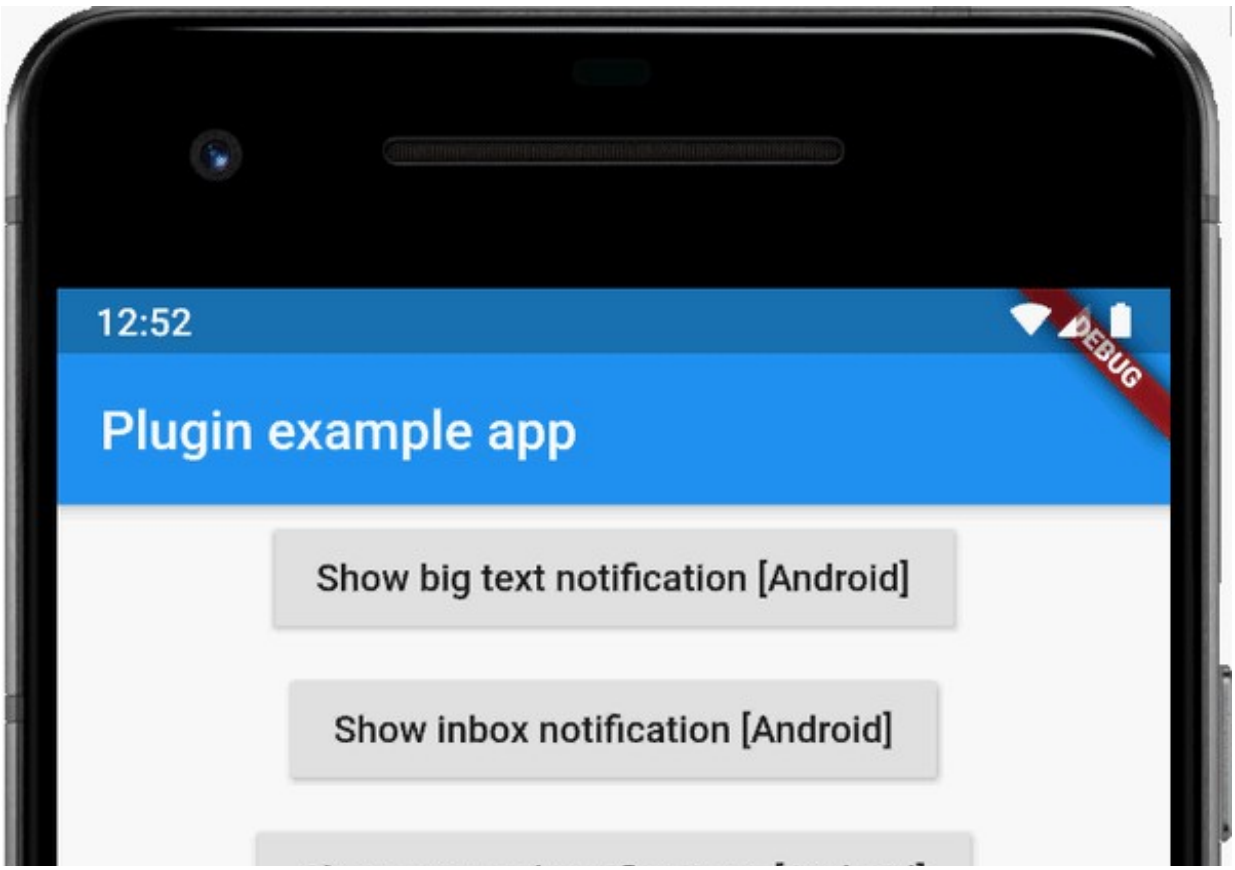
```
                htmlFormatContentTitle: true,
                summaryText: 'summary <i>text</i>',
                htmlFormatSummaryText: true); // BigPictureStyleIn

            notifications.show(
              id: 0,
              title: 'big text title',
              body: 'silent body',
              largeIcon: FilePathAndroidBitmap(largeIconPath),
              styleInformation: bigPictureStyleInformation,
            );
          },
        ), // _PaddedRaisedButton
        _PaddedRaisedButton(
          buttonText: 'Show media notification [Android]',
          onPressed: () async {
//
            var largeIconPath = await _downloadAndSaveFile(
                'http://via.placeholder.com/128x128/00FF00/000000',
                'largeIcon');

            notifications.show(
              id: 0,
              title: 'notification title',
              body: 'notification body',
```

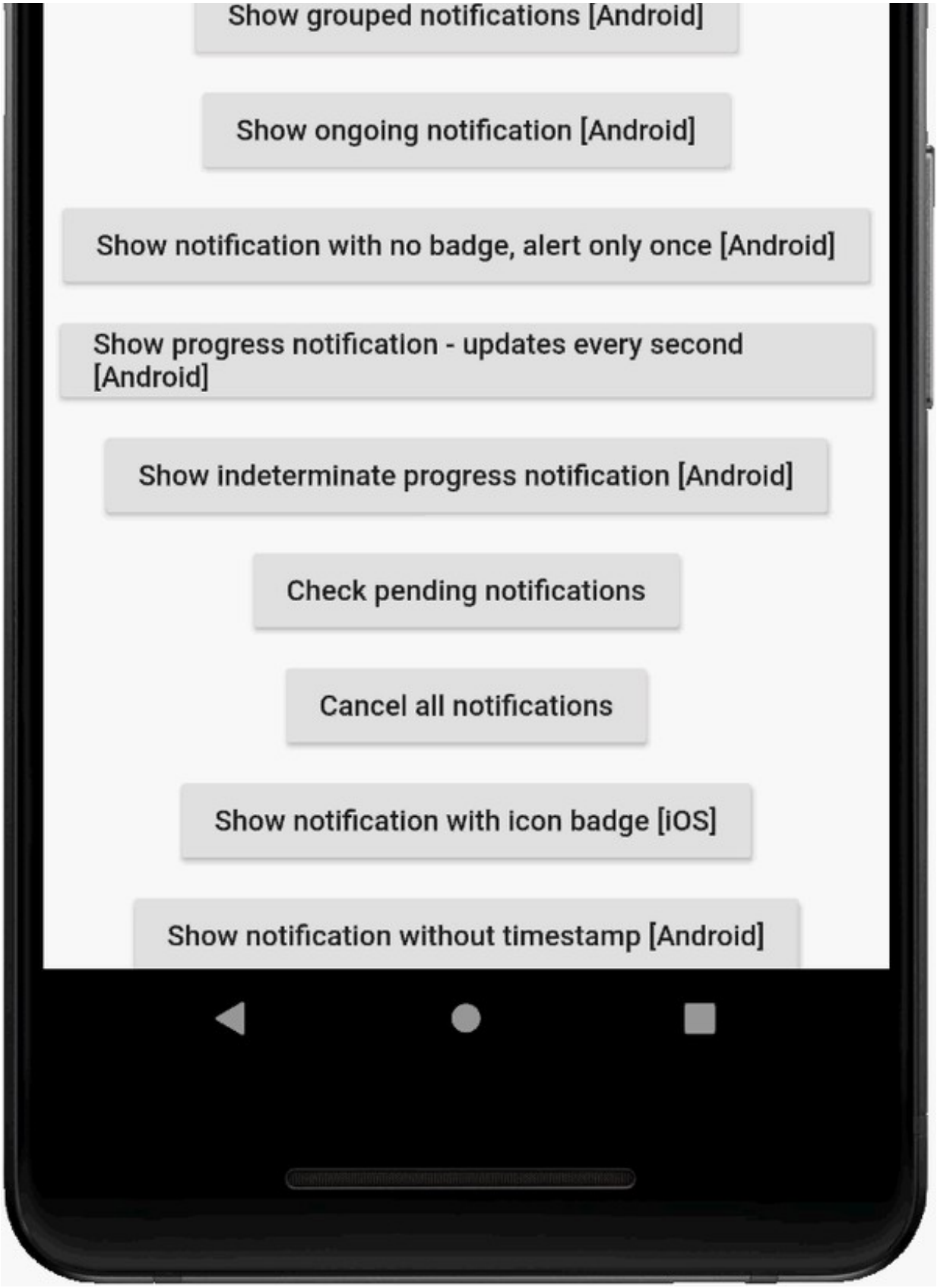




flutter_notifications_example.dart

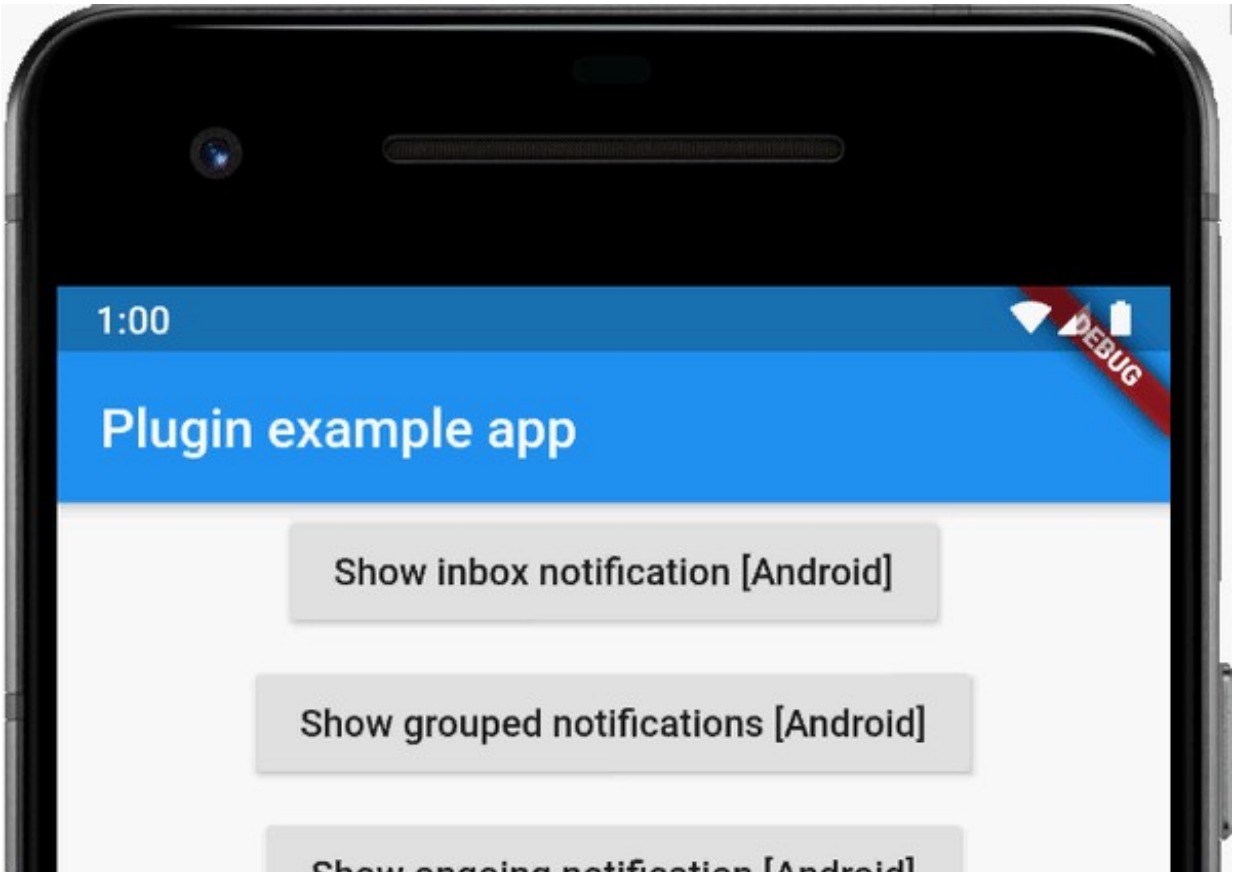
```
        largeIcon: FilePathAndroidBitmap(largeIconPath),
        styleInformation: MediaStyleInformation(),
      );
    },
  ), // _PaddedRaisedButton
  _PaddedRaisedButton(
    buttonText: 'Show big text notification [Android]',
    onPressed: () async {
      //
      var bigTextStyleInformation = BigTextStyleInformation(
          'Lorem <i>ipsum dolor sit</i> amet, consectetur <b>adi
          htmlFormatBigText: true,
          contentTitle: 'overridden <b>big</b> content title',
          htmlFormatContentTitle: true,
          summaryText: 'summary <i>text</i>',
          htmlFormatSummaryText: true); // BigTextStyleInformati

      notifications.show(
        id: 0,
        title: 'notification title',
        body: 'notification body',
        styleInformation: bigTextStyleInformation,
      );
    },
  ), // _PaddedRaisedButton
```

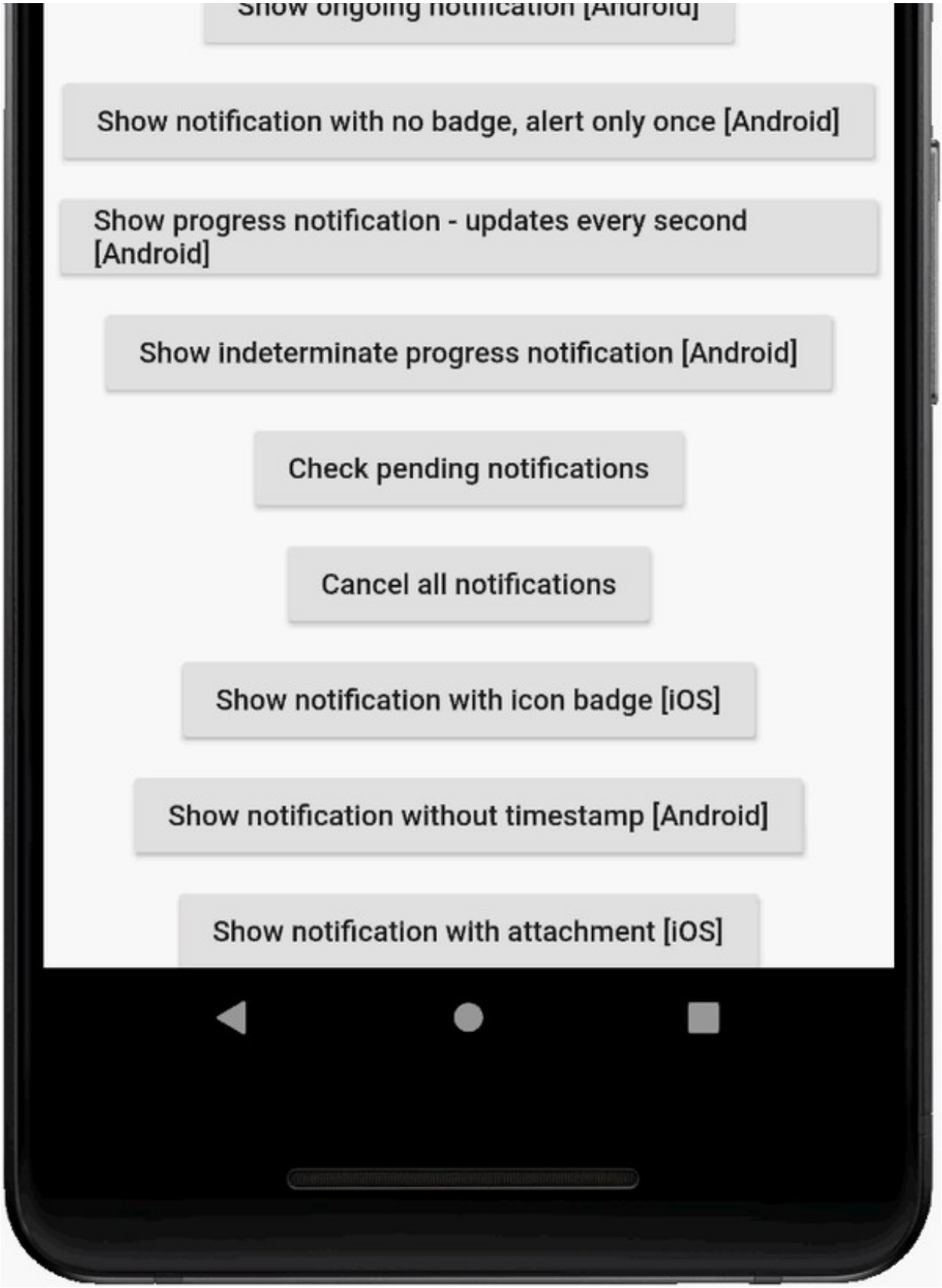




flutter_notifications_example.dart

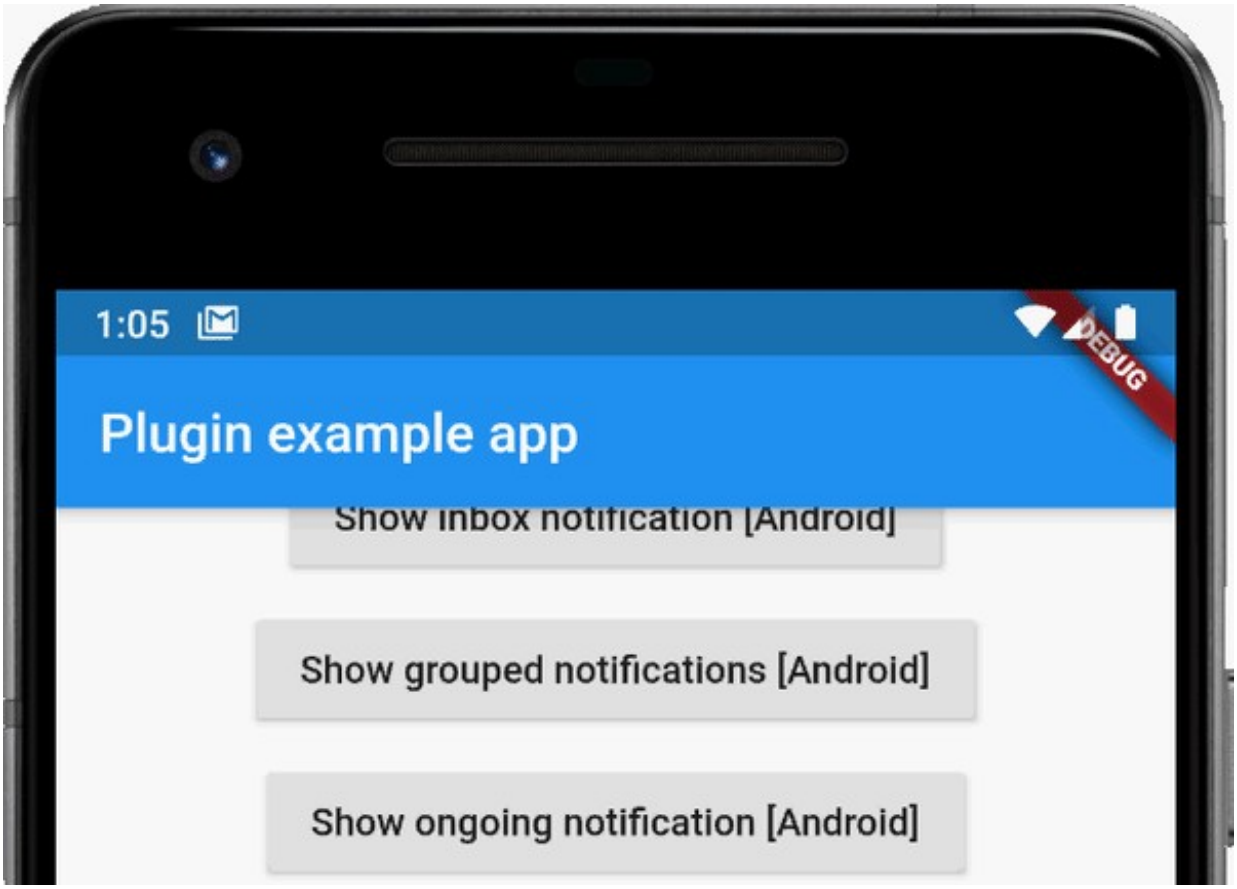
```
_PaddedRaisedButton(
  buttonText: 'Show inbox notification [Android]',
  onPressed: () async {
    //
    var lines = [];

    lines.add('line <b>1</b>');

    lines.add('line <i>2</i>');

    var inboxStyleInformation = InboxStyleInformation(lines,
        htmlFormatLines: true,
        contentTitle: 'overridden <b>inbox</b> context title',
        htmlFormatContentTitle: true,
        summaryText: 'summary <i>text</i>',
        htmlFormatSummaryText: true); // InboxStyleInformation

    notifications.show(
      id: 0,
      title: 'notification title',
      body: 'notification body',
      styleInformation: inboxStyleInformation,
    );
  },
), // _PaddedRaisedButton
```

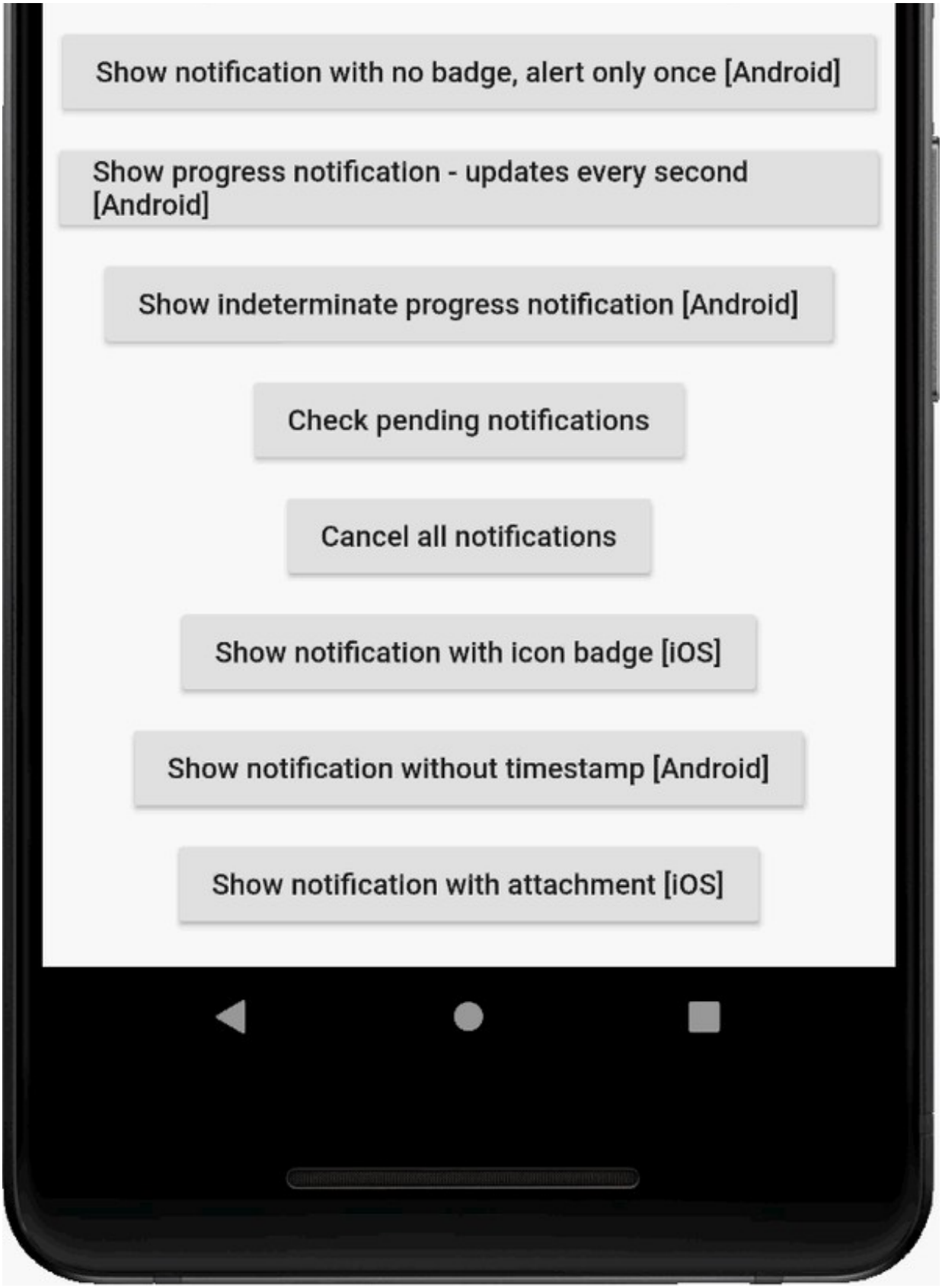




flutter_notifications_example.dart

```
), // _PaddedRaisedButton
_PaddedRaisedButton(
  buttonText: 'Show grouped notifications [Android]',
  onPressed: () async {
    //
    var groupKey = 'com.android.example.WORK_EMAIL';

    // example based on https://developer.android.com/training

    notifications.show(
      id: 1,
      title: 'Alex Faarborg',
      body: 'You will not believe...',
      importance: Importance.Max,
      priority: Priority.High,
      groupKey: groupKey,
    );

    notifications.show(
      id: 2,
      title: 'Jeff Chang',
      body: 'Please join us to celebrate the...',
      importance: Importance.Max,
      priority: Priority.High,
      groupKey: groupKey,
```



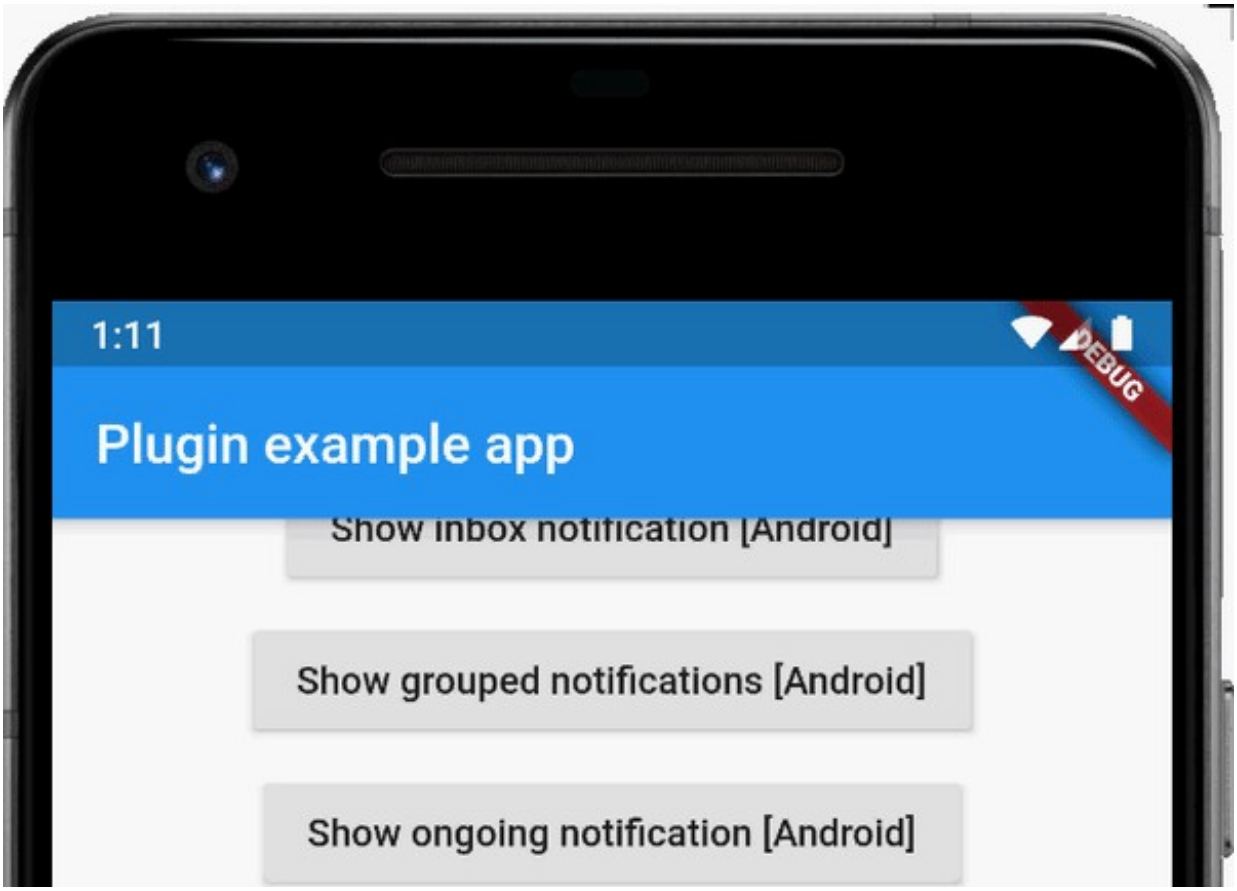


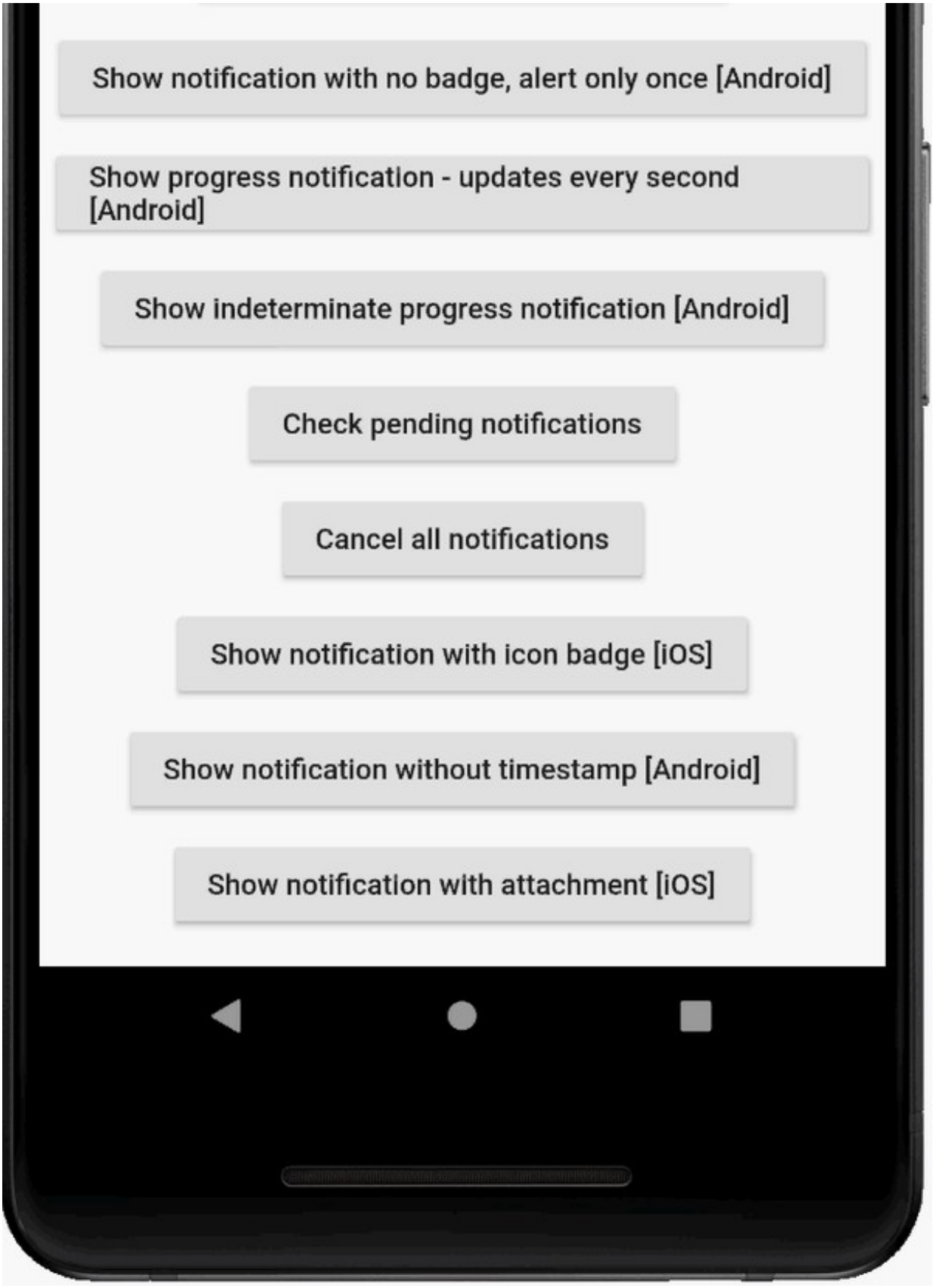
flutter_notifications_example.dart

```
        // create the summary notification to support older device
        // this is required is regardless of which versions of And
        var lines = List<String>();

        lines.add('Alex Faarborg  Check this out');

        lines.add('Jeff Chang    Launch Party');

        var inboxStyleInformation = InboxStyleInformation(lines,
            contentTitle: '2 messages',
            summaryText: 'janedoe@example.com'); // InboxStyleInfo

        notifications.show(
          id: 0,
          title: 'group notification title',
          body: 'group notification body',
          styleInformation: inboxStyleInformation,
          groupKey: groupKey,
          setAsGroupSummary: true,
        );
      },
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText: 'Show ongoing notification [Android]',
      onPressed: () => notifications.show(
```

flutter_notifications_example.dart

```
      onPressed: () => notifications.show(
          id: 0,
          title: 'ongoing notification title',
          body: 'ongoing notification body',
          importance: Importance.Max,
          priority: Priority.High,
          ongoing: true,
          autoCancel: false),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
          'Show notification with no badge, alert only once [Andro
      onPressed: () => notifications.show(
        id: 0,
        channelShowBadge: false,
        importance: Importance.Max,
        priority: Priority.High,
        onlyAlertOnce: true,
        title: 'no badge title',
        body: 'no badge body',
        payload: 'item x',
      ),
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText:
```

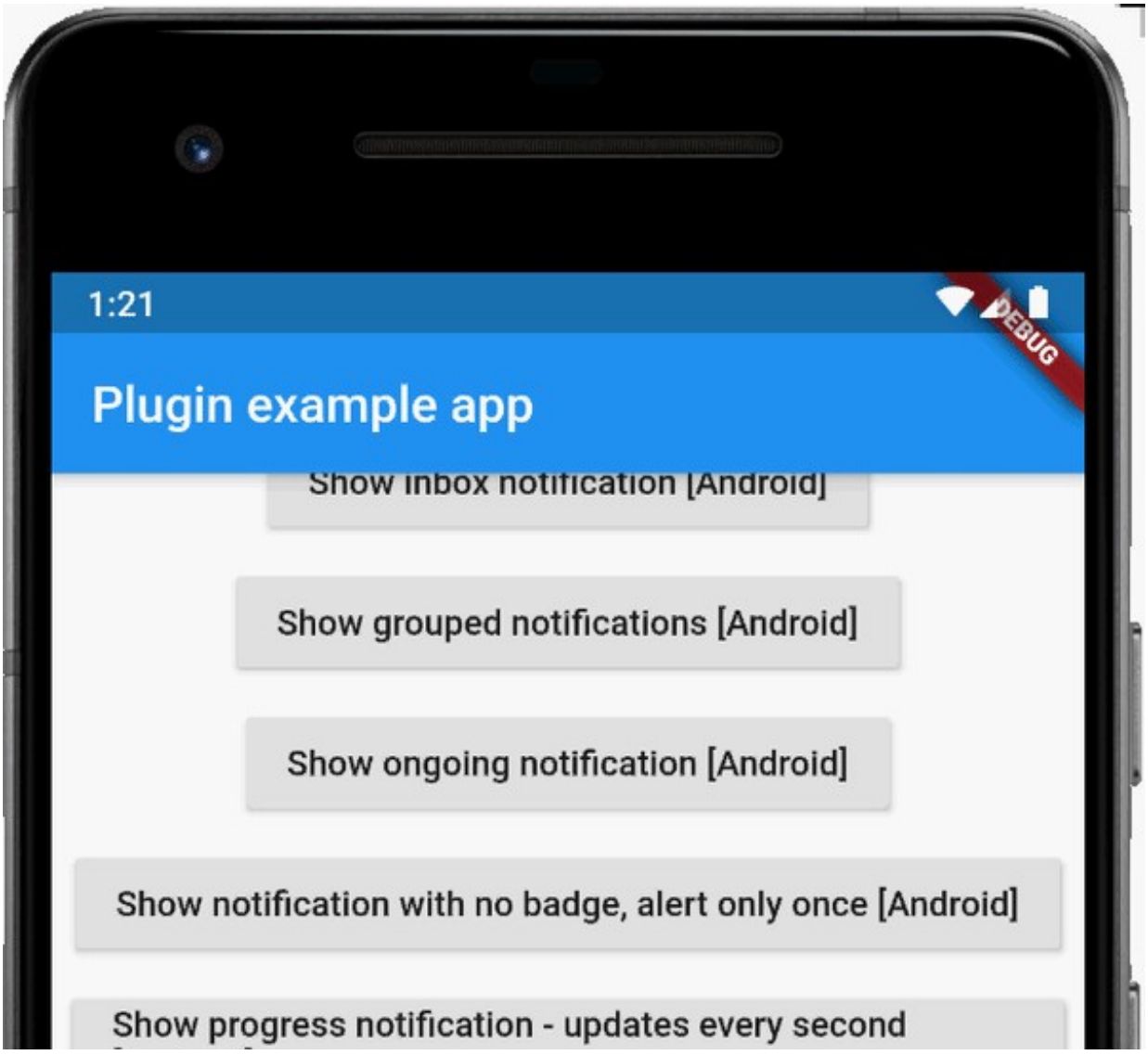


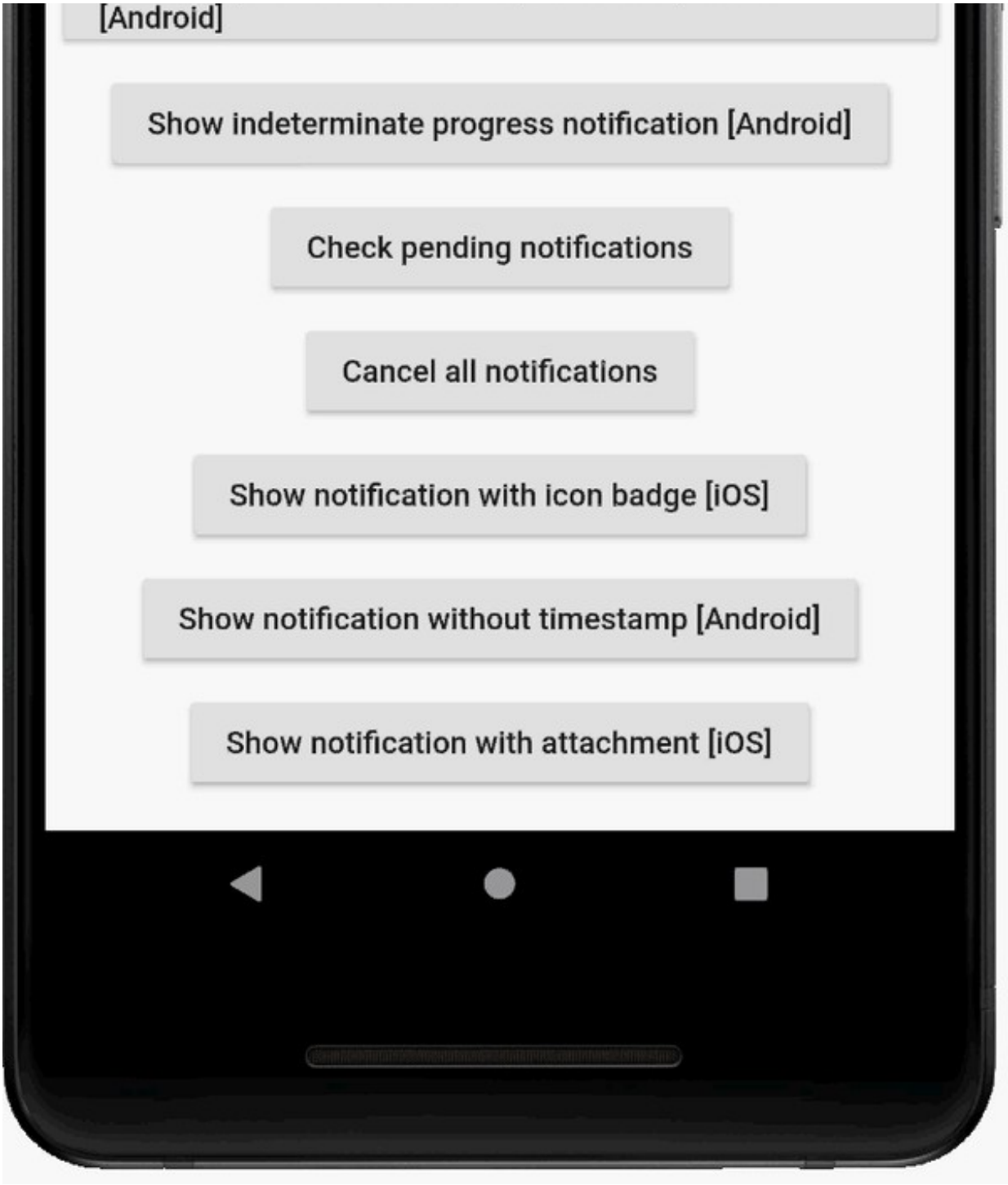


flutter_notifications_example.dart

```
buttonText:
    'Show progress notification - updates every second [Andr
onPressed: () async {
  var maxProgress = 5;

  for (var i = 0; i <= maxProgress; i++) {
```

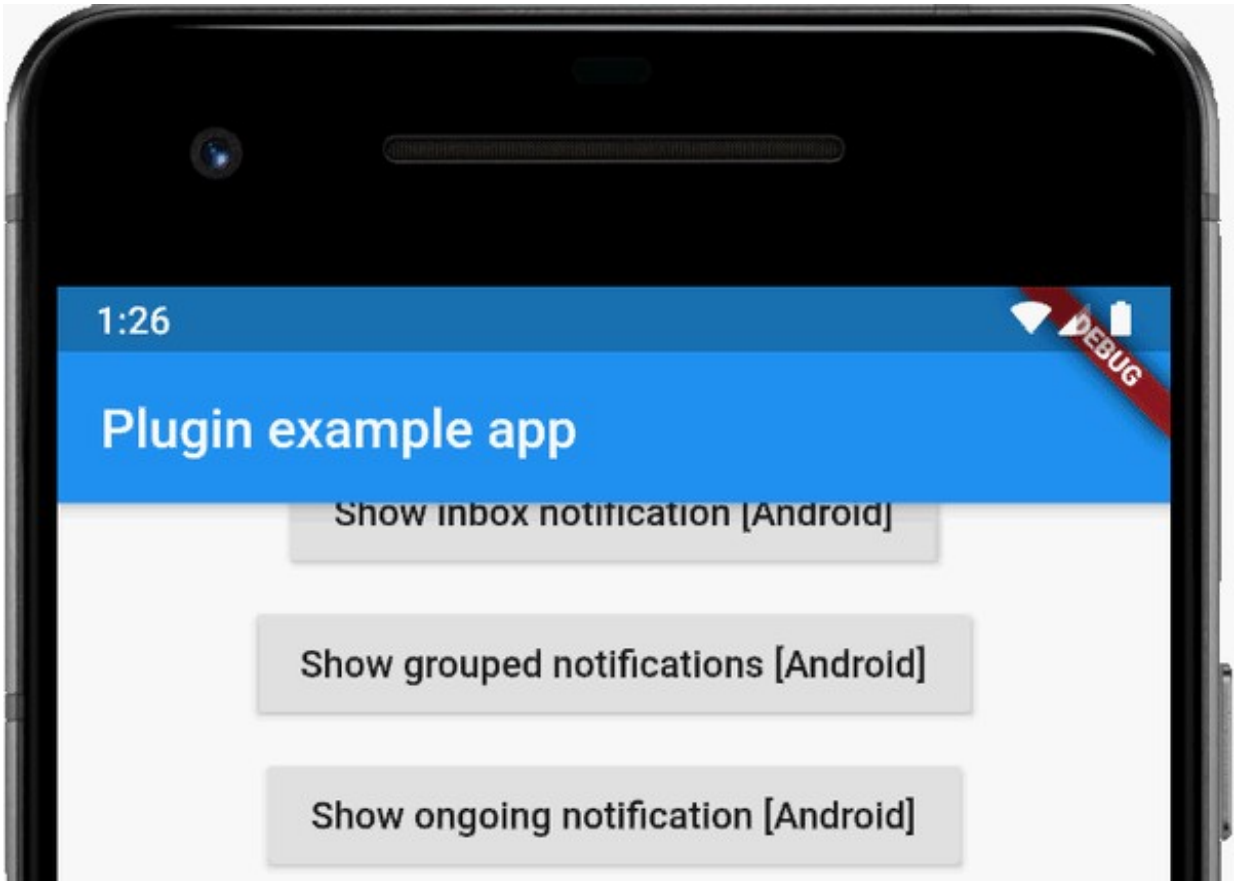
```
for (var i = 0; i <= maxProgress; i++) {
  await Future.delayed(Duration(seconds: 1), () async {
    notifications.show(
        id: 0,
        title: 'progress notification title',
        body: 'progress notification body',
        payload: 'item x',
        channelShowBadge: false,
        importance: Importance.Max,
        priority: Priority.High,
        onlyAlertOnce: true,
        showProgress: true,
        maxProgress: maxProgress,
        progress: i);
  }); // Future.delayed
 }
},
), // _PaddedRaisedButton
_PaddedRaisedButton(
    buttonText:
```

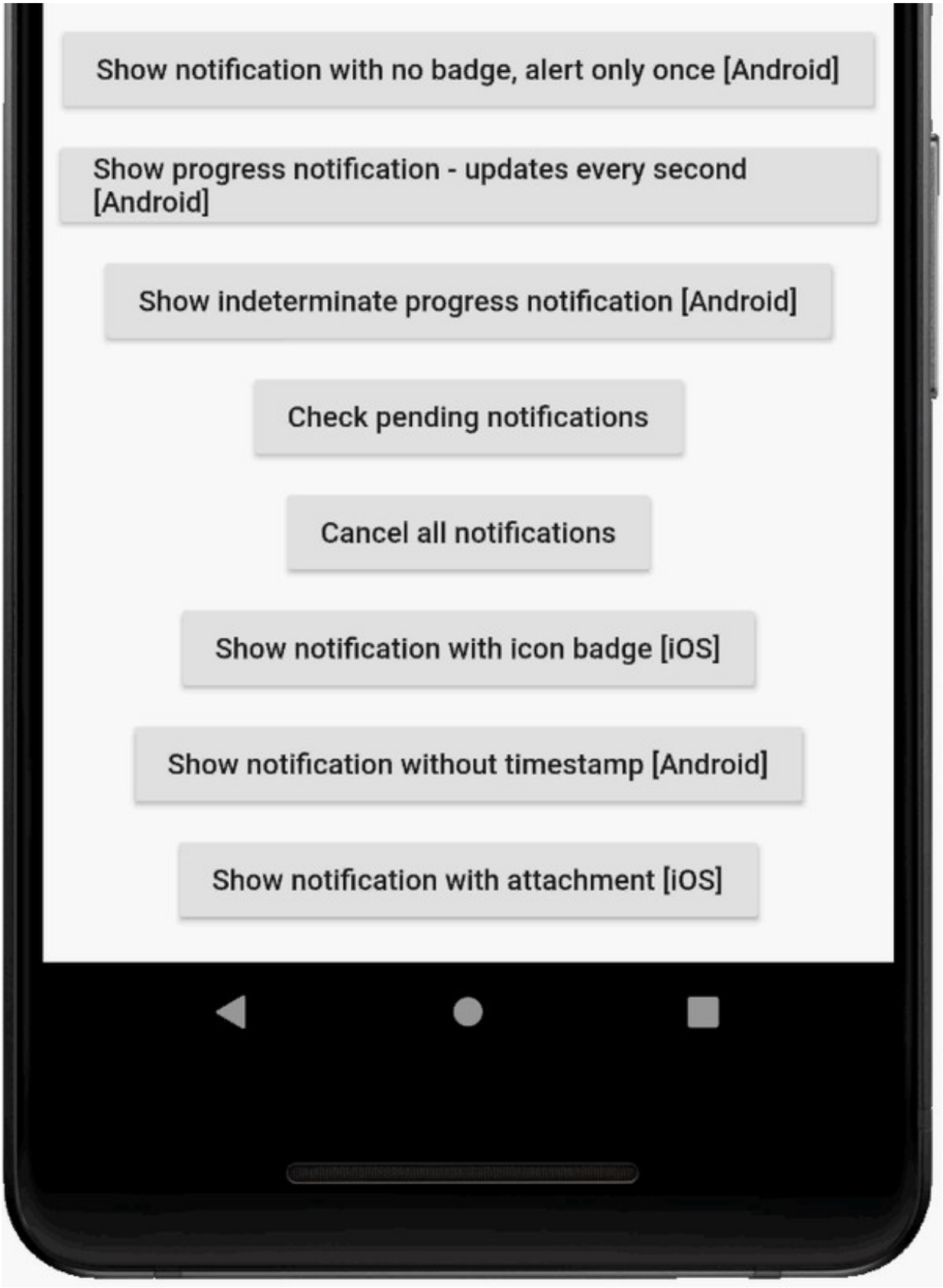




flutter_notifications_example.dart

```
        buttonText:
            'Show indeterminate progress notification [Android]',
        onPressed: () => notifications.show(
              id: 0,
              title: 'indeterminate progress notification title',
              body: 'indeterminate progress notification body',
              payload: 'item x',
              channelShowBadge: false,
              importance: Importance.Max,
              priority: Priority.High,
              onlyAlertOnce: true,
              showProgress: true,
              indeterminate: true,
            )), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText: 'Check pending notifications',
      onPressed: () async {
        //
        var pendingNotificationRequests =
            await notifications.pendingNotificationRequests();

        for (var pendingNotificationRequest
            in pendingNotificationRequests) {
          debugPrint(
              'pending notification: [id: ${pendingNotificationRec
```

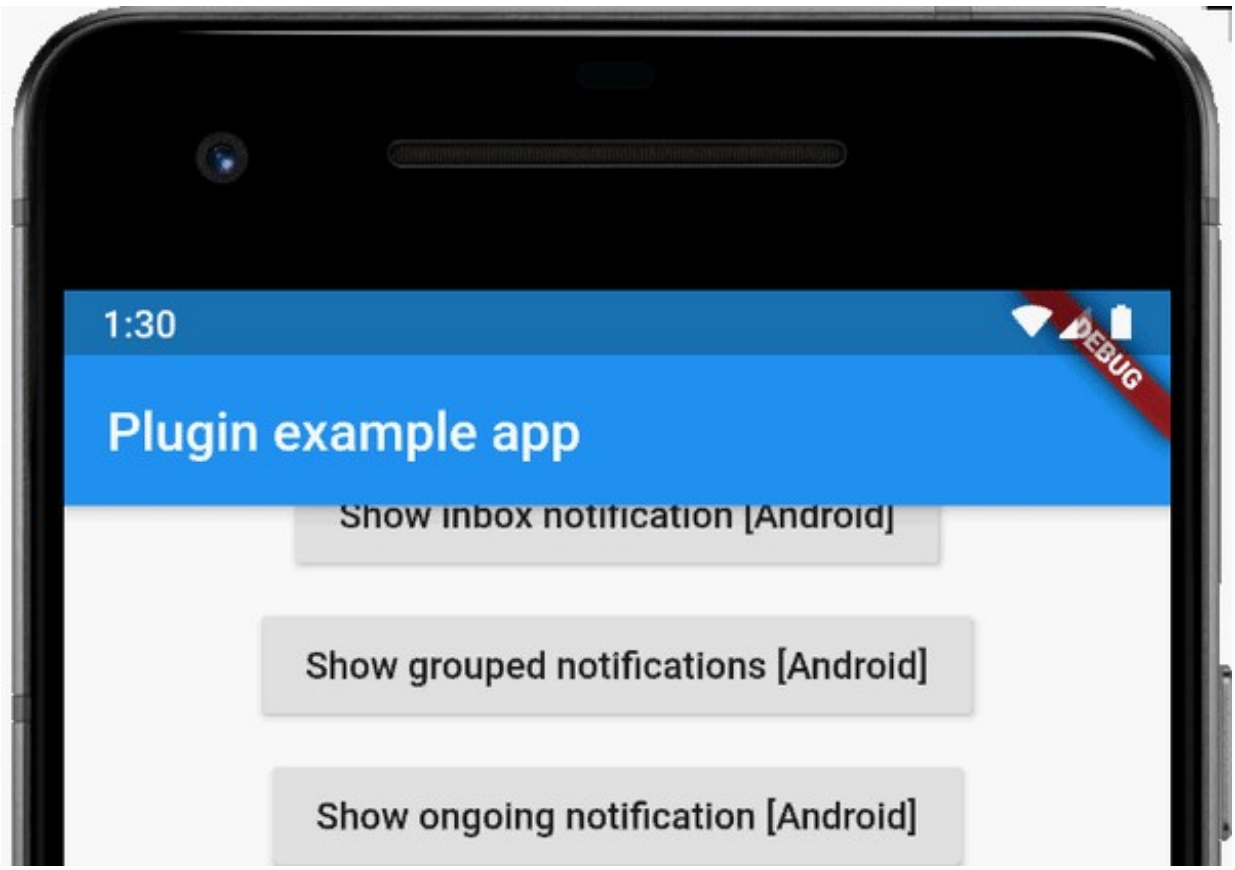


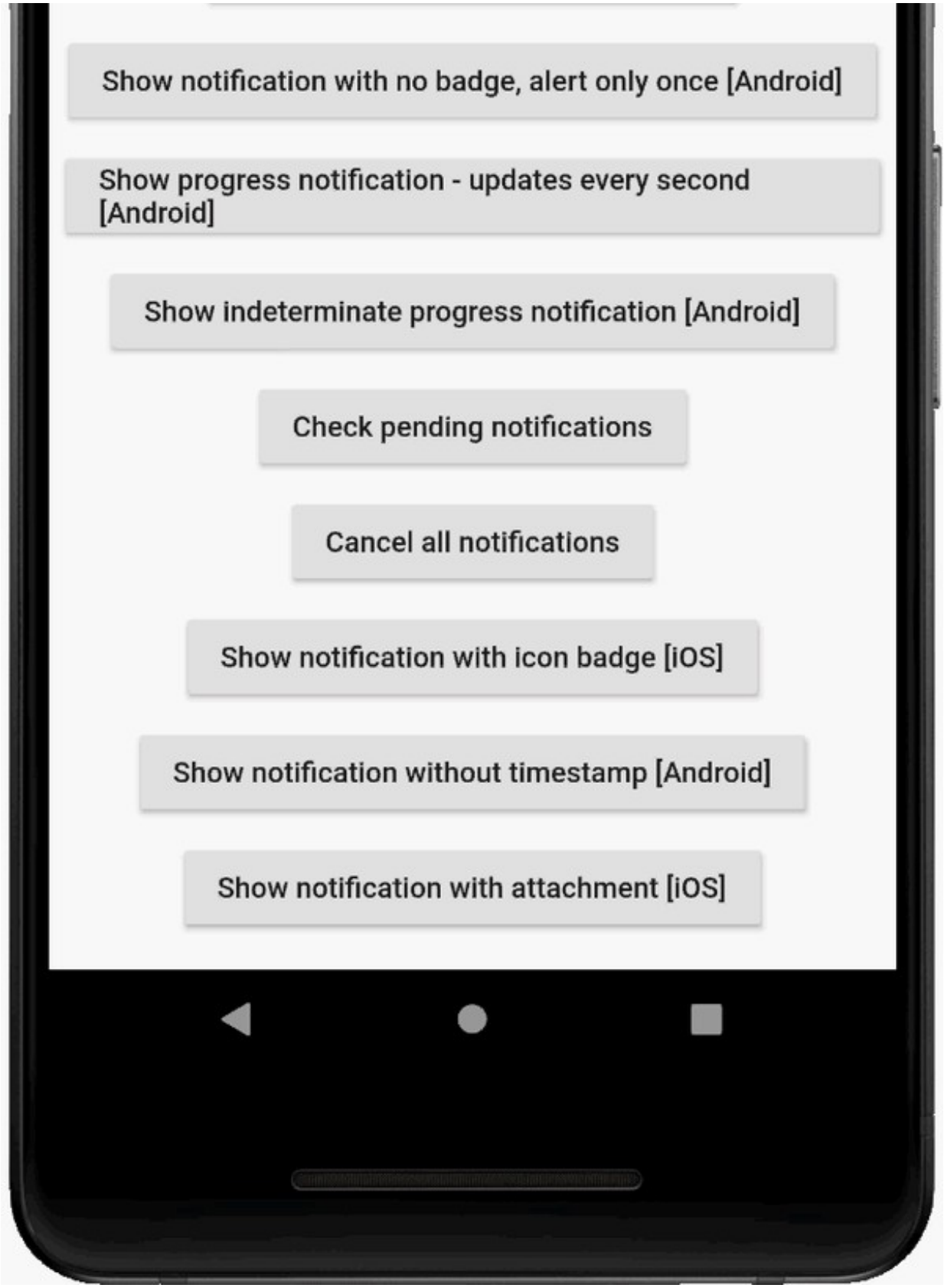


flutter_notifications_example.dart

The function, **pendingNotificationRequest**(), returns a List of objects describing the notifications pending for execution. Each object will have the following properties: *id*, *title*, *body*, and *payload*.

```
            'pending notification: [id: ${pendingNotificationRec
        }

        return showDialog<void>(
          context: context,
          builder: (BuildContext context) {
            return AlertDialog(
              content: Text(
                  '${pendingNotificationRequests.length} pending r
              actions: [
                FlatButton(
                  child: Text('OK'),
                  onPressed: () {
                    Navigator.of(context).pop();
                  },
                ), // FlatButton
              ],
            ); // AlertDialog
          },
        );
      },
    ), // _PaddedRaisedButton
    _PaddedRaisedButton(
      buttonText: 'Cancel all notifications',
      onPressed: () => notifications.cancelAll(),
```

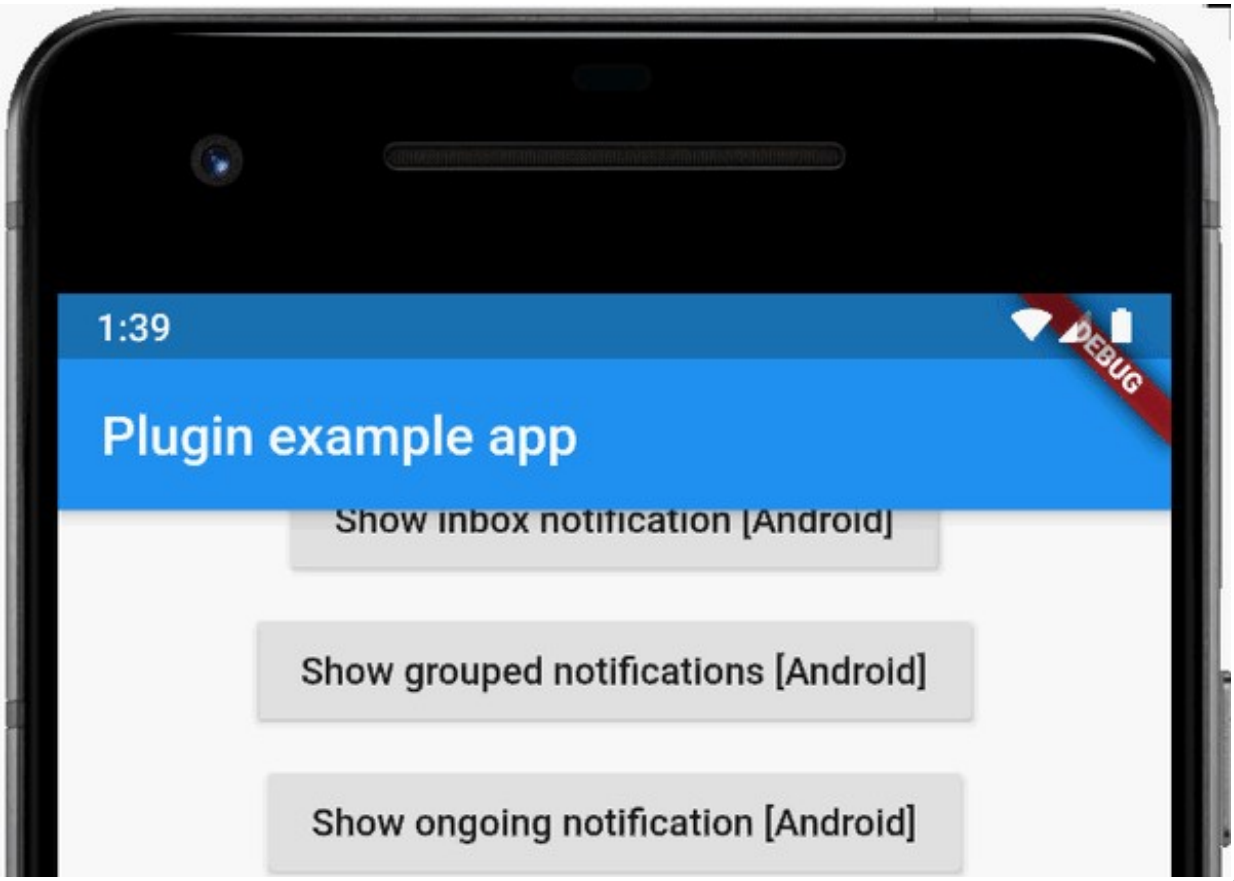


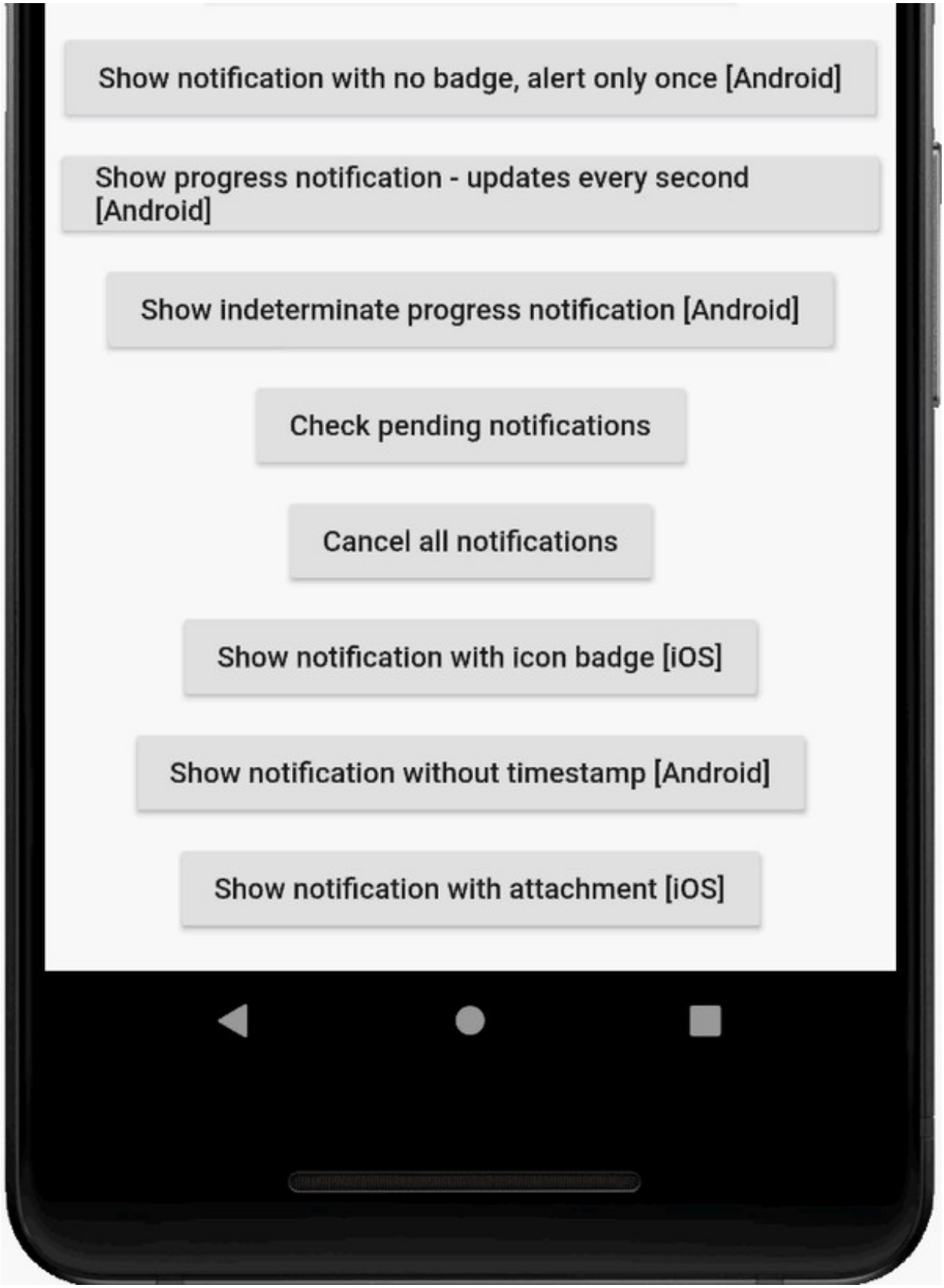


flutter_notifications_example.dart

The **cancelAll**() function, when executed, will terminate all those pending notifications so they won't ever be executed and displayed for example.

```
        onPressed: () => notifications.cancelAll(),
      ), // _PaddedRaisedButton
      _PaddedRaisedButton(
        buttonText: 'Show notification with icon badge [iOS]',
        onPressed: () => notifications.show(
          id: 0,
          title: 'icon badge title',
          body: 'icon badge body',
          payload: 'item x',
          badgeNumber: 1,
        ),
      ), // _PaddedRaisedButton
      _PaddedRaisedButton(
        buttonText: 'Show notification without timestamp [Android]',
        onPressed: () => notifications.show(
          id: 0,
          title: 'No Show When title',
          body: 'No Show When body',
          payload: 'item x',
          importance: Importance.Max,
          priority: Priority.High,
          showWhen: false,
        ),
      ), // _PaddedRaisedButton
      _PaddedRaisedButton(
```





flutter_notifications_example.dart

```
              _PaddedRaisedButton(
                buttonText: 'Show notification with attachment [iOS]',
                onPressed: () async {
                  //
                  var bigPicturePath = await _downloadAndSaveFile(
                      'http://via.placeholder.com/600x200',
                      'bigPicture.jpg');

                  var bigPictureAndroidStyle = BigPictureStyleInformation(
                      FilePathAndroidBitmap(bigPicturePath)); // BigPictureS

                  notifications.show(
                    id: 0,
                    title: 'notification with attachment title',
                    body: 'notification with attachment body',
                    importance: Importance.High,
                    priority: Priority.High,
                    styleInformation: bigPictureAndroidStyle,
                    attachments: [
                      IOSNotificationAttachment(bigPicturePath)
                    ],
                  );
                },
              ), // _PaddedRaisedButton
            ], // <Widget>[]
          ), // Column
        ), // Center
      ), // Padding
    ), // SingleChildScrollView
  ), // Scaffold
); // MaterialApp
}
```

flutter_notifications_example.dart

The Package

Let's now walk through the utility class, *ScheduleNotations*. Of course, like the plugin itself, you're required to enter three String values into the constructor when you instantiate a ScheduleNotations object. As it happens, when dealing with notations on the Android platform, you are to define and work with what is called a 'channel' — each identified by a unique string value. An excerpt of the Android documentation explains further:

> For each channel, you can set the visual and auditory behavior that is applied to all notifications in that channel. Then, users can change these settings and decide which notification channels from your app should be intrusive or visible at all.
>
> After you create a notification channel, you cannot change the notification behaviors — the user has complete control at that point. Though you can still change a channel's name and description.
>
> You should create a channel for each distinct type of notification you need to send. You can also create notification channels to reflect choices made by users of your app.
>
> - Create and Manage Notification Channels

And so, the first three parameters in the constructor are required positional parameters. In turn, there is the Android channel id, name and description. The remaining parameters, some 51 in all, are named parameters and so are optional. They represent the 'notification settings' for both the Android and iOS platforms. Most involve the Android platform, the last 13 involve iOS.

```
class ScheduleNotifications with HandleError {
//
  ScheduleNotifications(
    this.channelId,
    this.channelName,
    this.channelDescription, {
    String appIcon,
    DateTime schedule,
    String title,
    String body,
    String payload,
    bool androidAllowWhileIdle,
    String icon,
    Importance importance,
    Priority priority,
    StyleInformation styleInformation,
    bool playSound,
    AndroidNotificationSound sound,
    bool enableVibration,
    List<int> vibrationPattern,
    String groupKey,
    bool setAsGroupSummary,
    GroupAlertBehavior groupAlertBehavior,
    bool autoCancel,
```

schedule_notifications.dart

```
bool ongoing,
Color color,
AndroidBitmap largeIcon,
bool onlyAlertOnce,
bool showWhen,
bool channelShowBadge,
bool showProgress,
int maxProgress,
int progress,
bool indeterminate,
AndroidNotificationChannelAction channelAction,
bool enableLights,
Color ledColor,
int ledOnMs,
int ledOffMs,
String ticker,
NotificationVisibility visibility,
int timeoutAfter,
String category,
SelectNotificationCallback onSelectNotification,
bool requestAlertPermission,
bool requestSoundPermission,
bool requestBadgePermission,
bool defaultPresentAlert,
```

schedule_notifications.dart

The first named parameter, *appIcon*, is just that — the app's icon. You have the option to provide a specific icon, but by default, Android's *ic_launcher* is used.

```
    bool defaultPresentSound,
    bool defaultPresentBadge,
    DidReceiveLocalNotificationCallback onDidReceiveLocalNotification,
    bool presentAlert,
    bool presentSound,
    bool presentBadge,
    String soundFile,
    int badgeNumber,
    List<IOSNotificationAttachment> attachments,
}) {
    if (appIcon == null || appIcon.trim().isEmpty) {
      // Assign the app's icon.
      _appIcon = '@mipmap/ic_launcher';
    } else {
      _appIcon = appIcon.trim();
    }
    // Take in parameter values into private variables.
    _schedule = schedule;
    _title = title ?? '';
    _body = body ?? '';
    _payload = payload;
    _androidAllowWhileIdle = androidAllowWhileIdle ?? false;
    _icon = icon;
    _importance = importance ?? Importance.Default;
    _priority = priority ?? Priority.Default;
```

schedule_notifications.dart

The vast list of private variables is now next and most will store a default value if a specific value was not passed to the constructor. Note, I chose not to use syntactic sugar (eg. this.appIcon) and instead have them all as ‘final’ public variables. And so, it’s a long list of variables prefixed with underscores.

```
_styleInformation = styleInformation;
_playSound = playSound ?? false;
_sound = sound;
_enableVibration = enableVibration ?? false;
_vibrationPattern = vibrationPattern;
_groupKey = groupKey;
_setAsGroupSummary = setAsGroupSummary;
_groupAlertBehavior = groupAlertBehavior ?? GroupAlertBehavior.All;
_autoCancel = autoCancel ?? true;
_ongoing = ongoing;
_color = color;
_largeIcon = largeIcon;
_onlyAlertOnce = onlyAlertOnce;
_showWhen = showWhen ?? true;
_channelShowBadge = channelShowBadge ?? true;
_showProgress = showProgress ?? false;
_maxProgress = maxProgress ?? 0;
_progress = progress ?? 0;
_indeterminate = indeterminate ?? false;
_channelAction = channelAction ?? AndroidNotificationChannelAction.CreateIfN
_enableLights = enableLights ?? true;
_ledColor = ledColor ?? const Color.fromARGB(255, 255, 0, 0);
_ledOnMs = ledOnMs ?? 1000;
_ledOffMs = ledOffMs ?? 500;
_ticker = ticker;
```

schedule_notifications.dart

As you can see the 'if null' operator (??) is repeatedly used in the class' constructor. This is done in this fashion because a developer could pass a list of null values for example. Whether it was intentional or not is insignificant. The 'if null' operator is repeatedly used to resolve most of the private variables to non-null values.

```
  _visibility = visibility ?? NotificationVisibility.Private;
  _timeoutAfter = timeoutAfter;
  _category = category;
  _selectNotificationCallback = onSelectNotification;
  _requestAlertPermission = requestAlertPermission ?? true;
  _requestSoundPermission = requestSoundPermission ?? true;
  _requestBadgePermission = requestBadgePermission ?? true;
  _defaultPresentAlert = defaultPresentAlert ?? true;
  _defaultPresentSound = defaultPresentSound ?? true;
  _defaultPresentBadge = defaultPresentBadge ?? true;
  _onDidReceiveLocalNotification = onDidReceiveLocalNotification;
  _presentAlert = presentAlert ?? true;
  _presentSound = presentSound ?? true;
  _presentBadge = presentBadge ?? true;
  _soundFile = soundFile;
  _badgeNumber = badgeNumber;
  _attachments = attachments;
}

/// The icon representing the app implementing the notifications.
String _appIcon;

/// The channel's id.
/// Required for Android 8.0+.
final String channelId;
```

schedule_notifications.dart

Keep scrolling. The next two screenshots are declaring the private variables.

```dart
/// Required for Android 8.0+.
final String channelName;

/// The channel's description.
/// Required for Android 8.0+.
final String channelDescription;

DateTime _schedule;
String _title;
String _body;
String _payload;
bool _androidAllowWhileIdle;
String _icon;
Importance _importance;
Priority _priority;
StyleInformation _styleInformation;
bool _playSound;
AndroidNotificationSound _sound;
bool _enableVibration;
List<int> _vibrationPattern;
String _groupKey;
bool _setAsGroupSummary;
GroupAlertBehavior _groupAlertBehavior;
bool _autoCancel;
bool _ongoing;
```

schedule_notifications.dart

```
Color _color;
AndroidBitmap _largeIcon;
bool _onlyAlertOnce;
bool _showWhen;
bool _channelShowBadge;
bool _showProgress;
int _maxProgress;
int _progress;
bool _indeterminate;
AndroidNotificationChannelAction _channelAction;
bool _enableLights;
Color _ledColor;
int _ledOnMs;
int _ledOffMs;
String _ticker;
NotificationVisibility _visibility;
int _timeoutAfter;
String _category;
SelectNotificationCallback _selectNotificationCallback;
bool _requestAlertPermission;
bool _requestSoundPermission;
bool _requestBadgePermission;
bool _defaultPresentAlert;
bool _defaultPresentSound;
bool _defaultPresentBadge;
```

schedule_notifications.dart

Note the **init**() function, lists the same parameters found in the constructor. This means you have the option to assign values either at the constructor or at the **init**() function or both if necessary. In this class, the last value wins. Any values assigned at the constructor is 'overridden' by values assigned at the **init**() function. Note, values assigned at the **init**() function will be overridden by those passed to the specific notification functions: **show**(), **schedule**(), **periodicallyShow**(), etc.

```dart
DidReceiveLocalNotificationCallback _onDidReceiveLocalNotification;
bool _presentAlert;
bool _presentSound;
bool _presentBadge;
String _soundFile;
int _badgeNumber;
List<IOSNotificationAttachment> _attachments;

@mustCallSuper
Future<bool> init({
  DateTime schedule,
  String title,
  String body,
  String payload,
  bool androidAllowWhileIdle,
  String icon,
  Importance importance,
  Priority priority,
  StyleInformation styleInformation,
  bool playSound,
  AndroidNotificationSound sound,
  bool enableVibration,
  List<int> vibrationPattern,
  String groupKey,
  bool setAsGroupSummary,
```

schedule_notifications.dart

```
    GroupAlertBehavior groupAlertBehavior,
    bool autoCancel,
    bool ongoing,
    Color color,
    AndroidBitmap largeIcon,
    bool onlyAlertOnce,
    bool showWhen,
    bool channelShowBadge,
    bool showProgress,
    int maxProgress,
    int progress,
    bool indeterminate,
    AndroidNotificationChannelAction channelAction,
    bool enableLights,
    Color ledColor,
    int ledOnMs,
    int ledOffMs,
    String ticker,
    NotificationVisibility visibility,
    int timeoutAfter,
    String category,
    SelectNotificationCallback onSelectNotification,
    bool requestAlertPermission,
    bool requestSoundPermission,
    bool requestBadgePermission,
```

schedule_notifications.dart

It's in the **init**() function that the 'if null' operator is again repeatedly used. It's in this fashion that each private variable is either assigned the 'default' value from the constructor or assigned the value passed to the **init**() function.

```
  bool defaultPresentAlert,
  bool defaultPresentSound,
  bool defaultPresentBadge,
  DidReceiveLocalNotificationCallback onDidReceiveLocalNotification,
  bool presentAlert,
  bool presentSound,
  bool presentBadge,
  String soundFile,
  int badgeNumber,
  List<IOSNotificationAttachment> attachments,
}) async {
  // No need to continue.
  if (_init) return _init;

  // init parameters take over initial parameter values.
  _schedule = schedule ?? _schedule;
  _title = title ?? _title;
  _body = body ?? _body;
  _payload = payload ?? _payload;
  _androidAllowWhileIdle = androidAllowWhileIdle ?? _androidAllowWhileIdle;
  _icon = icon ?? _icon;
  _importance = importance ?? _importance;
  _priority = priority ?? _priority;
  _styleInformation = styleInformation ?? _styleInformation;
  _playSound = playSound ?? _playSound;
```

schedule_notifications.dart

This stretch of code updates the private variables if any parameter values were passed into the **init**() function. If the parameter value is null, it's merely re-assigned its own value. Granted, a series of **if** statements would possibly be more efficient (In fact, I've decided to make that change in its gist.).

```
_sound = sound ?? _sound;
_enableVibration = enableVibration ?? _enableVibration;
_vibrationPattern = vibrationPattern ?? _vibrationPattern;
_groupKey = groupKey ?? _groupKey;
_setAsGroupSummary = setAsGroupSummary ?? _setAsGroupSummary;
_groupAlertBehavior = groupAlertBehavior ?? _groupAlertBehavior;
_autoCancel = autoCancel ?? _autoCancel;
_ongoing = ongoing ?? _ongoing;
_color = color ?? _color;
_largeIcon = largeIcon ?? _largeIcon;
_onlyAlertOnce = onlyAlertOnce ?? _onlyAlertOnce;
_showWhen = showWhen ?? _showWhen;
_channelShowBadge = channelShowBadge ?? _channelShowBadge;
_showProgress = showProgress ?? _showProgress;
_maxProgress = maxProgress ?? _maxProgress;
_progress = progress ?? _progress;
_indeterminate = indeterminate ?? _indeterminate;
_channelAction = channelAction ?? _channelAction;
_enableLights = enableLights ?? _enableLights;
_ledColor = ledColor ?? _ledColor;
_ledOnMs = ledOnMs ?? _ledOnMs;
_ledOffMs = ledOffMs ?? _ledOffMs;
_ticker = ticker ?? _ticker;
_visibility = visibility ?? _visibility;
_timeoutAfter = timeoutAfter ?? _timeoutAfter;
```

schedule_notifications.dart

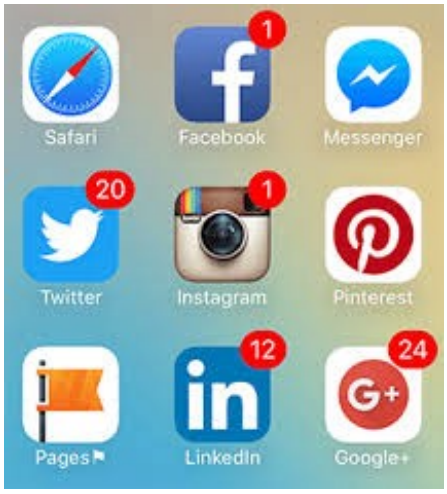
In the screenshot below, the **IOSInitializationSettings**() function is used to initialize the plugin settings for the iOS platform. It's supplied default values, of course, but you're free to pass its parameters to either the class constructor or to the **init**() function.

```
    _category = category ?? _category;
    onSelectNotification ??= _selectNotificationCallback;
    requestAlertPermission ??= _requestAlertPermission;
    requestSoundPermission ??= _requestSoundPermission;
    requestBadgePermission ??= _requestBadgePermission;
    defaultPresentAlert ??= _defaultPresentAlert;
    defaultPresentSound ??= _defaultPresentSound;
    defaultPresentBadge ??= _defaultPresentBadge;
    onDidReceiveLocalNotification ??= _onDidReceiveLocalNotification;
    _presentAlert = presentAlert ?? _presentAlert;
    _presentSound = presentSound ?? _presentSound;
    _presentBadge = presentBadge ?? _presentBadge;
    _soundFile = soundFile ?? _soundFile;
    _badgeNumber = badgeNumber ?? _badgeNumber;
    _attachments = attachments ?? _attachments;

    //
    try {
      // Note: permissions aren't requested here just to demonstrate that can be
      // of the `IOSFlutterLocalNotificationsPlugin` class
      var initializationSettingsIOS = IOSInitializationSettings(
        requestAlertPermission: requestAlertPermission,
        requestBadgePermission: requestSoundPermission,
        requestSoundPermission: requestBadgePermission,
        defaultPresentAlert: defaultPresentAlert,
```

schedule_notifications.dart

The parameter, *requestBadgePermission*, allows for the little red badges indicating how many notifications are currently being displayed.

The next stretch of code displayed below has the underlying plugin itself being instantiated with the function, **FlutterLocalNotificationsPlugin**(). An initialization object is first composed of both the Android and iOS platform settings and is delivered to the function, **InitializationSettings**(), and the plugin object is instantiated, it's initialized with those settings.



Notifications indicated by badge icons

```
        defaultPresentSound: defaultPresentSound,
        defaultPresentBadge: defaultPresentBadge,
        onDidReceiveLocalNotification: onDidReceiveLocalNotification ??
            (int id, String title, String body, String payload) =>
                onSelectNotification(payload),
      );

      var initializationSettings = InitializationSettings(
          AndroidInitializationSettings(_appIcon), initializationSettingsIOS); /

      _flutterLocalNotificationsPlugin = FlutterLocalNotificationsPlugin();

      _init = await _flutterLocalNotificationsPlugin.initialize(
          initializationSettings,
          onSelectNotification: onSelectNotification);
    } catch (ex) {
      getError(ex);
    }
    return _init;
}

@mustCallSuper
void dispose(){
  // Not really necessary but provides a dispose() function for the user.
  _flutterLocalNotificationsPlugin = null;
```

schedule_notifications.dart

```dart
}

/// Returns the underlying platform-specific implementation of given type [T],
/// must be a concrete subclass of [FlutterLocalNotificationsPlatform](https:/
Future<bool> resolveIOSImplementation({
  bool alert = true,
  bool badge = true,
  bool sound = true,
}) async {
  assert(_init, 'ScheduleNotifications: Failed to call init() first!');
  if (!_init) return false;
  var implementation;
  try {
    implementation = _flutterLocalNotificationsPlugin
        ?.resolvePlatformSpecificImplementation<
            IOSFlutterLocalNotificationsPlugin>();
  } catch (ex) {
    getError(ex);
  }
  var request = implementation != null;
  if (request) {
    request = await implementation.requestPermissions(
      alert: alert,
      badge: badge,
```

schedule_notifications.dart

The next stretch of code conveys the **show**() function. It is this function when called, that immediately displays the notification.

```
      sound: sound,
    );
  }
  return request;
}

// plugin
FlutterLocalNotificationsPlugin _flutterLocalNotificationsPlugin;

// Flag indicating it's initialized.
bool _init = false;
bool get initialized => _init;

int show({
  int id,
  String title,
  String body,
  String payload,
  bool androidAllowWhileIdle,
  String icon,
  Importance importance,
  Priority priority,
  StyleInformation styleInformation,
  bool playSound,
  AndroidNotificationSound sound,
  bool enableVibration,
```

schedule_notifications.dart

```
    List<int> vibrationPattern,
    String groupKey,
    bool setAsGroupSummary,
    GroupAlertBehavior groupAlertBehavior,
    bool autoCancel,
    bool ongoing,
    Color color,
    AndroidBitmap largeIcon,
    bool onlyAlertOnce,
    bool showWhen,
    bool channelShowBadge,
    bool showProgress,
    int maxProgress,
    int progress,
    bool indeterminate,
    AndroidNotificationChannelAction channelAction,
    bool enableLights,
    Color ledColor,
    int ledOnMs,
    int ledOffMs,
    String ticker,
    NotificationVisibility visibility,
    int timeoutAfter,
    String category,
```

schedule_notifications.dart

```
  bool presentAlert,
  bool presentSound,
  bool presentBadge,
  String soundFile,
  int badgeNumber,
  List<IOSNotificationAttachment> attachments,
}) {
  //
  var notificationSpecifics = _notificationDetails(
    title,
    body,
    payload,
    androidAllowWhileIdle,
    icon,
    importance,
    priority,
    styleInformation,
    playSound,
    sound,
    enableVibration,
    vibrationPattern,
    groupKey,
    setAsGroupSummary,
    groupAlertBehavior,
```

schedule_notifications.dart

```
    autoCancel,
    ongoing,
    color,
    largeIcon,
    onlyAlertOnce,
    showWhen,
    channelShowBadge,
    showProgress,
    maxProgress,
    progress,
    indeterminate,
    channelAction,
    enableLights,
    ledColor,
    ledOnMs,
    ledOffMs,
    ticker,
    visibility,
    timeoutAfter,
    category,
    presentAlert,
    presentSound,
    presentBadge,
    soundFile,
```

schedule_notifications.dart

You see the **show**() function is returning the integer value that is the id for that notification.

```
    badgeNumber,
    attachments,
  );

  if (notificationSpecifics == null) {
    id = -1;
  } else {
    //
    try {
      //
      if (id == null || id < 0) id = Random().nextInt(999);

      _flutterLocalNotificationsPlugin.show(
        id,
        title,
        body,
        notificationSpecifics,
        payload: payload,
      );
    } catch (ex) {
      id = -1;
      getError(ex);
    }
  }
  return id;
```

schedule_notifications.dart

The next stretch of code has the **schedule**() function. It takes in the required DateTime parameter.

```
}

int schedule(
  DateTime schedule, {
  int id,
  String title,
  String body,
  String payload,
  bool androidAllowWhileIdle,
  String icon,
  Importance importance,
  Priority priority,
  StyleInformation styleInformation,
  bool playSound,
  AndroidNotificationSound sound,
  bool enableVibration,
  List<int> vibrationPattern,
  String groupKey,
  bool setAsGroupSummary,
  GroupAlertBehavior groupAlertBehavior,
  bool autoCancel,
  bool ongoing,
  Color color,
  AndroidBitmap largeIcon,
  bool onlyAlertOnce,
```

schedule_notifications.dart

```
  bool showWhen,
  bool channelShowBadge,
  bool showProgress,
  int maxProgress,
  int progress,
  bool indeterminate,
  AndroidNotificationChannelAction channelAction,
  bool enableLights,
  Color ledColor,
  int ledOnMs,
  int ledOffMs,
  String ticker,
  NotificationVisibility visibility,
  int timeoutAfter,
  String category,
  bool presentAlert,
  bool presentSound,
  bool presentBadge,
  String soundFile,
  int badgeNumber,
  List<IOSNotificationAttachment> attachments,
}) {
  // Too late!
  if (schedule == null || DateTime.now().isAfter(schedule)) return -1;
```

schedule_notifications.dart

```dart
var notificationSpecifics = _notificationDetails(
  title,
  body,
  payload,
  androidAllowWhileIdle,
  icon,
  importance,
  priority,
  styleInformation,
  playSound,
  sound,
  enableVibration,
  vibrationPattern,
  groupKey,
  setAsGroupSummary,
  groupAlertBehavior,
  autoCancel,
  ongoing,
  color,
  largeIcon,
  onlyAlertOnce,
  showWhen,
  channelShowBadge,
  showProgress,
```

schedule_notifications.dart

```
    maxProgress,
    progress,
    indeterminate,
    channelAction,
    enableLights,
    ledColor,
    ledOnMs,
    ledOffMs,
    ticker,
    visibility,
    timeoutAfter,
    category,
    presentAlert,
    presentSound,
    presentBadge,
    soundFile,
    badgeNumber,
    attachments,
  );

  if (notificationSpecifics == null) {
    id = -1;
  } else {
    //
```

schedule_notifications.dart

Again, like the **show**() function, the **schedule**() function will generate an id value for you if you don't provide one. It'll be an integer value from the **Random**() function. The id returned can be used as a reference to later cancel that particular notification for example. Such an integer value is returned by all the notification funcitons: **periodicallyShow**(), **showDailyAtTime**(), **showWeeklyAtDayAndTime**()

```dart
    if (id == null || id < 0) id = Random().nextInt(999);

    try {
      //
      _flutterLocalNotificationsPlugin.schedule(
        id,
        title,
        body,
        schedule,
        notificationSpecifics,
        payload: payload,
        androidAllowWhileIdle: androidAllowWhileIdle,
      );
    } catch (ex) {
      id = -1;
      getError(ex);
    }
  }
  return id;
}

int periodicallyShow(
  RepeatInterval repeatInterval, {
  int id,
```

schedule_notifications.dart

```
String title,
String body,
String payload,
bool androidAllowWhileIdle,
String icon,
Importance importance,
Priority priority,
StyleInformation styleInformation,
bool playSound,
AndroidNotificationSound sound,
bool enableVibration,
List<int> vibrationPattern,
String groupKey,
bool setAsGroupSummary,
GroupAlertBehavior groupAlertBehavior,
bool autoCancel,
bool ongoing,
Color color,
AndroidBitmap largeIcon,
bool onlyAlertOnce,
bool showWhen,
bool channelShowBadge,
bool showProgress,
int maxProgress,
```

schedule_notifications.dart

```
  int progress,
  bool indeterminate,
  AndroidNotificationChannelAction channelAction,
  bool enableLights,
  Color ledColor,
  int ledOnMs,
  int ledOffMs,
  String ticker,
  NotificationVisibility visibility,
  int timeoutAfter,
  String category,
  bool presentAlert,
  bool presentSound,
  bool presentBadge,
  String soundFile,
  int badgeNumber,
  List<IOSNotificationAttachment> attachments,
}) {
  //
  var notificationSpecifics = _notificationDetails(
    title,
    body,
    payload,
    androidAllowWhileIdle,
```

schedule_notifications.dart

```
    icon,
    importance,
    priority,
    styleInformation,
    playSound,
    sound,
    enableVibration,
    vibrationPattern,
    groupKey,
    setAsGroupSummary,
    groupAlertBehavior,
    autoCancel,
    ongoing,
    color,
    largeIcon,
    onlyAlertOnce,
    showWhen,
    channelShowBadge,
    showProgress,
    maxProgress,
    progress,
    indeterminate,
    channelAction,
    enableLights,
```

schedule_notifications.dart

```
    ledColor,
    ledOnMs,
    ledOffMs,
    ticker,
    visibility,
    timeoutAfter,
    category,
    presentAlert,
    presentSound,
    presentBadge,
    soundFile,
    badgeNumber,
    attachments,
  );

  if (notificationSpecifics == null) {
    id = -1;
  } else {
    //
    try {
      //
      if (id == null || id < 0) id = Random().nextInt(999);

      _flutterLocalNotificationsPlugin.periodicallyShow(
```

schedule_notifications.dart

```
        id,
        title,
        body,
        repeatInterval,
        notificationSpecifics,
        payload: payload,
      );
    } catch (ex) {
      id = -1;
      getError(ex);
    }
  }
  return id;
}

int showDailyAtTime(
  Time notificationTime, {
  int id,
  String title,
  String body,
  String payload,
  bool androidAllowWhileIdle,
  String icon,
  Importance importance,
```

schedule_notifications.dart

```
Priority priority,
StyleInformation styleInformation,
bool playSound,
AndroidNotificationSound sound,
bool enableVibration,
List<int> vibrationPattern,
String groupKey,
bool setAsGroupSummary,
GroupAlertBehavior groupAlertBehavior,
bool autoCancel,
bool ongoing,
Color color,
AndroidBitmap largeIcon,
bool onlyAlertOnce,
bool showWhen,
bool channelShowBadge,
bool showProgress,
int maxProgress,
int progress,
bool indeterminate,
AndroidNotificationChannelAction channelAction,
bool enableLights,
Color ledColor,
int ledOnMs,
```

schedule_notifications.dart

```
  int ledOffMs,
  String ticker,
  NotificationVisibility visibility,
  int timeoutAfter,
  String category,
  bool presentAlert,
  bool presentSound,
  bool presentBadge,
  String soundFile,
  int badgeNumber,
  List<IOSNotificationAttachment> attachments,
}) {
  //
  var notificationSpecifics = _notificationDetails(
    title,
    body,
    payload,
    androidAllowWhileIdle,
    icon,
    importance,
    priority,
    styleInformation,
    playSound,
    sound,
    enableVibration,
```

schedule_notifications.dart

```
    vibrationPattern,
    groupKey,
    setAsGroupSummary,
    groupAlertBehavior,
    autoCancel,
    ongoing,
    color,
    largeIcon,
    onlyAlertOnce,
    showWhen,
    channelShowBadge,
    showProgress,
    maxProgress,
    progress,
    indeterminate,
    channelAction,
    enableLights,
    ledColor,
    ledOnMs,
    ledOffMs,
    ticker,
    visibility,
    timeoutAfter,
    category,
    presentAlert,
```

schedule_notifications.dart

```
    presentSound,
    presentBadge,
    soundFile,
    badgeNumber,
    attachments,
  );

  if (notificationSpecifics == null) {
    id = -1;
  } else {
    //
    try {
      //
      if (id == null || id < 0) id = Random().nextInt(999);

      _flutterLocalNotificationsPlugin.showDailyAtTime(
        id,
        title,
        body,
        notificationTime,
        notificationSpecifics,
        payload: payload,
      );
    } catch (ex) {
      id = -1;
```

schedule_notifications.dart

```
      getError(ex);
    }
  }
  return id;
}

int showWeeklyAtDayAndTime(
  Day day,
  Time notificationTime, {
  int id,
  String title,
  String body,
  String payload,
  bool androidAllowWhileIdle,
  String icon,
  Importance importance,
  Priority priority,
  StyleInformation styleInformation,
  bool playSound,
  AndroidNotificationSound sound,
  bool enableVibration,
  List<int> vibrationPattern,
  String groupKey,
  bool setAsGroupSummary,
  GroupAlertBehavior groupAlertBehavior,
```

schedule_notifications.dart

```
bool autoCancel,
bool ongoing,
Color color,
AndroidBitmap largeIcon,
bool onlyAlertOnce,
bool showWhen,
bool channelShowBadge,
bool showProgress,
int maxProgress,
int progress,
bool indeterminate,
AndroidNotificationChannelAction channelAction,
bool enableLights,
Color ledColor,
int ledOnMs,
int ledOffMs,
String ticker,
NotificationVisibility visibility,
int timeoutAfter,
String category,
bool presentAlert,
bool presentSound,
bool presentBadge,
String soundFile,
int badgeNumber,
```

schedule_notifications.dart

```
    List<IOSNotificationAttachment> attachments,
}) {
    //
    var notificationSpecifics = _notificationDetails(
        title,
        body,
        payload,
        androidAllowWhileIdle,
        icon,
        importance,
        priority,
        styleInformation,
        playSound,
        sound,
        enableVibration,
        vibrationPattern,
        groupKey,
        setAsGroupSummary,
        groupAlertBehavior,
        autoCancel,
        ongoing,
        color,
        largeIcon,
        onlyAlertOnce,
        showWhen,
```

schedule_notifications.dart

```
    channelShowBadge,
    showProgress,
    maxProgress,
    progress,
    indeterminate,
    channelAction,
    enableLights,
    ledColor,
    ledOnMs,
    ledOffMs,
    ticker,
    visibility,
    timeoutAfter,
    category,
    presentAlert,
    presentSound,
    presentBadge,
    soundFile,
    badgeNumber,
    attachments,
  );

  if (notificationSpecifics == null) {
    id = -1;
```

schedule_notifications.dart

```
  } else {
    //
    try {
      //
      if (id == null || id < 0) id = Random().nextInt(999);

      _flutterLocalNotificationsPlugin.showWeeklyAtDayAndTime(
        id,
        title,
        body,
        day,
        notificationTime,
        notificationSpecifics,
        payload: payload,
      );
    } catch (ex) {
      id = -1;
      getError(ex);
    }
  }
  return id;
}

NotificationDetails _notificationDetails(
  String title,
```

schedule_notifications.dart

The private function, **_notificationDetails**(), is called throughout the utility class. It configures the notification settings on both the Android and iOS platforms. It returns the object, *NotificationDetails*.

```
String body,
String payload,
bool androidAllowWhileIdle,
String icon,
Importance importance,
Priority priority,
StyleInformation styleInformation,
bool playSound,
AndroidNotificationSound sound,
bool enableVibration,
List<int> vibrationPattern,
String groupKey,
bool setAsGroupSummary,
GroupAlertBehavior groupAlertBehavior,
bool autoCancel,
bool ongoing,
Color color,
AndroidBitmap largeIcon,
bool onlyAlertOnce,
bool showWhen,
bool channelShowBadge,
bool showProgress,
int maxProgress,
int progress,
bool indeterminate,
```

schedule_notifications.dart

The private function, **_notificationDetails**(), will return null if the developer fails to call the **init**() function. The **init**() function, of course, does just that — it initializes the essential plugin, *FlutterLocalNotificationsPlugin*. I could call the **init**() function in these functions, but supplying some sort of conformity is another characteristic of a utility class. In this case, developers are to instantiate the class, call its **init**() function, call its 'notification' functions, and finally, call its **dispose**() function.

```
  AndroidNotificationChannelAction channelAction,
  bool enableLights,
  Color ledColor,
  int ledOnMs,
  int ledOffMs,
  String ticker,
  NotificationVisibility visibility,
  int timeoutAfter,
  String category,
  bool presentAlert,
  bool presentSound,
  bool presentBadge,
  String soundFile,
  int badgeNumber,
  List<IOSNotificationAttachment> attachments,
) {
  //
  NotificationDetails notificationSpecifics;

  // Failed to initialized.
  if (!_init) {
    if (hasError) {
      assert(false, errorMsg);
    } else {
      assert(false, 'ScheduleNotifications: Failed to call init() first!');
```

schedule_notifications.dart

Again, in this class, the last value wins. Consequently, in this class, any values assigned at the **init**() function is 'overridden' at the **show**() function, the **schedule**() function, the **periodicallyShow**() function, etc.

```
    }
    return notificationSpecifics;
  }

  title ??= _title;
  body ??= _body;
  payload ??= _payload;
  androidAllowWhileIdle ??= _androidAllowWhileIdle;
  icon ??= _icon;
  importance ??= _importance;
  priority ??= _priority;
  styleInformation ??= _styleInformation;
  playSound ??= _playSound;
  sound ??= _sound;
  enableVibration ??= _enableVibration;
  vibrationPattern ??= _vibrationPattern;
  groupKey ??= _groupKey;
  setAsGroupSummary ??= _setAsGroupSummary;
  groupAlertBehavior ??= _groupAlertBehavior;
  autoCancel ??= _autoCancel;
  ongoing ??= _ongoing;
  color ??= _color;
  largeIcon ??= _largeIcon;
  onlyAlertOnce ??= _onlyAlertOnce;
  showWhen ??= _showWhen;
```

schedule_notifications.dart

You see the assign-if-null operator, `??=` , is repeatedly used in the private function, **_notificationDetails**(). They're all to assist in the 'last one wins' approach used in this class. You see, if the notation function didn't provide parameter values, the **init**() function's parameter values are used. If the **init**() function's did not provide parameter values, the constructor's parameter values are used. If the constructor didn't provide parameter values, the 'default' values are finally used to return the object, NotificationDetails. See how that works? Why does it work that way? It's to provide the developer the option to supply specific parameter values at any stage of the implementation. If you know me, you know I like options. That's why.

```
channelShowBadge ??= _channelShowBadge;
showProgress ??= _showProgress;
maxProgress ??= _maxProgress;
progress ??= _progress;
indeterminate ??= _indeterminate;
channelAction ??= _channelAction;
enableLights ??= _enableLights;
ledColor ??= _ledColor;
ledOnMs ??= _ledOnMs;
ledOffMs ??= _ledOffMs;
ticker ??= _ticker;
visibility ??= _visibility;
timeoutAfter ??= _timeoutAfter;
category ??= _category;
presentAlert ??= _presentAlert;
presentSound ??= _presentSound;
presentBadge ??= _presentBadge;
soundFile ??= _soundFile;
badgeNumber ??= _badgeNumber;
attachments ??= _attachments;

// Play the sound if supplied a sound.
if (playSound == null && sound != null) playSound = true;
```

schedule_notifications.dart

Good Vibrations

In the first bit of code in the screenshot below, we see a ‘default’ vibration pattern is conceived if not already provided because the vibration is enabled. The class takes care of that. Finally, the Android Notification settings are passed in. You can see the required channel id, and such are used at this point.

```dart
// If to vibrate then do so.
if (enableVibration &&
    (vibrationPattern == null || vibrationPattern.isEmpty)) {
  vibrationPattern = Int64List(4);
  vibrationPattern[0] = 0;
  vibrationPattern[1] = 1000;
  vibrationPattern[2] = 5000;
  vibrationPattern[3] = 2000;
}

AndroidNotificationDetails androidSettings;
IOSNotificationDetails iOSSettings;

try {
  androidSettings = AndroidNotificationDetails(
    channelId,
    channelName,
    channelDescription,
    icon: icon,
    importance: importance,
    priority: priority,
    styleInformation: styleInformation,
    playSound: playSound,
    sound: sound,
```

schedule_notifications.dart

```
    enableVibration: enableVibration,
    vibrationPattern: vibrationPattern,
    groupKey: groupKey,
    setAsGroupSummary: setAsGroupSummary,
    groupAlertBehavior: groupAlertBehavior,
    autoCancel: autoCancel,
    ongoing: ongoing,
    color: color,
    largeIcon: largeIcon,
    onlyAlertOnce: onlyAlertOnce,
    showWhen: showWhen,
    channelShowBadge: channelShowBadge,
    showProgress: showProgress,
    maxProgress: maxProgress,
    progress: progress,
    indeterminate: indeterminate,
    channelAction: channelAction,
    enableLights: enableLights,
    ledColor: ledColor,
    ledOnMs: ledOnMs,
    ledOffMs: ledOffMs,
    ticker: ticker,
    visibility: visibility,
    timeoutAfter: timeoutAfter,
```

schedule_notifications.dart

In the screenshot below, the object, *NotificationDetails*, is finally instantiated with the settings for both the Android and iOS platforms. Further note, the **try-catch** statement is used repeatedly to catch and unexcepted errors. If such an error occurs, it's recorded (like any utility class should) and null is instead returned from the private function, **_notificationDetails**().

```
      category: category,
    );
  } catch (ex) {
    getError(ex);
    return notificationSpecifics;
  }

  try {
    iOSSettings = IOSNotificationDetails(
      presentAlert: presentAlert,
      presentSound: presentSound,
      presentBadge: presentBadge,
      sound: soundFile,
      badgeNumber: badgeNumber,
      attachments: attachments,
    );

    notificationSpecifics = NotificationDetails(androidSettings, iOSSettings);
  } catch (ex) {
    notificationSpecifics = null;
    getError(ex);
  }
  return notificationSpecifics;
}
```

schedule_notifications.dart

The next few functions take up the remaining functions found in the plugin. For example, the **cancel**() function below takes in the integer identifier to cancel a specific notification while the **cancelAll**() function affects all the notification your app may have initialized. Of course, their own comments displayed below sufficiently describe their purpose.

```dart
  /// Cancel a specific notification.
  Future<void> cancel(int id) async {
    if (id == null || id < 0) return;
    await _flutterLocalNotificationsPlugin.cancel(id);
    return;
  }

  /// Cancel all Notifications.
  Future<void> cancelAll() => _flutterLocalNotificationsPlugin.cancelAll();

  /// Returns info on if a notification created from this plugin had been used t
  Future<NotificationAppLaunchDetails> getNotificationAppLaunchDetails() =>
      _flutterLocalNotificationsPlugin.getNotificationAppLaunchDetails();

  /// Returns a list of notifications pending to be delivered/shown.
  Future<List<PendingNotificationRequest>> pendingNotificationRequests() =>
      _flutterLocalNotificationsPlugin.pendingNotificationRequests();
}

mixin HandleError {
  /// Return the 'last' error if any.
  Exception getError([dynamic error]) {
    var ex = _error;
    if (error == null) {
      _error = null;
    } else {
```

schedule_notifications.dart

The last bit of code is a mixin assigned to the class itself. It merely demonstrates how one could use a mixin — by supplying the error reporting capability to the class, *ScheduleNotifications*. In the screenshot above is the function, **getError**(), that's called in every **try-catch** statement found in the library class. Further, below are also the getter's used to determine if there is, in fact, an error as well as return the actual error message. Again, such a class indicates if there's an error and records the error.

```
      if (error is! Exception) {
        _error = Exception(error.toString());
      } else {
        _error = error;
      }
      ex ??= _error;
    }
    return ex;
  }

  /// Simply display the error.
  String get errorMsg => _error == null ? '' : _error.toString();

  /// Determine if app is 'in error.'
  bool get inError => _error != null;
  bool get hasError => _error != null;
  Exception _error;
}
```

schedule_notifications.dart

The purpose of such a class is to make life a little easier. You work with this class that then works with the underlying plugin, flutter_local_notifications. Although in this class, a great many parameters are allotted to you to use, all in a number of different functions, you don't have to use them as most are assigned the appropriate default values. Again, such a utility class is full of repetitive code. It's full of 'if null' operators and **try-catch** statements so that notifications are successfully displayed. And if something goes wrong, it will fail gracefully and not cause your app to crash. Yes, it's ugly on the inside, but beautiful to work with on the outside as is the case with such a class. Take it, make it better, and share.

Cheers.

→ Other Stories by Greg Perry

Written by

DECODE Flutter on YouTube

Greg Perry

Follow

Freelance Developer

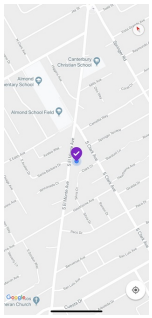
More From Medium

Error Handling in my Flutter App

Greg Perry in Follow Flutter

Add Custom Marker Images for your Google Maps in Flutter

Roman Jaquez in Flutter Community



Flutter Login & Registration Using Firebase

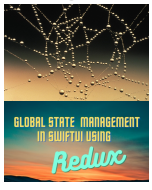
Mais Alheraki in The Startup

Your Next MVC Flutter Project

Greg Perry in Follow Flutter

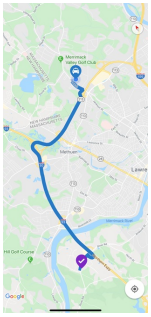
How to make HTTP requests in Flutter

Suragch in The Startup



Drawing Route Lines on Google Maps Between Two Locations in Flutter

Roman Jaquez in Flutter Community



my widget