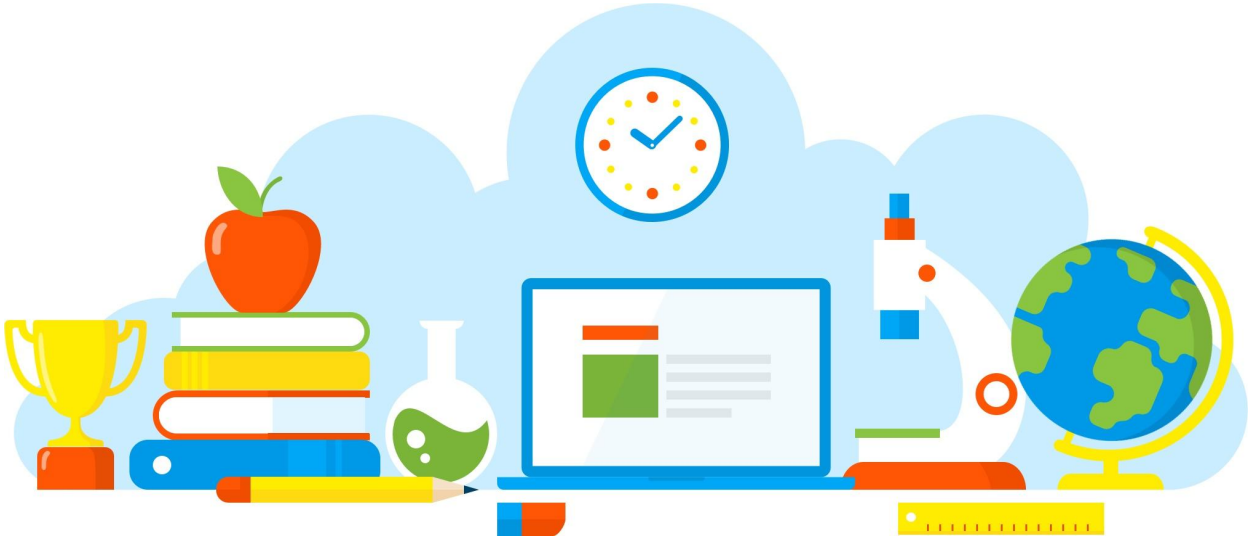


Implement an application that send a SMS and creates an alert upon receiving the SMS

Introduction

Android smartphones can send and receive messages to or from any other phone that supports Short Message Service (SMS). You have two choices for *sending* SMS messages:

- Use an implicit Intent to launch a messaging app with the ACTION_SENDTO intent action.
    - This is the simplest choice for sending messages. The user can add a picture or other attachment in the messaging app, if the messaging app supports adding attachments.
    - Your app doesn't need code to request permission from the user.
    - If the user has multiple SMS messaging apps installed on the Android phone, the App chooser will appear with a list of these apps, and the user can choose which one to use. (Android smartphones will have at least one, such as Messenger.)
    - The user can change the message in the messaging app before sending it.
    - The user navigates back to your app using the **Back** button.
- Send the SMS message using the sendTextMessage() method or other methods of the SmsManager class.
    - This is a good choice for sending messages from your app without having to use another installed app.
    - Your app must ask the user for permission before sending the SMS message, if the user hasn't already granted permission.
    - The user stays in your app during and after sending the message.
    - You can manage SMS operations such as dividing a message into fragments, sending a multipart message, get carrier-dependent

configuration values, and so on.

To *receive* SMS messages, use the onReceive() method of the BroadcastReceiver class.

Steps:

- Create an onClick method for a button with the `android:onClick` attribute.
- Use an implicit intent to perform a function with another app.
- Use a broadcast receiver to receive system events.

Detailed Implementation:

- Launch an SMS messaging app from your app with a phone number and message.
- Send an SMS message from within an app.
- Check for the SMS permission, and request permission if necessary.
- Receive SMS events using a broadcast receiver.
- Extract an SMS message from an SMS event.

Sending and receiving SMS messages

Access to the SMS features of an Android device is protected by user permissions. Just as your app needs the user's permission to use phone features, so also does an app need the user's permission to directly use SMS features.

However, your app doesn't need permission to pass a phone number to an installed SMS app, such as Messenger, for sending the message. The Messenger app itself is governed by user permission.

You have two choices for *sending* SMS messages:

- Use an implicit Intent to launch a messaging app such as Messenger, with the ACTION_SENDTO action.
    - This is the simplest choice for sending messages. The user can add a picture or other attachment in the messaging app, if the messaging app supports adding attachments.
    - Your app doesn't need code to request permission from the user.
    - If the user has multiple SMS messaging apps installed on the Android phone, the App chooser will appear with a list of these apps, and the user can choose which one to use. (Android smartphones will have at least one, such as Messenger.)
    - The user can change the message in the messaging app before sending it.
    - The user navigates back to your app using the **Back** button.
- Send the SMS message using the sendTextMessage() method or other methods of the SmsManager class.
    - This is a good choice for sending messages from your app without having to use another installed app.
    - Your code must ask the user for permission before sending the message if the user hasn't already granted permission.
    - The user stays in your app during and after sending the message.
    - You can manage SMS operations such as dividing a message into fragments, sending a multipart message, get carrier-dependent configuration values, and so on.

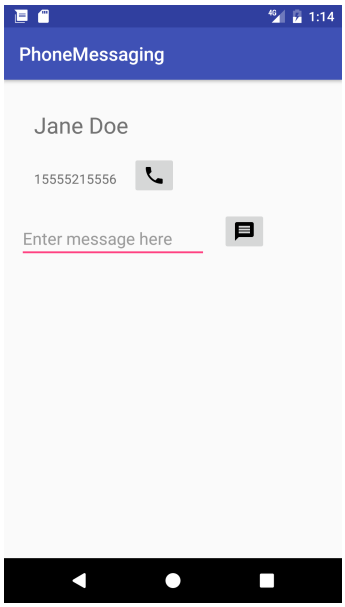
To *receive* SMS messages, the best practice is to use the onReceive() method of the BroadcastReceiver class. The Android framework sends out system broadcasts of events such as receiving an SMS message, containing intents that are meant to be received using a BroadcastReceiver. Your app receives SMS messages by listening for the SMS_RECEIVED_ACTION broadcast.

Using an intent to launch an SMS app

To use an Intent to launch an SMS app, your app needs to prepare a Uniform Resource Identifier (URI) for the phone number as a string prefixed by "smsto:" (as in `smsto:14155551212`). You can use a hardcoded phone number, such as the phone number of a support message center, or provide an EditText field in the layout to enable the user to enter a phone number.

Tip: For details about using methods in the PhoneNumberUtils class to format a phone number string, see the related concept Phone Calls.

Use a button (such as an ImageButton) that the user can tap to pass the phone number to the SMS app. For example, an app that enables a user make a phone call and/or send a message to the phone number might offer a simple layout with a phone icon button for calling, and a messaging icon button for sending a message, as shown in the figure below.



The following code demonstrates how to perform an implicit intent to send a message:

```
public void smsSendMessage(View view) {
    // Find the TextView number_to_call and assign it to textView.
    TextView textView = (TextView) findViewById(R.id.number_to_call);
    // Concatenate "smsto:" with phone number to create smsNumber.
    String smsNumber = "smsto:" + textView.getText().toString();
    // Find the sms_message view.
    EditText smsEditText = (EditText) findViewById(R.id.sms_message);
    // Get the text of the sms message.
    String sms = smsEditText.getText().toString();
    // Create the intent.
    Intent smsIntent = new Intent(Intent.ACTION_SENDTO);
    // Set the data for the intent as the phone number.
    smsIntent.setData(Uri.parse(smsNumber));
    // Add the message (sms) with the key ("sms_body").
    smsIntent.putExtra("sms_body", sms);
    // If package resolves (target app installed), send intent.
    if (smsIntent.resolveActivity(getPackageManager()) != null) {
        startActivity(smsIntent);
    } else {
        Log.e(TAG, "Can't resolve app for ACTION_SENDTO Intent.");
    }
}
```

Sending SMS messages from your app

To send an SMS message from your app, use the sendTextMessage() method of the SmsManager class. Perform these steps to enable sending messages from within your app:

1. Add the `SEND_SMS` permission to send SMS messages.
2. Check to see if the user continues to grant permission. If not, request permission.
3. Use the `sendTextMessage()` method of the SmsManager class.

Checking for user permission

Beginning in Android 6.0 (API level 23), users grant permissions to apps while the app is running, not when they install the app. This approach streamlines the app install process, since the user does not need to grant permissions when they install or update the app. It also gives the user more control over the app's functionality. However, your app must check for permission every time it does something that requires permission (such as sending an SMS message). If the user has used the Settings app to turn off SMS permissions for the app, your app can display a dialog to request permission.

Tip: For a complete description of the request permission process, see Requesting Permissions at Run Time.

Add the `SEND_SMS` permission to the AndroidManifest.xml file after the first line (with the `package` definition) and before the `<application>` section:

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Because the user can turn permissions on or off for each app, your app must check whether it still has permission every time it does something that requires permission (such as sending an SMS message). If the user has turned SMS permission off for the app, your app can display a dialog to request permission.

Follow these steps:

At the top of the activity that sends an SMS message, and below the activity's class definition, define a constant variable to hold the request code, and set it to an integer:

```
private static final int MY_PERMISSIONS_REQUEST_SEND_SMS = 1;
```

1. Why the integer 1? Each permission request needs three parameters: the `context`, a string array of permissions, and an integer `requestCode`. The `requestCode` is the integer attached to the request. When a result returns in the activity, it contains this code and uses it to differentiate multiple permission results from each other.

In the activity that makes a phone call, create a method that uses the checkSelfPermission() method to determine whether your app has been granted the permission:

```
private void checkForSmsPermission() {

   if (ActivityCompat.checkSelfPermission(this,

                    Manifest.permission.SEND_SMS) !=

                    PackageManager.PERMISSION_GRANTED) {

       Log.d(TAG, getString(R.string.permission_not_granted));

       // Permission not yet granted. Use requestPermissions().

       // MY_PERMISSIONS_REQUEST_SEND_SMS is an

       // app-defined int constant. The callback method gets the

       // result of the request.

       ActivityCompat.requestPermissions(this,

                    new String[]{Manifest.permission.SEND_SMS},

                    MY_PERMISSIONS_REQUEST_SEND_SMS);
```

```
    } else {
        // Permission already granted. Enable the message button.
        enableSmsButton();
    }
}
```

2. The code uses `checkSelfPermission()` to determine whether your app has been granted a particular permission by the user. If permission has *not* been granted, the code uses the requestPermissions() method to display a standard dialog for the user to grant permission.

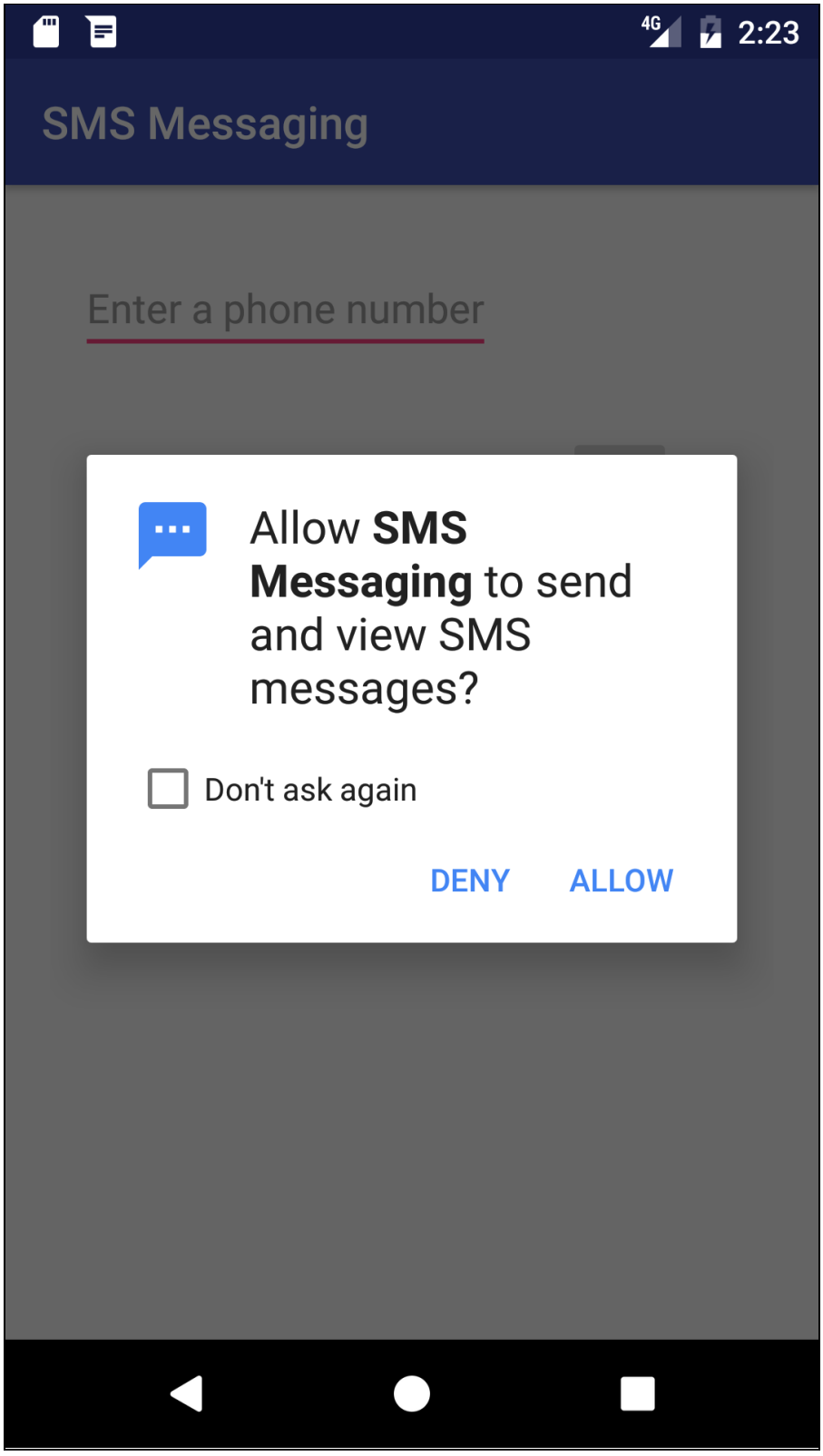
Use your `checkForSmsPermission()` method to check for permission at the following times:

- When the activity starts—in its `onCreate()` method.
- Every time before sending a message. Since the user might turn off the SMS permission while the app is still running, call the `checkForSmsPermission()` method in the `smsSendMessage()` method before using the `SmsManager` class.

Requesting user permission

If permission has *not* been granted by the user, use the requestPermissions() method of the `ActivityCompat` class. The `requestPermissions()` method needs three parameters: the context (`this`), a string array of permissions (`new String[]{Manifest.permission.SEND_SMS}`), and the predefined integer `MY_PERMISSIONS_REQUEST_SEND_SMS` for the `requestCode`.

When your app calls `requestPermissions()`, the system shows a standard dialog to the user, as shown in the figure below.



When the user responds to the request permission dialog by tapping **Deny** or **Allow**, the system invokes the onRequestPermissionsResult() method, passing it the user response. Your app has to override that method to find out whether the permission was granted.

The following code demonstrates how you can use a `switch` statement in your implementation of `onRequestPermissionsResult()` based on the value of `requestCode`. The user's response to the request dialog is returned in the `permissions` array (index `0` if only one permission is requested in the dialog). This is compared to the corresponding grant result, which is either `PERMISSION_GRANTED` or `PERMISSION_DENIED`.

If the user denies a permission request, your app should disable the functionality that depends on this permission and show a dialog explaining why it could not perform it. The code below logs a debug message, displays a toast to show that permission was not granted, and disables the message icon used as a button.

```
@Override
public void onRequestPermissionsResult(int requestCode,
                    String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_SEND_SMS: {
            if (permissions[0].equalsIgnoreCase(Manifest.permission.SEND_SMS)
                        && grantResults[0] ==
                        PackageManager.PERMISSION_GRANTED) {
                // Permission was granted.
            } else {
                // Permission denied.
                Log.d(TAG, getString(R.string.failure_permission));
                Toast.makeText(MainActivity.this,
                            getString(R.string.failure_permission),
                                    Toast.LENGTH_SHORT).show();
```

```
                // Disable the message button.
                disableSmsButton();
            }
        }
    }
}
```

Receiving SMS messages

To receive SMS messages, use the onReceive() method of the BroadcastReceiver class. The Android framework sends out system broadcasts of events such as `SMS_RECEIVED` for receiving an SMS message. You must also include `RECEIVE_SMS` permission in your project's AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

To use a broadcast receiver:

1. Add the broadcast receiver by choosing **File > New > Other > Broadcast Receiver**. The `<receiver...</receiver>` tags are automatically added to the AndroidManifest.xml file.
2. Register the receiver by adding an *intent filter* within the `<receiver...</receiver>` tags to specify the type of broadcast intent you want to receive.
3. Implement the `onReceive()` method.

Adding a broadcast receiver

You can perform the first step by selecting the package name in the Project:Android: view and choosing **File > New > Other > Broadcast Receiver**. Make sure "Exported"

and "Enabled" are checked. The "Exported" option allows your app to respond to outside broadcasts, while "Enabled" allows it to be instantiated by the system.

Android Studio automatically generates a `<receiver>` tag in the app's AndroidManifest.xml file, with your chosen options as attributes:

```
<receiver
    android:name="com.example.android.phonecallingsms.MySmsReceiver"
    android:enabled="true"
    android:exported="true">
</receiver>
```

Registering the broadcast receiver

In order to receive any broadcasts, you must register for specific broadcast intents. In the Intent documentation, under "Standard Broadcast Actions", you can find some of the common broadcast intents sent by the system.

The following intent filter registers the receiver for the `android.provider.Telephony.SMS_RECEIVED` intent:

```
<receiver
    android:name="com.example.android.smsmessaging.MySmsReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

Implementing the onReceive() method

Once your app's BroadcastReceiver intercepts a broadcast it is registered for (`SMS_RECEIVED`), the intent is delivered to the receiver's `onReceive()` method, along with the context in which the receiver is running.

The following shows the first part of the `onReceive()` method, which does the following:

- Retrieves the extras (the SMS message) from the intent.
- Stores it in a `bundle`.
- Defines the `msgs` array and `strMessage` string.
- Gets the `format` for the message from the `bundle` in order to use it with createFromPdu() to create the SmsMessage.

The `format` is the message's mobile telephony system format passed in an `SMS_RECEIVED_ACTION` broadcast. It is usually "3gpp" for GSM/UMTS/LTE messages in the 3GPP format, or "3gpp2" for CDMA/LTE messages in 3GPP2 format.

```
@Override
public void onReceive(Context context, Intent intent) {
    // Get the SMS message.
    Bundle bundle = intent.getExtras();
    SmsMessage[] msgs;
    String strMessage = "";
    String format = bundle.getString("format");
    // Retrieve the SMS message received.
    ...
}
```

The `onReceive()` method then retrieves from the bundle one or more pieces of data in the PDU:

```
...

// Retrieve the SMS message received.

Object[] pdus = (Object[]) bundle.get("pdus");

if (pdus != null) {

    // Fill the msgs array.

    msgs = new SmsMessage[pdus.length];

    for (int i = 0; i < msgs.length; i++) {
        ...
```