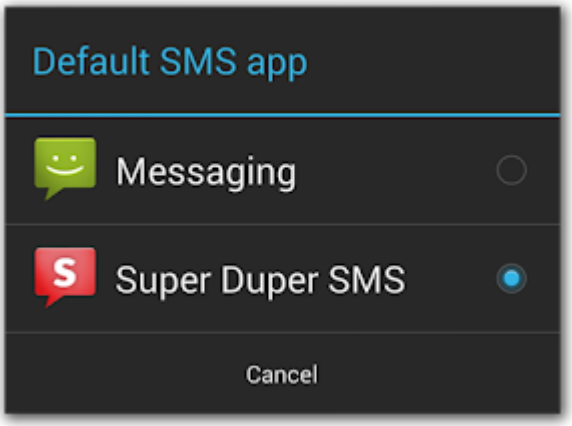
Getting Your SMS Apps Ready for KitKat

Posted by Scott Main and David Braun

Sending and receiving SMS messages are fundamental features on mobile devices and many developers have built successful apps that enhance this experience on Android. Some of you have built SMS apps using hidden APIs—a practice we discourage because hidden APIs may be changed or removed and new devices are not tested against them for compatibility. So, to provide you with a fully supported set of APIs for building SMS apps and to make the user experience for messaging more predictable, Android 4.4 (KitKat) makes the existing APIs public and adds the concept of a *default SMS app*, which the user can select in system settings.

This means that if you are using the hidden SMS APIs on previous platform versions, you need to make some adjustments so your app continues to work when Android 4.4 is released later this year.

Make your app the default SMS app



On Android 4.4, only one app can receive the new `SMS_DELIVER_ACTION` intent, which the system broadcasts when a new SMS message arrives. Which app receives this broadcast is determined by which app the user has selected as the default SMS app in system settings. Likewise, only the default SMS app receives the new `WAP_PUSH_DELIVER_ACTION` intent when a new MMS arrives.

Other apps that only want to read new messages can instead receive the `SMS_RECEIVED_ACTION` broadcast intent when a new SMS arrives. However, **only the app that receives the `SMS_DELIVER_ACTION` broadcast (the user-specified default SMS app) is able to write to the SMS Provider** defined by the `android.provider.Telephony` class and subclasses. As such, it's important that you update your messaging app as soon as possible to be available as a default SMS app, because although your existing app won't crash on an Android 4.4 device, it will silently fail when attempting to write to the SMS Provider.

In order for your app to appear in the system settings as an eligible default SMS app, your manifest file must declare some specific capabilities. So you must update your app to do the following things:

- In a broadcast receiver, include an intent filter for `SMS_DELIVER_ACTION` (`"android.provider.Telephony.SMS_DELIVER"`). The broadcast receiver must also require the BROADCAST_SMS permission.

  This allows your app to directly receive incoming SMS messages.

- In a broadcast receiver, include an intent filter for `WAP_PUSH_DELIVER_ACTION` (`"android.provider.Telephony.WAP_PUSH_DELIVER"`) with the MIME type `"application/vnd.wap.mms-message"` . The broadcast receiver must also require the `BROADCAST_WAP_PUSH` permission.

  This allows your app to directly receive incoming MMS messages.

- In your activity that delivers new messages, include an intent filter for `ACTION_SENDTO` (`"android.intent.action.SENDTO"`) with schemas, `sms:` , `smsto:` , `mms:` , and `mmsto:` .

  This allows your app to receive intents from other apps that want to deliver a message.

- In a service, include an intent filter for `ACTION_RESPONSE_VIA_MESSAGE` (`"android.intent.action.RESPOND_VIA_MESSAGE"`) with schemas, `sms:` , `smsto:` , `mms:` , and `mmsto:` . This service must also require the `SEND_RESPOND_VIA_MESSAGE` permission.

  This allows users to respond to incoming phone calls with an immediate text message using your app.

For example, here's a manifest file with the necessary components and intent filters:

```
<manifest>
    ...
    <application>
        <!-- BroadcastReceiver that listens for incoming SMS messages -->
        <receiver android:name=".SmsReceiver"
                android:permission="android.permission.BROADCAST_SMS">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_DELIVER" />
            </intent-filter>
        </receiver>

        <!-- BroadcastReceiver that listens for incoming MMS messages -->
        <receiver android:name=".MmsReceiver"
            android:permission="android.permission.BROADCAST_WAP_PUSH">
            <intent-filter>
                <action android:name="android.provider.Telephony.WAP_PUSH_DELIVER" />
                <data android:mimeType="application/vnd.wap.mms-message" />
            </intent-filter>
        </receiver>
```

```
        <!-- Activity that allows the user to send new SMS/MMS messages -->
        <activity android:name=".ComposeSmsActivity" >
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <action android:name="android.intent.action.SENDTO" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:scheme="sms" />
                <data android:scheme="smsto" />
                <data android:scheme="mms" />
                <data android:scheme="mmsto" />
            </intent-filter>
        </activity>

        <!-- Service that delivers messages from the phone "quick response" -->
        <service android:name=".HeadlessSmsSendService"
                 android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE"
                 android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.RESPOND_VIA_MESSAGE" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="sms" />
                <data android:scheme="smsto" />
                <data android:scheme="mms" />
                <data android:scheme="mmsto" />
            </intent-filter>
        </service>
    </application>
</manifest>
```

Any filters for the `SMS_RECEIVED_ACTION` broadcast in existing apps will continue to work the same on Android 4.4, but only as an observer of new messages, because unless your app also receives the `SMS_DELIVER_ACTION` broadcast, you cannot write to the SMS Provider on Android 4.4.

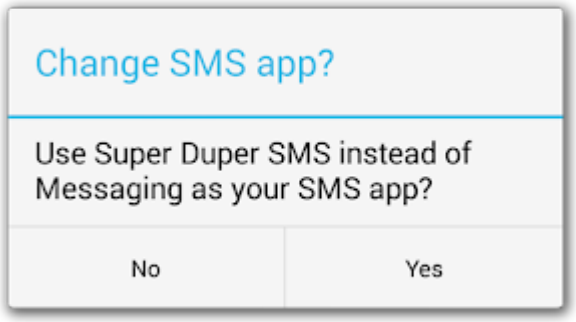
Beginning with Android 4.4, you should stop listening for the `SMS_RECEIVED_ACTION` broadcast, which you can do at runtime by checking the platform version then disabling your broadcast receiver for `SMS_RECEIVED_ACTION` with `PackageManager.setComponentEnabledSetting()`. However, you can continue listening for that broadcast if your app needs only to read special SMS messages, such as to perform phone number verification. Note that—beginning with Android 4.4—any attempt by your app to abort the `SMS_RECEIVED_ACTION` broadcast will be ignored so all apps interested have the chance to receive it.

Tip: To distinguish the two SMS broadcasts, imagine that the `SMS_RECEIVED_ACTION` simply says "the system received an SMS," whereas the `SMS_DELIVER_ACTION` says "the system is delivering your app an SMS, because you're the default SMS app."

Disable features when not the default SMS app

In consideration of some apps that do not want to behave as the default SMS app but still want to send messages, any app that has the `SEND_SMS` permission is still able to send SMS messages using `SmsManager`. If *and only if* an app is *not* selected as the default SMS app on Android 4.4, the system automatically writes the sent SMS messages to the SMS Provider (the default SMS app is always responsible for writing its sent messages to the SMS Provider).

However, if your app is designed to behave as the default SMS app, then while your app is not selected as the default, it's important that you understand the limitations placed upon your app and disable features as appropriate. Although the system writes sent SMS messages to the SMS Provider while your app is not the default SMS app, it does *not* write sent MMS messages and your app is *not* able to write to the SMS Provider for other operations, such as to mark messages as draft, mark them as read, delete them, etc.



So when your messaging activity resumes, check whether your app is the default SMS app by querying `Telephony.Sms.getDefaultSmsPackage()`, which returns the package name of the current default SMS app. If it doesn't match your package name, disable features such as the ability for users to send new messages.

When the user decides to use your app for messaging, you can display a dialog hosted by the system that allows the user to make your app the default SMS app. To display the dialog, call `startActivity()` with the `Telephony.Sms.Intents.ACTION_CHANGE_DEFAULT` intent, including an extra with the `Sms.Intents.EXTRA_PACKAGE_NAME` key and your package name as the string value.

For example, your activity might include code like this:

```
public class ComposeSmsActivity extends Activity {

    @Override
    protected void onResume() {
        super.onResume();

        final String myPackageName = getPackageName();
        if (!Telephony.Sms.getDefaultSmsPackage(this).equals(myPackageName)) {
            // App is not default.
            // Show the "not currently set as the default SMS app" interface
```

```
            View viewGroup = findViewById(R.id.not_default_app);
            viewGroup.setVisibility(View.VISIBLE);

            // Set up a button that allows the user to change the default SMS app
            Button button = (Button) findViewById(R.id.change_default_app);
            button.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
                    Intent intent =
                            new Intent(Telephony.Sms.Intents.ACTION_CHANGE_DEFAULT);
                    intent.putExtra(Telephony.Sms.Intents.EXTRA_PACKAGE_NAME,
                            myPackageName);
                    startActivity(intent);
                }
            });
        } else {
            // App is the default.
            // Hide the "not currently set as the default SMS app" interface
            View viewGroup = findViewById(R.id.not_default_app);
            viewGroup.setVisibility(View.GONE);
        }
    }
}
```

Advice for SMS backup & restore apps

Because the ability to write to the SMS Provider is restricted to the app the user selects as the default SMS app, any existing app designed purely to backup and restore SMS messages will currently be unable to restore SMS messages on Android 4.4. An app that backs up and restores SMS messages must also be set as the default SMS app so that it can write messages in the SMS Provider. However, if the app does not also send and receive SMS messages, then it should not remain set as the default SMS app. So, you can provide a functional user experience with the following design when the user opens your app to initiate a one-time restore operation:

1. Query the current default SMS app's package name and save it.

   ```
   String defaultSmsApp = Telephony.Sms.getDefaultSmsPackage(context);
   ```

2. Request the user change the default SMS app to your app in order to restore SMS messages (you must be the default SMS app in order to write to the SMS Provider).

   ```
   Intent intent = new Intent(context, Sms.Intents.ACTION_CHANGE_DEFAULT);
   intent.putExtra(Sms.Intents.EXTRA_PACKAGE_NAME, context.getPackageName());
   startActivity(intent);
   ```

3. When you finish restoring all SMS messages, request the user to change the default SMS app back to the previously selected app (saved during step 1).

```
Intent intent = new Intent(context, Sms.Intents.ACTION_CHANGE_DEFAULT);
intent.putExtra(Sms.Intents.EXTRA_PACKAGE_NAME, defaultSmsApp);
startActivity(intent);
```

Prepare to update your SMS app

We encourage you to update your apps as soon as possible to provide your users the best experience on Android. To help you make the changes, we'll soon be providing the necessary SDK components for Android 4.4 that allow you to compile and test your changes on Android 4.4. Stay tuned!
