52. An Android 8 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 26: Android 8.0 (Oreo). 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 Attributes 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.

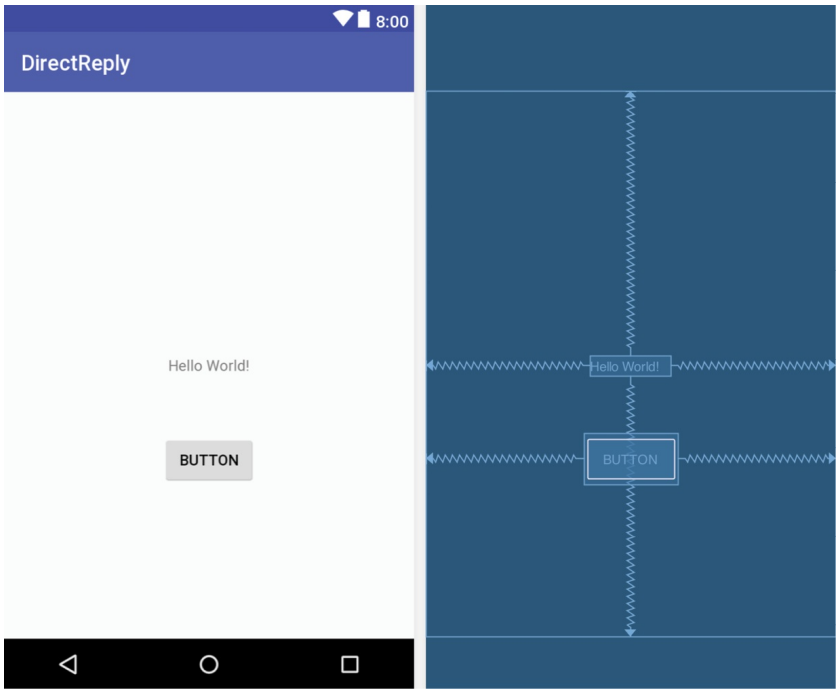

Figure 52-1

Before continuing, select the "Hello World!" TextView and change ID attribute to *textView*.

52.3 Creating the Notification Channel

As with the example in the previous chapter, a channel must be created before a notification can be sent. Edit the *DirectReplyActivity.java* file and add code to create a new channel as follows:

```
.
.
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.Context;
import android.graphics.Color;
.
.
public class DirectReplyActivity extends AppCompatActivity {

    NotificationManager notificationManager;
    private String channelID = "com.ebookfrenzy.directreply.news";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_direct_reply);

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

        createNotificationChannel(channelID,
                     "DirectReply News", "Example News Channel");
    }

    protected void createNotificationChannel(String id,
              String name, String description) {

        int importance = NotificationManager.IMPORTANCE_HIGH;
        NotificationChannel channel =
              new NotificationChannel(id, name, importance);

        channel.setDescription(description);
        channel.enableLights(true);
        channel.setLightColor(Color.RED);
        channel.enableVibration(true);
        channel.setVibrationPattern(new long[]{100, 200, 300, 400,
                            500, 400, 300, 200, 400});

        notificationManager.createNotificationChannel(channel);
    }
.
.
}
```

52.4 Building the RemoteInput Object

The key element that makes direct reply in-line 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 8 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.view.View;
import android.app.PendingIntent;
import android.app.RemoteInput;
import android.content.Intent;
.
.
public class DirectReplyActivity extends AppCompatActivity {

    private static int notificationId = 101;
    private static String KEY_TEXT_REPLY = "key_text_reply";
    private static String channelID =
                    "com.ebookfrenzy.directreply.news";

    NotificationManager notificationManager;
.
.
.
    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.

52.5 Creating the PendingIntent

The steps to creating the PendingIntent are the same as those outlined in the *"An Android 8 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
            );
    }
}
```

52.6 Creating the Reply Action

The in-line 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:

.
.

```
import android.graphics.drawable.Icon;
import android.app.Notification;
import android.graphics.Color;
import android.support.v4.content.ContextCompat;
.
.
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
            );

    final Icon icon =
              Icon.createWithResource(DirectReplyActivity.this,
                android.R.drawable.ic_dialog_info);

        Notification.Action replyAction =
                new Notification.Action.Builder(
                        icon,
                        "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
            );

    final Icon icon =
       Icon.createWithResource(DirectReplyActivity.this,
            android.R.drawable.ic_dialog_info);

    Notification.Action replyAction =
            new Notification.Action.Builder(
                    icon,
                    "Reply", resultPendingIntent)
                    .addRemoteInput(remoteInput)
                    .build();

    Notification newMessageNotification =
            new Notification.Builder(this, channelID)
                    .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 drawer, the notification should appear as shown in Figure 52-2:

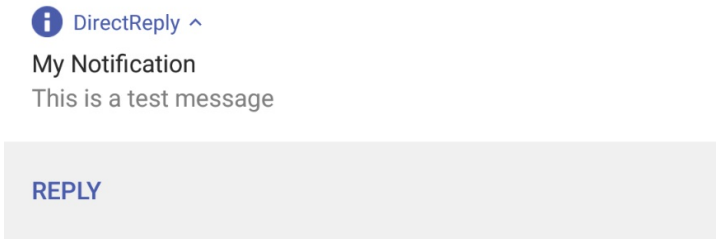

Figure 52-2

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.

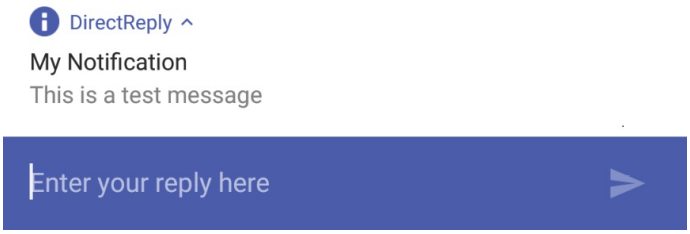

Figure 52-3

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

52.7 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:

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

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

    createNotificationChannel(channelID, "DirectReply News",
                            Example News Channel");

    handleIntent();

}

private void handleIntent() {

    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 in-line text that was entered.

52.8 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:

```
private void handleIntent() {

    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, channelID)
                        .setSmallIcon(
                                android.R.drawable.ic_dialog_info)
                        .setContentText("Reply received")
                        .build();

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

Test the app one last time and verify that the progress indicator goes away after the in-line reply text has been sent and that a new panel appears indicating that the reply has been received:

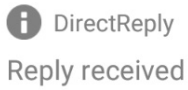

Figure 52-5

52.9 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.