51. An Android 7 Notifications Tutorial

Notifications provide a way for an app to convey a message to the user when the app is either not running or is currently in the background. A messaging app might, for example, issue a notification to let the user know that a new message has arrived from a contact. Notifications can be categorized as being either local or remote. A local notification is triggered by the app itself on the device on which it is running. Remote notifications, on the other hand, are initiated by a remote server and delivered to the device for presentation to the user.

Notifications appear in the notification shade that is pulled down from the status bar of the screen and each notification can include actions such as a button to open the app that sent the notification. Android 7 has also introduced Direct Reply, a feature that allows the user to type in and submit a response to a notification from within the notification panel.

The goal of this chapter is to outline and demonstrate the implementation of local notifications within an Android app. The next chapter (*An Android 7 Direct Reply Notification Tutorial*) will cover the implementation of direct reply notifications, while the use of Firebase to initiate and send remote notifications will be covered in the chapters entitled *Integrating Firebase Support into an Android Studio Project* and *An Android 7 Firebase Remote Notification Tutorial*.

51.1 An Overview of Notifications

When a notification is initiated on an Android device, it appears as an icon in the status bar. Figure 51-1, for example, shows a status bar with a number of notification icons:

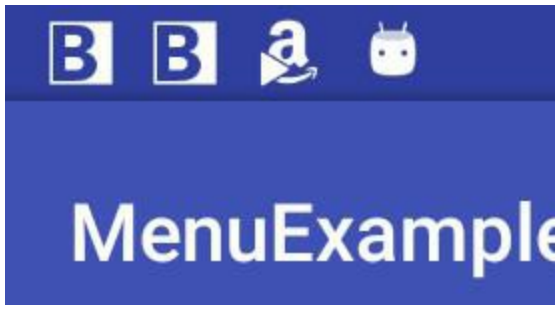

Figure 51-1

To view the notifications, the user makes a downward swiping motion starting at the status bar to pull down the notification shade as shown in Figure 51-2:

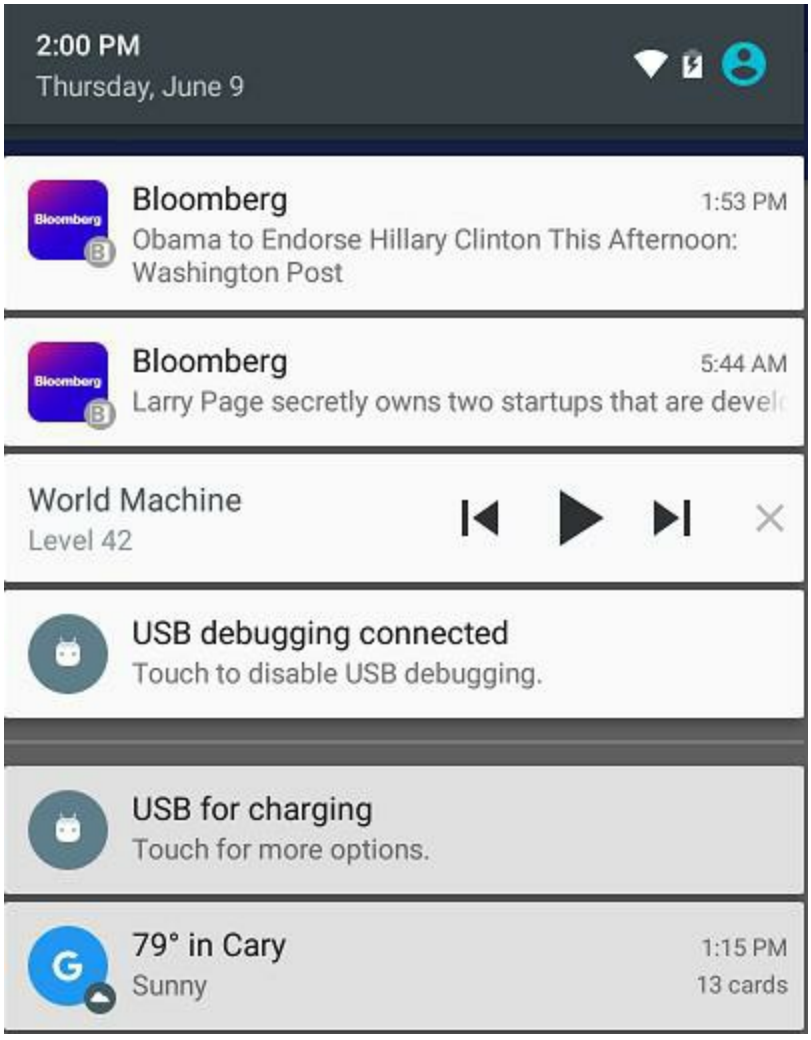

Figure 51-2

A typical notification will simply display a message and, when tapped, launch the app responsible for issuing the notification. Notifications may also contain action buttons which perform a task specific to the corresponding app when tapped. Figure 51-3, for example, shows a notification containing two action buttons allowing the user to either delete or save an incoming message.

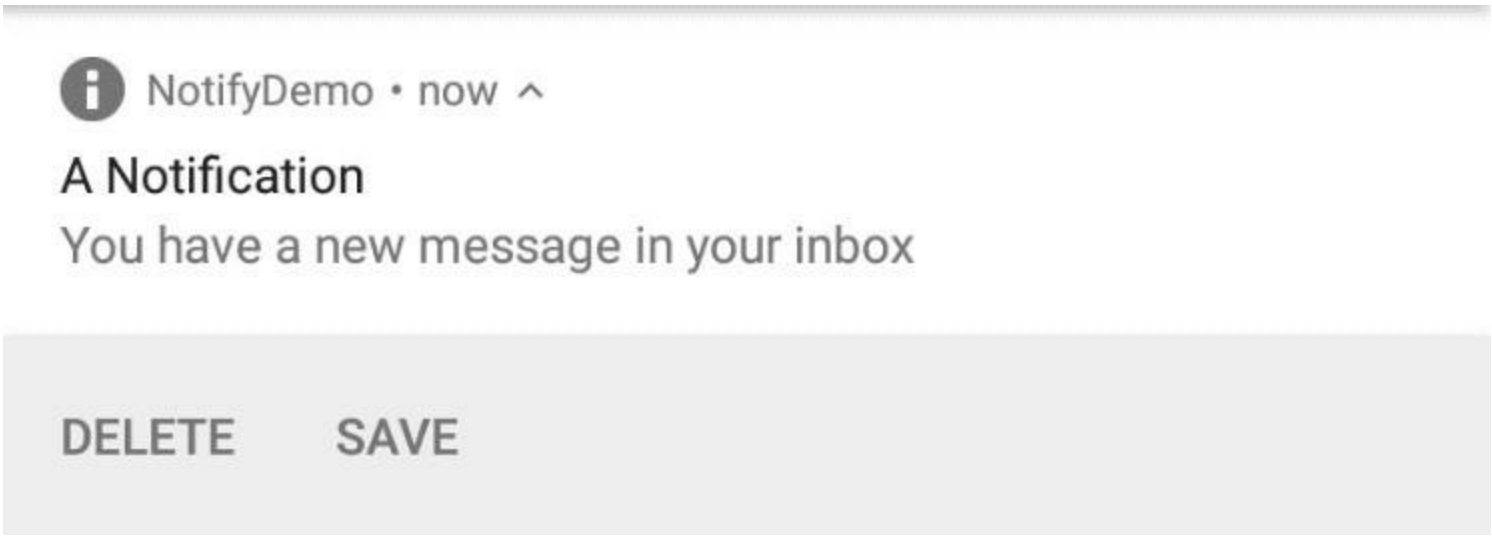

Figure 51-3

With Android 7, it is now also possible for the user to enter an inline text reply into the notification and send it to the app, as is the case in Figure 51-4 below. This allows the user to respond to a notification without having to launch the corresponding app into the foreground.

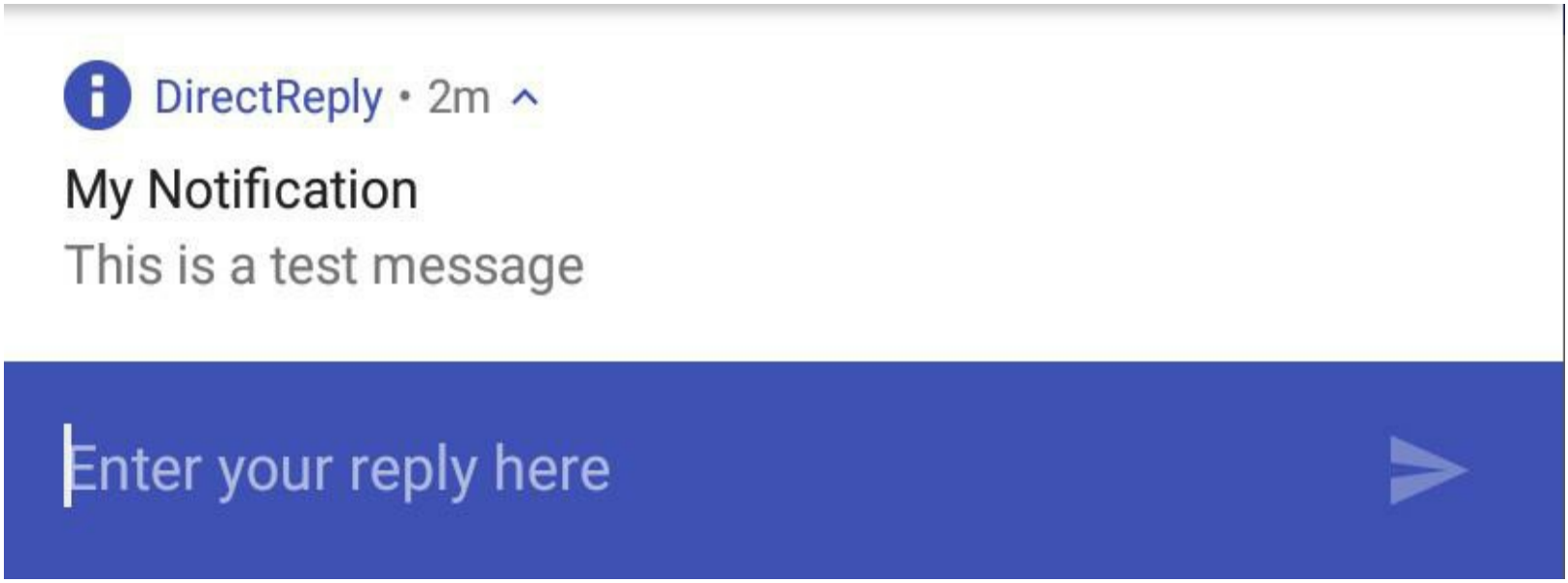

Figure 51-4

The remainder of this chapter will work through the steps involved in creating and issuing a simple notification containing actions. The topic of direct reply support will then be covered in the next chapter entitled *An Android 7 Direct Reply Notification Tutorial*.

51.2 Creating the NotifyDemo Project

Start Android Studio and create a new project, entering *NotifyDemo* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 24: Android 7.0 (Nougat). Continue through the screens, requesting the creation of an Empty Activity named *NotifyDemoActivity* with a corresponding layout file named *activity_notify_demo*.

51.3 Designing the User Interface

The main activity will contain a single button, the purpose of which is to create and issue an intent. Locate and load the *activity_notify_demo.xml* file into the Layout Editor tool and delete the default TextView widget.

With Autoconnect enabled, drag and drop a Button object from the panel onto the center of the layout canvas as illustrated in Figure 51-5.

With the Button widget selected in the layout, use the Properties panel to configure the onClick property to call a method named *sendNotification*.

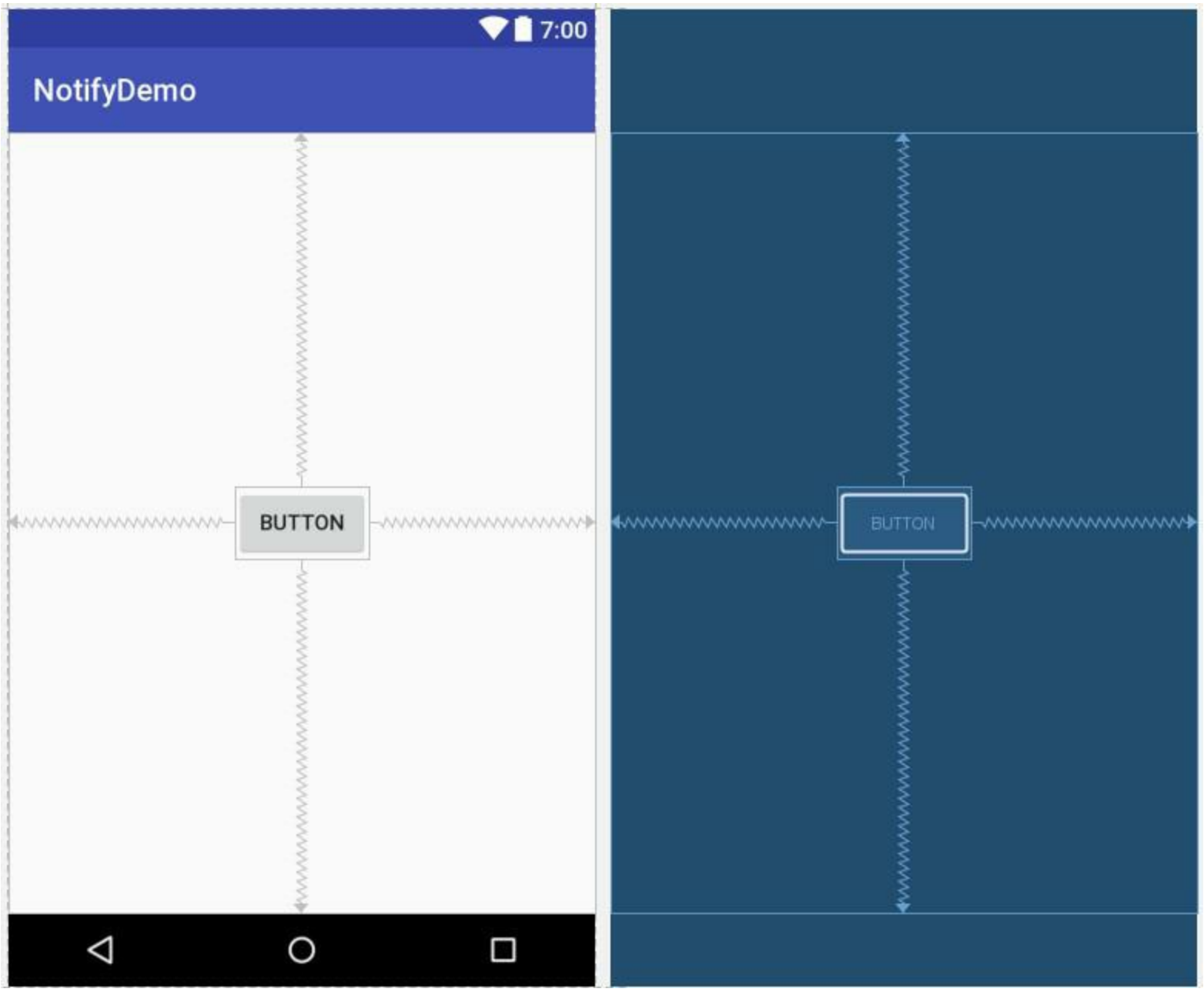

Figure 51-5

51.4 Creating the Second Activity

For the purposes of this example, the app will contain a second activity which will be launched by the user from within the notification. Add this new activity to the project by right-clicking on the *com.ebookfrenzy.notifydemo* package name located in *app -> java* and select the *New -> Activity -> Empty Activity* menu option to display the *New Android Activity* dialog.

Enter *ResultActivity* into the Activity Name field and name the layout file *activity_result*. Since this activity will not be started when the application is launched (it will instead be launched via an intent from within the notification), it is important to make sure that the *Launcher Activity* option is disabled before clicking on the Finish button.

Open the layout for the second activity (*app -> res -> layout -> activity_result.xml*) and drag and drop a TextView widget so that it is positioned in the center of the layout. Edit the text of the TextView so that it reads “Result Activity”, extract the property value to a string resource and change the layout_width property to *wrap_content*.

51.5 Creating and Issuing a Basic Notification

Notifications are created using the NotificationCompat.Builder class which allows properties such as the icon, title and content of the notification to be specified. Open the *NotifyDemoActivity.java* file and implement the *sendNotification()* method as follows to build a basic notification:

```
package com.ebookfrenzy.notifydemo;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.NotificationManager;
import android.support.v4.app.NotificationCompat;
import android.view.View;
import android.content.Intent;
import android.app.PendingIntent;

public class NotifyDemoActivity extends AppCompatActivity {
.
.
.
    protected void sendNotification(View view) {

        NotificationCompat.Builder builder =
          new NotificationCompat.Builder(this)
              .setSmallIcon(android.R.drawable.ic_dialog_info)
              .setContentTitle("A Notification")
              .setContentText("This is an example notification");
    }
}
```

The icon setting in the above code makes use of a built-in Android icon which is displayed both within the status bar and the notification panel when the notification is issued.

Once a notification has been built, it needs to be issued using the *notify()* method of the NotificationManager instance. The NotificationManager, a reference to which can be obtained via a call to the *getSystemService()* method, is a service that runs on Android devices and is responsible for managing notifications. The code to access the NotificationManager and issue the notification needs to be added to the *sendNotification()* method as follows:

```
protected void sendNotification(View view) {

    NotificationCompat.Builder builder =
          new NotificationCompat.Builder(this)
              .setSmallIcon(android.R.drawable.ic_dialog_info)
              .setContentTitle("A Notification")
              .setContentText("This is an example notification");

    int notificationId = 101;

    NotificationManager notifyMgr =
            (NotificationManager)
              getSystemService(NOTIFICATION_SERVICE);

    notifyMgr.notify(notificationId, builder.build());
}
```

Note that when the notification is issued, it is assigned a notification ID. This can be any integer and may be used later when updating or deleting the notification.

Compile and run the app and tap the button on the main activity. When the notification icon appears in the status bar, touch and drag down from the status bar to view the full notification:

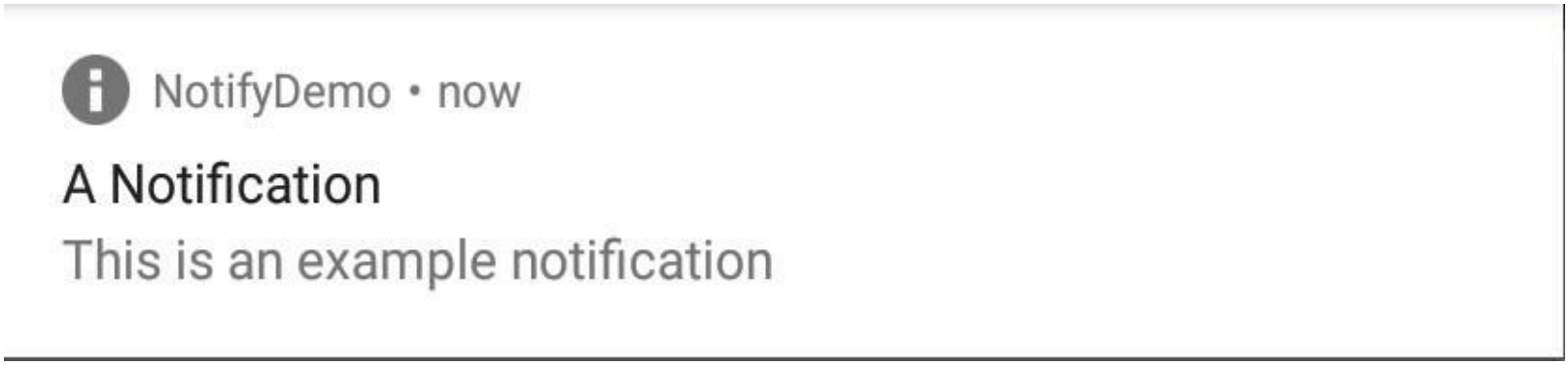

Figure 51-6

As currently implemented, tapping on the notification has no effect. The next step is to configure the notification to launch an activity when tapped.

51.6 Launching an Activity from a Notification

A notification should ideally allow the user to perform some form of action, such as launching the corresponding app, or taking some other form of action in response to the notification. A common requirement is to simply launch an activity belonging to the app when the user taps the notification.

This approach requires an activity to be launched and an Intent configured to launch that activity. Assuming an app that contains an activity named ResultActivity, the intent would be created as follows:

```
Intent resultIntent = new Intent(this, ResultActivity.class);
```

This intent needs to then be wrapped in a PendingIntent instance. PendingIntent objects are designed to allow an intent to be passed to other applications, essentially granting those applications permission to perform the intent at some point in the future. In this case, the PendingIntent object is being used to provide the Notification system with a way to launch the ResultActivity activity when the user taps the notification panel:

```
PendingIntent pendingIntent =
    PendingIntent.getActivity(
    this,
    0,
    resultIntent,
    PendingIntent.FLAG_UPDATE_CURRENT
);
```

All that remains is to assign the PendingIntent object to the notification builder instance created previously:

```
builder.setContentIntent(pendingIntent);
```

Bringing these changes together results in a modified *sendNotification()* method which reads as follows:

```
protected void sendNotification(View view) {

    NotificationCompat.Builder builder =
         new NotificationCompat.Builder(this)
             .setSmallIcon(android.R.drawable.ic_dialog_info)
```

```
                .setContentTitle("A Notification")
                .setContentText("This is an example notification");

        Intent resultIntent = new Intent(this, ResultActivity.class);

        PendingIntent pendingIntent =
                PendingIntent.getActivity(
                        this,
                        0,
                        resultIntent,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );

        builder.setContentIntent(pendingIntent);

        int notificationId = 101;

        NotificationManager notifyMgr =
          (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

        notifyMgr.notify(notificationId, builder.build());
    }
```

Compile and run the app once again, tap the button and display the notification shade. This time, however, tapping the notification will cause the ResultActivity to launch.

51.7 Adding Actions to a Notification

Another way to add interactivity to a notification is to create actions. These appear as buttons beneath the notification message and are programmed to trigger specific intents when tapped by the user. The following code, if added to the *sendNotification()* method, will add an action button labeled "Open" which launches the referenced pending intent when selected:

```
    NotificationCompat.Action action =
            new NotificationCompat.Action.Builder(
                android.R.drawable.sym_action_chat,
                        "Open", pendingIntent)
                        .build();

    builder.addAction(action);
```

Add the above code to the method and run the app. Issue the notification and note the appearance of the Open action within the notification:

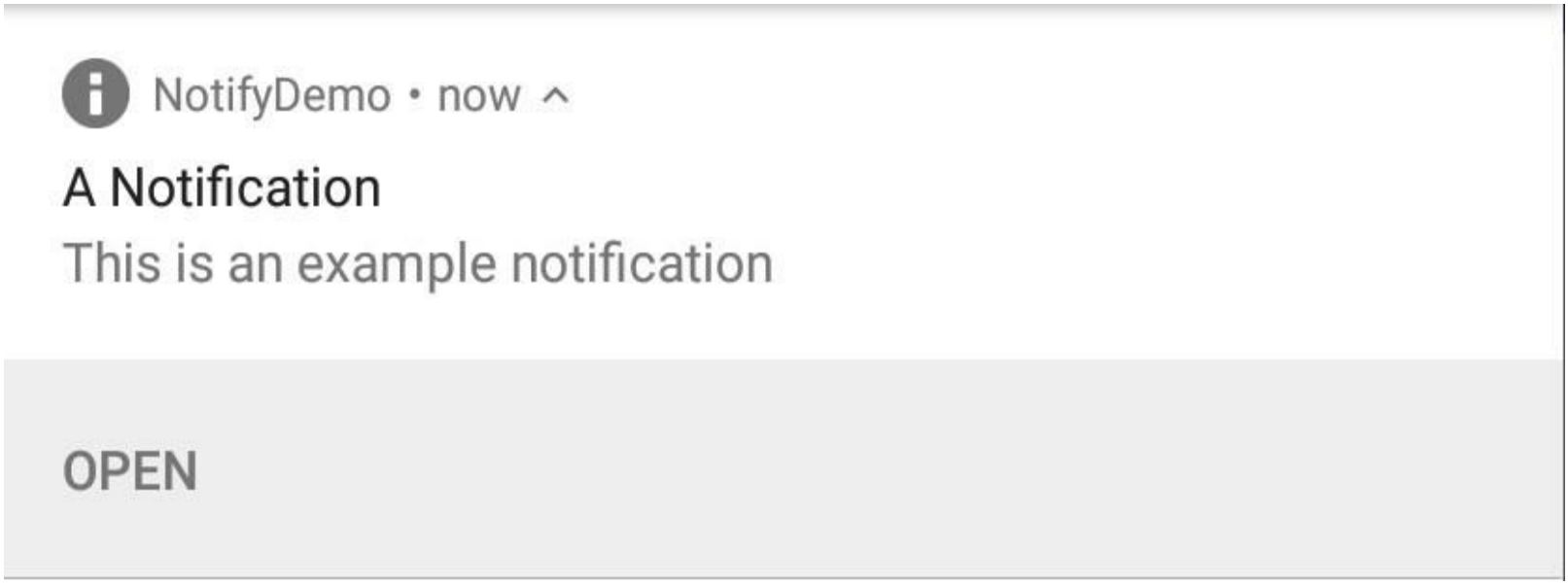

Figure 51-7

Tapping the action will trigger the pending intend and launch the ResultActivity.

51.8 Adding Sound to a Notification

Sound can be added to a notification using the *setSound()* method when creating the notification builder object. The following code fragment modifies the example to configure the default notification sound to accompany the notification:

```
package com.ebookfrenzy.notifydemo;

import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v4.app.NotificationCompat;
import android.view.View;
import android.media.RingtoneManager;
import android.net.Uri;
.
.
.
     Uri defaultSoundUri= RingtoneManager.getDefaultUri(
                              RingtoneManager.TYPE_NOTIFICATION);

     NotificationCompat.Builder builder =
           new NotificationCompat.Builder(this)
              .setSmallIcon(android.R.drawable.ic_dialog_info)
              .setContentTitle("A Notification")
              .setContentText("This is an example notification")
              .setSound(defaultSoundUri);
.
.
.
}
```

51.9 Bundled Notifications

If an app has a tendency to regularly issue notifications there is a danger that those notifications will

rapidly clutter both the status bar and the notification shade providing a less than optimal experience for the user. This can be particularly true of news or messaging apps that send a notification every time there is either a breaking news story or a new message arrives from a contact. Consider, for example, the notifications in Figure 51-8:

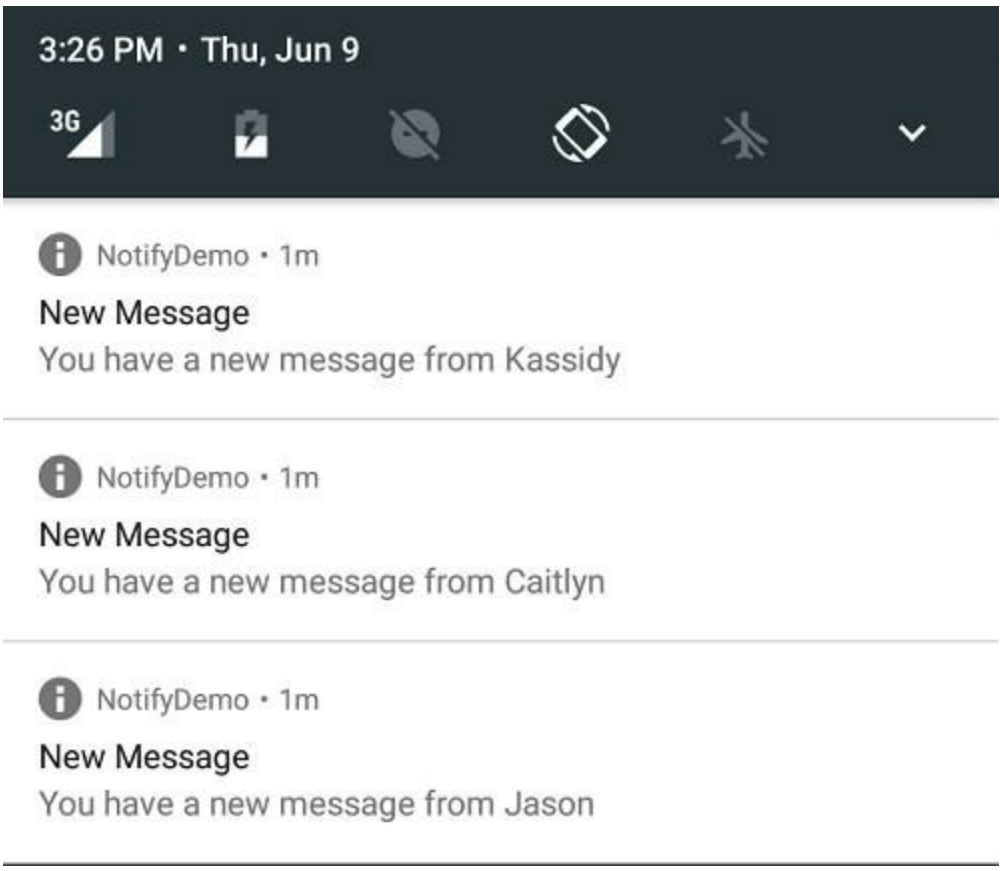

Figure 51-8

Now imagine if ten or even twenty new messages had arrived. To avoid this kind of problem Android 7 allows notifications to be bundled together into groups.

To bundle notifications, each notification must be designated as belonging to the same group via the *setGroup()* method, and an additional notification must be issued and configured as being the *summary notification*. The following code, for example, creates and issues the three notifications shown in Figure 51-8 above, but bundles them into the same group. The code also issues a notification to act as the summary:

```
final static String GROUP_KEY_NOTIFY = "group_key_notify";

NotificationCompat.Builder builderSummary =
        new NotificationCompat.Builder(this)
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .setContentTitle("A Bundle Example")
                .setContentText("You have 3 new messages")
                .setGroup(GROUP_KEY_NOTIFY)
                .setGroupSummary(true);

NotificationCompat.Builder builder1 =
        new NotificationCompat.Builder(this)
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .setContentTitle("New Message")
                .setContentText("You have a new message from Kassidy")
                .setGroup(GROUP_KEY_NOTIFY);
```

```
NotificationCompat.Builder builder2 =
        new NotificationCompat.Builder(this)
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .setContentTitle("New Message")
                .setContentText("You have a new message from Caitlyn")
                .setGroup(GROUP_KEY_NOTIFY);

NotificationCompat.Builder builder3 =
        new NotificationCompat.Builder(this)
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .setContentTitle("New Message")
                .setContentText("You have a new message from Jason")
                .setGroup(GROUP_KEY_NOTIFY);
.
.
.
int notificationId0 = 100;
int notificationId1 = 101;
int notificationId2 = 102;
int notificationId3 = 103;

NotificationManager notifyMgr =
        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

notifyMgr.notify(notificationId1, builder1.build());
notifyMgr.notify(notificationId2, builder2.build());
notifyMgr.notify(notificationId3, builder3.build());
notifyMgr.notify(notificationId0, builderSummary.build());
```

When the code is executed, a single notification icon will appear in the status bar even though four notifications have actually been issued by the app. Within the notification shade, a single summary notification is displayed listing the information in each of the bundled notifications:

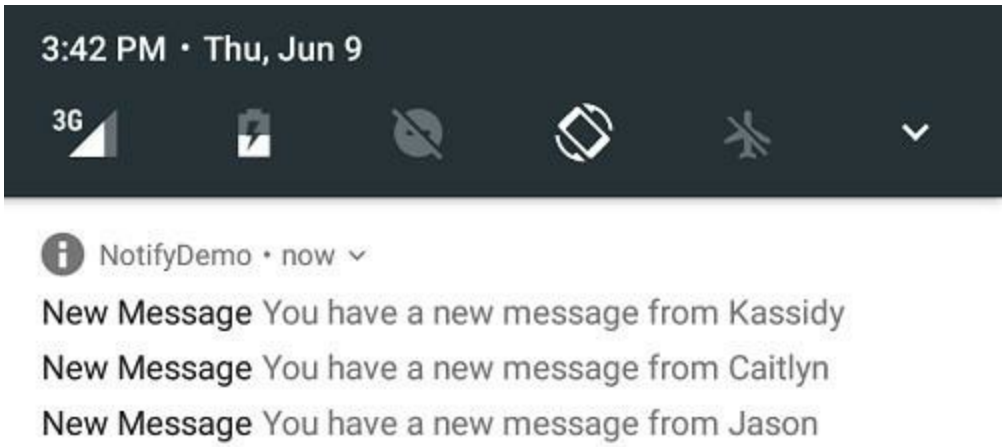

Figure 51-9

Pulling further downward on the notification shade expands the panel to show the details of each of the bundled notifications:

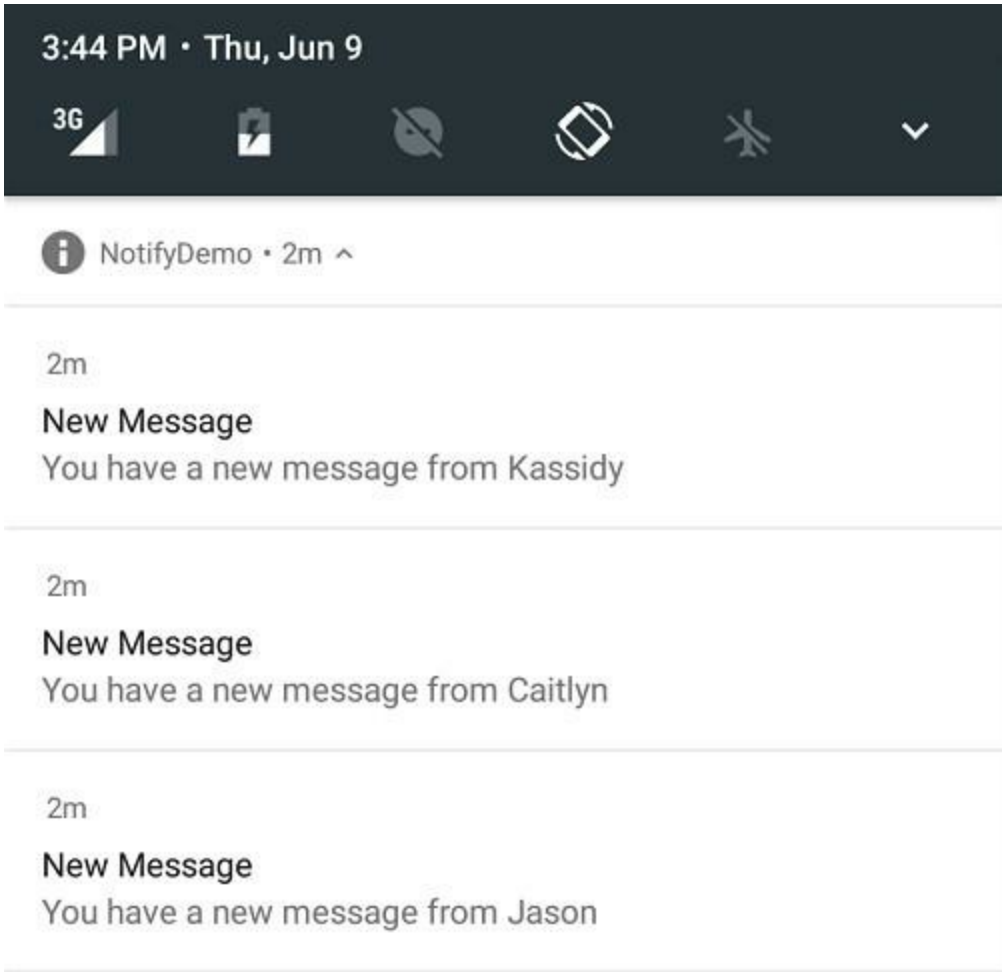

Figure 51-10

51.10 **Summary**

Notifications provide a way for an app to deliver a message to the user when the app is not running, or is currently in the background. Notifications appear in the status bar and notification shade. Local notifications are triggered on the device by the running app while remote notifications are initiated by a remote server and delivered to the device. Local notifications are created using the NotificationCompat.Builder class and issued using the NotificationManager service.

As demonstrated in this chapter, notifications can be configured to provide the user with options (such as launching an activity or saving a message) by making use of actions, intents and the PendingIntent class. Notification bundling provides a mechanism for grouping together notifications to provide an improved experience for apps that issue a greater number of notifications.

52. An Android 7 Direct Reply Notification Tutorial

Direct reply is a feature introduced in Android 7 that allows the user to enter text into a notification and send it to the app associated with that notification. This allows the user to reply to a message in the notification without the need to launch an activity within the app. This chapter will build on the knowledge gained in the previous chapter to create an example app that makes use of this notification feature.

52.1 Creating the DirectReply Project

Start Android Studio and create a new project, entering *DirectReply* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 24: Android 7.0 (Nougat). Continue through the setup screens, requesting the creation of an Empty Activity named *DirectReplyActivity* with a corresponding layout file named *activity_direct_reply*.

52.2 Designing the User Interface

Load the *activity_direct_reply.xml* layout file into the layout tool. With Autoconnect enabled, add a Button object beneath the existing "Hello World!" label. With the Button widget selected in the layout, use the Properties tool window to set the onClick property to call a method named *sendNotification*. If necessary, use the Infer Constraints button to add any missing constraints to the layout.

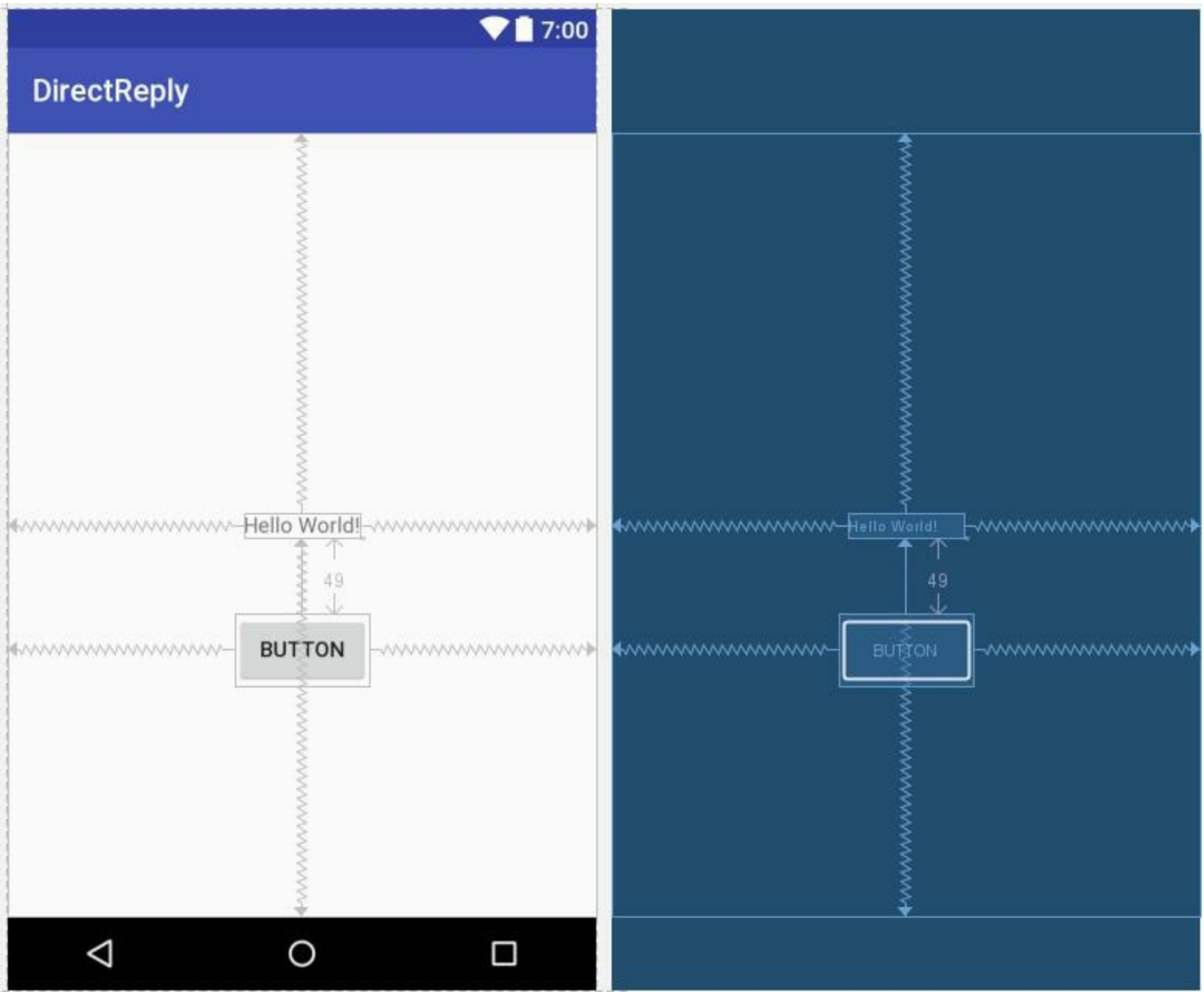

Figure 52-1

52.3 Building the RemoteInput Object

The key element that makes direct reply inline text possible within a notification is the RemoteInput class. The previous chapters introduced the PendingIntent class and explained the way in which it allows one application to create an intent and then grant other applications or services the ability to launch that intent from outside the original app. In that chapter, entitled *An Android 7 Notifications Tutorial*, a pending intent was created that allowed an activity in the original app to be launched from within a notification. The RemoteInput class allows a request for user input to be included in the PendingIntent object along with the intent. When the intent within the PendingIntent object is triggered, for example launching an activity, that activity is also passed any input provided by the user.

The first step in implementing direct reply within a notification is to create the RemoteInput object. This is achieved using the *RemoteInput.Builder()* method. To build a RemoteInput object, a key string is required that will be used to extract the input from the resulting intent. The object also needs a label string that will appear within the text input field of the notification. Edit the *DirectReplyAction.java* file and begin implementing the *sendNotification()* method. Note also the addition of some import directives and variables that will be used later as the chapter progresses:

```
package com.ebookfrenzy.directreply;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.support.v4.app.RemoteInput;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class DirectReplyActivity extends AppCompatActivity {

    private static int notificationId = 101;
    private static String KEY_TEXT_REPLY = "key_text_reply";
.
.
.
    public void sendNotification(View view) {

        String replyLabel = "Enter your reply here";
        RemoteInput remoteInput =
           new RemoteInput.Builder(KEY_TEXT_REPLY)
                .setLabel(replyLabel)
                .build();
    }
.
.
.
}
```

Now that the RemoteInput object has been created and initialized with a key and a label string it will need to be placed inside a notification action object. Before that step can be performed, however, the PendingIntent instance needs to be created.

52.4 Creating the PendingIntent

The steps to creating the PendingIntent are the same as those outlined in the *An Android 7 Notifications Tutorial* chapter, with the exception that the intent will be configured to launch the main DirectReplyActivity activity. Remaining within the *DirectReplyActivity.java* file, add the code to create the PendingIntent as follows:

```
public void sendNotification(View view) {

    String replyLabel = "Enter your reply here";
    RemoteInput remoteInput =
            new RemoteInput.Builder(KEY_TEXT_REPLY)
            .setLabel(replyLabel)
            .build();

    Intent resultIntent = new Intent(this, DirectReplyActivity.class);
```

```
        PendingIntent resultPendingIntent =
                PendingIntent.getActivity(
                        this,
                        0,
                        resultIntent,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );
        }
    .
    .
    .
    }
```

52.5 Creating the Reply Action

The inline reply will be accessible within the notification via an action button. This action now needs to be created and configured with an icon, a label to appear on the button, the PendingIntent object and the RemoteInput object. Modify the *sendNotification()* method to add the code to create this action:

```
    public void sendNotification(View view) {

        String replyLabel = "Enter your reply here";
        RemoteInput remoteInput =
                new RemoteInput.Builder(KEY_TEXT_REPLY)
                .setLabel(replyLabel)
                .build();

        Intent resultIntent = new Intent(this, DirectReplyActivity.class);

        PendingIntent resultPendingIntent =
                PendingIntent.getActivity(
                        this,
                        0,
                        resultIntent,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );

        NotificationCompat.Action replyAction =
                new NotificationCompat.Action.Builder(
                        android.R.drawable.ic_dialog_info,
                        "Reply", resultPendingIntent)
                        .addRemoteInput(remoteInput)
                        .build();
        }
    .
    .
    .
    }
```

At this stage in the tutorial we have the RemoteInput, PendingIntent and Notification Action objects built and ready to be used. The next stage is to build the notification and issue it:

```
    public void sendNotification(View view) {
```

```
        String replyLabel = "Enter your reply here";
        RemoteInput remoteInput =
                new RemoteInput.Builder(KEY_TEXT_REPLY)
                .setLabel(replyLabel)
                .build();

        Intent resultIntent =
             new Intent(this, DirectReplyActivity.class);

        PendingIntent resultPendingIntent =
                PendingIntent.getActivity(
                        this,
                        0,
                        resultIntent,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );

        NotificationCompat.Action replyAction =
                new NotificationCompat.Action.Builder(
                        android.R.drawable.ic_dialog_info,
                        "Reply", resultPendingIntent)
                        .addRemoteInput(remoteInput)
                        .build();

        Notification newMessageNotification =
                new NotificationCompat.Builder(this)
                    .setColor(ContextCompat.getColor(this,
                          R.color.colorPrimary))
                    .setSmallIcon(
                          android.R.drawable.ic_dialog_info)
                    .setContentTitle("My Notification")
                    .setContentText("This is a test message")
                    .addAction(replyAction).build();

        NotificationManager notificationManager =
                (NotificationManager)
                   getSystemService(Context.NOTIFICATION_SERVICE);

        notificationManager.notify(notificationId,
                      newMessageNotification);
    }
```

With the changes made, compile and run the app and test that tapping the button successfully issues the notification. When viewing the notification shade, the notification should appear as shown in Figure 52-2:

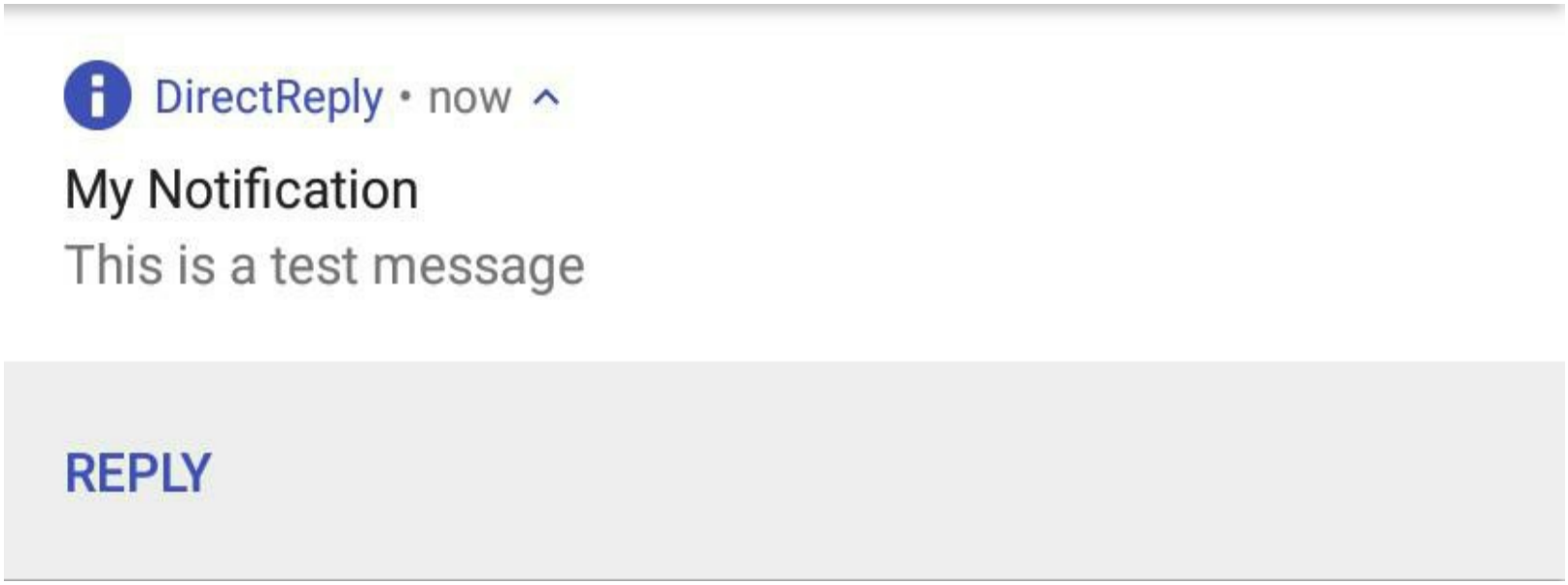

Figure 52-2

Tap the Reply action button so that the text input field appears displaying the reply label that was embedded into the RemoteInput object when it was created.

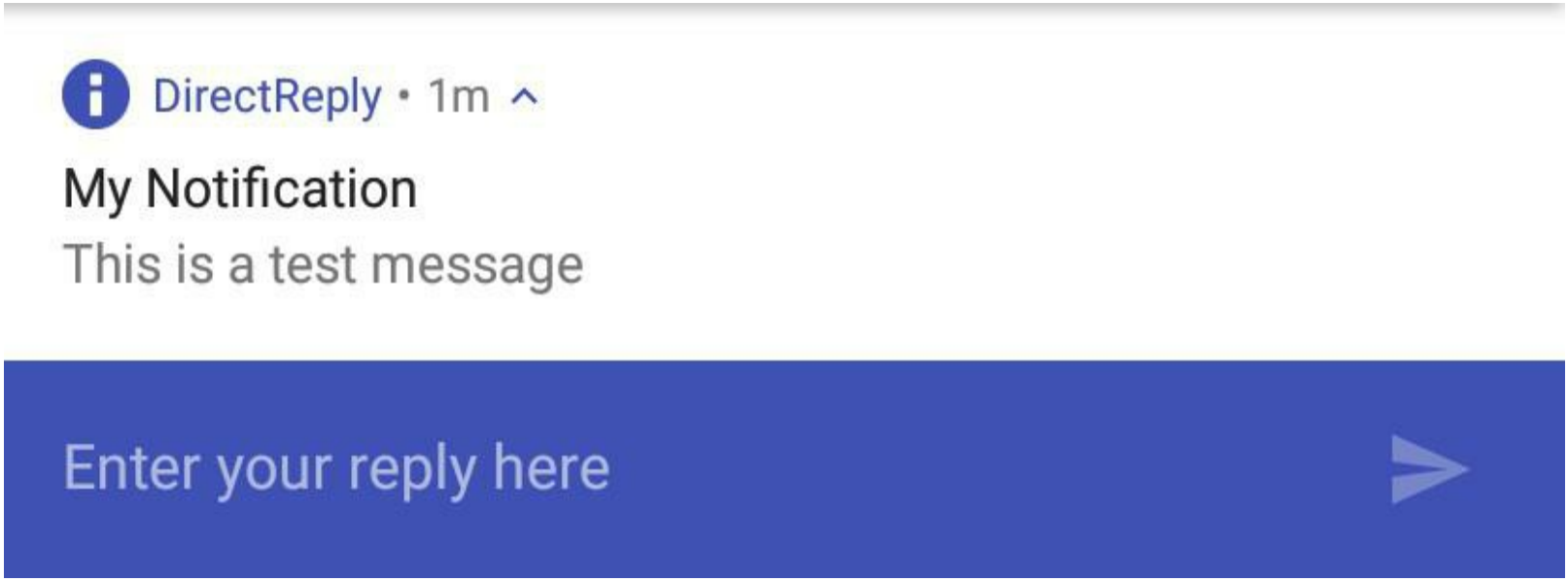

Figure 52-3

Enter some text, tap the send arrow button located at the end of the input field.

52.6 Receiving Direct Reply Input

Now that the notification is successfully seeking input from the user, the app needs to do something with that input. The goal of this particular tutorial is to have the text entered by the user into the notification appear on the TextView widget in the activity user interface.

When the user enters text and taps the send button the DirectReplyActivity activity is launched via the intent contained in the PendingIntent object. Embedded in this intent is the text entered by the user via the notification. Within the *onCreate()* method of the activity, a call to the *getIntent()* method will return a copy of the intent that launched the activity. Passing this through to the *RemoteInput.getResultsFromIntent()* method will, in turn, return a Bundle object containing the reply text which can be extracted and assigned to the TextView widget. This results in a modified *onCreate()* method within the *DirectReplyActivity.java* file which reads as follows:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_direct_reply);

    Intent intent = this.getIntent();

    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);

    if (remoteInput != null) {

        TextView myTextView = (TextView) findViewById(R.id.textView);
        String inputString = remoteInput.getCharSequence(
              KEY_TEXT_REPLY).toString();

        myTextView.setText(inputString);
    }
}
```

After making these code changes build and run the app once again. Click the button to issue the notification and enter and send some text from within the notification panel. Note that the TextView widget in the DirectReplyActivity activity is updated to display the inline text that was entered.

52.7 Updating the Notification

After sending the reply within the notification you may have noticed that the progress indicator continues to spin within the notification panel as highlighted in Figure 52-4:

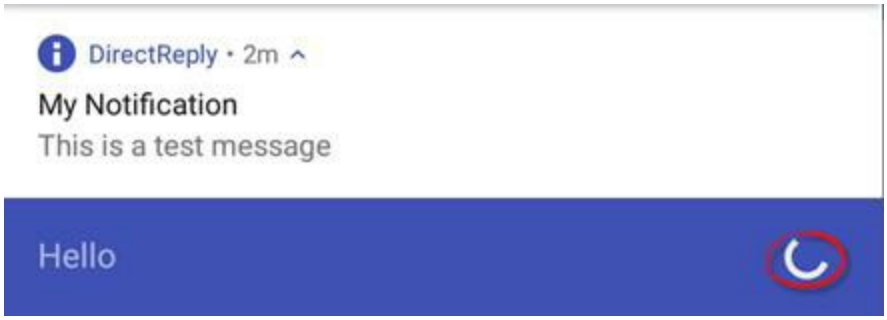

Figure 52-4

The notification is showing this indicator because it is waiting for a response from the activity confirming receipt of the sent text. The recommended approach to performing this task is to update the notification with a new message indicating that the reply has been received and handled. Since the original notification was assigned an ID when it was issued, this can be used once again to perform an update. Add the following code to the *onCreate()* method to perform this task:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_direct_reply);

    Intent intent = this.getIntent();

    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);

    if (remoteInput != null) {
```

```
            TextView myTextView = (TextView) findViewById(R.id.textView);
            String inputString = remoteInput.getCharSequence(
                        KEY_TEXT_REPLY).toString();

            myTextView.setText(inputString);

            Notification repliedNotification =
                    new Notification.Builder(this)
                            .setSmallIcon(
                                  android.R.drawable.ic_dialog_info)
                            .setContentText("Reply received")
                            .build();

            NotificationManager notificationManager =
                    (NotificationManager)
                          getSystemService(Context.NOTIFICATION_SERVICE);
            notificationManager.notify(notificationId,
                          repliedNotification);
        }
    }
```

Test the app one last time and verify that the progress indicator goes away after the inline reply text has been sent.

52.8 **Summary**

The direct reply notification feature allows text to be entered by the user within a notification and passed via an intent to an activity of the corresponding application. Direct reply is made possible by the RemoteInput class, an instance of which can be embedded within an action and bundled with the notification. When working with direct reply notifications, it is important to let the NotificationManager service know that the reply has been received and processed. The best way to achieve this is to simply update the notification message using the notification ID provided when the notification was first issued.