Preserving Navigation when Starting an Activity

Part of designing a notification is preserving the user's expected navigation experience. For a detailed discussion of this topic, see the Notifications (/guide/topics/ui/notifiers/notifications.html#NotificationResponse) API guide. There are two general situations:

Regular activity
: You're starting an `Activity` that's part of the application's normal workflow.

Special activity
: The user only sees this `Activity` if it's started from a notification. In a sense, the `Activity` extends the notification by providing information that would be hard to display in the notification itself.

THIS LESSON TEACHES YOU TO

1. Set up a regular activity PendingIntent
2. Set up a special activity PendingIntent

YOU SHOULD ALSO READ

- Notifications API Guide
- Intents and Intent Filters
- Notifications Design Guide

Set Up a Regular Activity PendingIntent

To set up a `PendingIntent (/reference/android/app/PendingIntent.html)` that starts a direct entry `Activity (/reference/android/app/Activity.html)`, follow these steps:

1. Define your application's `Activity` hierarchy in the manifest. The final XML should look like this:

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".ResultActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

2. Create a back stack based on the `Intent` that starts the `Activity`. For example:

```
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
```

```
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

Set Up a Special Activity PendingIntent

A special Activity (/reference/android/app/Activity.html) doesn't need a back stack, so you don't have to define its Activity (/reference/android/app/Activity.html) hierarchy in the manifest, and you don't have to call addParentStack() (/reference/android/support/v4/app/TaskStackBuilder.html#addParentStack(android.app.Activity)) to build a back stack. Instead, use the manifest to set up the Activity (/reference/android/app/Activity.html) task options, and create the PendingIntent (/reference/android/app/PendingIntent.html) by calling getActivity() (/reference/android/app/PendingIntent.html#getActivity(android.content.Context, int, android.content.Intent, int)):

1. In your manifest, add the following attributes to the <activity> element for the Activity:

   android:name="*activityclass*"
   : The activity's fully-qualified class name.

   android:taskAffinity=""
   : Combined with the FLAG_ACTIVITY_NEW_TASK flag that you set in code, this ensures that this Activity doesn't go into the application's default task. Any existing tasks that have the application's default affinity are not affected.

   android:excludeFromRecents="true"
   : Excludes the new task from *Recents*, so that the user can't accidentally navigate back to it.

   This snippet shows the element:

   ```
   <activity
       android:name=".ResultActivity"
   ...
       android:launchMode="singleTask"
       android:taskAffinity=""
       android:excludeFromRecents="true">
   </activity>
   ...
   ```

2. Build and issue the notification:
   a. Create an Intent that starts the Activity.
   b. Set the Activity to start in a new, empty task by calling setFlags() with the flags FLAG_ACTIVITY_NEW_TASK and FLAG_ACTIVITY_CLEAR_TASK.
   c. Set any other options you need for the Intent.
   d. Create a PendingIntent from the Intent by calling getActivity(). You can then use this PendingIntent as the argument to setContentIntent().

   The following code snippet demonstrates the process:

   ```
   // Instantiate a Builder object.
   NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
   // Creates an Intent for the Activity
   Intent notifyIntent =
           new Intent(new ComponentName(this, ResultActivity.class));
   // Sets the Activity to start in a new, empty task
   notifyIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
           Intent.FLAG_ACTIVITY_CLEAR_TASK);
   // Creates the PendingIntent
   PendingIntent notifyIntent =
   ```

```
        PendingIntent.getActivity(
        this,
        0,
        notifyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```