

福昕PDF编辑器

个人版

• 永久 • 轻巧 • 自由

立即下载

购买会员



永久使用

无限制使用次数



极速轻巧

超低资源占用，告别卡顿慢



自由编辑

享受Word一样的编辑自由



扫一扫，关注公众号

<http://edit.foxitreader.cn>



Android 简介

Android 提供了一个内容丰富的应用框架，支持您在 Java 语言环境中为移动设备开发创新应用和游戏。在左侧导航窗格列出的文档中，提供了有关如何使用各种 Android API 开发应用的详细信息。

如果您是 Android 应用开发新手，则需了解以下有关 Android 应用框架的基本概念，这一点至关重要：

应用提供多个入口点

Android 应用都是将各种可单独调用的不同组件加以组合开发而成。例如，组件可以是为用户界面提供一个屏幕的单个“Activity”，也可以是在后台独立执行工作的“服务”。

您可以使用 *intent* 从一个组件启动另一个组件。甚至，您还可以启动不同应用中的组件，例如，启动地图应用中的 Activity 以显示地址。此模式可为单个应用提供多个入口点，并使任何应用均能够像用户“默认设置”一样处理其他应用可能调用的操作。

了解详情：

[应用基础知识](#)

[Intent 和 Intent 过滤器](#)

[Activity](#)

应用可适应不同的设备

Android 提供了一个自适应应用框架，可用以为不同的设备配置提供独特的资源。例如，您可以针对不同的屏幕尺寸创建不同的 XML 布局文件，系统将根据当前设备的屏幕尺寸确定要应用的布局。

如有任何应用功能需要相机等特定的硬件，则可在运行时查询设备功能的可用性。如有必要，您还可以声明您的应用所必需的功能，使 Google Play 商店等应用市场不得在不支持这些功能的设备上安装您的应用。

了解详情：

[设备兼容性](#)

[资源概览](#)

[UI 概览](#)

要了解应用的工作原理，请从[应用基础知识](#)开始入手。

要立即开始编码，请仔细阅读[构建您的第一个应用](#)。



应用基础知识

本文内容

- › [应用组件](#)
- › [启动组件](#)
- › [清单文件](#)
- › [声明组件](#)
- › [声明应用要求](#)
- › [应用资源](#)

Android 应用采用 Java 编程语言编写。Android SDK 工具将您的代码 — 连同任何数据和资源文件 — 编译到一个 APK：Android 软件包，即带有 .apk 后缀的存档文件中。一个 APK 文件包含 Android 应用的所有内容，它是基于 Android 系统的设备用来安装应用的文件。

安装到设备后，每个 Android 应用都运行在自己的安全沙箱内：

- Android 操作系统是一种多用户 Linux 系统，其中的每个应用都是一个不同的用户；
- 默认情况下，系统会为每个应用分配一个唯一的 Linux 用户 ID（该 ID 仅由系统使用，应用并不知晓）。系统为应用中的所有文件设置权限，使得只有分配给该应用的用户 ID 才能访问这些文件；
- 每个进程都具有自己的虚拟机 (VM)，因此应用代码是在与其他应用隔离的环境中运行；
- 默认情况下，每个应用都在其自己的 Linux 进程内运行。Android 会在需要执行任何应用组件时启动该进程，然后在不再需要该进程或系统必须为其他应用恢复内存时关闭该进程。

Android 系统可以通过这种方式实现**最小权限原则**。也就是说，默认情况下，每个应用都只能访问执行其工作所需的组件，而不能访问其他组件。这样便营造出一个非常安全的环境，在这个环境中，应用无法访问系统中其未获得权限的部分。

不过，应用仍然可以通过一些途径与其他应用共享数据以及访问系统服务：

- 可以安排两个应用共享同一 Linux 用户 ID，在这种情况下，它们能够相互访问彼此的文件。为了节省系统资源，可以安排具有相同用户 ID 的应用在同一 Linux 进程中运行，并共享同一 VM（应用还必须使用相同的证书签署）。
- 应用可以请求访问设备数据（如用户的联系人、短信、可装载存储装置 [SD 卡]、相机、蓝牙等）的权限。用户必须明确授予这些权限。如需了解详细信息，请参阅 [使用系统权限](#)。

以上内容阐述了有关 Android 应用在系统内存在方式的基础知识。本文的其余部分将向您介绍以下内容：

- 用于定义应用的核心框架组件
- 您用来声明组件和应用必需设备功能的清单文件
- 与应用代码分离并允许您的应用针对各种设备配置适当优化其行为的资源

应用组件

应用组件是 Android 应用的基本构建基块。每个组件都是一个不同的点，系统可以通过它进入您的应用。并非所有组件都是用户的实际入口点，有些组件相互依赖，但每个组件都以独立实体形式存在，并发挥特定作用 — 每个组件都是唯一的构建基块，有助于定义应用的总体行为。

共有四种不同的应用组件类型。每种类型都服务于不同的目的，并且具有定义组件的创建和销毁方式的不同生命周期。

以下便是这四种应用组件类型：

Activity

Activity 表示具有用户界面的单一屏幕。例如，电子邮件应用可能具有一个显示新电子邮件列表的 *Activity*、一个用于撰写电子邮件的 *Activity* 以及一个用于阅读电子邮件的 *Activity*。尽管这些 *Activity* 通过协作在电子邮件应用中形成了一种紧密结合的用户体验，但每一个 *Activity* 都独立于其他 *Activity* 而存在。因此，其他应用可以启动其中任何一个 *Activity*（如果电子邮件应用允许）。例如，相机应用可以启动电子邮件应用内用于撰写新电子邮件的 *Activity*，以便用户共享图片。

Activity 作为 [Activity](#) 的子类实现，您可以在 [Activity](#) 开发者指南中了解有关它的更多详情。

服务

服务是一种在后台运行的组件，用于执行长时间运行的操作或为远程进程执行作业。服务不提供用户界面。例如，当用户位于其他应用中时，服务可能在后台播放音乐或者通过网络获取数据，但不会阻断用户与 *Activity* 的交互。诸如 *Activity* 等其他组件可以启动服务，让其运行或与其绑定以便与其进行交互。

服务作为 [Service](#) 的子类实现，您可以在[服务](#)开发者指南中了解有关它的更多详情。

内容提供程序

内容提供程序管理一组共享的应用数据。您可以将数据存储在文件系统、SQLite 数据库、网络上或您的应用可以访问的任何其他永久性存储位置。其他应用可以通过内容提供程序查询数据，甚至修改数据（如果内容提供程序允许）。例如，Android 系统可提供管理用户联系人信息的内容提供程序。因此，任何具有适当权限的应用都可以查询内容提供程序的某一部分（如 [ContactsContract.Data](#)），以读取和写入有关特定人员的信息。

内容提供程序也适用于读取和写入您的应用不共享的私有数据。例如，[记事本](#)示例应用使用内容提供程序来保存笔记。

内容提供程序作为 [ContentProvider](#) 的子类实现，并且必须实现让其他应用能够执行事务的一组标准 API。如需了解详细信息，请参阅[内容提供程序](#)开发者指南。

广播接收器

广播接收器是一种用于响应系统范围广播通知的组件。许多广播都是由系统发起的 — 例如，通知屏幕已关闭、电池电量不足或已拍摄照片的广播。应用也可以发起广播 — 例如，通知其他应用某些数据已下载至设备，并且可供其使用。尽管广播接收器不会显示用户界面，但它们可以[创建状态栏通知](#)，在发生广播事件时提醒用户。但广播接收器更常见的用途只是作为通向其他组件的“通道”，设计用于执行极少量的工作。例如，它可能会基于事件发起一项服务来执行某项工作。

广播接收器作为 [BroadcastReceiver](#) 的子类实现，并且每条广播都作为 [Intent](#) 对象进行传递。如需了解详细信息，请参阅 [BroadcastReceiver](#) 类。

Android 系统设计的独特之处在于，任何应用都可以启动其他应用的组件。例如，如果您想让用户使用设备的相机拍摄照片，很可能有另一个应用可以执行该操作，那么您的应用就可以利用该应用，而不是开发一个 *Activity* 来自行拍摄照片。您不需要集成甚至链接到该相机应用的代码，而是只需启动拍摄照片的相机应用中的 *Activity*。完成拍摄时，系统甚至会将照片返回您的应用，以便您使用。对用户而言，就好像相机真正是您应用的组成部分。

当系统启动某个组件时，会启动该应用的进程（如果尚未运行），并实例化该组件所需的类。例如，如果您的应用启动相机应用中拍摄照片的 *Activity*，则该 *Activity* 会在属于相机应用的进程中运行，而不是您的应用的进程中运行。因此，与大多数其他系统上的应用不同，Android 应用并没有单一入口点（例如，没有 `main()` 函数）。

由于系统在单独的进程中运行每个应用，且其文件权限会限制对其他应用的访问，因此您的应用无法直接启动其他应用中的组件，但 Android 系统却可以。因此，要想启动其他应用中的组件，您必须向系统传递一则消息，说明您想启动特定组件的 *Intent*。系统随后便会为您启动该组件。

启动组件

四种组件类型中的三种 — *Activity*、服务和广播接收器 — 通过名为 *Intent* 的异步消息进行启动。*Intent* 会在运行时将各个组件相互绑定（您可以将 *Intent* 视为从其他组件请求操作的信使），无论组件属于您的应用还是其他应用。

Intent 使用 [Intent](#) 对象创建，它定义的消息用于启动特定组件或特定类型的组件 — *Intent* 可以是显式的，也可以是隐式的。

对于 *Activity* 和服务，*Intent* 定义要执行的操作（例如，“查看”或“发送”某个内容），并且可以指定要执行操作的数据的 URI（以及正在启动的组件可能需要了解的信息）。例如，*Intent* 传达的请求可以是启动一个显示图像或打开网页的 *Activity*。在某些情况下，您可以启动 *Activity* 来接收结果，在这种情况下，*Activity* 也会在 [Intent](#) 中返回结果（例如，您可以发出一个 *Intent*，让用户选取某位联系人并将其返回给您 —

返回 Intent 包括指向所选联系人的 URI)。

对于广播接收器，Intent 只会定义要广播的通知（例如，指示设备电池电量不足的广播只包括指示“电池电量不足”的已知操作字符串）。

Intent 不会启动另一个组件类型 - 内容提供程序，后者会在成为 ContentResolver 的请求目标时启动。内容解析程序通过内容提供程序处理所有直接事务，使得通过提供程序执行事务的组件可以无需执行事务，而是改为在 ContentResolver 对象上调用方法。这会在内容提供程序与请求信息的组件之间留出一个抽象层（以确保安全）。

每种类型的组件有不同的启动方法：

- 您可以通过将 Intent 传递到 `startActivity()` 或 `startActivityForResult()`（当您想让 Activity 返回结果时）来启动 Activity（或为其安排新任务）。
- 您可以通过将 Intent 传递到 `startService()` 来启动服务（或对执行中的服务下达新指令）。或者，您也可以通过将 Intent 传递到 `bindService()` 来绑定到该服务。
- 您可以通过将 Intent 传递到 `sendBroadcast()`、`sendOrderedBroadcast()` 或 `sendStickyBroadcast()` 等方法来发起广播；
- 您可以通过在 ContentResolver 上调用 `query()` 来对内容提供程序执行查询。

如需了解有关 Intent 用法的详细信息，请参阅 Intent 和 Intent 过滤器 文档。以下文档中还提供了有关启动特定组件的详细信息：[Activity](#)、[服务](#)、[BroadcastReceiver](#) 和[内容提供程序](#)。

清单文件

在 Android 系统启动应用组件之前，系统必须通过读取应用的 `AndroidManifest.xml` 文件（“清单”文件）确认组件存在。您的应用必须在此文件中声明其所有组件，该文件必须位于应用项目目录的根目录中。

除了声明应用的组件外，清单文件还有许多其他作用，如：

- 确定应用需要的任何用户权限，如互联网访问权限或对用户联系人的读取权限
- 根据应用使用的 API，声明应用所需的最低 API 级别
- 声明应用使用或需要的硬件和软件功能，如相机、蓝牙服务或多点触摸屏幕
- 应用需要链接的 API 库（Android 框架 API 除外），如 [Google 地图库](#)
- 其他功能

声明组件

清单文件的主要任务是告知系统有关应用组件的信息。例如，清单文件可以像下面这样声明 Activity：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

在 `<application>` 元素中，`android:icon` 属性指向标识应用的图标所对应的资源。

在 `<activity>` 元素中，`android:name` 属性指定 [Activity](#) 子类的完全限定类名，`android:label` 属性指定用作 Activity 的用户可见标签的字符串。

您必须通过以下方式声明所有应用组件：

- Activity 的 `<activity>` 元素
- 服务的 `<service>` 元素
- 广播接收器的 `<receiver>` 元素

- 内容提供程序的 `<provider>` 元素

您包括在源代码中，但未在清单文件中声明的 Activity、服务和内容提供程序对系统不可见，因此也永远不会运行。不过，广播接收器可以在清单文件中声明或在代码中动态创建（如 `BroadcastReceiver` 对象）并通过调用 `registerReceiver()` 在系统中注册。

如需了解有关如何为您的应用构建清单文件的详细信息，请参阅 [AndroidManifest.xml 文档](#)。

声明组件功能

如上文[启动组件](#)中所述，您可以使用 `Intent` 来启动 Activity、服务和广播接收器。您可以通过在 `Intent` 中显式命名目标组件（使用组件类名）来执行此操作。不过，`Intent` 的真正强大之处在于[隐式 Intent](#)概念。隐式 `Intent` 的作用无非是描述要执行的操作类型（还可选择描述您想执行的操作所针对的数据），让系统能够在设备上找到可执行该操作的组件，并启动该组件。如果有多个组件可以执行 `Intent` 所描述的操作，则由用户选择使用哪一个组件。

系统通过将接收到的 `Intent` 与设备上的其他应用的清单文件中提供的 `Intent` 过滤器进行比较来确定可以响应 `Intent` 的组件。

当您在应用的清单文件中声明 `Activity` 时，可以选择性地加入声明 `Activity` 功能的 `Intent` 过滤器，以便响应来自其他应用的 `Intent`。您可以通过将 `<intent-filter>` 元素作为组件声明元素的子项进行添加来为您的组件声明 `Intent` 过滤器。

例如，如果您开发的电子邮件应用包含一个用于撰写新电子邮件的 `Activity`，则可以像下面这样声明一个 `Intent` 过滤器来响应“send”`Intent`（以发送新电子邮件）：

```
<manifest ... >
  ...
  <application ... >
    <activity android:name="com.example.project.ComposeEmailActivity">
      <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="*/*" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

然后，如果另一个应用创建了一个包含[ACTION_SEND](#)操作的 `Intent`，并将其传递到 `startActivity()`，则系统可能会启动您的 `Activity`，以便用户能够草拟并发送电子邮件。

如需了解有关创建 `Intent` 过滤器的详细信息，请参阅 [Intent 和 Intent 过滤器](#) 文档。

声明应用要求

基于 Android 系统的设备多种多样，并非所有设备都提供相同的特性和功能。为防止将您的应用安装在缺少应用所需特性的设备上，您必须通过在清单文件中声明设备和软件要求，为您的应用支持的设备类型明确定义一个配置文件。其中的大多数声明只是为了提供信息，系统不会读取它们，但 Google Play 等外部服务会读取它们，以便当用户在其设备中搜索应用时为用户提供过滤功能。

例如，如果您的应用需要相机，并使用 Android 2.1（[API 级别 7](#)）中引入的 API，您应该像下面这样在清单文件中以要求形式声明这些信息：

```
<manifest ... >
  <uses-feature android:name="android.hardware.camera.any"
                android:required="true" />
  <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
  ...
</manifest>
```

现在，没有相机且 Android 版本低于 2.1 的设备将无法从 Google Play 安装您的应用。

不过，您也可以声明您的应用使用相机，但并不要求必须使用。在这种情况下，您的应用必须将 `required` 属性设置为 `"false"`，并在运行时检查设备是否具有相机，然后根据需要停用任何相机功能。

[设备兼容性](#) 文档中提供了有关如何管理应用与不同设备兼容性的详细信息。

应用资源

Android 应用并非只包含代码 — 它还需要与源代码分离的资源，如图像、音频文件以及任何与应用的视觉呈现有关的内容。例如，您应该通过 XML 文件定义 Activity 用户界面的动画、菜单、样式、颜色和布局。使用应用资源能够在不修改代码的情况下轻松地更新应用的各种特性，并可通过提供备用资源集让您能够针对各种设备配置（如不同的语言和屏幕尺寸）优化您的应用。

对于您的 Android 项目中包括的每一项资源，SDK 构建工具都会定义一个唯一的整型 ID，您可以利用它来引用应用代码或 XML 中定义的其他资源中的资源。例如，如果您的应用包含一个名为 `logo.png` 的图像文件（保存在 `res/drawable/` 目录中），则 SDK 工具会生成一个名为 `R.drawable.logo` 的资源 ID，您可以利用它来引用该图像并将其插入您的用户界面。

提供与源代码分离的资源的其中一个最重要优点在于，您可以提供针对不同设备配置的备用资源。例如，通过在 XML 中定义 UI 字符串，您可以将字符串翻译为其他语言，并将这些字符串保存在单独的文件中。然后，Android 系统会根据向资源目录名称追加的语言限定符（如为法语字符串值追加 `res/values-fr/`）和用户的语言设置，对您的 UI 应用相应的语言字符串。

Android 支持许多不同的备用资源限定符。限定符是一种加入到资源目录名称中，用来定义这些资源适用的设备配置的简短字符串。再举一例，您应该经常会根据设备的屏幕方向和尺寸为 Activity 创建不同的布局。例如，当设备屏幕为纵向（长型）时，您可能想要一种垂直排列按钮的布局；但当屏幕为横向（宽型）时，应按水平方向排列按钮。要想根据方向更改布局，您可以定义两种不同的布局，然后对每个布局的目录名称应用相应的限定符。然后，系统会根据当前设备方向自动应用相应的布局。

如需了解有关可以在应用中包括的不同资源类型以及如何针对不同设备配置创建备用资源的详细信息，请阅读[提供资源](#)。

继续阅读以下方面的内容：

[Intent 和 Intent 过滤器](#)

有关如何使用 `Intent` API 来启动应用组件（如 Activity 和服务）以及如何使您的应用组件可供其他应用使用的信息。

[Activity](#)

有关如何创建 `Activity` 类实例的信息，该类可在您的应用内提供一个具有用户界面的独立屏幕。

[提供资源](#)

有关如何通过适当构建 Android 应用来使应用资源与应用代码分离的信息，包括如何针对特定设备配置提供备用资源。

您可能还对以下内容感兴趣：

[设备兼容性](#)

有关 Android 在不同设备类型上工作方式的信息，并介绍了如何针对不同设备优化您的应用，或如何限制您的应用在不同设备上的可用性。

[系统权限](#)

有关 Android 如何通过一种权限系统来限制应用对特定 API 访问权限的信息，该系统要求征得用户同意，才允许您的应用使用这些 API。



Device Compatibility

In this document

- [What Does "Compatibility" Mean?](#)
- [Controlling Your App's Availability to Devices](#)
 - [Device features](#)
 - [Platform version](#)
 - [Screen configuration](#)
- [Controlling Your App's Availability for Business Reasons](#)

See also

- [Filtering on Google Play](#)
- [Providing Resources](#)
- [Android Compatibility](#)

Android is designed to run on many different types of devices, from phones to tablets and televisions. As a developer, the range of devices provides a huge potential audience for your app. In order for your app to be successful on all these devices, it should tolerate some feature variability and provide a flexible user interface that adapts to different screen configurations.

To facilitate your effort toward that goal, Android provides a dynamic app framework in which you can provide configuration-specific [app resources](#) in static files (such as different XML layouts for different screen sizes). Android then loads the appropriate resources based on the current device configuration. So with some forethought to your app design and some additional app resources, you can publish a single application package (APK) that provides an optimized user experience on a variety of devices.

If necessary, however, you can specify your app's feature requirements and control which types of devices can install your app from Google Play Store. This page explains how you can control which devices have access to your apps, and how to prepare your apps to make sure they reach the right audience. For more information about how you can make your app adapt to different devices, read [Supporting Different Devices](#).

What Does "Compatibility" Mean?

As you read more about Android development, you'll probably encounter the term "compatibility" in various situations. There are two types of compatibility: *device compatibility* and *app compatibility*.

Because Android is an open source project, any hardware manufacturer can build a device that runs the Android operating system. Yet, a device is "**Android compatible**" only if it can correctly run apps written for the *Android execution environment*. The exact details of the Android execution environment are defined by the [Android compatibility program](#) and each device must pass the Compatibility Test Suite (CTS) in order to be considered compatible.

As an app developer, you don't need to worry about whether a device is Android compatible, because only devices that are Android compatible include Google Play Store. So you can rest assured that users who install your app from Google Play Store are using an Android compatible device.

However, you do need to consider whether your **app is compatible** with each potential device configuration. Because Android runs on a wide range of device configurations, some features are not available on all devices. For example, some devices may not include a compass sensor. If your app's core functionality requires the use of a compass sensor, then your app is compatible only with devices that include a compass sensor.

Controlling Your App's Availability to Devices

Android supports a variety of features your app can leverage through platform APIs. Some features are hardware-based (such as a compass sensor), some are software-based (such as app widgets), and some are dependent on the platform version. Not every device supports every feature, so you may need to control your app's availability to devices based on your app's required features.

To achieve the largest user-base possible for your app, you should strive to support as many device configurations as possible using a single APK. In most situations, you can do so by disabling optional features at runtime and [providing app resources](#) with alternatives for different configurations (such as different layouts for different screen sizes). If necessary, however, you can restrict your app's availability to devices through Google Play Store based on the following device characteristics:

- [Device features](#)
- [Platform version](#)
- [Screen configuration](#)

Device features

In order for you to manage your app's availability based on device features, Android defines *feature IDs* for any hardware or software feature that may not be available on all devices. For instance, the feature ID for the compass sensor is `FEATURE_SENSOR_COMPASS` and the feature ID for app widgets is `FEATURE_APP_WIDGETS`.

If necessary, you can prevent users from installing your app when their devices don't provide a given feature by declaring it with a `<uses-feature>` element in your app's [manifest file](#).

For example, if your app does not make sense on a device that lacks a compass sensor, you can declare the compass sensor as required with the following manifest tag:

```
<manifest ... >
    <uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true" />
    ...
</manifest>
```

Google Play Store compares the features your app requires to the features available on each user's device to determine whether your app is compatible with each device. If the device does not provide all the features your app requires, the user cannot install your app.

However, if your app's primary functionality does not *require* a device feature, you should set the `required` attribute to `"false"` and check for the device feature at runtime. If the app feature is not available on the current device, gracefully degrade the corresponding app feature. For example, you can query whether a feature is available by calling `hasSystemFeature()` like this:

```
PackageManager pm = getPackageManager();
if (!pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_COMPASS)) {
    // This device does not have a compass, turn off the compass feature
    disableCompassFeature();
}
```

For information about all the filters you can use to control the availability of your app to users through Google Play Store, see the [Filters on Google Play](#) document.

Note: Some [system permissions](#) implicitly require the availability of a device feature. For example, if your app requests permission to access to `BLUETOOTH`, this implicitly requires the `FEATURE_BLUETOOTH` device feature. You can disable filtering based on this feature and make your app available to devices without Bluetooth by setting the `required` attribute to `"false"` in the `<uses-feature>` tag. For more information about implicitly required device features, read [Permissions that Imply Feature Requirements](#).

Platform version

Different devices may run different versions of the Android platform, such as Android 4.0 or Android 4.4. Each successive platform version often adds new APIs not available in the previous version. To indicate which set of APIs are available, each platform version specifies an [API level](#). For instance, Android 1.0 is API level 1 and Android 4.4 is API level 19.

The API level allows you to declare the minimum version with which your app is compatible, using the `<uses-sdk>` manifest tag and its `minSdkVersion` attribute.

For example, the [Calendar Provider](#) APIs were added in Android 4.0 (API level 14). If your app cannot function without these APIs, you should declare API level 14 as your app's minimum supported version like this:

```
<manifest ... >
  <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="19" />
  ...
</manifest>
```

The `minSdkVersion` attribute declares the minimum version with which your app is compatible and the `targetSdkVersion` attribute declares the highest version on which you've optimized your app.

Each successive version of Android provides compatibility for apps that were built using the APIs from previous platform versions, so your app should always be compatible with future versions of Android while using the documented Android APIs.

Note: The `targetSdkVersion` attribute does not prevent your app from being installed on platform versions that are higher than the specified value, but it is important because it indicates to the system whether your app should inherit behavior changes in newer versions. If you don't update the `targetSdkVersion` to the latest version, the system assumes that your app requires some backward-compatibility behaviors when running on the latest version. For example, among the [behavior changes in Android 4.4](#), alarms created with the [AlarmManager](#) APIs are now inexact by default so the system can batch app alarms and preserve system power, but the system will retain the previous API behavior for your app if your target API level is lower than "19".

However, if your app uses APIs added in a more recent platform version, but does not require them for its primary functionality, you should check the API level at runtime and gracefully degrade the corresponding features when the API level is too low. In this case, set the `minSdkVersion` to the lowest value possible for your app's primary functionality, then compare the current system's version, `SDK_INT`, to one of the codename constants in `Build.VERSION_CODES` that corresponds to the API level you want to check. For example:

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
    // Running on something older than API level 11, so disable
    // the drag/drop features that use ClipboardManager APIs
    disableDragAndDrop();
}
```

Screen configuration

Android runs on devices of various sizes, from phones to tablets and TVs. In order to categorize devices by their screen type, Android defines two characteristics for each device: screen size (the physical size of the screen) and screen density (the physical density of the pixels on the screen, known as [DPI](#)). To simplify the different configurations, Android generalizes these variants into groups that make them easier to target:

- Four generalized sizes: small, normal, large, and `xlarge`.
- And several generalized densities: `mdpi` (medium), `hdpi` (`hdpi`), `xhdpi` (extra high), `xxhdpi` (extra-extra high), and others.

By default, your app is compatible with all screen sizes and densities, because the system makes the appropriate adjustments to your UI layout and image resources as necessary for each screen. However, you should optimize the user experience for each screen configuration by adding specialized layouts for different screen sizes and optimized bitmap images for common screen densities.

For information about how to create alternative resources for different screens and how to restrict your app to certain screen sizes when necessary, read [Supporting Different Screens](#).

Controlling Your App's Availability for Business Reasons

In addition to restricting your app's availability based on device characteristics, it's possible you may need to restrict your app's availability for business or legal reasons. For instance, an app that displays train schedules for the London Underground is unlikely to be useful to users outside the United Kingdom. For this type of situation, Google Play Store provides filtering options in the Play Console that allow you to control your app's availability for non-technical reasons such as the user's locale or wireless carrier.

Filtering for technical compatibility (such as required hardware components) is always based on information contained within your APK file. But filtering for non-technical reasons (such as geographic locale) is always handled in the Google Play Console.

CONTINUE READING ABOUT:

[Providing Resources](#)

Information about how Android apps are structured to separate app resources from the app code, including how you can provide alternative resources for specific device configurations.

[Filters on Google Play](#)

Information about the different ways that Google Play Store can prevent your app from being installed on different devices.

YOU MIGHT ALSO BE INTERESTED IN:

[System Permissions](#)

How Android restricts app access to certain APIs with a permission system that requires the user's consent for your app to use those APIs.



系统权限

本文内容

- › [安全架构](#)
- › [应用签署](#)
- › [用户 ID 和文件访问](#)
- › [使用权限](#)
- › [正常权限和危险权限](#)
 - › [权限组](#)
 - › [定义和实施权限](#)
 - › [自定义权限建议](#)
 - › [...在 AndroidManifest.xml 中](#)
 - › [...发送广播时](#)
 - › [其他权限实施](#)
 - › [URI 权限](#)

关键类

- › [Manifest.permission](#)
- › [Manifest.permission_group](#)

另请参阅

- › [使用系统权限](#)



设计模式
权限



视频

[Google I/O 2015 - Android M 权限：开发者最佳做法](#)

Android 是一个权限分隔的操作系统，其中每个应用都有其独特的系统标识（Linux 用户 ID 和组 ID）。系统各部分也分隔为不同的标识。Linux 据此将不同的应用以及应用与系统分隔开来。

其他更详细的安全功能通过“权限”机制提供，此机制会限制特定进程可以执行的具体操作，并且根据 URI 权限授权临时访问特定的数据段。

本文档介绍应用开发者可以如何使用 Android 提供的安全功能。一般的 [Android 安全性概览](#) 在“Android 开源项目”中提供。

安全架构

Android 安全架构的中心设计点是：在默认情况下任何应用都没有权限执行对其他应用、操作系统或用户有不利影响的任何操作。这包括读取或写入用户的私有数据（例如联系人或电子邮件）、读取或写入其他应用程序的文件、执行网络访问、使设备保持唤醒状态等。

由于每个 Android 应用都是在进程沙盒中运行，因此应用必须显式共享资源和数据。它们的方法是声明需要哪些权限来获取基本沙盒未提供的额外功能。应用以静态方式声明它们需要的权限，然后 Android 系统提示用户同意。

应用沙盒不依赖用于开发应用的技术。特别是，Dalvik VM 不是安全边界，任何应用都可运行原生代码（请参阅 [Android NDK](#)）。各类应用 — Java、原生和混合 — 以同样的方式放在沙盒中，彼此采用相同程度的安全防护。

应用签署

所有 APK (`.apk` 文件) 都必须使用证书签署，其私钥由开发者持有。此证书用于识别应用的作者。证书不需要由证书颁发机构签署；Android 应用在理想情况下可以而且通常也是使用自签名证书。证书在 Android 中的作用是识别应用的作者。这允许系统授予或拒绝应用对 [签名级权限](#) 的访问，以及授予或拒绝应用 [获得与另一应用相同的 Linux 身份的请求](#)。

用户 ID 和文件访问

在安装时，Android 为每个软件包提供唯一的 Linux 用户 ID。此 ID 在软件包在该设备上的使用寿命期间保持不变。在不同设备上，相同软件包可能有不同的 UID；重要的是每个软件包在指定设备上的 UID 是唯一的。

由于在进程级实施安全性，因此任何两个软件包的代码通常都不能在同一进程中运行，因为它们需要作为不同的 Linux 用户运行。您可以在每个软件包的 `AndroidManifest.xml` 的 `manifest` 标记中使用 `sharedUserId` 属性，为它们分配相同的用户 ID。这样做以后，出于安全目的，两个软件包将被视为同一个应用，具有相同的用户 ID 和文件权限。请注意，为保持安全性，只有两个签署了相同签名（并且请求相同的 `sharedUserId`）的应用才被分配同一用户 ID。

应用存储的任何数据都会被分配该应用的用户 ID，并且其他软件包通常无法访问这些数据。使用 `getSharedPreferences(String, int)`、`openFileOutput(String, int)` 或 `openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)` 创建新文件时，可以使用 `MODE_WORLD_READABLE` 和/或 `MODE_WORLD_WRITEABLE` 标记允许任何其他软件包读取/写入文件。设置这些标记时，文件仍归您的应用所有，但其全局读取和/或写入权限已适当设置，使任何其他应用都可看见它。

使用权限

基本 Android 应用默认情况下未关联权限，这意味着它无法执行对用户体验或设备上任何数据产生不利影响的任何操作。要利用受保护的设备功能，必须在 [应用清单](#) 中包含一个或多个 `<uses-permission>` 标记。

例如，需要监控传入的短信的应用要指定：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

如果您的应用在其清单中列出 [正常权限](#)（即，不会对用户隐私或设备操作造成很大风险的权限），系统会自动授予这些权限。如果您的应用在其清单中列出 [危险权限](#)（即，可能影响用户隐私或设备正常操作的权限），系统会要求用户明确授予这些权限。Android 发出请求的方式取决于系统版本，而系统版本是应用的目标：

- 如果设备运行的是 Android 6.0 (API 级别 23) 或更高版本，并且应用的 `targetSdkVersion` 是 23 或更高版本，则应用在运行时向用户请求权限。用户可随时调用权限，因此应用在每次运行时均需检查自身是否具备所需的权限。如需了解有关在应用中请求权限的详细信息，请参阅[使用系统权限培训指南](#)。
- 如果设备运行的是 Android 5.1 (API 级别 22) 或更低版本，并且应用的 `targetSdkVersion` 是 22 或更低版本，则系统会在用户安装应用时要求用户授予权限。如果将新权限添加到更新的应用版本，系统会在用户更新应用时要求授予该权限。用户一旦安装应用，他们撤销权限的唯一方式是卸载应用。

权限级别

如需了解有关不同保护级别权限的详细信息，请参阅[正常权限和危险权限](#)。

通常，权限失效会导致 `SecurityException` 被扔回应用。但不能保证每个地方都是这样。例如，`sendBroadcast(Intent)` 方法在数据传递到每个接收者时会检查权限，在方法调用返回后，即使权限失效，您也不会收到异常。但在几乎所有情况下，权限失效会记入系统日志。

Android 系统提供的权限请参阅 [Manifest.permission](#)。此外，任何应用都可定义并实施自己的权限，因此这不是所有可能权限的详尽列表。

可能在程序运行期间的多个位置实施特定权限：

- 在调用系统时，防止应用执行某些功能。
- 在启动 Activity 时，防止应用启动其他应用的 Activity。
- 在发送和接收广播时，控制谁可以接收您的广播，谁可以向您发送广播。

- 在访问和操作内容提供程序时。
- 绑定至服务或启动服务。

自动权限调整

随着时间的推移，平台中可能会加入新的限制，要想使用特定 API，您的应用可能必须请求之前不需要的权限。因为现有应用假设可随意获取这些 API 应用的访问权限，所以 Android 可能会将新的权限请求应用到应用清单，以免在新平台版本上中断应用。Android 将根据 `targetSdkVersion` 属性提供的值决定应用是否需要权限。如果该值低于在其中添加权限的版本，则 Android 会添加该权限。

例如，API 级别 4 中加入了 `WRITE_EXTERNAL_STORAGE` 权限，用以限制访问共享存储空间。如果您的 `targetSdkVersion` 为 3 或更低版本，则会向更新 Android 版本设备上的应用添加此权限。

注意：如果某权限自动添加到应用，则即使您的应用可能实际并不需要这些附加权限，Google Play 上的应用列表也会列出它们。

为避免这种情况，并且删除您不需要的默认权限，请始终将 `targetSdkVersion` 更新至最高版本。可在 `Build.VERSION_CODES` 文档中查看各版本添加的权限。

正常权限和危险权限

系统权限分为几个保护级别。需要了解的两个最重要保护级别是 **正常权限** 和 **危险权限**：

- 正常权限** 涵盖应用需要访问其沙盒外部数据或资源，但对用户隐私或其他应用操作风险很小的区域。例如，设置时区的权限就是正常权限。如果应用声明其需要正常权限，系统会自动向应用授予该权限。如需当前正常权限的完整列表，请参阅[正常权限](#)。
- 危险权限** 涵盖应用需要涉及用户隐私信息的数据或资源，或者可能对用户存储的数据或其他应用的操作产生影响的区域。例如，能够读取用户的联系人属于危险权限。如果应用声明其需要危险权限，则用户必须明确向应用授予该权限。

权限组

所有危险的 Android 系统权限都属于权限组。如果设备运行的是 Android 6.0 (API 级别 23)，并且应用的 `targetSdkVersion` 是 23 或更高版本，则当用户请求危险权限时系统会发生以下行为：

- 如果应用请求其清单中列出的危险权限，而应用目前在权限组中没有任何权限，则系统会向用户显示一个对话框，描述应用要访问的权限组。对话框不描述该组内的具体权限。例如，如果应用请求 `READ_CONTACTS` 权限，系统对话框只说明该应用需要访问设备的联系信息。如果用户批准，系统将向应用授予其请求的权限。
- 如果应用请求其清单中列出的危险权限，而应用在同一权限组中已有另一项危险权限，则系统会立即授予该权限，而无需与用户进行任何交互。例如，如果某应用已经请求并且被授予了 `READ_CONTACTS` 权限，然后它又请求 `WRITE_CONTACTS`，系统将立即授予该权限。

特殊权限

有许多权限其行为方式与正常权限及危险权限都不相同。`SYSTEM_ALERT_WINDOW` 和 `WRITE_SETTINGS` 特别敏感，因此大多数应用不应该使用它们。如果某应用需要其中一种权限，必须在清单中声明该权限，并且发送请求用户授权的 intent。系统将向用户显示详细管理屏幕，以响应该 intent。如需了解有关如何请求这些权限的详情，请参阅 [SYSTEM_ALERT_WINDOW](#) 和 [WRITE_SETTINGS](#) 参考条目。

任何权限都可属于一个权限组，包括正常权限和应用定义的权限。但权限组仅当权限危险时才影响用户体验。可以忽略正常权限的权限组。

如果设备运行的是 Android 5.1 (API 级别 22) 或更低版本，并且应用的 `targetSdkVersion` 是 22 或更低版本，则系统会在安装时要求用户授予权限。再次强调，系统只告诉用户应用需要的权限组，而不告知具体权限。

表 1. 危险权限和权限组。

权限组	权限
CALENDAR	<ul style="list-style-type: none"> <code>READ_CALENDAR</code> <code>WRITE_CALENDAR</code>
CAMERA	<code>CAMERA</code>
CONTACTS	<code>READ_CONTACTS</code>

	<ul style="list-style-type: none"> ● <code>WRITE_CONTACTS</code> ● <code>GET_ACCOUNTS</code>
LOCATION	<ul style="list-style-type: none"> ● <code>ACCESS_FINE_LOCATION</code> ● <code>ACCESS_COARSE_LOCATION</code>
MICROPHONE	<ul style="list-style-type: none"> ● <code>RECORD_AUDIO</code>
PHONE	<ul style="list-style-type: none"> ● <code>READ_PHONE_STATE</code> ● <code>CALL_PHONE</code> ● <code>READ_CALL_LOG</code> ● <code>WRITE_CALL_LOG</code> ● <code>ADD_VOICEMAIL</code> ● <code>USE_SIP</code> ● <code>PROCESS_OUTGOING_CALLS</code>
SENSORS	<ul style="list-style-type: none"> ● <code>BODY_SENSORS</code>
SMS	<ul style="list-style-type: none"> ● <code>SEND_SMS</code> ● <code>RECEIVE_SMS</code> ● <code>READ_SMS</code> ● <code>RECEIVE_WAP_PUSH</code> ● <code>RECEIVE_MMS</code>
STORAGE	<ul style="list-style-type: none"> ● <code>READ_EXTERNAL_STORAGE</code> ● <code>WRITE_EXTERNAL_STORAGE</code>

定义和实施权限

要实施您自己的权限，必须先使用一个或多个 `<permission>` 元素在 `AndroidManifest.xml` 中声明它们。

例如，想要控制谁可以开始其中一个 Activity 的应用可如下所示声明此操作的权限：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp" >
    <permission android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permLab_deadlyActivity"
        android:description="@string/permDesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>
```

注：系统不允许多个软件包使用同一名称声明权限，除非所有软件包都使用同一证书签署。如果软件包声明权限，则系统不允许用户安装具有相同权限名称的其他软件包，除非这些软件包使用与第一个软件包相同的证书签署。为避免命名冲突，建议对自定义权限使用相反域名样式命名，例如 `com.example.myapp.ENGAGE_HYPERSPACE`。

`protectionLevel` 属性是必要属性，用于指示系统如何向用户告知需要权限的应用，或者谁可以拥有该权限，具体如链接的文档中所述。

`android:permissionGroup` 属性是可选属性，只是用于帮助系统向用户显示权限。大多数情况下，您要将此设为标准系统组（列在 `android.Manifest.permission_group` 中），但您也可以自己定义一个组。建议使用现有的组，因为这样可简化向用户显示的权限 UI。

需要为权限提供标签和描述。这些是用户在查看权限列表（`android:label`）或单一权限详细信息（`android:description`）时可以看到的字符串资源。标签应简短；用几个词描述权限保护的功能的关键部分。描述应该用几个句子描述权限允许持有人执行的操作。我们的约定是用两个句子描述：第一句描述权限，第二句向用户提醒为应用授予权限后可能出现的错误类型。

下面是 CALL_PHONE 权限的标签和描述示例：

```
<string name="permLab_callPhone">directly call phone numbers</string>
<string name="permDesc_callPhone">Allows the application to call
phone numbers without your intervention. Malicious applications may
cause unexpected calls on your phone bill. Note that this does not
allow the application to call emergency numbers.</string>
```

您可以使用 Settings 应用和 shell 命令 `adb shell pm list permissions` 查看系统中当前定义的权限。要使用 Settings 应用，请转到 **Settings > Applications**。选择一个应用并向下滑动查看该应用使用的权限。对于开发者，`adb -s` 选项以类似于用户将会看到的形式显示权限：

```
$ adb shell pm list permissions -s
All Permissions:

Network communication: view Wi-Fi state, create Bluetooth connections, full
Internet access, view network state

Your location: access extra location provider commands, fine (GPS) location,
mock location sources for testing, coarse (network-based) location

Services that cost you money: send SMS messages, directly call phone numbers

...
```

自定义权限建议

应用可以定义自己的自定义权限，并通过定义 `<uses-permission>` 元素请求其他应用的自定义权限。不过，您应该仔细评估您的应用是否有必要这样做。

- 如果要设计一套向彼此显示功能的应用，请尽可能将应用设计为每个权限只定义一次。如果所有应用并非使用同一证书签署，则必须这样做。即使所有应用使用同一证书签署，最佳做法也是每个权限只定义一次。
- 如果功能仅适用于使用与提供应用相同的签名所签署的应用，您可能可以使用签名检查避免定义自定义权限。当一个应用向另一个应用发出请求时，第二个应用可在遵从该请求之前验证这两个应用是否使用同一证书签署。
- 如果您要开发一套只在您自己的设备上运行的应用，则应开发并安装管理该套件中所有应用权限的软件包。此软件包本身无需提供任何服务。它只是声明所有权限，然后套件中的其他应用通过 `<uses-permission>` 元素请求这些权限。

实施 AndroidManifest.xml 中的权限

您可以通过 `AndroidManifest.xml` 应用高级权限，限制访问系统或应用的全部组件。要执行此操作，在所需的组件上包含 `android:permission` 属性，为用于控制访问它的权限命名。

`Activity` 权限（应用于 `<activity>` 标记）限制谁可以启动相关的 Activity。在 `Context.startActivity()` 和 `Activity.startActivityForResult()` 时会检查权限；如果调用方没有所需的权限，则调用会抛出 `SecurityException`。

`Service` 权限（应用于 `<service>` 标记）限制谁可以启动或绑定到相关的服务。在 `Context.startService()`、`Context.stopService()` 和 `Context.bindService()` 时会检查权限；如果调用方没有所需的权限，则调用会抛出 `SecurityException`。

`BroadcastReceiver` 权限（应用于 `<receiver>` 标记）限制谁可以发送广播给相关的接收方。在 `Context.sendBroadcast()` 返回后检查权限，因为系统会尝试将提交的广播传递到指定的接收方。因此，权限失效不会导致向调用方抛回异常；只是不会传递该 intent。同样，可以向 `Context.registerReceiver()` 提供权限来控制谁可以广播到以编程方式注册的接收方。另一方面，可以在调用 `Context.sendBroadcast()` 时提供权限来限制允许哪些 BroadcastReceiver 对象接收广播（请参阅下文）。

`ContentProvider` 权限（应用于 `<provider>` 标记）限制谁可以访问 `ContentProvider` 中的数据。（内容提供程序有重要的附加安全工具可用，称为 `URI` 权限，将在后面介绍。）与其他组件不同，您可以设置两个单独的权限属性：`android:readPermission` 限制谁可以读取提供程序，`android:writePermission` 限制谁可以写入提供程序。请注意，如果提供程序有读取和写入权限保护，仅拥有写入权限并不表示您可以读取提供程序。第一次检索提供程序时将会检查权限（如果没有任何权限，将会抛出 `SecurityException`），对提供程序执行操作时也会检查权限。使用 `ContentResolver.query()` 需要拥有读取权限；使用 `ContentResolver.insert()`、`ContentResolver.update()`、`ContentResolver.delete()` 需要写入权限。在所有这些情况下，没有所需

的权限将导致调用抛出 `SecurityException`。

发送广播时实施权限

除了实施谁可以向注册的 `BroadcastReceiver` 发送 intent 的权限（如上所述），您还可以指定在发送广播时需要的权限。通过使用权限字符串调用 `Context.sendBroadcast()`，您可以要求接收方的应用必须拥有该权限才可接收您的广播。

请注意，接收者和广播者可能需要权限。此时，这两项权限检查都必须通过后方可将 intent 传递到相关的目标。

其他权限实施

可对任何服务调用实施任意细化的权限。这可通过 `Context.checkSelfPermission()` 方法完成。使用所需的权限字符串调用，它将返回一个整数，表示权限是否已授予当前的调用进程。请注意，仅在执行从另一个进程传入的调用（通常是通过从服务发布的 IDL 界面或者指定给另一进程的某种其他方式完成）时才可使用此方法。

检查权限还有许多其他有用的方法。如果您有另一个进程的 pid，可以使用 Context 方法 `Context.checkSelfPermission(String, int, int)` 检查针对该 pid 的权限。如果您有另一个应用的软件包名称，可以使用直接的 PackageManager 方法 `PackageManager.checkSelfPermission(String, String)` 了解是否已为特定软件包授予特定权限。

URI 权限

到目前为止所述的是标准权限系统，内容提供程序仅仅使用此系统通常是不够的。内容提供程序可能需要通过读取和写入权限保护自己，而其直接客户端也需要将特定 URI 传给其他应用以便于它们运行。邮件应用中的附件是一个典型的示例。应通过权限保护对邮件的访问，因为这是敏感的用户数据。但是，如果将图像附件的 URI 提供给图像查看程序，该图像查看程序不会有打开附件的权限，因为它没有理由拥有访问所有电子邮件的权限。

此问题的解决方法是采用 per-URI 权限机制：在启动 Activity 或返回结果给 Activity 时，调用方可以设置 `Intent.FLAG_GRANT_READ_URI_PERMISSION` 和/或 `Intent.FLAG_GRANT_WRITE_URI_PERMISSION`。这将授予接收 Activity 权限访问 intent 中的特定数据 URI，而不管它是否具有访问 intent 对应的内容提供程序中数据的任何权限。

此机制支持常见的能力式模型，其中用户交互（打开附件、从列表中选择联系人等）驱动临时授予细化的权限。这是一项关键功能，可将应用所需的权限缩小至只与其行为直接相关的权限。

但授予细化的 URI 权限需要与拥有这些 URI 的内容提供程序进行一定的合作。强烈建议内容提供程序实施此功能，并且通过 `android:grantUriPermissions` 属性或 `<grant-uri-permissions>` 标记声明支持此功能。

在 `Context.grantUriPermission()`、`Context.revokeUriPermission()` 和 `Context.checkSelfPermission()` 方法中可以找到更多信息。

继续阅读以下方面的内容：

隐含功能要求的权限

有关如何请求某些权限的信息会隐式将您的应用限制于包含相应硬件或软件功能的设备。

`<uses-permission>`

声明应用所需系统权限的清单标记的 API 参考。

`Manifest.permission`

所有系统权限的 API 参考。

您可能还对以下内容感兴趣：

设备兼容性

有关 Android 在不同设备类型上工作方式的信息，并介绍了如何针对不同设备优化您的应用，或如何限制您的应用在不同设备上的可

用性。

[Android 安全性概览](#)

有关 Android 平台安全模式的详细论述。

平台架构

本文内容

- » [Linux 内核](#)
- » [硬件抽象层 \(HAL\)](#)
- » [Android Runtime](#)
- » [原生 C/C++ 库](#)
- » [Java API 框架](#)
- » [系统应用](#)

Android 是一种基于 Linux 的开放源代码软件栈，为广泛的设备和机型而创建。下图所示为 Android 平台的主要组件。

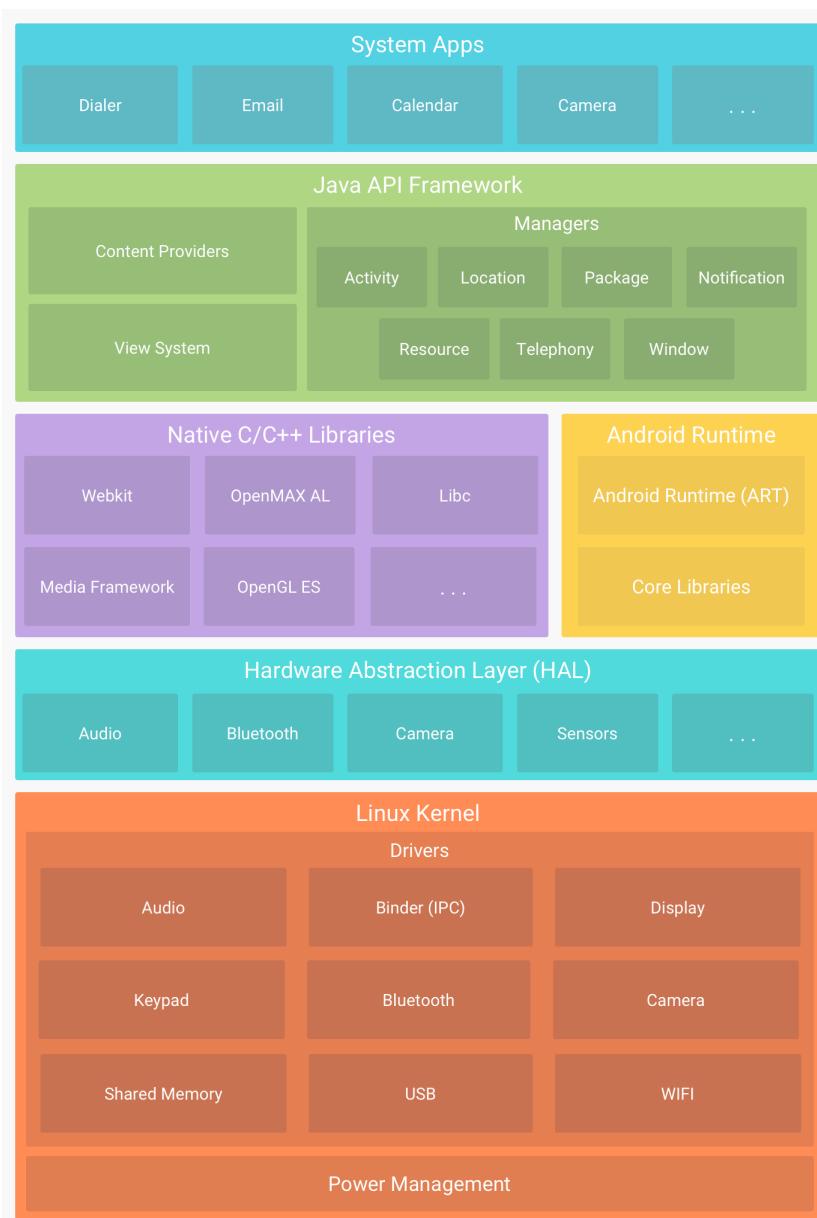


图 1. Android 软件栈。

Linux 内核

Android 平台的基础是 Linux 内核。例如，[Android Runtime \(ART\)](#) 依靠 Linux 内核来执行底层功能，例如线程和低层内存管理。

使用 Linux 内核可让 Android 利用[主要安全功能](#)，并且允许设备制造商为著名的内核开发硬件驱动程序。

硬件抽象层 (HAL)

[硬件抽象层 \(HAL\)](#) 提供标准界面，向更高级别的 [Java API 框架](#) 显示设备硬件功能。HAL 包含多个库模块，其中每个模块都为特定类型的硬件组件实现一个界面，例如[相机](#)或[蓝牙](#)模块。当框架 API 要求访问设备硬件时，Android 系统将为该硬件组件加载库模块。

Android Runtime

对于运行 Android 5.0 (API 级别 21) 或更高版本的设备，每个应用都在其自己的进程中运行，并且有其自己的 [Android Runtime \(ART\)](#) 实例。ART 编写为通过执行 DEX 文件在低内存设备上运行多个虚拟机，DEX 文件是一种专为 Android 设计的字节码格式，经过优化，使用的内存很少。编译工具链（例如 [Jack](#)）将 Java 源代码编译为 DEX 字节码，使其可在 Android 平台上运行。

ART 的部分主要功能包括：

- 预先 (AOT) 和即时 (JIT) 编译
- 优化的垃圾回收 (GC)
- 更好的调试支持，包括专用采样分析器、详细的诊断异常和崩溃报告，并且能够设置监视点以监控特定字段

在 Android 版本 5.0 (API 级别 21) 之前，Dalvik 是 Android Runtime。如果您的应用在 ART 上运行效果很好，那么它应该也可在 Dalvik 上运行，但[反过来不一定](#)。

Android 还包含一套核心运行时库，可提供 Java API 框架使用的 Java 编程语言大部分功能，包括一些 [Java 8 语言功能](#)。

原生 C/C++ 库

许多核心 Android 系统组件和服务（例如 ART 和 HAL）构建自原生代码，需要以 C 和 C++ 编写的原生库。Android 平台提供 Java 框架 API 以向应用显示其中部分原生库的功能。例如，您可以通过 Android 框架的 [Java OpenGL API](#) 访问 [OpenGL ES](#)，以支持在应用中绘制和操作 2D 和 3D 图形。

如果开发的是需要 C 或 C++ 代码的应用，可以使用 [Android NDK](#) 直接从原生代码访问某些[原生平台库](#)。

Java API 框架

您可通过以 Java 语言编写的 API 使用 Android OS 的整个功能集。这些 API 形成创建 Android 应用所需的构建块，它们可简化核心模块化系统组件和服务的重复使用，包括以下组件和服务：

- 丰富、可扩展的[视图系统](#)，可用以构建应用的 UI，包括列表、网格、文本框、按钮甚至可嵌入的网络浏览器
- [资源管理器](#)，用于访问非代码资源，例如本地化的字符串、图形和布局文件
- [通知管理器](#)，可让所有应用在状态栏中显示自定义提醒
- [Activity 管理器](#)，用于管理应用的生命周期，提供常见的[导航返回栈](#)
- [内容提供程序](#)，可让应用访问其他应用（例如“联系人”应用）中的数据或者共享其自己的数据

开发者可以完全访问 Android 系统应用使用的[框架 API](#)。

系统应用

Android 随附一套用于电子邮件、短信、日历、互联网浏览和联系人等的核心应用。平台随附的应用与用户可以选择安装的应用一样，没有特殊状态。因此第三方应用可成为用户的默认网络浏览器、短信 Messenger 甚至默认键盘（有一些例外，例如系统的“设置”应用）。

系统应用可用作用户的应用，以及提供开发者可从其自己的应用访问的主要功能。例如，如果您的应用要发短信，您无需自己构建该功能，可以改为调用已安装的短信应用向您指定的接收者发送消息。



使用 Java 8 语言功能

本文内容：

- [支持的 Java 8 语言功能和 API](#)
- [启用 Java 8 功能和 Jack 工具链](#)
 - [配置 Gradle](#)
 - [已知问题](#)

Android 支持所有 Java 7 语言功能，以及一部分 Java 8 语言功能（具体因平台版本而异）。本页介绍您可以使用的新语言功能、如何正确配置项目以使用这些功能，以及您可能遇到的任何已知问题。

注：在为 Android 开发应用时，可以选择使用 Java 8 语言功能。您可以将项目的源和目标兼容性值保留为 Java 7，但仍须使用 JDK 8 进行编译。

支持 Java 8 语言功能需要一个名为 [Jack](#) 的新编译。Jack 仅在 Android Studio 2.1 和更高版本上才受支持。因此，如果要使用 Java 8 语言功能，则需使用 Android Studio 2.1 开发应用。

如果您已经安装了 Android Studio，请通过点击 [Help > Check for Update](#)（在 Mac 上，点击 [Android Studio > Check for Updates](#)）来确保您已更新到最新版本。如果您的工作站尚未安装 IDE，可[在此下载 Android Studio](#)。

支持的 Java 8 语言功能和 API

Android 并非支持所有 Java 8 语言功能。不过，以下功能在开发面向 Android 7.0 (API 级别 24) 的应用时可用：

- [默认和静态接口方法](#)
- [Lambda 表达式](#) (在 API 级别 23 及更低版本中也可用)
- [重复注解](#)
- [方法引用](#) (在 API 级别 23 及更低版本中也可用)
- [类型注解](#) (在 API 级别 23 及更低版本中也可用)

注：类型注解信息仅在编译时可用，而在运行时不可用。

要在 Android 的较早版本中测试 Lambda 表达式、方法引用和类型注解，请前往您的 `build.gradle` 文件，将 `compileSdkVersion` 和 `targetSdkVersion` 设置为 23 或更低。您仍需要[启用 Jack 工具链](#)以使用这些 Java 8 功能。

此外，也可使用以下 Java 8 语言 API：

- 反反映和语言相关 API：
 - [java.lang.FunctionalInterface](#)
 - [java.lang.annotation.Repeatable](#)
 - [java.lang.reflect.Method.isDefault\(\)](#)
 - 以及与重复注解关联的反映 API，例如 [AnnotatedElement.getAnnotationsByType\(Class\)](#)
- 实用程序 API：
 - [java.util.function](#)
 - [java.util.stream](#)

启用 Java 8 功能和 Jack 工具链

要使用新的 Java 8 语言功能，还需使用新的 [Jack 工具链](#)。新的 Android 工具链将 Java 源语言编译成 Android 可读取的 Dalvik 可执行文件字节码，且有其自己的 `.jack` 库格式，在一个工具中提供了大多数工具链功能：重新打包、压缩、模糊化以及 Dalvik 可执行文件分包。

以下是构建 Android Dalvik 可执行文件可用的两种工具链的对比：

- 旧版 `javac` 工具链：

```
javac (.java → .class) → dx (.class → .dex)
```

- 新版 Jack 工具链：

```
Jack (.java → .jack → .dex)
```

配置 Gradle

要为您的项目启用 Java 8 语言功能和 Jack，请在模块级别的 `build.gradle` 文件中输入以下内容：

```
android {  
    ...  
    defaultConfig {  
        ...  
        jackOptions {  
            enabled true  
        }  
    }  
    compileOptions {  
        sourceCompatibility JavaVersion.VERSION_1_8  
        targetCompatibility JavaVersion.VERSION_1_8  
    }  
}
```

已知问题

[Instant Run](#) 目前不能用于 Jack，在使用新的工具链时将被停用。

由于 Jack 在编译应用时不生成中间类文件，依赖这些文件的工具目前不能用于 Jack。下面是一些工具示例：

- 对类文件进行操作的 Lint 检测工具
- 需要应用类文件的工具和库（例如使用 JaCoCo 进行仪器测试中）

如果您在使用 Jack 的过程中发现其他问题，[请提交错误](#)。



在 Android Runtime (ART) 上验证应用行为

本文内容

- › [解决垃圾回收 \(GC\) 问题](#)
- › [预防 JNI 问题](#)
 - › [检查 JNI 代码中的垃圾回收问题](#)
 - › [错误处理](#)
 - › [对象模型更改](#)
- › [预防堆栈大小问题](#)
- › [修复 AOT 编译问题](#)
- › [报告问题](#)

另请参阅

- › [ART 简介](#)
- › [使用 CheckJNI 调试 Android JNI](#)

Android Runtime (ART) 是运行 Android 5.0 (API 级别 21) 及更高版本的设备的默认运行时。此运行时提供了多种可改善 Android 平台和应用的性能和流畅度的功能。您可以在 [ART 简介](#) 中找到关于 ART 新功能的更多信息。

不过，部分适合 Dalvik 的技术并不适用于 ART。本文档可帮助您了解在迁移现有应用，使其与 ART 兼容时需要注意的事项。大多数应用在使用 ART 运行时都能正常工作。

解决垃圾回收 (GC) 问题

在 Dalvik 中，应用常常发现显式调用 `System.gc()` 非常有用，可促进垃圾回收 (GC)。对 ART 而言这种做法的必要性低得多，尤其是当您需要通过垃圾回收来预防出现 `GC_FOR_ALLOC` 类型或减少碎片时。您可以通过调用 `System.getProperty("java.vm.version")` 来验证正在使用哪种运行时。如果使用的是 ART，则该属性值将是 "`2.0.0`" 或更高。

而且，[Android 开源项目 \(AOSP\)](#) 中正在开发一种紧凑型垃圾回收器，以改善内存管理。因此，您应该避免使用与紧凑型 GC 不兼容的方法（例如保存对象实例数据的指针）。这对于使用 Java 原生接口 (JNI) 的应用而言尤其重要。如需了解详细信息，请参阅[预防 JNI 问题](#)。

预防 JNI 问题

ART 的 JNI 比 Dalvik 的 JNI 更为严格一些。使用 CheckJNI 模式来捕获常见问题是一种特别实用的方法。如果您的应用使用 C/C++ 代码，您应该阅读以下文章：

[使用 CheckJNI 调试 Android JNI](#)

检查 JNI 代码中的垃圾回收问题

ART 在 Android 开源项目 (AOSP) 有正在开发中的紧凑型垃圾回收器。一旦该紧凑型垃圾回收器投入使用，便可在内存中移动对象。如果您使用 C/C++ 代码，请勿执行与紧凑型 GC 不兼容的操作。我们对 CheckJNI 进行了增强，以识别一些潜在的问题（如 [ICS 中的 JNI 局部引用更改](#) 中所述）。

需要特别注意的一个方面是 `Get...ArrayElements()` 和 `Release...ArrayElements()` 函数的使用。在包含非紧凑型 GC 的运行时中，`Get...ArrayElements()` 函数通常返回支持数组对象的实际内存的引用。如果对其中一个返回的数组元素执行更改，数组对象本身将被更改（并且 `Release...ArrayElements()` 的参数往往会被忽略）。但如果正在使用的是紧凑型 GC，则 `Get...ArrayElements()` 函数可能返回内存的副本。如果您在使用紧凑型 GC 的情况下误用引用方法，可能会导致内存崩溃或其他问题。例如：

- 如果您对返回的数组元素执行任何更改，则在完成更改后必须调用相应的 `Release...ArrayElements()` 函数，以确保您所做的更改已正确地复制回基础数组对象。
- 在您释放内存数组元素时，必须根据所做的更改使用相应的模式：
 - 如果您没有对数组元素执行任何更改，请使用 `JNI_ABORT` 模式，该模式会释放内存，而不将更改复制回基础数组元素。
 - 如果您对数组执行了更改，并且不再需要该引用，请使用代码 `0`（它将更新数组对象并释放内存副本）。
 - 如果您对您想要提交的数组执行了更改，并且您希望保留该数组的副本，请使用 `JNI_COMMIT`（它将更新基础数组对象并保留该副本）。
- 调用 `Release...ArrayElements()` 时，将返回最初由 `Get...ArrayElements()` 返回的相同指针。例如，递增原始指针（以扫描所有返回的数组元素），然后将递增的指针传递至 `Release...ArrayElements()` 是不安全的做法。传递此修改后的指针可能导致释放错误的内存，进而导致内存崩溃。

错误处理

ART 的 JNI 会在多种情况下引发错误，而 Dalvik 则不然。（同样地，您可以通过使用 `CheckJNI` 执行测试来捕获大量此种情况）。

例如，如果使用不存在的方法（可能由于该方法已被 `ProGuard` 等工具移除）调用 `RegisterNatives`，ART 现在会正确地引发 `NoSuchMethodError`：

```
08-12 17:09:41.082 13823 13823 E AndroidRuntime: FATAL EXCEPTION: main
08-12 17:09:41.082 13823 13823 E AndroidRuntime: java.lang.NoSuchMethodError:
    no static or non-static method
    "Lcom/foo/Bar;.native_frob(Ljava/lang/String;)I"
08-12 17:09:41.082 13823 13823 E AndroidRuntime:
    at java.lang.Runtime.nativeLoad(Native Method)
08-12 17:09:41.082 13823 13823 E AndroidRuntime:
    at java.lang.Runtime.doLoad(Runtime.java:421)
08-12 17:09:41.082 13823 13823 E AndroidRuntime:
    at java.lang.Runtime.loadLibrary(Runtime.java:362)
08-12 17:09:41.082 13823 13823 E AndroidRuntime:
    at java.lang.System.loadLibrary(System.java:526)
```

如果不使用任何方法调用 `RegisterNatives`，ART 也会记录错误（在 `logcat` 中可见）：

```
W/art      ( 1234): JNI RegisterNativeMethods: attempt to register 0 native
methods for <classname>
```

此外，JNI 函数 `GetFieldID()` 和 `GetStaticFieldID()` 现在会正确地引发 `NoSuchFieldError`，而不是仅仅返回 `null`。类似地，`GetMethodID()` 和 `GetStaticMethodID()` 现在会正确地引发 `NoSuchMethodError`。这可能会导致 `CheckJNI` 由于未处理的异常或引发至原生代码的 Java 调用函数的异常而失败。这让使用 `CheckJNI` 模式测试 ART 兼容型应用变得格外重要。

ART 预期 JNI `CallNonvirtual...Method()` 方法（例如 `CallNonvirtualVoidMethod()`）的用户按照 JNI 规范的要求，使用该方法的声明类而不是子类。

预防堆栈大小问题

Dalvik 具有单独的原生代码堆栈和 Java 代码堆栈，并且默认的 Java 堆栈大小为 32KB，默认的原生堆栈大小为 1MB。ART 具有统一的堆栈以改善局部性。通常情况下，ART `Thread` 堆栈大小应该与 Dalvik 堆栈大小近乎相同。但如果显式设置了堆栈大小，则可能需要针对 ART 中运行的应用重新访问这些值。

- 在 Java 中，查看用于指定显式堆栈大小的 `Thread` 构造函数的调用。例如，如果发生 `StackOverflowError`，您将需要增加该大小。
 - 在 C/C++ 中，查看如何将 `pthread_attr_setstack()` 和 `pthread_attr_setstacksize()` 用于同时通过 JNI 运行 Java 代码的线程。
- 以下是某个应用在 `pthread` 过小的情况下尝试调用 JNI `AttachCurrentThread()` 时记录的错误示例：

```
F/art: art/runtime/thread.cc:435]
    Attempt to attach a thread with a too-small stack (16384 bytes)
```

对象模型更改

Dalvik 错误地允许子类覆盖包私有的方法。ART 在这类情况下会发出警告：

```
Before Android 4.1, method void com.foo.Bar.quux()
would have incorrectly overridden the package-private method in
com.quux.Quux
```

如果您希望在另一个包中覆盖某个类的方法，请将该方法声明为 `public` 或 `protected`。

`Object` 现在包含私有字段。对于反射其类层次中的字段的应用，应小心避免尝试查看 `Object` 的字段。例如，如果您正在向上迭代某个作为串行化框架一部分的类层次，以下情况下请停止迭代操作

```
Class.getSuperclass() == java.lang.Object.class
```

而不是继续操作，直至该方法返回 `null`。

如果没有任何参数，代理 `InvocationHandler.invoke()` 现在将会收到 `null`，而不是空数组。之前记录过此行为，但在 Dalvik 中未得到正确处理。之前版本的 `Mockito` 难以处理这一问题，因此在使用 ART 测试时请使用更新的 Mockito 版本。

修复 AOT 编译问题

ART 的提前 (AOT) Java 编译应适用于所有标准 Java 代码。编译由 ART 的 `dex2oat` 工具执行，如果您在安装时遇到任何与 `dex2oat` 有关的问题，请联系我们（请参阅[报告问题](#)），以便我们能够尽快将其修复。需要注意的几个问题：

- ART 会在安装时执行比 Dalvik 更严格的字节代码验证。Android 构建工具生成的代码应该没有问题。但一些后期处理工具（尤其是执行模糊处理的工具）可能会生成被 Dalvik 容忍而被 ART 拒绝的无效文件。我们已经与工具供应商合作，查找并修复此类问题。在许多情况下，获取最新版本的工具并重新生成 DEX 文件可以修复这些问题。
- 一些被 ART 验证器标记的典型问题包括：
 - 无效的控制流
 - 失衡的 `moniterenter/moniterexit`
 - 0 长度参数类型列表大小
- 一些应用对 `/system/framework`、`/data/dalvik-cache` 中或 `DexClassLoader` 的优化输出目录中的安装的 `.odex` 文件格式具有依赖性。这些文件现在是 ELF 文件，而不是 DEX 文件的扩展形式。尽管 ART 努力遵循与 Dalvik 相同的命名和锁定规则，但应用不能依赖于文件格式，因为该格式可能未经通知便发生更改。

报告问题

如果您遇到任何不是由于应用 JNI 问题而导致的问题，请通过位于 <https://code.google.com/p/android/issues/list> 的 Android 开源项目问题跟踪器报告这些问题。请包含 `"adb bugreport"` 和 Google Play 商店中的应用链接（如果可用）。否则，如果可能，请附加用于重现该问题的 APK。请注意，这些问题（包括附件）是公开可见的。



应用组件

利用 Android 应用框架，您可以使用一组可重复使用的组件创建丰富的创新应用。此部分阐述您可以如何构建用于定义应用构建基块的组件，以及如何使用 Intent 将这些组件连接在一起。

博客文章

使用 DialogFragments

在这篇博文中，我将介绍如何使用带有 v4 支持库（旨在支持 Honeycomb 之前的设备实现向后兼容）的 DialogFragments 显示一个简单的编辑对话框，并使用一个接口向调用 Activity 返回一个结果。

通用片段

今天，我们已发布一个展示相同 Fragments API 的静态库（以及新的 LoaderManager 和其他几个类）。因此，与 Android 1.6 或更高版本兼容的应用可以使用片段来创建与平板电脑兼容的用户界面。

多线程处理，性能卓越

创建快速响应的应用的有效方法是：确保最大程度地减少主 UI 线程的工作负载。任何可能会导致应用挂起的、耗时较长的任务均应在其他线程中进行处理。

培训

管理 Activity 生命周期

本课程介绍每个 Activity 实例将收到的重要生命周期回调方法，阐述可以如何利用这些方法使 Activity 达到用户预期，且避免它们在 Activity 不需要使用时消耗系统资源。

利用片段构建动态 UI

本课程向您介绍如何利用片段创造动态的用户体验、针对不同屏幕尺寸的设备优化应用的用户体验，以及在实现以上目的的同时继续为运行低至 Android 1.6 版本系统的设备提供支持。

共享内容

本课程阐述您可以通过使用 Intent API 和 ActionProvider 对象在应用之间收发内容的一些常见方法。



进程和线程

本文内容

- › [进程](#)
 - › [进程生命周期](#)
 - › [线程](#)
 - › [工作线程](#)
 - › [线程安全方法](#)
 - › [进程间通信](#)

当某个应用组件启动且该应用没有运行其他任何组件时，Android 系统会使用单个执行线程为应用启动新的 Linux 进程。默认情况下，同一应用的所有组件在相同的进程和线程（称为“主”线程）中运行。如果某个应用组件启动且该应用已存在进程（因为存在该应用的其他组件），则该组件会在此进程中启动并使用相同的执行线程。但是，您可以安排应用中的其他组件在单独的进程中运行，并为任何进程创建额外的线程。

本文档介绍进程和线程在 Android 应用中的工作方式。

进程

默认情况下，同一应用的所有组件均在相同的进程中运行，且大多数应用都不会改变这一点。但是，如果您发现需要控制某个组件所属的进程，则可在清单文件中执行此操作。

各类组件元素的清单文件条目—`<activity>`、`<service>`、`<receiver>` 和 `<provider>`—均支持 `android:process` 属性，此属性可以指定该组件应在哪个进程运行。您可以设置此属性，使每个组件均在各自的进程中运行，或者使一些组件共享一个进程，而其他组件则不共享。此外，您还可以设置 `android:process`，使不同应用的组件在相同的进程中运行，但前提是这些应用共享相同的 Linux 用户 ID 并使用相同的证书进行签署。

此外，`<application>` 元素还支持 `android:process` 属性，以设置适用于所有组件的默认值。

如果内存不足，而其他为用户提供更紧急服务的进程又需要内存时，Android 可能会决定在某一时刻关闭某一进程。在被终止进程中运行的应用组件也会随之销毁。当这些组件需要再次运行时，系统将为它们重启进程。

决定终止哪个进程时，Android 系统将权衡它们对用户的相对重要程度。例如，相对于托管可见 Activity 的进程而言，它更有可能关闭托管屏幕上不再可见的 Activity 的进程。因此，是否终止某个进程的决定取决于该进程中所运行组件的状态。下面，我们介绍决定终止进程所用的规则。

进程生命周期

Android 系统将尽量长时间地保持应用进程，但为了新建进程或运行更重要的进程，最终需要移除旧进程来回收内存。为了确定保留或终止哪些进程，系统会根据进程中正在运行的组件以及这些组件的状态，将每个进程放入“重要性层次结构”中。必要时，系统会首先消除重要性最低的进程，然后是重要性略逊的进程，依此类推，以回收系统资源。

重要性层次结构一共有 5 级。以下列表按照重要程度列出了各类进程（第一个进程最重要，将是最后一个被终止的进程）：

1. 前台进程

用户当前操作所必需的进程。如果一个进程满足以下任一条件，即视为前台进程：

- 托管用户正在交互的 `Activity`（已调用 `Activity` 的 `onResume()` 方法）
- 托管某个 `Service`，后者绑定到用户正在交互的 `Activity`
- 托管正在“前台”运行的 `Service`（服务已调用 `startForeground()`）

- 托管正执行一个生命周期回调的 `Service` (`onCreate()`、`onStart()` 或 `onDestroy()`)
- 托管正执行其 `onReceive()` 方法的 `BroadcastReceiver`

通常，在任意给定时间前台进程都为数不多。只有在内存不足以支持它们同时继续运行这一不得已的情况下，系统才会终止它们。此时，设备往往已达到内存分页状态，因此需要终止一些前台进程来确保用户界面正常响应。

2. 可见进程

没有任何前台组件、但仍会影响用户在屏幕上所见内容的进程。如果一个进程满足以下任一条件，即视为可见进程：

- 托管不在前台、但仍对用户可见的 `Activity` (已调用其 `onPause()` 方法)。例如，如果前台 `Activity` 启动了一个对话框，允许在其后显示上一 `Activity`，则有可能会发生这种情况。
- 托管绑定到可见（或前台）`Activity` 的 `Service`。

可见进程被视为是极其重要的进程，除非为了维持所有前台进程同时运行而必须终止，否则系统不会终止这些进程。

3. 服务进程

正在运行已使用 `startService()` 方法启动的服务且不属于上述两个更高类别进程的进程。尽管服务进程与用户所见内容没有直接关联，但是它们通常在执行一些用户关心的操作（例如，在后台播放音乐或从网络下载数据）。因此，除非内存不足以维持所有前台进程和可见进程同时运行，否则系统会让服务进程保持运行状态。

4. 后台进程

包含目前对用户不可见的 `Activity` 的进程（已调用 `Activity` 的 `onStop()` 方法）。这些进程对用户体验没有直接影响，系统可能随时终止它们，以回收内存供前台进程、可见进程或服务进程使用。通常会有很多后台进程在运行，因此它们会保存在 LRU（最近最少使用）列表中，以确保包含用户最近查看的 `Activity` 的进程最后一个被终止。如果某个 `Activity` 正确实现了生命周期方法，并保存了其当前状态，则终止其进程不会对用户体验产生明显影响，因为当用户导航回该 `Activity` 时，`Activity` 会恢复其所有可见状态。有关保存和恢复状态的信息，请参阅 `Activity` 文档。

5. 空进程

不含任何活动应用组件的进程。保留这种进程的唯一目的是用作缓存，以缩短下次在其中运行组件所需的启动时间。为使总体系统资源在进程缓存和底层内核缓存之间保持平衡，系统往往会终止这些进程。

根据进程中当前活动组件的重要程度，Android 会将进程评定为它可能达到的最高级别。例如，如果某进程托管着服务和可见 `Activity`，则会将此进程评定为可见进程，而不是服务进程。

此外，一个进程的级别可能会因其他进程对它的依赖而有所提高，即服务于另一进程的进程其级别永远不会低于其所服务的进程。例如，如果进程 A 中的内容提供程序为进程 B 中的客户端提供服务，或者如果进程 A 中的服务绑定到进程 B 中的组件，则进程 A 始终被视为至少与进程 B 同样重要。

由于运行服务的进程其级别高于托管后台 `Activity` 的进程，因此启动长时间运行操作的 `Activity` 最好为该操作启动 `服务`，而不是简单地创建工作线程，当操作有可能比 `Activity` 更加持久时尤要如此。例如，正在将图片上传到网站的 `Activity` 应该启动服务来执行上传，这样一来，即使用户退出 `Activity`，仍可在后台继续执行上传操作。使用服务可以保证，无论 `Activity` 发生什么情况，该操作至少具备“服务进程”优先级。同理，广播接收器也应使用服务，而不是简单地将耗时冗长的操作放入线程中。

线程

应用启动时，系统会为应用创建一个名为“主线程”的执行线程。此线程非常重要，因为它负责将事件分派给相应的用户界面小部件，其中包括绘图事件。此外，它也是应用与 Android UI 工具包组件（来自 `android.widget` 和 `android.view` 软件包的组件）进行交互的线程。因此，主线程有时也称为 UI 线程。

系统不会为每个组件实例创建单独的线程。运行于同一进程的所有组件均在 UI 线程中实例化，并且对每个组件的系统调用均由该线程进行分派。因此，响应系统回调的方法（例如，报告用户操作的 `onKeyDown()` 或生命周期回调方法）始终在进程的 UI 线程中运行。

例如，当用户触摸屏幕上的按钮时，应用的 UI 线程会将触摸事件分派给小部件，而小部件反过来又设置其按下状态，并将失效请求发布到事件队列中。UI 线程从队列中取消该请求并通知小部件应该重绘自身。

在应用执行繁重的任务以响应用户交互时，除非正确实现应用，否则这种单线程模式可能会导致性能低下。具体地讲，如果 UI 线程需要处理所有任务，则执行耗时很长的操作（例如，网络访问或数据库查询）将会阻塞整个 UI。一旦线程被阻塞，将无法分派任何事件，包括绘图事件。从用户的角度来看，应用显示为挂起。更糟糕的是，如果 UI 线程被阻塞超过几秒钟时间（目前大约是 5 秒钟），用户就会看到一个让人

厌烦的“[应用无响应](#)”(ANR) 对话框。如果引起用户不满，他们可能就会决定退出并卸载此应用。

此外，Android UI 工具包**并非**线程安全工具包。因此，您不得通过工作线程操纵 UI，而只能通过 UI 线程操纵用户界面。因此，Android 的单线程模式必须遵守两条规则：

1. 不要阻塞 UI 线程
2. 不要在 UI 线程之外访问 Android UI 工具包

工作线程

根据上述单线程模式，要保证应用 UI 的响应能力，关键是不能阻塞 UI 线程。如果执行的操作不能很快完成，则应确保它们在单独的线程（“后台”或“工作”线程）中运行。

例如，以下代码演示了一个点击侦听器从单独的线程下载图像并将其显示在 [ImageView](#) 中：

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

乍看起来，这段代码似乎运行良好，因为它创建了一个新线程来处理网络操作。但是，它违反了单线程模式的第二条规则：[不要在 UI 线程之外访问 Android UI 工具包](#)— 此示例从工作线程（而不是 UI 线程）修改了 [ImageView](#)。这可能导致出现不明确、不可预见的行为，但要跟踪此行为困难而又费时。

为解决此问题，Android 提供了几种途径来从其他线程访问 UI 线程。以下列出了几种有用的方法：

- [Activity.runOnUiThread\(Runnable\)](#)
- [View.post\(Runnable\)](#)
- [View.postDelayed\(Runnable, long\)](#)

例如，您可以通过使用 [View.post\(Runnable\)](#) 方法修复上述代码：

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
                loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

现在，上述实现属于线程安全型：在单独的线程中完成网络操作，而在 UI 线程中操纵 [ImageView](#)。

但是，随着操作日趋复杂，这类代码也会变得复杂且难以维护。要通过工作线程处理更复杂的交互，可以考虑在工作线程中使用 [Handler](#) 处理来自 UI 线程的消息。当然，最好的解决方案或许是扩展 [AsyncTask](#) 类，此类简化了与 UI 进行交互所需执行的工作线程任务。

使用 AsyncTask

[AsyncTask](#) 允许对用户界面执行异步操作。它会先阻塞工作线程中的操作，然后在 UI 线程中发布结果，而无需您亲自处理线程和/或处理程序。

要使用它，必须创建 [AsyncTask](#) 的子类并实现 [doInBackground\(\)](#) 回调方法，该方法将在后台线程池中运行。要更新 UI，应该实现 [onPostExecute\(\)](#) 以传递 [doInBackground\(\)](#) 返回的结果并在 UI 线程中运行，以便您安全地更新 UI。稍后，您可以通过从 UI 线程调用 [execute\(\)](#) 来运行任务。

例如，您可以通过以下方式使用 [AsyncTask](#) 来实现上述示例：

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
     * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
     * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

现在 UI 是安全的，代码也得到简化，因为任务分解成了两部分：一部分应在工作线程内完成，另一部分应在 UI 线程内完成。

下面简要概述了 [AsyncTask](#) 的工作方法，但要全面了解如何使用此类，您应阅读 [AsyncTask](#) 参考文档：

- 可以使用泛型指定参数类型、进度值和任务最终值
- 方法 [doInBackground\(\)](#) 会在工作线程上自动执行
- [onPreExecute\(\)](#)、[onPostExecute\(\)](#) 和 [onProgressUpdate\(\)](#) 均在 UI 线程中调用
- [doInBackground\(\)](#) 返回的值将发送到 [onPostExecute\(\)](#)
- 您可以随时在 [doInBackground\(\)](#) 中调用 [publishProgress\(\)](#)，以在 UI 线程中执行 [onProgressUpdate\(\)](#)
- 您可以随时取消任何线程中的任务

注意：使用工作线程时可能会遇到另一个问题，即：[运行时配置变更](#)（例如，用户更改了屏幕方向）导致 Activity 意外重启，这可能会销毁工作线程。要了解如何在这种重启情况下坚持执行任务，以及如何在 Activity 被销毁时正确地取消任务，请参阅[书架](#)示例应用的源代码。

线程安全方法

在某些情况下，您实现的方法可能会从多个线程调用，因此编写这些方法时必须确保其满足线程安全的要求。

这一点主要适用于可以远程调用的方法，如[绑定服务](#)中的方法。如果对 [IBinder](#) 中所实现方法的调用源自运行 [IBinder](#) 的同一进程，则该方法在调用方的线程中执行。但是，如果调用源自其他进程，则该方法将在从线程池选择的某个线程中执行（而不是在进程的 UI 线程中执行），线程池由系统在与 [IBinder](#) 相同的进程中维护。例如，即使服务的 [onBind\(\)](#) 方法将从服务进程的 UI 线程调用，在 [onBind\(\)](#) 返回的对象中实现的方法（例如，实现 RPC 方法的子类）仍会从线程池中的线程调用。由于一个服务可以有多个客户端，因此可能会有多个池线程在同一时间使用同一 [IBinder](#) 方法。因此，[IBinder](#) 方法必须实现为线程安全方法。

同样，内容提供程序也可接收来自其他进程的数据请求。尽管 [ContentResolver](#) 和 [ContentProvider](#) 类隐藏了如何管理进程间通信的细节，但响应这些请求的 [ContentProvider](#) 方法（[query\(\)](#)、[insert\(\)](#)、[delete\(\)](#)、[update\(\)](#) 和 [getType\(\)](#) 方法）将从内容提供程序所在进程的线程池中调用，而不是从进程的 UI 线程调用。由于这些方法可能会同时从任意数量的线程调用，因此它们也必须实现为线程安全方法。

进程间通信

Android 利用远程过程调用 (RPC) 提供了一种进程间通信 (IPC) 机制，通过这种机制，由 Activity 或其他应用组件调用的方法将（在其他进程中）远程执行，而所有结果将返回给调用方。这就要求把方法调用及其数据分解至操作系统可以识别的程度，并将其从本地进程和地址空间传输至远程进程和地址空间，然后在远程进程中重新组装并执行该调用。然后，返回值将沿相反方向传输回来。Android 提供了执行这些 IPC 事务所需的全部代码，因此您只需集中精力定义和实现 RPC 编程接口即可。

要执行 IPC，必须使用 [bindService\(\)](#) 将应用绑定到服务上。如需了解详细信息，请参阅[服务开发者指南](#)。

Intent 和 Intent 过滤器

本文内容

- › [Intent 类型](#)
- › [构建 Intent](#)
 - › [显式 Intent 示例](#)
 - › [隐式 Intent 示例](#)
 - › [强制使用应用选择器](#)
- › [接收隐式 Intent](#)
 - › [过滤器示例](#)
- › [使用待定 Intent](#)
- › [Intent 解析](#)
 - › [操作测试](#)
 - › [类别测试](#)
 - › [数据测试](#)
 - › [Intent 匹配](#)

另请参阅

- › [与其他应用交互](#)
- › [共享内容](#)

[Intent](#) 是一个消息传递对象，您可以使用它从其他[应用组件](#)请求操作。尽管 Intent 可以通过多种方式促进组件之间的通信，但其基本用例主要包括以下三个：

- **启动 Activity：**

[Activity](#) 表示应用中的一个屏幕。通过将 Intent 传递给 `startActivity()`，您可以启动新的 [Activity](#) 实例。Intent 描述了要启动的 Activity，并携带了任何必要的数据。

如果您希望在 Activity 完成后收到结果，请调用 `startActivityForResult()`。在 Activity 的 `onActivityResult()` 回调中，您的 Activity 将结果作为单独的 Intent 对象接收。如需了解详细信息，请参阅 [Activity](#) 指南。

- **启动服务：**

[Service](#) 是一个不使用用户界面而在后台执行操作的组件。通过将 Intent 传递给 `startService()`，您可以启动服务执行一次性操作（例如，下载文件）。Intent 描述了要启动的服务，并携带了任何必要的数据。

如果服务旨在使用客户端-服务器接口，则通过将 Intent 传递给 `bindService()`，您可以从其他组件绑定到此服务。如需了解详细信息，请参阅 [服务](#) 指南。

- **传递广播：**

广播是任何应用均可接收的消息。系统将针对系统事件（例如：系统启动或设备开始充电时）传递各种广播。通过将 Intent 传递给 `sendBroadcast()`、`sendOrderedBroadcast()` 或 `sendStickyBroadcast()`，您可以将广播传递给其他应用。

Intent 类型

Intent 分为两种类型：

- **显式 Intent**：按名称（完全限定类名）指定要启动的组件。通常，您会在自己的应用中使用显式 Intent 来启动组件，这是因为您知道要启动的 Activity 或服务的类名。例如，启动新 Activity 以响应用户操作，或者启动服务以在后台下载文件。

- 隐式 Intent：不会指定特定的组件，而是声明要执行的常规操作，从而允许其他应用中的组件来处理它。例如，如需在地图上向用户显示位置，则可以使用隐式 Intent，请求另一具有此功能的应用在地图上显示指定的位置。

创建显式 Intent 启动 Activity 或服务时，系统将立即启动 Intent 对象中指定的应用组件。

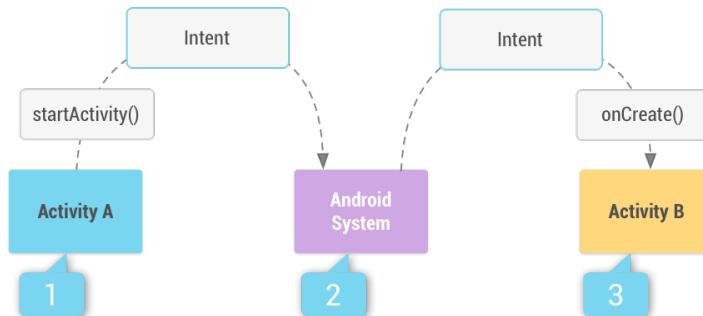


图 1. 隐式 Intent 如何通过系统传递以启动其他 Activity 的图解：[1] Activity A 创建包含操作描述的 Intent，并将其传递给 `startActivity()`。[2] Android 系统搜索所有应用中与 Intent 匹配的 Intent 过滤器。找到匹配项之后，[3] 该系统通过调用匹配 Activity (Activity B) 的 `onCreate()` 方法并将其实传递给 Intent，以此启动匹配 Activity。

创建隐式 Intent 时，Android 系统通过将 Intent 的内容与在设备上其他应用的清单文件中声明的 Intent 过滤器进行比较，从而找到要启动的相关组件。如果 Intent 与 Intent 过滤器匹配，则系统将启动该组件，并向其传递 Intent 对象。如果多个 Intent 过滤器兼容，则系统会显示一个对话框，支持用户选取要使用的应用。

Intent 过滤器是应用清单文件中的一个表达式，它指定该组件要接收的 Intent 类型。例如，通过为 Activity 声明 Intent 过滤器，您可以使其他应用能够直接使用某一特定类型的 Intent 启动 Activity。同样，如果您没有为 Activity 声明任何 Intent 过滤器，则 Activity 只能通过显式 Intent 启动。

注意：为了确保应用的安全性，启动 Service 时，请始终使用显式 Intent，且不要为服务声明 Intent 过滤器。使用隐式 Intent 启动服务存在安全隐患，因为您无法确定哪些服务将响应 Intent，且用户无法看到哪些服务已启动。从 Android 5.0 (API 级别 21) 开始，如果使用隐式 Intent 调用 `bindService()`，系统会引发异常。

构建 Intent

Intent 对象携带了 Android 系统用来确定要启动哪个组件的信息（例如，准确的组件名称或应当接收该 Intent 的组件类别），以及收件人组件为了正确执行操作而使用的信息（例如，要采取的操作以及要处理的数据）。

Intent 中包含的主要信息如下：

组件名称

要启动的组件名称。

这是可选项，但也是构建显式 Intent 的一项重要信息，这意味着 Intent 应当仅传递给由组件名称定义的应用组件。如果没有组件名称，则 Intent 是隐式的，且系统将根据其他 Intent 信息（例如，以下所述的操作、数据和类别）决定哪个组件应当接收 Intent。因此，如需在应用中启动特定的组件，则应指定该组件的名称。

注：启动 Service 时，您应始终指定组件名称。否则，您无法确定哪项服务会响应 Intent，且用户无法看到哪项服务已启动。

Intent 的这一字段是一个 ComponentName 对象，您可以使用目标组件的完全限定类名指定此对象，其中包括应用的软件包名称。例如，`com.example.ExampleActivity`。您可以使用 `setComponent()`、`setClass()`、`setClassName()` 或 Intent 构造函数设置组件名称。

操作

指定要执行的通用操作（例如，“查看”或“选取”）的字符串。

对于广播 Intent，这是指已发生且正在报告的操作。操作在很大程度上决定了其余 Intent 的构成，特别是数据和 extra 中包含的内容。

您可以指定自己的操作，供 Intent 在您的应用内使用（或者供其他应用在您的应用中调用组件）。但是，您通常应该使用由 [Intent](#) 类或其他框架类定义的操作常量。以下是一些用于启动 Activity 的常见操作：

[ACTION_VIEW](#)

如果您拥有一些某项 Activity 可向用户显示的信息（例如，要使用图库应用查看的照片；或者要使用地图应用查看的地址），请使用 Intent 将此操作与 [startActivity\(\)](#) 结合使用。

[ACTION_SEND](#)

这也称为“共享”Intent。如果您拥有一些用户可通过其他应用（例如，电子邮件应用或社交共享应用）共享的数据，则应使用 Intent 将此操作与 [startActivity\(\)](#) 结合使用。

有关更多定义通用操作的常量，请参阅 [Intent](#) 类参考文档。其他操作在 Android 框架中的其他位置定义。例如，对于在系统的设置应用中打开特定屏幕的操作，将在 [Settings](#) 中定义。

您可以使用 [setAction\(\)](#) 或 [Intent](#) 构造函数为 Intent 指定操作。

如果定义自己的操作，请确保将应用的软件包名称作为前缀。例如：

```
static final String ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL";
```

数据

引用待操作数据和/或该数据 MIME 类型的 URI（[Uri](#) 对象）。提供的数据类型通常由 Intent 的操作决定。例如，如果操作是 [ACTION_EDIT](#)，则数据应包含待编辑文档的 URI。

创建 Intent 时，除了指定 URI 以外，指定数据类型（其 MIME 类型）往往也很重要。例如，能够显示图像的 Activity 可能无法播放音频文件，即便 URI 格式十分类似时也是如此。因此，指定数据的 MIME 类型有助于 Android 系统找到接收 Intent 的最佳组件。但有时，MIME 类型可以从 URI 中推断得出，特别当数据是 [content:](#) URI 时尤其如此。这表明数据位于设备中，且由 [ContentProvider](#) 控制，这使得数据 MIME 类型对系统可见。

要仅设置数据 URI，请调用 [setData\(\)](#)。要仅设置 MIME 类型，请调用 [setType\(\)](#)。如有必要，您可以使用 [setDataAndType\(\)](#) 同时显式设置二者。

注意：若要同时设置 URI 和 MIME 类型，请勿调用 [setData\(\)](#) 和 [setType\(\)](#)，因为它们会互相抵消彼此的值。请始终使用 [setDataAndType\(\)](#) 同时设置 URI 和 MIME 类型。

类别

一个包含应处理 Intent 组件类型的附加信息的字符串。您可以将任意数量的类别描述放入一个 Intent 中，但大多数 Intent 均不需要类别。以下是一些常见类别：

[CATEGORY_BROWSABLE](#)

目标 Activity 允许本身通过网络浏览器启动，以显示链接引用的数据，如图像或电子邮件。

[CATEGORY_LAUNCHER](#)

该 Activity 是任务的初始 Activity，在系统的应用启动器中列出。

有关类别的完整列表，请参阅 [Intent](#) 类描述。

您可以使用 [addCategory\(\)](#) 指定类别。

以上列出的这些属性（组件名称、操作、数据和类别）表示 Intent 的既定特征。通过读取这些属性，Android 系统能够解析应当启动哪个应用组件。

但是，Intent 也有可能会一些携带不影响其如何解析为应用组件的信息。Intent 还可以提供：

Extra

携带完成请求操作所需的附加信息的键值对。正如某些操作使用特定类型的数据 URI 一样，有些操作也使用特定的 extra。

您可以使用各种 `putExtra()` 方法添加 extra 数据，每种方法均接受两个参数：键名和值。您还可以创建一个包含所有 extra 数据的 `Bundle` 对象，然后使用 `putExtras()` 将 `Bundle` 插入 `Intent` 中。

例如，使用 `ACTION_SEND` 创建用于发送电子邮件的 Intent 时，可以使用 `EXTRA_EMAIL` 键指定“目标”收件人，并使用 `EXTRA_SUBJECT` 键指定“主题”。

`Intent` 类将为标准化的数据类型指定多个 `EXTRA_*` 常量。如需声明自己的 extra 键（对于应用接收的 Intent），请确保将应用的软件包名称作为前缀。例如：

```
static final String EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS";
```

标志

在 `Intent` 类中定义的、充当 Intent 元数据的标志。标志可以指示 Android 系统如何启动 Activity（例如，Activity 应属于哪个任务），以及启动之后如何处理（例如，它是否属于最近的 Activity 列表）。

如需了解详细信息，请参阅 `setFlags()` 方法。

显式 Intent 示例

显式 Intent 是指用于启动某个特定应用组件（例如，应用中的某个特定 Activity 或服务）的 Intent。要创建显式 Intent，请为 `Intent` 对象定义组件名称 — Intent 的所有其他属性均为可选属性。

例如，如果在应用中构建了一个名为 `DownloadService`、旨在从网页下载文件的服务，则可使用以下代码启动该服务：

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

`Intent(Context, Class)` 构造函数分别为应用和组件提供 `Context` 和 `Class` 对象。因此，此 Intent 将显式启动该应用中的 `DownloadService` 类。

如需了解有关构建和启动服务的详细信息，请参阅 [服务指南](#)。

隐式 Intent 示例

隐式 Intent 指定能够在可以执行相应操作的设备上调用任何应用的操作。如果您的应用无法执行该操作而其他应用可以，且您希望用户选取要使用的应用，则使用隐式 Intent 非常有用。

例如，如果您希望用户与他人共享您的内容，请使用 `ACTION_SEND` 操作创建 Intent，并添加指定共享内容的 extra。使用该 Intent 调用 `startActivity()` 时，用户可以选取共享内容所使用的应用。

注意：用户可能没有任何应用处理您发送到 `startActivity()` 的隐式 Intent。如果出现这种情况，则调用将会失败，且应用会崩溃。要验证 Activity 是否会接收 Intent，请对 `Intent` 对象调用 `resolveActivity()`。如果结果为非空，则至少有一个应用能够处理该 Intent，且可以安全调用 `startActivity()`。如果结果为空，则不应使用该 Intent。如有可能，您应停用发出该 Intent 的功能。

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getApplicationContext()) != null) {
    startActivity(sendIntent);
}
```

注：在这种情况下，系统并没有使用 URI，但已声明 Intent 的数据类型，用于指定 extra 携带的内容。

调用 `startActivity()` 时，系统将检查已安装的所有应用，确定哪些应用能够处理这种 Intent（即：含 `ACTION_SEND` 操作并携带“text/plain”数据的 Intent）。如果只有一个应用能够处理，则该应用将立即打开并为其提供 Intent。如果多个 Activity 接受 Intent，则系统将显示一个对话框，使用户能够选取要使用的应用。



图 2. 选择器对话框。

强制使用应用选择器

如果有多个应用响应隐式 Intent，则用户可以选择要使用的应用，并将其设置为该操作的默认选项。如果用户可能希望今后一直使用相同的应用执行某项操作（例如，打开网页时，用户往往倾向于仅使用一种网络浏览器），则这一点十分有用。

但是，如果多个应用可以响应 Intent，且用户可能希望每次使用不同的应用，则应采用显式方式显示选择器对话框。选择器对话框每次都会要求用户选择用于操作的应用（用户无法为该操作选择默认应用）。例如，当应用使用 `ACTION_SEND` 操作执行“共享”时，用户根据目前的状况可能需要使用另一不同的应用，因此应当始终使用选择器对话框，如图 2 中所示。

要显示选择器，请使用 `createChooser()` 创建 `Intent`，并将其传递给 `startActivity()`。例如：

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...
// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show the chooser dialog
Intent chooser = Intent.createChooser(sendIntent, title);

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

这将显示一个对话框，其中有响应传递给 `createChooser()` 方法的 Intent 的应用列表，并且将提供的文本用作对话框标题。

接收隐式 Intent

要公布应用可以接收哪些隐式 Intent，请在[清单文件](#)中使用 `<intent-filter>` 元素为每个应用组件声明一个或多个 Intent 过滤器。每个 Intent 过滤器均根据 Intent 的操作、数据和类别指定自身接受的 Intent 类型。仅当隐式 Intent 可以通过 Intent 过滤器之一传递时，系统才会将该 Intent 传递给应用组件。

注：显式 Intent 始终会传递给其目标，无论组件声明的 Intent 过滤器如何均是如此。

应用组件应当为自身可执行的每个独特作业声明单独的过滤器。例如，图像库应用中的一个 Activity 可能会有两个过滤器，分别用于查看图像和编辑图像。当 Activity 启动时，它将检查 Intent 并根据 Intent 中的信息决定具体的行为（例如，是否显示编辑器控件）。

每个 Intent 过滤器均由应用清单文件中的 `<intent-filter>` 元素定义，并嵌套在相应应用组件（例如，`<activity>` 元素）中。在 `<intent-filter>` 内部，您可以使用以下三个元素中的一个或多个指定要接受的 Intent 类型：

`<action>`

在 `name` 属性中，声明接受的 Intent 操作。该值必须是操作的文本字符串值，而不是类常量。

`<data>`

使用一个或多个指定数据 URI 各个方面 (`scheme`、`host`、`port`、`path` 等) 和 MIME 类型的属性，声明接受的数据类型。

`<category>`

在 `name` 属性中，声明接受的 Intent 类别。该值必须是操作的文本字符串值，而不是类常量。

注：为了接收隐式 Intent，必须将 `CATEGORY_DEFAULT` 类别包括在 Intent 过滤器中。方法 `startActivity()` 和 `startActivityForResult()` 将按照已申明 `CATEGORY_DEFAULT` 类别的方式处理所有 Intent。如果未在 Intent 过滤器中声明此类别，则隐式 Intent 不会解析为您的 Activity。

例如，以下是一个使用包含 Intent 过滤器的 Activity 声明，当数据类型为文本时，系统将接收 `ACTION_SEND` Intent：

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

您可以创建一个包括多个 `<action>`、`<data>` 或 `<category>` 实例的过滤器。创建时，仅需确定组件能够处理这些过滤器元素的任何及所有组合即可。

如需仅以操作、数据和类别类型的特定组合来处理多种 Intent，则需创建多个 Intent 过滤器。

系统通过将 Intent 与所有这三个元素进行比较，根据过滤器测试隐式 Intent。隐式 Intent 若要传递给组件，必须通过所有这三项测试。如果 Intent 甚至无法匹配其中任何一项测试，则 Android 系统不会将其传递给组件。但是，由于一个组件可能有多个 Intent 过滤器，因此未能通过某一组件过滤器的 Intent 可能会通过另一过滤器。如需了解有关系统如何解析 Intent 的详细信息，请参阅下文的 [Intent 解析](#) 部分。

限制对组件的访问

使用 Intent 过滤器时，无法安全地防止其他应用启动组件。尽管 Intent 过滤器将组件限制为仅响应特定类型的隐式 Intent，但如果开发者确定您的组件名称，则其他应用有可能通过使用显式 Intent 启动您的应用组件。如果必须确保只有您自己的应用才能启动您的某一组件，请针对该组件将 `exported` 属性设置为 `"false"`。

注：对于所有 Activity，您必须在清单文件中声明 Intent 过滤器。但是，广播接收器的过滤器可以通过调用 `registerReceiver()` 动态注册。稍后，您可以使用 `unregisterReceiver()` 注销该接收器。这样一来，应用便可仅在应用运行时的某一指定时间段内侦听特定的广播。

过滤器示例

为了更好地了解一些 Intent 过滤器的行为，我们一起来看看从社交共享应用的清单文件中截取的以下片段。

```

<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>

```

第一个 Activity `MainActivity` 是应用的主要入口点。当用户最初使用启动器图标启动应用时，该 Activity 将打开：

- `ACTION_MAIN` 操作指示这是主要入口点，且不要求输入任何 Intent 数据。
- `CATEGORY_LAUNCHER` 类别指示此 Activity 的图标应放入系统的应用启动器。如果 `<activity>` 元素未使用 `icon` 指定图标，则系统将使用 `<application>` 元素中的图标。

这两个元素必须配对使用，Activity 才会显示在应用启动器中。

第二个 Activity `ShareActivity` 旨在便于共享文本和媒体内容。尽管用户可以通过从 `MainActivity` 导航进入此 Activity，但也可以从发出隐式 Intent（与两个 Intent 过滤器之一匹配）的另一应用中直接进入 `ShareActivity`。

注：MIME 类型 `application/vnd.google.panorama360+jpg` 是一个指定全景照片的特殊数据类型，您可以使用 Google panorama API 对其进行处理。

使用待定 Intent

`PendingIntent` 对象是 `Intent` 对象的包装器。`PendingIntent` 的主要目的是授权外部应用使用包含的 `Intent`，就像是它从您应用本身的进程中执行的一样。

待定 Intent 的主要用例包括：

- 声明用户使用您的[通知](#)执行操作时所要执行的 Intent（Android 系统的 `NotificationManager` 执行 `Intent`）。
- 声明用户使用您的[应用小部件](#)执行操作时要执行的 Intent（主屏幕应用执行 `Intent`）。
- 声明未来某一特定时间要执行的 Intent（Android 系统的 `AlarmManager` 执行 `Intent`）。

由于每个 `Intent` 对象均设计为由特定类型的应用组件（`Activity`、`Service` 或 `BroadcastReceiver`）进行处理，因此还必须基于相同的考虑因素创建 `PendingIntent`。使用待定 Intent 时，应用不会使用调用（如 `startActivity()`）执行该 Intent。相反，通过调用相应的创建器方法创建 `PendingIntent` 时，您必须声明所需的组件类型：

- `PendingIntent.getActivity()`，适用于启动 `Activity` 的 `Intent`。
- `PendingIntent.getService()`，适用于启动 `Service` 的 `Intent`。
- `PendingIntent.getBroadcast()`，适用于启动 `BroadcastReceiver` 的 `Intent`。

除非您的应用正在从其他应用中接收待定 Intent，否则上述用于创建 `PendingIntent` 的方法可能是您所需的唯一 `PendingIntent` 方法。

每种方法均会提取当前的应用 `Context`、您要包装的 `Intent` 以及一个或多个指定应如何使用该 `Intent` 的标志（例如，是否可以多次使用该 `Intent`）。

如需了解有关使用待定 `Intent` 的详细信息，请参阅[通知和应用小部件 API 指南](#)等手册中每个相应用例的相关文档。

Intent 解析

当系统收到隐式 `Intent` 以启动 `Activity` 时，它根据以下三个方面将该 `Intent` 与 `Intent` 过滤器进行比较，搜索该 `Intent` 的最佳 `Activity`：

- `Intent` 操作
- `Intent` 数据（`URI` 和数据类型）
- `Intent` 类别

下文根据如何在应用的清单文件中声明 `Intent` 过滤器，描述 `Intent` 如何与相应的组件匹配。

操作测试

要指定接受的 `Intent` 操作，`Intent` 过滤器既可以不声明任何 `<action>` 元素，也可以声明多个此类元素。例如：

```
<intent-filter>
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.VIEW" />
    ...
</intent-filter>
```

要通过此过滤器，您在 `Intent` 中指定的操作必须与过滤器中列出的某一操作匹配。

如果该过滤器未列出任何操作，则 `Intent` 没有任何匹配项，因此所有 `Intent` 均无法通过测试。但是，如果 `Intent` 未指定操作，则会通过测试（只要过滤器至少包含一个操作）。

类别测试

要指定接受的 `Intent` 类别，`Intent` 过滤器既可以不声明任何 `<category>` 元素，也可以声明多个此类元素。例如：

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    ...
</intent-filter>
```

若要使 `Intent` 通过类别测试，则 `Intent` 中的每个类别均必须与过滤器中的类别匹配。反之则未必然，`Intent` 过滤器声明的类别可以超出 `Intent` 中指定的数量，且 `Intent` 仍会通过测试。因此，不含类别的 `Intent` 应当始终会通过此测试，无论过滤器中声明何种类别均是如此。

注：Android 会自动将 `CATEGORY_DEFAULT` 类别应用于传递给 `startActivity()` 和 `startActivityForResult()` 的所有隐式 `Intent`。因此，如需 `Activity` 接收隐式 `Intent`，则必须将 `"android.intent.category.DEFAULT"` 的类别包括在其 `Intent` 过滤器中（如上文的 `<intent-filter>` 示例所示）。

数据测试

要指定接受的 `Intent` 数据，`Intent` 过滤器既可以不声明任何 `<data>` 元素，也可以声明多个此类元素。例如：

```
<intent-filter>
    <data android:mimeType="video/mpeg" android:scheme="http" ... />
    <data android:mimeType="audio/mpeg" android:scheme="http" ... />
    ...
</intent-filter>
```

每个 `<data>` 元素均可指定 `URI` 结构和数据类型（MIME 媒体类型）。`URI` 的每个部分均包含单独的 `scheme`、`host`、`port` 和 `path` 属性：

```
<scheme>://<host>:<port>/<path>
```

例如：

```
content://com.example.project:200/folder/subfolder/etc
```

在此 URI 中，架构是 `content`，主机是 `com.example.project`，端口是 `200`，路径是 `folder/subfolder/etc`。

在 `<data>` 元素中，上述每个属性均为可选，但存在线性依赖关系：

- 如果未指定架构，则会忽略主机。
- 如果未指定主机，则会忽略端口。
- 如果未指定架构和主机，则会忽略路径。

将 Intent 中的 URI 与过滤器中的 URI 规范进行比较时，它仅与过滤器中包含的部分 URI 进行比较。例如：

- 如果过滤器仅指定架构，则具有该架构的所有 URI 均与该过滤器匹配。
- 如果过滤器指定架构和权限，但未指定路径，则具有相同架构和权限的所有 URI 都会通过过滤器，无论其路径如何均是如此。
- 如果过滤器指定架构、权限和路径，则仅具有相同架构、权限和路径的 URI 才会通过过滤器。

注：路径规范可以包含星号通配符 (*)，因此仅需部分匹配路径名即可。

数据测试会将 Intent 中的 URI 和 MIME 类型与过滤器中指定的 URI 和 MIME 类型进行比较。规则如下：

- 仅当过滤器未指定任何 URI 或 MIME 类型时，不含 URI 和 MIME 类型的 Intent 才会通过测试。
- 对于包含 URI 但不含 MIME 类型（既未显式声明，也无法通过 URI 推断得出）的 Intent，仅当其 URI 与过滤器的 URI 格式匹配、且过滤器同样未指定 MIME 类型时，才会通过测试。
- 仅当过滤器列出相同的 MIME 类型且未指定 URI 格式时，包含 MIME 类型、但不含 URI 的 Intent 才会通过测试。
- 仅当 MIME 类型与过滤器中列出的类型匹配时，同时包含 URI 类型和 MIME 类型（通过显式声明，或可以通过 URI 推断得出）的 Intent 才会通过测试的 MIME 类型部分。如果 Intent 的 URI 与过滤器中的 URI 匹配，或者如果 Intent 具有 `content:` 或 `file:` URI 且过滤器未指定 URI，则 Intent 会通过测试的 URI 部分。换言之，如果过滤器只是列出 MIME 类型，则假定组件支持 `content:` 和 `file:` 数据。

最后一条规则，即规则 (d)，反映了期望组件能够从文件中或内容提供程序获得本地数据。因此，其过滤器可以仅列出数据类型，而不必显式命名 `content:` 和 `file:` 架构。这是一个典型的案例。例如，下文中的 `<data>` 元素向 Android 指出，组件可从内容提供商处获得并显示图像数据：

```
<intent-filter>
    <data android:mimeType="image/*" />
    ...
</intent-filter>
```

由于大部分可用数据均由内容提供商分发，因此指定数据类型（而非 URI）的过滤器也许最为常见。

另一常见的配置是具有架构和数据类型的过滤器。例如，下文中的 `<data>` 元素向 Android 指出，组件可从网络中检索视频数据以执行操作：

```
<intent-filter>
    <data android:scheme="http" android:type="video/*" />
    ...
</intent-filter>
```

Intent 匹配

通过 Intent 过滤器匹配 Intent，这不仅有助于发现要激活的目标组件，还有助于发现设备上组件集的相关信息。例如，主页应用通过使用指定 `ACTION_MAIN` 操作和 `CATEGORY_LAUNCHER` 类别的 Intent 过滤器查找所有 Activity，以此填充应用启动器。

您的应用可以采用类似的方式使用 Intent 匹配。`PackageManager` 提供了一整套 `query...()` 方法来返回所有能够接受特定 Intent 的组件。此外，它还提供了一系列类似的 `resolve...()` 方法来确定响应 Intent 的最佳组件。例如，`queryIntentActivities()` 将返回能够执行那些作为参数传递的 Intent 的所有 Activity 列表，而 `queryIntentServices()` 则可返回类似的服务列表。这两种方法均不会激活组件，而只是列出能够响应的组件。对于广播接收器，有一种类似的方法：`queryBroadcastReceivers()`。



通用 Intent

本文内容

[显示详细信息](#)

- › [闹钟](#)
- › [日历](#)
- › [相机](#)
- › [联系人/人员应用](#)
- › [电子邮件](#)
- › [文件存储](#)
- › [本地操作](#)
- › [地图](#)
- › [音乐或视频](#)
- › [新笔记](#)
- › [电话](#)
- › [搜索](#)
- › [设置](#)
- › [发送短信](#)
- › [网络浏览器](#)
- › [使用 Android 调试桥验证 Intent](#)

另请参阅

- › [Intent 和 Intent 过滤器](#)

Intent 用于通过描述您想在某个 [Intent](#) 对象中执行的简单操作（如“查看地图”或“拍摄照片”）来启动另一应用中的某个 Activity。这种 Intent 称作**隐式 Intent**，因为它并不指定要启动的应用组件，而是指定一项**操作**并提供执行该操作所需的一些数据。

当您调用 `startActivity()` 或 `startActivityForResult()` 并向其传递隐式 Intent 时，系统会将 Intent 解析为可处理该 Intent 的应用并启动其对应的 [Activity](#)。如果有多个应用可处理 Intent，系统会为用户显示一个对话框，供其选择要使用的应用。

本页面介绍几种可用于执行常见操作的隐式 Intent，按处理 Intent 的应用类型分成不同部分。此外，每个部分还介绍如何创建 [Intent 过滤器](#) 来公布您的应用执行相应操作的能力。

注意：如果设备上没有可接收隐式 Intent 的应用，您的应用将在调用 `startActivity()` 时崩溃。如需事先验证是否存在可接收 Intent 的应用，请对 Intent 对象调用 `resolveActivity()`。如果结果为非空，则至少有一个应用能够处理该 Intent，并且可以安全调用 `startActivity()`。如果结果为空，则您不应使用该 Intent。如有可能，您应停用调用该 Intent 的功能。

如果您不熟悉如何创建 Intent 或 Intent 过滤器，应该先阅读 [Intent 和 Intent 过滤器](#)。

如需了解如何从开发主机触发本页面上所列的 Intent，请参阅[使用 Android 调试桥验证 Intent](#)。

Google Voice Actions

[Google Voice Actions](#) 会触发本页面上所列的一些 Intent 来响应语音命令。如需了解详细信息，请参阅 [Google Voice Actions 触发的 Intent](#)。

闹钟

创建闹铃

如需创建新闹铃，请使用 `ACTION_SET_ALARM` 操作并使用下文介绍的 extra 指定时间和消息等闹铃详细信息。



Google Voice Actions

- “设置一个上午 7 点的闹铃”

操作

`ACTION_SET_ALARM`

数据 URI

无

MIME 类型

无

Extra

`EXTRA_HOUR`

闹铃的小时。

`EXTRA_MINUTES`

闹铃的分钟。

`EXTRA_MESSAGE`

用于标识闹铃的自定义消息。

`EXTRA_DAYS`

一个 `ArrayList`，其中包括应重复触发该闹铃的每个周日。每一天都必须使用 `Calendar` 类中的某个整型值（如 `MONDAY`）进行声明。

对于一次性闹铃，无需指定此 extra。

`EXTRA_RINGTONE`

一个 `content`: URI，用于指定闹铃使用的铃声，也可指定 `VALUE_RINGTONE_SILENT` 以不使用铃声。

如需使用默认铃声，则无需指定此 extra。

`EXTRA_VIBRATE`

一个布尔型值，用于指定该闹铃触发时是否振动。

`EXTRA_SKIP_UI`

一个布尔型值，用于指定响应闹铃的应用在设置闹铃时是否应跳过其 UI。若为 `true`，则应用应跳过任何确认 UI，直接设置指定的闹铃。

示例 Intent :

```
public void createAlarm(String message, int hour, int minutes) {  
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)  
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)  
        .putExtra(AlarmClock.EXTRA_HOUR, hour)  
        .putExtra(AlarmClock.EXTRA_MINUTES, minutes);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

注：

为了调用 ACTION_SET_ALARM Intent，您的应用必须具有 SET_ALARM 权限：

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

示例 Intent 过滤器：

```
<activity ...>  
    <intent-filter>  
        <action android:name="android.intent.action.SET_ALARM" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>
```

创建定时器

如需创建倒计时器，请使用 ACTION_SET_TIMER 操作并使用下文介绍的 extra 指定持续时间等定时器详细信息。



Google Voice Actions

- “设置 5 分钟定时器”

操作

ACTION_SET_TIMER

数据 URI

无

MIME 类型

无

Extra

EXTRA_LENGTH

以秒为单位的定时器定时长度。

EXTRA_MESSAGE

用于标识定时器的自定义消息。

EXTRA_SKIP_UI

一个布尔型值，用于指定响应定时器的应用在设置定时器时是否应跳过其 UI。若为 true，则应用应跳过任何确认 UI，直接启动指定的定时器。

示例 Intent：

```
public void startTimer(String message, int seconds) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_TIMER)
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)
        .putExtra(AlarmClock.EXTRA_LENGTH, seconds)
        .putExtra(AlarmClock.EXTRA_SKIP_UI, true);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

注：

为了调用 ACTION_SET_TIMER Intent，您的应用必须具有 SET_ALARM 权限：

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SET_TIMER" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

显示所有闹铃

如需显示闹铃列表，请使用 ACTION_SHOWALARMS 操作。

尽管调用此 Intent 的应用并不多（使用它的主要是系统应用），但任何充当闹钟的应用都应实现此 Intent 过滤器，并通过显示现有闹铃列表作出响应。

注：此 Intent 是在 Android 4.4（API 级别 19）中添加的。

操作

ACTION_SHOWALARMS

数据 URI

无

MIME 类型

无

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SHOWALARMS" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

日历

添加日历事件

如需向用户的日历添加新事件，请使用 ACTION_INSERT 操作指定具有 Events.CONTENT_URI 的数据 URI。然后您就可以使用下文介绍的 extra 指定事件的各类详细信息。

操作

`ACTION_INSERT`

数据 URI

`Events.CONTENT_URI`

MIME 类型

`"vnd.android.cursor.dir/event"`

Extra

`EXTRA_EVENT_ALL_DAY`

一个布尔型值，指定此事件是否为全天事件。

`EXTRA_EVENT_BEGIN_TIME`

事件的开始时间（从新纪年开始计算的毫秒数）。

`EXTRA_EVENT_END_TIME`

事件的结束时间（从新纪年开始计算的毫秒数）。

`TITLE`

事件标题。

`DESCRIPTION`

事件说明。

`EVENT_LOCATION`

事件地点。

`EXTRA_EMAIL`

以逗号分隔的受邀者电子邮件地址列表。

可使用 `CalendarContract.EventsColumns` 类中定义的常量指定许多其他事件详细信息。

示例 Intent：

```
public void addEvent(String title, String location, Calendar begin, Calendar end) {
    Intent intent = new Intent(Intent.ACTION_INSERT)
        .setData(Events.CONTENT_URI)
        .putExtra(Events.TITLE, title)
        .putExtra(Events.EVENT_LOCATION, location)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, begin)
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, end);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.INSERT" />
        <data android:mimeType="vnd.android.cursor.dir/event" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

相机

拍摄照片或视频并将其返回

如需打开相机应用并接收拍摄的照片或视频，请使用 `ACTION_IMAGE_CAPTURE` 或 `ACTION_VIDEO_CAPTURE` 操作。此外，还可在 `EXTRA_OUTPUT` extra 中指定您希望相机将照片或视频保存到的 URI 位置。

操作

`ACTION_IMAGE_CAPTURE` 或
`ACTION_VIDEO_CAPTURE`

数据 URI 架构

无

MIME 类型

无

Extra

`EXTRA_OUTPUT`

相机应用应将照片或视频文件保存到的 URI 位置（`Uri` 对象形式）。

当相机应用成功将焦点归还给您的 Activity（您的应用收到 `onActivityResult()` 回调）时，您可以按通过 `EXTRA_OUTPUT` 值指定的 URI 访问照片或视频。

注：当您使用 `ACTION_IMAGE_CAPTURE` 拍摄照片时，相机可能还会在结果 `Intent` 中返回缩小尺寸的照片副本（缩略图），这个副本以 `Bitmap` 形式保存在名为 "data" 的 extra 字段中。

示例 Intent：

```

static final int REQUEST_IMAGE_CAPTURE = 1;
static final Uri mLocationForPhotos;

public void capturePhoto(String targetFilename) {
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT,
        Uri.withAppendedPath(mLocationForPhotos, targetFilename));
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_CAPTURE);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        // Do other work with full size photo saved in mLocationForPhotos
        ...
    }
}

```

如需了解有关如何使用此 Intent 拍摄照片的详细信息，包括如何创建与输出位置相适应的 Uri，请阅读[只拍摄照片](#)或[只拍摄视频](#)。

示例 Intent 过滤器：

```

<activity ...>
    <intent-filter>
        <action android:name="android.media.action.IMAGE_CAPTURE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

```

处理此 Intent 时，您的 Activity 应检查传入 Intent 中有无 EXTRA_OUTPUT extra，然后将拍摄的图像或视频保存在该 extra 指定的位置，并调用带 Intent 的 setResult()，该 Intent 将经过压缩的缩略图包括在名为 "data" 的 extra 中。

以静态图像模式启动相机应用

如需以静态图像模式打开相机应用，请使用 [INTENT_ACTION_STILL_IMAGE_CAMERA](#) 操作。

操作

[INTENT_ACTION_STILL_IMAGE_CAMERA](#)



Google Voice Actions

- “拍摄照片”

数据 URI 架构

无

MIME 类型

无

Extra

无

示例 Intent：

```

public void capturePhoto() {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivityForResult(intent);
    }
}

```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.STILL_IMAGE_CAMERA" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

以视频模式启动相机应用

如需以视频模式打开相机应用，请使用 [INTENT_ACTION_VIDEO_CAMERA](#) 操作。

操作

[INTENT_ACTION_VIDEO_CAMERA](#)



Google Voice Actions

- “录制视频”

数据 URI 架构

无

MIME 类型

无

Extra

无

示例 Intent：

```
public void capturePhoto() {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_VIDEO_CAMERA);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivityForResult(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.VIDEO_CAMERA" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

联系人/人员应用

选择联系人

如需让用户选择联系人和为您的应用提供对所有联系人信息的访问权限，请使用 [ACTION_PICK](#) 操作，并将 MIME 类型指定为 [Contacts.CONTENT_TYPE](#)。

传送至您的 [onActivityResult\(\)](#) 回调的结果 Intent 包含指向所选联系人的 [content:](#) URI。响应会利用 [Contacts Provider API](#) 为您的应用授予该联系人的临时读取权限，即使您的应用不具备 [READ_CONTACTS](#) 权限也没有关系。

提示：如果您只需要访问某一条联系人信息（如电话号码或电子邮件地址），请改为参见下一节的内容，其中介绍了如何[选择特定联系人数据](#)。

操作

`ACTION_PICK`

数据 URI 架构

无

MIME 类型

`Contacts.CONTENT_TYPE`

示例 Intent：

```
static final int REQUEST_SELECT_CONTACT = 1;

public void selectContact() {
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(ContactsContract.Contacts.CONTENT_TYPE);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivityForResult(intent, REQUEST_SELECT_CONTACT);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_SELECT_CONTACT && resultCode == RESULT_OK) {
        Uri contactUri = data.getData();
        // Do something with the selected contact at contactUri
        ...
    }
}
```

如需了解有关在获得联系人 URI 后如何检索联系人详情的信息，请阅读[检索联系人详情](#)。请谨记，使用以上 Intent 检索联系人 URI 时，读取该联系人的详情**并不需要** `READ_CONTACTS` 权限。

选择特定联系人数据

如需让用户选择某一条联系人信息，如电话号码、电子邮件地址或其他数据类型，请使用`ACTION_PICK` 操作，并将 MIME 类型指定为下列其中一个内容类型（如 `CommonDataKinds.Phone.CONTENT_TYPE`），以获取联系人的电话号码。

如果您只需要检索一种类型的联系人数据，则将此方法与来自 `ContactsContract.CommonDataKinds` 类的 `CONTENT_TYPE` 配合使用要比使用 `Contacts.CONTENT_TYPE`（如上一部分中所示）更高效，因为结果可让您直接访问所需数据，无需对[联系人提供程序](#)执行更复杂的查询。

传送至您的 `onActivityResult()` 回调的结果 `Intent` 包含指向所选联系人数据的 `content:` URI。响应会为您的应用授予该联系人数据的临时读取权限，即使您的应用不具备 `READ_CONTACTS` 权限也没有关系。

操作

`ACTION_PICK`

数据 URI 架构

无

MIME 类型

`CommonDataKinds.Phone.CONTENT_TYPE`

从有电话号码的联系人中选取。

`CommonDataKinds.Email.CONTENT_TYPE`

从有电子邮件地址的联系人中选取。

`CommonDataKinds.StructuredPostal.CONTENT_TYPE`

从有邮政地址的联系人中选取。

或者 `ContactsContract` 下众多其他 `CONTENT_TYPE` 值中的一个。

示例 Intent：

```
static final int REQUEST_SELECT_PHONE_NUMBER = 1;

public void selectContact() {
    // Start an activity for the user to pick a phone number from contacts
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(CommonDataKinds.Phone.CONTENT_TYPE);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivityForResult(intent, REQUEST_SELECT_PHONE_NUMBER);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_SELECT_PHONE_NUMBER && resultCode == RESULT_OK) {
        // Get the URI and query the content provider for the phone number
        Uri contactUri = data.getData();
        String[] projection = new String[]{CommonDataKinds.Phone.NUMBER};
        Cursor cursor = getContentResolver().query(contactUri, projection,
            null, null, null);
        // If the cursor returned is valid, get the phone number
        if (cursor != null && cursor.moveToFirst()) {
            int numberIndex = cursor.getColumnIndex(CommonDataKinds.Phone.NUMBER);
            String number = cursor.getString(numberIndex);
            // Do something with the phone number
            ...
        }
    }
}
```

查看联系人

如需显示已知联系人的详情，请使用 `ACTION_VIEW` 操作，并使用 `content:` URI 作为 Intent 数据指定联系人。

初次检索联系人 URI 的方法主要有两种：

- 使用 `ACTION_PICK` 返回的联系人 URI，如上一节所示（此方法不需要任何应用权限）
- 直接访问所有联系人的列表，如[检索联系人列表](#)所述（此方法需要 `READ_CONTACTS` 权限）

操作

`ACTION_VIEW`

数据 URI 架构

`content:<URI>`

MIME 类型

无。该类型是从联系人 URI 推断得出。

示例 Intent：

```
public void viewContact(Uri contactUri) {
    Intent intent = new Intent(Intent.ACTION_VIEW, contactUri);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

编辑现有联系人

如需编辑已知联系人，请使用 `ACTION_EDIT` 操作，使用 `content:` URI 作为 Intent 数据指定联系人，并将 extra 中由常量指定的任何已知联系人信息包括在 `ContactsContract.Intents.Insert` 中。

初次检索联系人 URI 的方法主要有两种：

- 使用 `ACTION_PICK` 返回的联系人 URI，如上一节所示（此方法不需要任何应用权限）
- 直接访问所有联系人的列表，如[检索联系人列表](#)所述（此方法需要 `READ_CONTACTS` 权限）

操作

`ACTION_EDIT`

数据 URI 架构

`content:<URI>`

MIME 类型

该类型是从联系人 URI 推断得出。

Extra

`ContactsContract.Intents.Insert` 中定义的一个或多个 extra，以便您填充联系人详情字段。

示例 Intent：

```
public void editContact(Uri contactUri, String email) {
    Intent intent = new Intent(Intent.ACTION_EDIT);
    intent.setData(contactUri);
    intent.putExtra(Intents.Insert.EMAIL, email);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

如需了解有关如何编辑联系人的详细信息，请阅读[使用 Intent 修改联系人](#)。

插入联系人

如需插入新联系人，请使用 `ACTION_INSERT` 操作，将 `Contacts.CONTENT_TYPE` 指定为 MIME 类型，并将 extra 中由常量指定的任何已知联系人信息包括在 `ContactsContract.Intents.Insert` 中。

操作

`ACTION_INSERT`

数据 URI 架构

无

MIME 类型

`Contacts.CONTENT_TYPE`

Extra

`ContactsContract.Intents.Insert` 中定义的一个或多个 extra。

示例 Intent：

```
public void insertContact(String name, String email) {  
    Intent intent = new Intent(Intent.ACTION_INSERT);  
    intent.setType(Contacts.CONTENT_TYPE);  
    intent.putExtra(Intents.Insert.NAME, name);  
    intent.putExtra(Intents.Insert.EMAIL, email);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

如需了解有关如何插入联系人的详细信息，请阅读[使用 Intent 修改联系人](#)。

电子邮件

撰写带有可选附件的电子邮件

如需撰写电子邮件，请根据其是否包括附件使用以下其中一项操作，并使用下列 extra 键加入收件人和主题等电子邮件详情。

操作

`ACTION_SENDTO` (适用于不带附件)

`ACTION_SEND` (适用于带一个附件)

`ACTION_SEND_MULTIPLE` (适用于带多个附件)

数据 URI 架构

无

MIME 类型

`"text/plain"`

`"*/*"`

Extra

`Intent.EXTRA_EMAIL`

包含所有“主送”收件人电子邮件地址的字符串数组。

`Intent.EXTRA_CC`

包含所有“抄送”收件人电子邮件地址的字符串数组。

`Intent.EXTRA_BCC`

包含所有“密件抄送”收件人电子邮件地址的字符串数组。

`Intent.EXTRA_SUBJECT`

包含电子邮件主题的字符串。

`Intent.EXTRA_TEXT`

包含电子邮件正文的字符串。

`Intent.EXTRA_STREAM`

指向附件的 `Uri`。如果使用的是 `ACTION_SEND_MULTIPLE` 操作，应将其改为包含多个 `Uri` 对象的 `ArrayList`。

示例 Intent：

```
public void composeEmail(String[] addresses, String subject, Uri attachment) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("*/");
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

如果您想确保 Intent 只由电子邮件应用（而非其他短信或社交应用）进行处理，则需使用 `ACTION_SENDTO` 操作并加入 "mailto:" 数据架构。例如：

```
public void composeEmail(String[] addresses, String subject) {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:")); // only email apps should handle this
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="*/" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SENDTO" />
        <data android:scheme="mailto" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

文件存储

检索特定类型的文件

如需请求用户选择文档或照片等文件并向您的应用返回文件引用，请使用 `ACTION_GET_CONTENT` 操作并指定所需 MIME 类型。向您的应用返回的文件引用对 Activity 的当前生命周期而言是瞬态引用，因此如果您想稍后进行访问，就必须导入可在稍后读取的副本。用户还可利用此 Intent 在进程中创建新文件（例如，用户可以不选择现有照片，而是用相机拍摄新照片）。

传送至您的 `onActivityResult()` 方法的结果 Intent 包括的数据具有指向该文件的 URI。该 URI 可以是任何类型，如 `http:` URI、`file:` URI 或 `content:` URI。不过，如果您想将可选择的文件限定为可从内容提供程序 (`content:` URI) 访问的文件，以及通过 `openFileDescriptor()` 以文件流形式提供的文件，则您应该为 Intent 添加 `CATEGORY_OPENABLE` 类别。

在 Android 4.3 (API 级别 18) 及更高版本上，您还可以通过为 Intent 添加 `EXTRA_ALLOW_MULTIPLE` 并将其设置为 `true`，允许用户选择多个文件。然后您就可以在 `getClipData()` 返回的 `ClipData` 对象中访问每一个选定的文件。

操作

ACTION_GET_CONTENT

数据 URI 架构

无

MIME 类型

与用户应选择的文件类型对应的 MIME 类型。

Extra

EXTRA_ALLOW_MULTIPLE

一个布尔型值，声明用户是否可以一次选择多个文件。

EXTRA_LOCAL_ONLY

一个布尔型值，声明是否返回的文件必须直接存在于设备上，而不是需要从远程服务下载。

类别（可选）

CATEGORY_OPENABLE

只返回可通过 `openFileDescriptor()` 以文件流形式表示的“可打开”文件。

用于获取照片的示例 Intent：

```
static final int REQUEST_IMAGE_GET = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_GET);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_GET && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        Uri fullPhotoUri = data.getData();
        // Do work with photo saved at fullPhotoUri
        ...
    }
}
```

用于返回照片的示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.GET_CONTENT" />
        <data android:type="image/*" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- The OPENABLE category declares that the returned file is accessible
            from a content provider that supports OpenableColumns
            and ContentResolver.openFileDescriptor() -->
        <category android:name="android.intent.category.OPENABLE" />
    </intent-filter>
</activity>
```

打开特定类型的文件

在 Android 4.4 或更高版本上运行时，您可以不必检索必须导入应用的文件副本（使用 `ACTION_GET_CONTENT` 操作），而是使用 `ACTION_OPEN_DOCUMENT` 操作并指定 MIME 类型，请求打开由另一个应用管理的文件。如果还需要允许用户创建应用可写入的新文档，可改用 `ACTION_CREATE_DOCUMENT` 操作。例如，`ACTION_CREATE_DOCUMENT` Intent 允许用户选择他们想在哪里创建新 PDF 文档（在另一个管理文档存储的应用内），而不是从现有文档中进行选择 — 您的应用随后会收到其可以写入新文档的 URI 位置。

尽管从 `ACTION_GET_CONTENT` 操作传递至您的 `onActivityResult()` 方法的 Intent 可能返回任何类型的 URI，来自 `ACTION_OPEN_DOCUMENT` 和 `ACTION_CREATE_DOCUMENT` 的结果 Intent 始终将所选文件指定为 `DocumentsProvider` 支持的 `content:` URI。您可以通过 `openFileDescriptor()` 打开该文件，并使用 `DocumentsContract.Document` 中的列查询其详细信息。

返回的 URI 会为您的应用授予对文件的长期读取权限（还可能会授予写入权限）。因此，如果您想读取现有文件而不将其副本导入您的应用，或者您想就地打开和编辑文件，特别适合使用 `ACTION_OPEN_DOCUMENT` 操作（而不是使用 `ACTION_GET_CONTENT`）。

您还可以通过为 Intent 添加 `EXTRA_ALLOW_MULTIPLE` 并将其设置为 `true`，允许用户选择多个文件。如果用户只选择一项，您就可以从 `getData()` 检索该项目。如果用户选择多项，则 `getData()` 返回 null，此时您必须改为从 `getClipData()` 返回的 `ClipData` 对象检索每个项目。

注：您的 Intent 必须指定 MIME 类型，并且必须声明 `CATEGORY_OPENABLE` 类别。必要时，您可以使用 `EXTRA_MIME_TYPES` extra 添加一个 MIME 类型数组来指定多个 MIME 类型 — 如果您这样做，必须将 `setType()` 中的主 MIME 类型设置为 `"*/*"`。

操作

`ACTION_OPEN_DOCUMENT` 或
`ACTION_CREATE_DOCUMENT`

数据 URI 架构

无

MIME 类型

与用户应选择的文件类型对应的 MIME 类型。

Extra

`EXTRA_MIME_TYPES`

与您的应用请求的文件类型对应的 MIME 类型数组。当您使用此 extra 时，必须在 `setType()` 中将主 MIME 类型设置为 `"*/*"`。

`EXTRA_ALLOW_MULTIPLE`

一个布尔型值，声明用户是否可以一次选择多个文件。

`EXTRA_TITLE`

供与 `ACTION_CREATE_DOCUMENT` 配合使用，用于指定初始文件名。

`EXTRA_LOCAL_ONLY`

一个布尔型值，声明是否返回的文件必须直接存在于设备上，而不是需要从远程服务下载。

类别

`CATEGORY_OPENABLE`

只返回可通过 `openFileDescriptor()` 以文件流形式表示的“可打开”文件。

用于获取照片的示例 Intent：

```

static final int REQUEST_IMAGE_OPEN = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
    intent.setType("image/*");
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    // Only the system receives the ACTION_OPEN_DOCUMENT, so no need to test.
    startActivityForResult(intent, REQUEST_IMAGE_OPEN);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_OPEN && resultCode == RESULT_OK) {
        Uri fullPhotoUri = data.getData();
        // Do work with full size photo saved at fullPhotoUri
        ...
    }
}

```

第三方应用实际上无法通过 `ACTION_OPEN_DOCUMENT` 操作响应 Intent，而是由系统接收此 Intent，然后在统一用户界面中显示各类应用提供的所有文件。

如需在该 UI 中提供您的应用的文件，并允许其他应用打开它们，您必须实现一个 `DocumentsProvider`，并加入一个 `PROVIDER_INTERFACE` Intent 过滤器 ("`android.content.action.DOCUMENTS_PROVIDER`")。例如：

```

<provider ...
    android:grantUriPermissions="true"
    android:exported="true"
    android:permission="android.permission.MANAGE_DOCUMENTS">
    <intent-filter>
        <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
    </intent-filter>
</provider>

```

如需了解有关如何实现从其他应用打开您的应用管理的文件的详细信息，请阅读[存储访问框架](#)指南。

本地操作

叫车

如需叫一台出租车，请使用 `ACTION_RESERVE_TAXI_RESERVATION` 操作。



Google Voice Actions

- “给我叫一台出租车”
- “给我叫一台车”

(仅限 Android Wear)

操作

`ACTION_RESERVE_TAXI_RESERVATION`

数据 URI

无

MIME 类型

无

Extra

无

示例 Intent :

```
public void callCar() {  
    Intent intent = new Intent(ReserveIntents.ACTION_RESERVE_TAXI_RESERVATION);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

示例 Intent 过滤器：

```
<activity ...>  
    <intent-filter>  
        <action android:name="com.google.android.gms.actions.RESERVE_TAXI_RESERVATION" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>
```

地图

显示地图上的位置

如需打开地图，请使用 [ACTION_VIEW](#) 操作，并通过下文介绍的其中一个架构在 Intent 数据中指定位置信息。

操作

[ACTION_VIEW](#)

数据 URI 架构

`geo:latitude,longitude`

显示给定经度和纬度处的地图。

示例：`"geo:47.6, -122.3"`

`geo:latitude,longitude?z=zoom`

按特定缩放级别显示给定经度和纬度处的地图。缩放级别为 1 时显示以给定纬度、经度为中心的全球地图。最高（最精确）缩放级别为 23。

示例：`"geo:47.6, -122.3?z=11"`

`geo:0,0?q=lat,lng(label)`

显示给定经度和纬度处带字符串标签的地图。

示例：`"geo:0,0?q=34.99, -106.61(Treasure)"`

`geo:0,0?q=my+street+address`

显示“我的街道地址”的位置（可能是具体地址或位置查询）。

示例：`"geo:0,0?q=1600+Amphitheatre+Parkway%2C+CA"`

注：geo URI 中传递的所有字符串都必须编码。例如，字符串 `1st & Pike, Seattle` 应编码为 `1st%20%26%20Pike%2C%20Seattle`。字符串中的空格可使用 `%20` 编码或替换为加号 (+)。

MIME 类型

无

示例 Intent：

```
public void showMap(Uri geoLocation) {  
    Intent intent = new Intent(Intent.ACTION_VIEW);  
    intent.setData(geoLocation);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

示例 Intent 过滤器：

```
<activity ...>  
    <intent-filter>  
        <action android:name="android.intent.action.VIEW" />  
        <data android:scheme="geo" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>
```

音乐或视频

播放媒体文件

如需播放音乐文件，请使用 `ACTION_VIEW` 操作，并在 Intent 数据中指定文件的 URI 位置。

操作

`ACTION_VIEW`

数据 URI 架构

`file:<URI>`

`content:<URI>`

`http:<URL>`

MIME 类型

`"audio/*"`

`"application/ogg"`

`"application/x-ogg"`

`"application/itunes"`

或者您的应用可能需要的任何其他类型。

示例 Intent：

```
public void playMedia(Uri file) {  
    Intent intent = new Intent(Intent.ACTION_VIEW);  
    intent.setData(file);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <data android:type="audio/*" />
        <data android:type="application/ogg" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

基于搜索查询播放音乐

如需基于搜索查询播放音乐，请使用 `INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH` Intent。应用可能会触发此 Intent 来响应用户的音乐播放语音命令。接收此 Intent 的应用会在其库存音乐内搜索与给定查询匹配的现有内容，并在找到后开始播放该内容。

此 Intent 应该包括 `EXTRA_MEDIA_FOCUS` 字符串 extra，以指定预期搜索模式。例如，搜索模式可指定搜索的目标是艺术家姓名还是歌曲名称。



Google Voice Actions

- “播放 michael jackson 的 billie jean”

操作

`INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH`

数据 URI 架构

无

MIME 类型

无

Extra

`MediaStore.EXTRA_MEDIA_FOCUS` (必需)

表示搜索模式（用户是否在寻找特定艺术家、专辑、歌曲或播放列表）。大多数搜索模式都需要额外的 extra。例如，如果用户有意收听某一首歌曲，Intent 可能需要额外增加三个 extra：歌曲名称、艺术家和专辑。对于 `EXTRA_MEDIA_FOCUS` 的每个值，此 Intent 都支持下列搜索模式：

任意 - `"vnd.android.cursor.item/*"`

播放任意音乐。接收 Intent 的应用应该根据智能选择（如用户最后收听的播放列表）播放音乐。

额外 extra：

- `QUERY` (必需) - 一个空字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

非结构化 - `"vnd.android.cursor.item/*"`

播放通过非结构化搜索查询找到的特定歌曲、专辑或类型。当应用无法识别用户想要收听的内容类型时，可能会生成一个具有此搜索模式的 Intent。应用应尽可能使用更确切的搜索模式。

额外 extra：

- `QUERY` (必需) - 一个包含艺术家、专辑、歌曲名称或类型任意组合的字符串。

类型 - `Audio.Genres.ENTRY_CONTENT_TYPE`

播放特定类型的音乐。

额外 extra：

- `"android.intent.extra.genre"` (必需) - 类型。
- `QUERY` (必需) - 类型。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

艺术家 - [Audio.Artists.ENTRY_CONTENT_TYPE](#)

播放特定艺术家的音乐。

额外 extra：

- `EXTRA_MEDIA_ARTIST` (必需) - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `QUERY` (必需) - 一个包含艺术家或类型任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

专辑 - [Audio.Albums.ENTRY_CONTENT_TYPE](#)

播放特定专辑的音乐。

额外 extra：

- `EXTRA_MEDIA_ALBUM` (必需) - 专辑。
- `EXTRA_MEDIA_ARTIST` - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `QUERY` (必需) - 一个包含专辑或艺术家任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

歌曲 - ["vnd.android.cursor.item/audio"](#)

播放特定歌曲。

额外 extra：

- `EXTRA_MEDIA_ALBUM` - 专辑。
- `EXTRA_MEDIA_ARTIST` - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `EXTRA_MEDIA_TITLE` (必需) - 歌曲名称。
- `QUERY` (必需) - 一个包含专辑、艺术家、类型或名称任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

播放列表 - [Audio.Playlists.ENTRY_CONTENT_TYPE](#)

播放特定播放列表或符合额外 extra 指定的某些条件的播放列表。

额外 extra：

- `EXTRA_MEDIA_ALBUM` - 专辑。
- `EXTRA_MEDIA_ARTIST` - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `"android.intent.extra.playlist"` - 播放列表。
- `EXTRA_MEDIA_TITLE` - 播放列表所基于的歌曲名称。

- **QUERY** (必需) - 一个包含专辑、艺术家、类型、播放列表或名称任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

示例 Intent：

如果用户想收听特定艺术家的音乐，搜索应用可生成以下 Intent：

```
public void playSearchArtist(String artist) {  
    Intent intent = new Intent(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH);  
    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,  
                  MediaStore.Audio.Artists.ENTRY_CONTENT_TYPE);  
    intent.putExtra(MediaStore.EXTRA_MEDIA_ARTIST, artist);  
    intent.putExtra(SearchManager.QUERY, artist);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

示例 Intent 过滤器：

```
<activity ...>  
    <intent-filter>  
        <action android:name="android.media.action.MEDIA_PLAY_FROM_SEARCH" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>
```

处理此 Intent 时，您的 Activity 应通过检查传入 Intent 中 EXTRA_MEDIA_FOCUS extra 的值来确定搜索模式。您的 Activity 识别出搜索模式后，应该读取该特定搜索模式额外 extra 的值。您的应用随后便可利用这些信息在其库存音乐内进行搜索，以播放与搜索查询匹配的内容。例如：

```

protected void onCreate(Bundle savedInstanceState) {
    ...
    Intent intent = this.getIntent();
    if (intent.getAction().compareTo(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH) == 0) {

        String mediaFocus = intent.getStringExtra(MediaStore.EXTRA_MEDIA_FOCUS);
        String query = intent.getStringExtra(SearchManager.QUERY);

        // Some of these extras may not be available depending on the search mode
        String album = intent.getStringExtra(MediaStore.EXTRA_MEDIA_ALBUM);
        String artist = intent.getStringExtra(MediaStore.EXTRA_MEDIA_ARTIST);
        String genre = intent.getStringExtra("android.intent.extra.genre");
        String playlist = intent.getStringExtra("android.intent.extra.playlist");
        String title = intent.getStringExtra(MediaStore.EXTRA_MEDIA_TITLE);

        // Determine the search mode and use the corresponding extras
        if (mediaFocus == null) {
            // 'Unstructured' search mode (backward compatible)
            playUnstructuredSearch(query);

        } else if (mediaFocus.compareTo("vnd.android.cursor.item/*") == 0) {
            if (query.isEmpty()) {
                // 'Any' search mode
                playResumeLastPlaylist();
            } else {
                // 'Unstructured' search mode
                playUnstructuredSearch(query);
            }

        } else if (mediaFocus.compareTo(MediaStore.Audio.Genres.ENTRY_CONTENT_TYPE) == 0) {
            // 'Genre' search mode
            playGenre(genre);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Artists.ENTRY_CONTENT_TYPE) == 0) {
            // 'Artist' search mode
            playArtist(artist, genre);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Albums.ENTRY_CONTENT_TYPE) == 0) {
            // 'Album' search mode
            playAlbum(album, artist);

        } else if (mediaFocus.compareTo("vnd.android.cursor.item/audio") == 0) {
            // 'Song' search mode
            playSong(album, artist, genre, title);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Playlists.ENTRY_CONTENT_TYPE) == 0) {
            // 'Playlist' search mode
            playPlaylist(album, artist, genre, playlist, title);
        }
    }
}

```

新笔记

创建笔记

如需创建新笔记，请使用 `ACTION_CREATE_NOTE` 操作并使用下文定义的 extra 指定笔记详情，例如主题和正文。

注：应用必须请求用户确认，然后才能完成操作。

操作

`ACTION_CREATE_NOTE`

数据 URI 架构

无

MIME 类型

```
PLAIN_TEXT_TYPE  
/*/*
```

Extra

`EXTRA_NAME`

一个表示笔记标题或主题的字符串。

`EXTRA_TEXT`

一个表示笔记正文的字符串。

示例 Intent：

```
public void createNote(String subject, String text) {  
    Intent intent = new Intent(NoteIntents.ACTION_CREATE_NOTE)  
        .putExtra(NoteIntents.EXTRA_NAME, subject)  
        .putExtra(NoteIntents.EXTRA_TEXT, text);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

示例 Intent 过滤器：

```
<activity ...>  
    <intent-filter>  
        <action android:name="com.google.android.gms.actions.CREATE_NOTE" />  
        <category android:name="android.intent.category.DEFAULT" />  
        <data android:mimeType="*/*">  
    </intent-filter>  
</activity>
```

电话

发起通话

如需打开电话应用并拨打电话号码，请使用 `ACTION_DIAL` 操作，并使用下文定义的 URI 架构指定电话号码。电话应用打开时会显示电话号码，但用户必需按**拨打/电话**按钮才能开始通话。

如需直接拨打电话，请使用 `ACTION_CALL` 操作，并使用下文定义的 URI 架构指定电话号码。电话应用打开时便会拨打电话，用户无需按**拨打/电话**按钮。

`ACTION_CALL` 操作需要您在清单文件中添加 `CALL_PHONE` 权限：

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



Google Voice Actions

- “致电 555-5555”
- “致电 bob”
- “致电语音邮件”

操作

- `ACTION_DIAL` - 打开拨号器或电话应用。
- `ACTION_CALL` - 拨打电话（需要 `CALL_PHONE` 权限）

数据 URI 架构

- `tel:<phone-number>`
- `voicemail:<phone-number>`

MIME 类型

无

有效电话号码是指符合 [IETF RFC 3966](#) 规定的号码。举例来说，有效电话号码包括下列号码：

- `tel:2125551212`
- `tel:(212) 555 1212`

电话的拨号器能够很好地对架构进行标准化，如电话号码。因此并不严格要求 `Uri.parse()` 方法中必须使用所述架构。不过，如果您尚未试用过架构，或者不确定是否可以处理架构，请改用 `Uri.fromParts()` 方法。

示例 Intent：

```
public void dialPhoneNumber(String phoneNumber) {  
    Intent intent = new Intent(Intent.ACTION_DIAL);  
    intent.setData(Uri.parse("tel:" + phoneNumber));  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

搜索

使用特定应用搜索



Google Voice Actions

- “在我的视频应用中搜索有关猫的视频”



视频

在您的应用中进行语音搜索

如需支持在您的应用环境内进行搜索，请使用 `SEARCH_ACTION` 操作在您的应用中声明一个 Intent 过滤器，如下文示例 Intent 过滤器中所示。

操作

`"com.google.android.gms.actions.SEARCH_ACTION"`

支持来自 Google Voice Actions 的搜索查询。

Extra

`QUERY`

一个包含搜索查询的字符串。

示例 Intent 过滤器：

```
<activity android:name=".SearchActivity">
    <intent-filter>
        <action android:name="com.google.android.gms.actions.SEARCH_ACTION"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

执行网页搜索

如需发起网页搜索，请使用 `ACTION_WEB_SEARCH` 操作，并在 `SearchManager.QUERY` extra 中指定搜索字符串。

操作

`ACTION_WEB_SEARCH`

数据 URI 架构

无

MIME 类型

无

Extra

`SearchManager.QUERY`

搜索字符串。

示例 Intent：

```
public void searchWeb(String query) {
    Intent intent = new Intent(Intent.ACTION_SEARCH);
    intent.putExtra(SearchManager.QUERY, query);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

设置

打开特定设置部分

如需在您的应用要求用户更改内容时打开某个系统设置屏幕，请使用下列其中一个 Intent 操作打开与操作名称对应的设置屏幕。

操作

`ACTION_SETTINGS`
`ACTION_WIRELESS_SETTINGS`
`ACTION_AIRPLANE_MODE_SETTINGS`
`ACTION_WIFI_SETTINGS`
`ACTION_APN_SETTINGS`
`ACTION_BLUETOOTH_SETTINGS`
`ACTION_DATE_SETTINGS`
`ACTION_LOCALE_SETTINGS`
`ACTION_INPUT_METHOD_SETTINGS`
`ACTION_DISPLAY_SETTINGS`
`ACTION_SECURITY_SETTINGS`

```
ACTION_LOCATION_SOURCE_SETTINGS  
ACTION_INTERNAL_STORAGE_SETTINGS  
ACTION_MEMORY_CARD_SETTINGS
```

有关其他可用的设置屏幕，请参见 [Settings](#) 文档。

数据 URI 架构

无

MIME 类型

无

示例 Intent：

```
public void openWifiSettings() {  
    Intent intent = new Intent(Intent.ACTION_WIFI_SETTINGS);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

发送短信

撰写带附件的短信/彩信

如需发起短信或彩信，请使用以下其中一个 Intent 操作，并使用下列 extra 键指定电话号码、主题和消息正文等消息详情。

操作

```
ACTION_SENDTO 或  
ACTION_SEND 或  
ACTION_SEND_MULTIPLE
```

数据 URI 架构

`sms:<phone_number>`

`smsto:<phone_number>`

`mms:<phone_number>`

`mmsto:<phone_number>`

以上每一个架构的处理方式都相同。

MIME 类型

```
"text/plain"  
  
"image/*"  
  
"video/*"
```

Extra

"subject"

表示消息主题的字符串（通常只适用于彩信）。

"sms_body"

表示消息正文的字符串。

EXTRA_STREAM

指向要附加的图像或视频的 `Uri`。如果使用的是 `ACTION_SEND_MULTIPLE` 操作，此 extra 应为指向要附加的图像/视频的 `Uri ArrayList`。

示例 Intent：

```
public void composeMmsMessage(String message, Uri attachment) {  
    Intent intent = new Intent(Intent.ACTION_SENDTO);  
    intent.setType(HTTP.PLAIN_TEXT_TYPE);  
    intent.putExtra("sms_body", message);  
    intent.putExtra(Intent.EXTRA_STREAM, attachment);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

如果您想确保 Intent 只由短信应用（而非其他电子邮件或社交应用）进行处理，则需使用 `ACTION_SENDTO` 操作并加入 "smsto:" 数据架构。例如：

```
public void composeMmsMessage(String message, Uri attachment) {  
    Intent intent = new Intent(Intent.ACTION_SEND);  
    intent.setData(Uri.parse("smsto:")); // This ensures only SMS apps respond  
    intent.putExtra("sms_body", message);  
    intent.putExtra(Intent.EXTRA_STREAM, attachment);  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    }  
}
```

示例 Intent 过滤器：

```
<activity ...>  
    <intent-filter>  
        <action android:name="android.intent.action.SEND" />  
        <data android:type="text/plain" />  
        <data android:type="image/*" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>
```

注：如果您要开发短信/彩信应用，必须为几项额外操作实现 Intent 过滤器，才能在 Android 4.4 及更高版本上成为默认短信应用。如需了解详细信息，请参见 [Telephony](#) 处的文档。

网络浏览器

加载网址

如需打开网页，请使用 `ACTION_VIEW` 操作，并在 Intent 数据中指定网址。

操作

`ACTION_VIEW`



Google Voice Actions

- “打开 example.com”

数据 URI 架构

```
http:<URL>
https:<URL>
```

MIME 类型

```
"text/plain"
"text/html"
"application/xhtml+xml"
"application/vnd.wap.xhtml+xml"
```

示例 Intent：

```
public void openWebPage(String url) {
    Uri webpage = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <!-- Include the host attribute if you want your app to respond
            only to URLs with your app's domain. -->
        <data android:scheme="http" android:host="www.example.com" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- The BROWSABLE category is required to get links from web pages. -->
        <category android:name="android.intent.category.BROWSABLE" />
    </intent-filter>
</activity>
```

提示：如果您的 Android 应用提供与您的网站相似的功能，请为指向您的网站的 URL 加入一个 Intent 过滤器。之后，如果用户安装了您的应用，点击电子邮件或其他网页中指向您的网站的链接时，将会打开您的 Android 应用而不是您的网页。

使用 Android 调试桥验证 Intent

如需验证您的应用可以对您想支持的 Intent 作出响应，可以使用 `adb` 工具来触发特定 Intent：

1. 设置一台用于[开发](#)的 Android 设备，或使用一台[虚拟设备](#)。
2. 安装一个处理您想支持的 Intent 的应用版本。
3. 使用 `adb` 触发一个 Intent：

```
adb shell am start -a <ACTION> -t < MIME_TYPE > -d < DATA > \
-e < EXTRA_NAME > < EXTRA_VALUE > -n < ACTIVITY >
```

例如：

```
adb shell am start -a android.intent.action.DIAL \
-d tel:555-5555 -n org.example.MyApp/.MyActivity
```

4. 如果您定义了必需的 Intent 过滤器，您的应用应该会处理 Intent。

如需了解详细信息，请参见 [ADB Shell 命令](#)。



Activity

本文内容

- › [创建 Activity](#)
- › [实现用户界面](#)
- › [在清单文件中声明 Activity](#)
- › [启动 Activity](#)
- › [启动 Activity 以获得结果](#)
- › [结束 Activity](#)
- › [管理 Activity 生命周期](#)
- › [实现生命周期回调](#)
- › [保存 Activity 状态](#)
- › [处理配置变更](#)
- › [协调 Activity](#)

关键类

- › [Activity](#)

另请参阅

- › [任务和返回栈](#)

[Activity](#) 是一个应用组件，用户可与其提供的屏幕进行交互，以执行拨打电话、拍摄照片、发送电子邮件或查看地图等操作。每个 Activity 都会获得一个用于绘制其用户界面的窗口。窗口通常会充满屏幕，但也可小于屏幕并浮动在其他窗口之上。

一个应用通常由多个彼此松散联系的 Activity 组成。一般会指定应用中的某个 Activity 为“主”Activity，即首次启动应用时呈现给用户的那个 Activity。而且每个 Activity 均可启动另一个 Activity，以便执行不同的操作。每次新 Activity 启动时，前一 Activity 便会停止，但系统会在堆栈（“返回栈”）中保留该 Activity。当新 Activity 启动时，系统会将其推送到返回栈上，并取得用户焦点。返回栈遵循基本的“后进先出”堆栈机制，因此，当用户完成当前 Activity 并按“返回”按钮时，系统会从堆栈中将其弹出（并销毁），然后恢复前一 Activity。（[任务和返回栈](#)文档中对返回栈有更详细的阐述。）

当一个 Activity 因某个新 Activity 启动而停止时，系统会通过该 Activity 的生命周期回调方法通知其这一状态变化。Activity 因状态变化—系统是创建 Activity、停止 Activity、恢复 Activity 还是销毁 Activity—而收到的回调方法可能有若干种，每一种回调都会为您提供执行与该状态变化相应的特定操作的机会。例如，停止时，您的 Activity 应释放任何大型对象，例如网络或数据库连接。当 Activity 恢复时，您可以重新获取所需资源，并恢复执行中断的操作。这些状态转变都是 Activity 生命周期的一部分。

本文的其余部分阐述有关如何创建和使用 Activity 的基础知识（包括对 Activity 生命周期工作方式的全面阐述），以便您正确管理各种 Activity 状态之间的转变。

创建 Activity

要创建 Activity，您必须创建 [Activity](#) 的子类（或使用其现有子类）。您需要在子类中实现 Activity 在其生命周期的各种状态之间转变时（例如创建 Activity、停止 Activity、恢复 Activity 或销毁 Activity 时）系统调用的回调方法。两个最重要的回调方法是：

`onCreate()`

您必须实现此方法。系统会在创建您的 Activity 时调用此方法。您应该在实现内初始化 Activity 的必需组件。最重要的是，您必须在此方法内调用 [setContentView\(\)](#)，以定义 Activity 用户界面的布局。

onPause()

系统将此方法作为用户离开 Activity 的第一个信号（但并不总是意味着 Activity 会被销毁）进行调用。您通常应该在此方法内确认在当前用户会话结束后仍然有效的任何更改（因为用户可能不会返回）。

您还应使用几种其他生命周期回调方法，以便提供流畅的 Activity 间用户体验，以及处理导致您的 Activity 停止甚至被销毁的意外中断。后文的管理 Activity 生命周期部分对所有生命周期回调方法进行了阐述。

实现用户界面

Activity 的用户界面是由层级式视图 — 衍生自 `View` 类的对象 — 提供的。每个视图都控制 Activity 窗口内的特定矩形空间，可对用户交互作出响应。例如，视图可以是在用户触摸时启动某项操作的按钮。

您可以利用 Android 提供的许多现成视图设计和组织您的布局。“小部件”是提供按钮、文本字段、复选框或仅仅是一幅图像等屏幕视觉（交互式）元素的视图。“布局”是衍生自 `ViewGroup` 的视图，为其子视图提供唯一布局模型，例如线性布局、网格布局或相对布局。您还可以为 `View` 类和 `ViewGroup` 类创建子类（或使用其现有子类）来自行创建小部件和布局，然后将它们应用于您的 Activity 布局。

利用视图定义布局的最常见方法是借助保存在您的应用资源内的 XML 布局文件。这样一来，您就可以将用户界面的设计与定义 Activity 行为的源代码分开维护。您可以通过 `setContentView()` 将布局设置为 Activity 的 UI，从而传递布局的资源 ID。不过，您也可以在 Activity 代码中创建新 `View`，并通过将新 `View` 插入 `ViewGroup` 来创建视图层次，然后通过将根 `ViewGroup` 传递到 `setContentView()` 来使用该布局。

如需了解有关创建用户界面的信息，请参阅[用户界面文档](#)。

在清单文件中声明 Activity

您必须在清单文件中声明您的 Activity，这样系统才能访问它。要声明您的 Activity，请打开您的清单文件，并将 `<activity>` 元素添加为 `<application>` 元素的子项。例如：

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```

您还可以在此元素中加入几个其他特性，以定义 Activity 标签、Activity 图标或风格主题等用于设置 Activity UI 风格的属性。`android:name` 属性是唯一必需的属性—它指定 Activity 的类名。应用一旦发布，即不应更改此类名，否则，可能会破坏诸如应用快捷方式等一些功能（请阅读博客文章 [Things That Cannot Change](#) [不能更改的内容]）。

请参阅 `<activity>` 元素参考文档，了解有关在清单文件中声明 Activity 的详细信息。

使用 Intent 过滤器

`<activity>` 元素还可指定各种 Intent 过滤器—使用 `<intent-filter>` 元素—以声明其他应用组件激活它的方法。

当您使用 Android SDK 工具创建新应用时，系统自动为您创建的存根 Activity 包含一个 Intent 过滤器，其中声明了该 Activity 响应“主”操作且应置于“launcher”类别内。Intent 过滤器的内容如下所示：

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

`<action>` 元素指定这是应用的“主”入口点。`<category>` 元素指定此 Activity 应列入系统的应用启动器内（以便用户启动该 Activity）。

如果您打算让应用成为独立应用，不允许其他应用激活其 Activity，则您不需要任何其他 Intent 过滤器。正如前例所示，只应有一个 Activity 具有“主”操作和“launcher”类别。您不想提供给其他应用的 Activity 不应有任何 Intent 过滤器，您可以利用显式 Intent 自行启动它们（下文对此做了阐述）。

不过，如果您想让 Activity 对衍生自其他应用（以及您的自有应用）的隐式 Intent 作出响应，则必须为 Activity 定义其他 Intent 过滤器。对于

您想要作出响应的每一个 Intent 类型，您都必须加入相应的 `<intent-filter>`，其中包括一个 `<action>` 元素，还可选择性地包括一个 `<category>` 元素和/或一个 `<data>` 元素。这些元素指定您的 Activity 可以响应的 Intent 类型。

如需了解有关您的 Activity 如何响应 Intent 的详细信息，请参阅 [Intent 和 Intent 过滤器](#) 文档。

启动 Activity

您可以通过调用 `startActivity()`，并将其传递给描述您想启动的 Activity 的 Intent 来启动另一个 Activity。Intent 对象会指定您想启动的具体 Activity 或描述您想执行的操作类型（系统会为您选择合适的 Activity，甚至是来自其他应用的 Activity）。Intent 对象还可能携带少量供所启动 Activity 使用的数据。

在您的自有应用内工作时，您经常只需要启动某个已知 Activity。您可以通过使用类名创建一个显式定义您想启动的 Activity 的 Intent 对象来实现此目的。例如，可以通过以下代码让一个 Activity 启动另一个名为 `SignInActivity` 的 Activity：

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

不过，您的应用可能还需要利用您的 Activity 数据执行某项操作，例如发送电子邮件、短信或状态更新。在这种情况下，您的应用自身可能不具有执行此类操作所需的 Activity，因此您可以改为利用设备上其他应用提供的 Activity 为您执行这些操作。这便是 Intent 对象的真正价值所在——您可以创建一个 Intent 对象，对您想执行的操作进行描述，系统会从其他应用启动相应的 Activity。如果有多个 Activity 可以处理 Intent，则用户可以选择要使用哪一个。例如，如果您想允许用户发送电子邮件，可以创建以下 Intent：

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

添加到 Intent 中的 `EXTRA_EMAIL` extra 是一个字符串数组，其中包含应将电子邮件发送到的电子邮件地址。当电子邮件应用响应此 Intent 时，它会读取 extra 中提供的字符串数组，并将它们放入电子邮件撰写窗体的“收件人”字段。在这种情况下，电子邮件应用的 Activity 启动，并且当用户完成操作时，您的 Activity 会恢复执行。

启动 Activity 以获得结果

有时，您可能需要从启动的 Activity 获得结果。在这种情况下，请通过调用 `startActivityForResult()`（而非 `startActivity()`）来启动 Activity。要想在随后收到后续 Activity 的结果，请实现 `onActivityResult()` 回调方法。当后续 Activity 完成时，它会使用 Intent 向您的 `onActivityResult()` 方法返回结果。

例如，您可能希望用户选取其中一位联系人，以便您的 Activity 对该联系人中的信息执行某项操作。您可以通过以下代码创建此类 Intent 并处理结果：

```
private void pickContact() {
    // Create an intent to "pick" a contact, as defined by the content provider URI
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
    startActivityForResult(intent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // If the request went well (OK) and the request was PICK_CONTACT_REQUEST
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
        // Perform a query to the contact's content provider for the contact's name
        Cursor cursor = getContentResolver().query(data.getData(),
            new String[] {Contacts.DISPLAY_NAME}, null, null, null);
        if (cursor.moveToFirst()) { // True if the cursor is not empty
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
            String name = cursor.getString(columnIndex);
            // Do something with the selected contact's name...
        }
    }
}
```

上例显示的是，您在处理 Activity 结果时应该在 `onActivityResult()` 方法中使用的基本逻辑。第一个条件检查请求是否成功（如果成功，则 `resultCode` 将为 `RESULT_OK`）以及此结果响应的请求是否已知——在此情况下，`requestCode` 与随 `startActivityForResult()` 发送的

第二个参数匹配。代码通过查询 Intent 中返回的数据（`data` 参数）从该处开始处理 Activity 结果。

实际情况是，ContentResolver 对一个内容提供程序执行查询，后者返回一个 Cursor，让查询的数据能够被读取。如需了解详细信息，请参阅[内容提供程序](#)文档。

如需了解有关 Intent 用法的详细信息，请参阅[Intent 和 Intent 过滤器](#)文档。

结束 Activity

您可以通过调用 Activity 的 `finish()` 方法来结束该 Activity。您还可以通过调用 `finishActivity()` 结束您之前启动的另一个 Activity。

注：在大多数情况下，您不应使用这些方法显式结束 Activity。正如下文有关 Activity 生命周期的部分所述，Android 系统会为您管理 Activity 的生命周期，因此您无需结束自己的 Activity。调用这些方法可能对预期的用户体验产生不良影响，因此只应在您确实不想让用户返回此 Activity 实例时使用。

管理 Activity 生命周期

通过实现回调方法管理 Activity 的生命周期对开发强大而又灵活的应用至关重要。Activity 的生命周期会直接受到 Activity 与其他 Activity、其任务及返回栈的关联性的影响。

Activity 基本上以三种状态存在：

继续

此 Activity 位于屏幕前台并具有用户焦点。（有时也将此状态称作“运行中”。）

暂停

另一个 Activity 位于屏幕前台并具有用户焦点，但此 Activity 仍可见。也就是说，另一个 Activity 显示在此 Activity 上方，并且该 Activity 部分透明或未覆盖整个屏幕。暂停的 Activity 处于完全活动状态（Activity 对象保留在内存中，它保留了所有状态和成员信息，并与窗口管理器保持连接），但在内存极度不足的情况下，可能会被系统终止。

停止

该 Activity 被另一个 Activity 完全遮盖（该 Activity 目前位于“后台”）。已停止的 Activity 同样仍处于活动状态（Activity 对象保留在内存中，它保留了所有状态和成员信息，但未与窗口管理器连接）。不过，它对用户不再可见，在他处需要内存时可能会被系统终止。

如果 Activity 处于暂停或停止状态，系统可通过要求其结束（调用其 `finish()` 方法）或直接终止其进程，将其从内存中删除。（将其结束或终止后）再次打开 Activity 时，必须重建。

实现生命周期回调

当一个 Activity 转入和转出上述不同状态时，系统会通过各种回调方法向其发出通知。所有回调方法都是挂钩，您可以在 Activity 状态发生变化时替代这些挂钩来执行相应操作。以下框架 Activity 包括每一个基本生命周期方法：

```
public class ExampleActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        // The activity is being created.  
    }  
    @Override  
    protected void onStart() {  
        super.onStart();  
        // The activity is about to become visible.  
    }  
    @Override  
    protected void onResume() {  
        super.onResume();  
        // The activity has become visible (it is now "resumed").  
    }  
    @Override  
    protected void onPause() {  
        super.onPause();  
        // Another activity is taking focus (this activity is about to be "paused").  
    }  
    @Override  
    protected void onStop() {  
        super.onStop();  
        // The activity is no longer visible (it is now "stopped")  
    }  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        // The activity is about to be destroyed.  
    }  
}
```

注：正如以上示例所示，您在实现这些生命周期方法时必须始终先调用超类实现，然后再执行任何操作。

这些方法共同定义 Activity 的整个生命周期。您可以通过实现这些方法监控 Activity 生命周期中的三个嵌套循环：

- Activity 的**整个生命周期**发生在 `onCreate()` 调用与 `onDestroy()` 调用之间。您的 Activity 应在 `onCreate()` 中执行“全局”状态设置（例如定义布局），并释放 `onDestroy()` 中的所有其余资源。例如，如果您的 Activity 有一个在后台运行的线程，用于从网络上下载数据，它可能会在 `onCreate()` 中创建该线程，然后在 `onDestroy()` 中停止该线程。
- Activity 的**可见生命周期**发生在 `onStart()` 调用与 `onStop()` 调用之间。在这段时间，用户可以在屏幕上看到 Activity 并与其交互。例如，当一个新 Activity 启动，并且此 Activity 不再可见时，系统会调用 `onStop()`。您可以在调用这两个方法之间保留向用户显示 Activity 所需的资源。例如，您可以在 `onStart()` 中注册一个 `BroadcastReceiver` 以监控影响 UI 的变化，并在用户无法再看到您显示的内容时在 `onStop()` 中将其取消注册。在 Activity 的整个生命周期，当 Activity 在对用户可见和隐藏两种状态中交替变化时，系统可能会多次调用 `onStart()` 和 `onStop()`。
- Activity 的**前台生命周期**发生在 `onResume()` 调用与 `onPause()` 调用之间。在这段时间，Activity 位于屏幕上的所有其他 Activity 之前，并具有用户输入焦点。Activity 可频繁转入和转出前台 — 例如，当设备转入休眠状态或出现对话框时，系统会调用 `onPause()`。由于此状态可能经常发生转变，因此这两个方法中应采用适度轻量级的代码，以避免因转变速度慢而让用户等待。

图 1 说明了这些循环以及 Activity 在状态转变期间可能经过的路径。矩形表示回调方法，当 Activity 在不同状态之间转变时，您可以实现这些方法来执行操作。

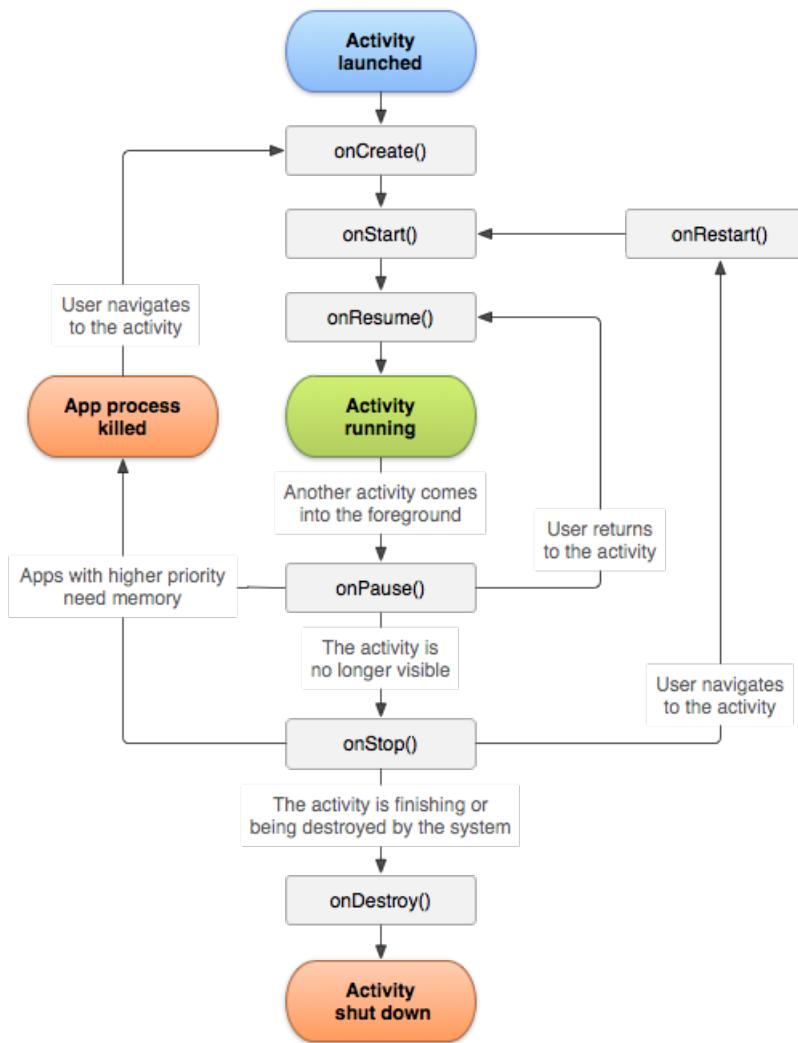


图 1. Activity 生命周期。

表 1 列出了相同的生命周期回调方法，其中对每一种回调方法做了更详细的描述，并说明了每一种方法在 Activity 整个生命周期内的位置，包括在回调方法完成后系统能否终止 Activity。

表 1. Activity 生命周期回调方法汇总表。

方法	说明	是否能事后终止？	后接
<code>onCreate()</code>	首次创建 Activity 时调用。您应该在此方法中执行所有正常的静态设置 — 创建视图、将数据绑定到列表等等。系统向此方法传递一个 Bundle 对象，其中包含 Activity 的上一状态，不过前提是捕获了该状态（请参阅后文的 保存 Activity 状态 ）。 始终后接 <code>onStart()</code> 。	否	<code>onStart()</code>
<code>onRestart()</code>	在 Activity 已停止并即将再次启动前调用。 始终后接 <code>onStart()</code>	否	<code>onStart()</code>
<code>onStart()</code>	在 Activity 即将对用户可见之前调用。 如果 Activity 转入前台，则后接 <code>onResume()</code> ，如果 Activity 转入隐藏状态，则后接 <code>onStop()</code> 。	否	<code>onResume()</code> 或 <code>onStop()</code>
<code>onResume()</code>	在 Activity 即将开始与用户进行交互之前调用。此时，Activity 处于 Activity 堆栈的顶层，并具有用户输入焦点。 始终后接 <code>onPause()</code> 。	否	<code>onPause()</code>
<code>onPause()</code>	当系统即将开始继续另一个 Activity 时调用。此方法通常用于确认对持久性数据的未保存更改、停止动画以及其他可能消耗 CPU 的内容，诸如此类。它应该非常	是	<code>onResume()</code> 或

方法		说明	是否能事后终止	后接 <code>Stop()</code>
	<code>onStop()</code>	说明地执行所需操作，因为它返回后，下一个 Activity 才能继续执行。 如果 Activity 返回前台，则后接 <code>onResume()</code> ，如果 Activity 转入对用户不可见状态，则后接 <code>onStop()</code> 。	是	
		在 Activity 对用户不再可见时调用。如果 Activity 被销毁，或另一个 Activity（一个现有 Activity 或新 Activity）继续执行并将其覆盖，就可能发生这种情况。 如果 Activity 恢复与用户的交互，则后接 <code>onRestart()</code> ，如果 Activity 被销毁，则后接 <code>onDestroy()</code> 。	否？	或 <code>onDestroy()</code>
	<code>onDestroy()</code>	在 Activity 被销毁前调用。这是 Activity 将收到的最后调用。当 Activity 结束（有人对 Activity 调用了 <code>finish()</code> ），或系统为节省空间而暂时销毁该 Activity 实例时，可能会调用它。您可以通过 <code>isFinishing()</code> 方法区分这两种情形。	是	无

名为“是否能事后终止？”的列表示系统是否能在不执行另一行 Activity 代码的情况下，在方法返回后随时终止承载 Activity 的进程。有三个方法带有“是”标记：`(onPause(), onStop() 和 onDestroy())`。由于 `onPause()` 是这三个方法中的第一个，因此 Activity 创建后，`onPause()` 必定成为最后调用的方法，然后才能终止进程 — 如果系统在紧急情况下必须恢复内存，则可能不会调用 `onStop()` 和 `onDestroy()`。因此，您应该使用 `onPause()` 向存储设备写入至关重要的持久性数据（例如用户编辑）。不过，您应该对 `onPause()` 调用期间必须保留的信息有所选择，因为该方法中的任何阻止过程都会妨碍向下一个 Activity 的转变并拖慢用户体验。

在是否能在事后终止？列中标记为“否”的方法可从系统调用它们的一刻起防止承载 Activity 的进程被终止。因此，在从 `onPause()` 返回的时间到 `onResume()` 被调用的时间，系统可以终止 Activity。在 `onPause()` 被再次调用并返回前，将无法再次终止 Activity。

注：根据表 1 中的定义属于技术上无法“终止”的 Activity 仍可能被系统终止 — 但这种情况只有在无任何其他资源的极端情况下才会发生。[进程和线程处理文档](#)对可能会终止 Activity 的情况做了更详尽的阐述。

保存 Activity 状态

管理 [Activity 生命周期](#) 的引言部分简要提及，当 Activity 暂停或停止时，Activity 的状态会得到保留。确实如此，因为当 Activity 暂停或停止时，[Activity](#) 对象仍保留在内存中 — 有关其成员和当前状态的所有信息仍处于活动状态。因此，用户在 Activity 内所做的任何更改都会得到保留，这样一来，当 Activity 返回前台（当它“继续”）时，这些更改仍然存在。

不过，当系统为了恢复内存而销毁某项 Activity 时，[Activity](#) 对象也会被销毁，因此系统在继续 Activity 时根本无法让其状态保持完好，而是必须在用户返回 Activity 时重建 [Activity](#) 对象。但用户并不知道系统销毁 Activity 后又对其进行了重建，因此他们很可能认为 Activity 状态毫无变化。在这种情况下，您可以实现另一个回调方法对有关 Activity 状态的信息进行保存，以确保有关 Activity 状态的重要信息得到保留：`onSaveInstanceState()`。

系统会先调用 `onSaveInstanceState()`，然后再使 Activity 变得易于销毁。系统会向该方法传递一个 [Bundle](#)，您可以在其中使用 `putString()` 和 `putInt()` 等方法以名称-值对形式保存有关 Activity 状态的信息。然后，如果系统终止您的应用进程，并且用户返回您的 Activity，则系统会重建该 Activity，并将 [Bundle](#) 同时传递给 `onCreate()` 和 `onRestoreInstanceState()`。您可以使用上述任一方法从 [Bundle](#) 提取您保存的状态并恢复该 Activity 状态。如果没有状态信息需要恢复，则传递给您的 [Bundle](#) 是空值（如果是首次创建该 Activity，就会出现这种情况）。

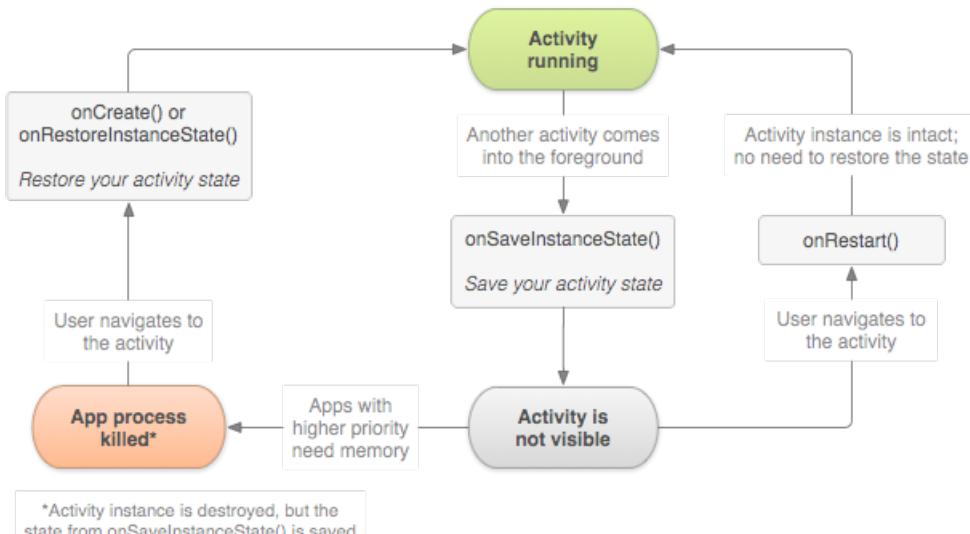


图 2. 在两种情况下，Activity 重获用户焦点时可保持状态完好：系统在销毁 Activity 后重建 Activity，Activity 必须恢复之前保存的状态；系统停止 Activity

后继续执行 Activity，并且 Activity 状态保持完好。

注：无法保证系统会在销毁您的 Activity 前调用 `onSaveInstanceState()`，因为存在不需要保存状态的情况（例如用户使用“返回”按钮离开您的 Activity 时，因为用户的行为是在显式关闭 Activity）。如果系统调用 `onSaveInstanceState()`，它会在调用 `onStop()` 之前，并且可能会在调用 `onPause()` 之前进行调用。

不过，即使您什么都不做，也不实现 `onSaveInstanceState()`，`Activity` 类的 `onSaveInstanceState()` 默认实现也会恢复部分 Activity 状态。具体地讲，默认实现会为布局中的每个 `View` 调用相应的 `onSaveInstanceState()` 方法，让每个视图都能提供有关自身的应保存信息。Android 框架中几乎每个小部件都会根据需要实现此方法，以便在重建 Activity 时自动保存和恢复对 UI 所做的任何可见更改。例如，`EditText` 小部件保存用户输入的任何文本，`CheckBox` 小部件保存复选框的选中或未选中状态。您只需为想要保存其状态的每个小部件提供一个唯一的 ID（通过 `android:id` 属性）。如果小部件没有 ID，则系统无法保存其状态。

尽管 `onSaveInstanceState()` 的默认实现会保存有关您的 Activity UI 的有用信息，您可能仍需替换它以保存更多信息。例如，您可能需要保存在 Activity 生命周期内发生了变化的成员值（它们可能与 UI 中恢复的值有关联，但默认情况下系统不会恢复储存这些 UI 值的成员）。

由于 `onSaveInstanceState()` 的默认实现有助于保存 UI 的状态，因此如果您为了保存更多状态信息而替换该方法，应始终先调用 `onSaveInstanceState()` 的超类实现，然后再执行任何操作。同样，如果您替换 `onRestoreInstanceState()` 方法，也应调用它的超类实现，以便默认实现能够恢复视图状态。

您还可以通过将 `android:saveEnabled` 属性设置为 `"false"` 或通过调用 `setSaveEnabled()` 方法显式阻止布局内的视图保存其状态。您通常不应将该属性停用，但如果想以不同方式恢复 Activity UI 的状态，就可能需要这样做。

注：由于无法保证系统会调用 `onSaveInstanceState()`，因此您只应利用它来记录 Activity 的瞬态（UI 的状态）—切勿使用它来存储持久性数据，而应使用 `onPause()` 在用户离开 Activity 后存储持久性数据（例如应保存到数据库的数据）。

您只需旋转设备，让屏幕方向发生变化，就能有效地测试您的应用的状态恢复能力。当屏幕方向变化时，系统会销毁并重建 Activity，以便应用可供新屏幕配置使用的备用资源。单凭这一理由，您的 Activity 在重建时能否完全恢复其状态就显得非常重要，因为用户在使用应用时经常需要旋转屏幕。

处理配置变更

有些设备配置可能会在运行时发生变化（例如屏幕方向、键盘可用性及语言）。发生此类变化时，Android 会重建运行中的 Activity（系统调用 `onDestroy()`，然后立即调用 `onCreate()`）。此行为旨在通过利用您提供的备用资源（例如适用于不同屏幕方向和屏幕尺寸的不同布局）自动重新加载您的应用来帮助它适应新配置。

如果您对 Activity 进行了适当设计，让它能够按以上所述处理屏幕方向变化带来的重启并恢复 Activity 状态，那么在遭遇 Activity 生命周期中的其他意外事件时，您的应用将具有更强的适应性。

正如上文所述，处理此类重启的最佳方法是利用 `onSaveInstanceState()` 和 `onRestoreInstanceState()`（或 `onCreate()`）保存并恢复 Activity 的状态。

如需了解有关运行时发生的配置变更以及应对方法的详细信息，请阅读 [处理运行时变更指南](#)。

协调 Activity

当一个 Activity 启动另一个 Activity 时，它们都会体验到生命周期转变。第一个 Activity 暂停并停止（但如果它在后台仍然可见，则不会停止）时，同时系统会创建另一个 Activity。如果这些 Activity 共用保存到磁盘或其他地方的数据，必须了解的是，在创建第二个 Activity 前，第一个 Activity 不会完全停止。更确切地说，启动第二个 Activity 的过程与停止第一个 Activity 的过程存在重叠。

生命周期回调的顺序经过明确定义，当两个 Activity 位于同一进程，并且由一个 Activity 启动另一个 Activity 时，其定义尤其明确。以下是当 Activity A 启动 Activity B 时一系列操作的发生顺序：

1. Activity A 的 `onPause()` 方法执行。
2. Activity B 的 `onCreate()`、`onStart()` 和 `onResume()` 方法依次执行。（Activity B 现在具有用户焦点。）
3. 然后，如果 Activity A 在屏幕上不再可见，则其 `onStop()` 方法执行。

您可以利用这种可预测的生命周期回调顺序管理从一个 Activity 到另一个 Activity 的信息转变。例如，如果您必须在第一个 Activity 停止时向数据库写入数据，以便下一个 Activity 能够读取该数据，则应在 `onPause()` 而不是 `onStop()` 执行期间向数据库写入数据。

片段

本文内容

- › [设计原理](#)
- › [创建片段](#)
 - › [添加用户界面](#)
 - › [向 Activity 添加片段](#)
- › [管理片段](#)
- › [执行片段事务](#)
- › [与 Activity 通信](#)
 - › [创建对 Activity 的事件回调](#)
 - › [向应用栏添加项目](#)
- › [处理片段生命周期](#)
 - › [与 Activity 生命周期协调一致](#)
- › [示例](#)

关键类

- › [Fragment](#)
- › [FragmentManager](#)
- › [FragmentTransaction](#)

另请参阅

- › [利用片段构建动态 UI](#)
- › [支持平板电脑和手机](#)

[Fragment](#) 表示 [Activity](#) 中的行为或用户界面部分。您可以将多个片段组合在一个 [Activity](#) 中来构建多窗格 UI，以及在多个 [Activity](#) 中重复使用某个片段。您可以将片段视为 [Activity](#) 的模块化组成部分，它具有自己的生命周期，能接收自己的输入事件，并且您可以在 [Activity](#) 运行时添加或移除片段（有点像您可以在不同 [Activity](#) 中重复使用的“子 [Activity](#)”）。

片段必须始终嵌入在 [Activity](#) 中，其生命周期直接受宿主 [Activity](#) 生命周期的影响。例如，当 [Activity](#) 暂停时，其中的所有片段也会暂停；当 [Activity](#) 被销毁时，所有片段也会被销毁。不过，当 [Activity](#) 正在运行（处于已恢复生命周期状态）时，您可以独立操纵每个片段，如添加或移除它们。当您执行此类片段事务时，您也可以将其添加到由 [Activity](#) 管理的返回栈 — [Activity](#) 中的每个返回栈条目都是一条已发生片段事务的记录。返回栈让用户可以通过按[返回](#)按钮撤消片段事务（后退）。

当您将片段作为 [Activity](#) 布局的一部分添加时，它存在于 [Activity](#) 视图层次结构的某个 [ViewGroup](#) 内部，并且片段会定义其自己的视图布局。您可以通过在 [Activity](#) 的布局文件中声明片段，将其作为 `<fragment>` 元素插入您的 [Activity](#) 布局中，或者通过将其添加到某个现有 [ViewGroup](#)，利用应用代码进行插入。不过，片段并非必须成为 [Activity](#) 布局的一部分；您还可以将没有自己 UI 的片段用作 [Activity](#) 的不可见工作线程。

本文描述如何在开发您的应用时使用片段，包括将片段添加到 [Activity](#) 返回栈时如何保持其状态、如何与 [Activity](#) 及 [Activity](#) 中的其他片段共享事件、如何为 [Activity](#) 的操作栏发挥作用等等。

设计原理

Android 在 Android 3.0 (API 级别 11) 中引入了片段，主要是为了给大屏幕（如平板电脑）上更加动态和灵活的 UI 设计提供支持。由于平板电脑的屏幕比手机屏幕大得多，因此可用于组合和交换 UI 组件的空间更大。利用片段实现此类设计时，您无需管理对视图层次结构的复杂更改。通过将 [Activity](#) 布局分成片段，您可以在运行时修改 [Activity](#) 的外观，并在由 [Activity](#) 管理的返回栈中保留这些更改。

例如，新闻应用可以使用一个片段在左侧显示文章列表，使用另一个片段在右侧显示文章 — 两个片段并排显示在一个 Activity 中，每个片段都具有自己的一套生命周期回调方法，并各自处理自己的用户输入事件。因此，用户不需要使用一个 Activity 来选择文章，然后使用另一个 Activity 来阅读文章，而是可以在同一个 Activity 内选择文章并进行阅读，如图 1 中的平板电脑布局所示。

您应该将每个片段都设计为可重复使用的模块化 Activity 组件。也就是说，由于每个片段都会通过各自的生命周期回调来定义其自己的布局和行为，您可以将一个片段加入多个 Activity，因此，您应该采用可复用式设计，避免直接从某个片段直接操纵另一个片段。这特别重要，因为模块化片段让您可以通过更改片段的组合方式来适应不同的屏幕尺寸。在设计可同时支持平板电脑和手机的应用时，您可以在不同的布局配置中重复使用您的片段，以根据可用的屏幕空间优化用户体验。例如，在手机上，如果不能在同一 Activity 内储存多个片段，可能必须利用单独片段来实现单窗格 UI。

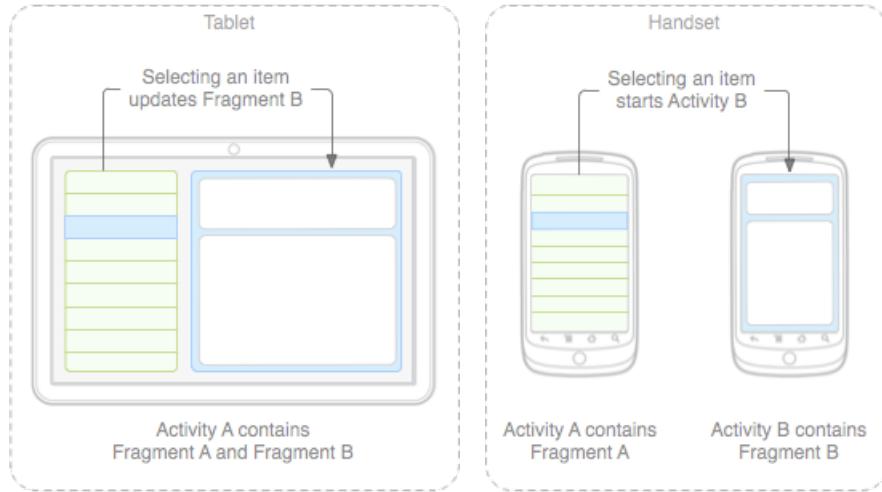


图 1. 有关由片段定义的两个 UI 模块如何适应不同设计的示例：通过组合成一个 Activity 来适应平板电脑设计，通过单独片段来适应手机设计。

例如 — 仍然以新闻应用为例 — 在平板电脑尺寸的设备上运行时，该应用可以在 *Activity A* 中嵌入两个片段。不过，在手机尺寸的屏幕上，没有足以储存两个片段的空间，因此 *Activity A* 只包括用于显示文章列表的片段，当用户选择文章时，它会启动 *Activity B*，其中包括用于阅读文章的第二个片段。因此，应用可通过重复使用不同组合的片段来同时支持平板电脑和手机，如图 1 所示。

如需了解有关通过利用不同片段组合来适应不同屏幕配置这种方法设计应用的详细信息，请参阅[支持平板电脑和手机指南](#)。

创建片段

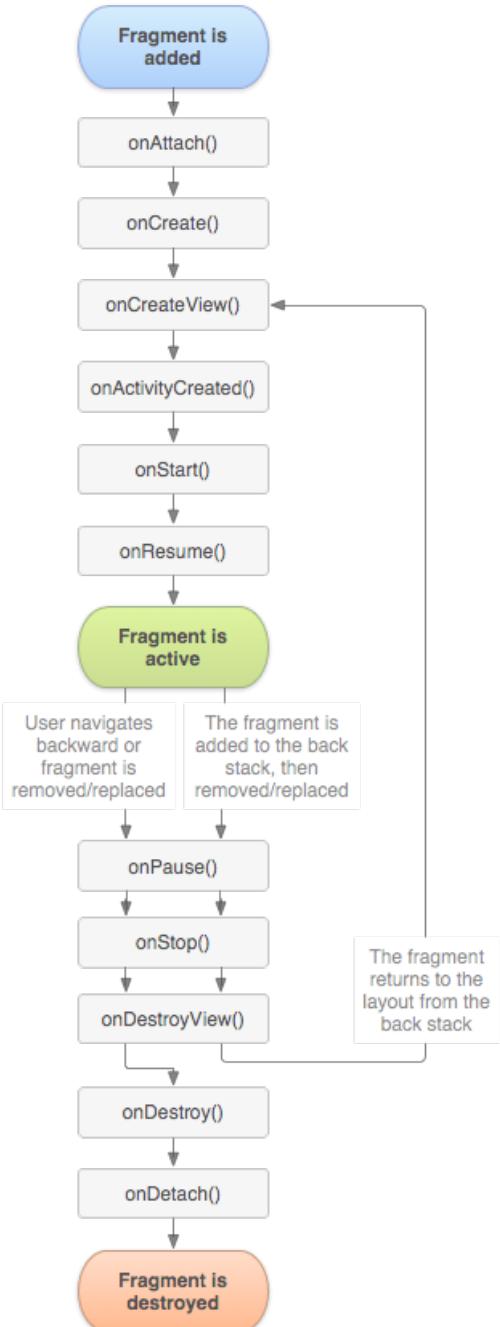


图 2. 片段的生命周期（其 Activity 运行时）。

要想创建片段，您必须创建 `Fragment` 的子类（或已有其子类）。`Fragment` 类的代码与 `Activity` 非常相似。它包含与 `Activity` 类似的回调方法，如 `onCreate()`、`onStart()`、`onPause()` 和 `onStop()`。实际上，如果您要将现有 Android 应用转换为使用片段，可能只需将代码从 `Activity` 的回调方法移入片段相应的回调方法中。

通常，您至少应实现以下生命周期方法：

`onCreate()`

系统会在创建片段时调用此方法。您应该在实现内初始化您想在片段暂停或停止后恢复时保留的必需片段组件。

`onCreateView()`

系统会在片段首次绘制其用户界面时调用此方法。要想为您的片段绘制 UI，您从此方法中返回的 `View` 必须是片段布局的根视图。如果片段未提供 UI，您可以返回 null。

`onPause()`

系统将此方法作为用户离开片段的第一个信号（但并不总是意味着此片段会被销毁）进行调用。您通常应该在此方法内确认在当前用户会话结束后仍然有效的任何更改（因为用户可能不会返回）。

大多数应用都应该至少为每个片段实现这三个方法，但您还应该使用几种其他回调方法来处理片段生命周期的各个阶段。[处理片段生命周期](#)部分对所有生命周期回调方法做了更详尽的阐述。

您可能还想扩展几个子类，而不是 [Fragment](#) 基类：

[DialogFragment](#)

显示浮动对话框。使用此类创建对话框可有效地替代使用 [Activity](#) 类中的对话框帮助程序方法，因为您可以将片段对话框纳入由 [Activity](#) 管理的片段返回栈，从而使用户能够返回清除的片段。

[ListFragment](#)

显示由适配器（如 [SimpleCursorAdapter](#)）管理的一系列项目，类似于 [ListActivity](#)。它提供了几种管理列表视图的方法，如用于处理点击事件的 [onListItemClick\(\)](#) 回调。

[PreferenceFragment](#)

以列表形式显示 [Preference](#) 对象的层次结构，类似于 [PreferenceActivity](#)。这在为您的应用创建“设置”Activity 时很有用处。

添加用户界面

片段通常用作 Activity 用户界面的一部分，将其自己的布局融入 Activity。

要想为片段提供布局，您必须实现 [onCreateView\(\)](#) 回调方法，Android 系统会在片段需要绘制其布局时调用该方法。您对此方法的实现返回的 [View](#) 必须是片段布局的根视图。

注：如果您的片段是 [ListFragment](#) 的子类，则默认实现会从 [onCreateView\(\)](#) 返回一个 [ListView](#)，因此您无需实现它。

要想从 [onCreateView\(\)](#) 返回布局，您可以通过 XML 中定义的 [布局资源](#) 来扩展布局。为帮助您执行此操作，[onCreateView\(\)](#) 提供了一个 [LayoutInflater](#) 对象。

例如，以下这个 [Fragment](#) 子类从 `example_fragment.xml` 文件加载布局：

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

传递至 [onCreateView\(\)](#) 的 `container` 参数是您的片段布局将插入到的父 [ViewGroup](#)（来自 Activity 的布局）。`savedInstanceState` 参数是在恢复片段时，提供上一片段实例相关数据的 [Bundle](#)（[处理片段生命周期](#) 部分对恢复状态做了详细阐述）。

[inflate\(\)](#) 方法带有三个参数：

- 您想要扩展的布局的资源 ID；
- 将作为扩展布局父项的 [ViewGroup](#)。传递 `container` 对系统向扩展布局的根视图（由其所属的父视图指定）应用布局参数具有重要意义；
- 指示是否应该在扩展期间将扩展布局附加至 [ViewGroup](#)（第二个参数）的布尔值。（在本例中，其值为 `false`，因为系统已经将扩展布局插入 `container` — 传递 `true` 值会在最终布局中创建一个多余的视图组。）

现在，您已经了解了如何创建提供布局的片段。接下来，您需要将该片段添加到您的 Activity 中。

向 Activity 添加片段

通常，片段向宿主 Activity 贡献一部分 UI，作为 Activity 总体视图层次结构的一部分嵌入到 Activity 中。可以通过两种方式向 Activity 布局添加片段：

创建布局

在上例中，`R.layout.example_fragment` 是对应用资源中保存的名为 `example_fragment.xml` 的布局资源的引用。如需了解有关如何在 XML 中创建布局的信息，请参阅[用户界面文档](#)。

• 在 Activity 的布局文件内声明片段

在本例中，您可以将片段当作视图来为其指定布局属性。例如，以下是一个具有两个片段的 Activity 的布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

<fragment> 中的 android:name 属性指定要在布局中实例化的 Fragment 类。

当系统创建此 Activity 布局时，会实例化在布局中指定的每个片段，并为每个片段调用 `onCreateView()` 方法，以检索每个片段的布局。系统会直接插入片段返回的 View 来替代 <fragment> 元素。

注：每个片段都需要一个唯一的标识符，重启 Activity 时，系统可以使用该标识符来恢复片段（您也可以使用该标识符来捕获片段以执行某些事务，如将其移除）。可以通过三种方式为片段提供 ID：

- 为 android:id 属性提供唯一 ID。
- 为 android:tag 属性提供唯一字符串。
- 如果您未给以上两个属性提供值，系统会使用容器视图的 ID。

• 或者通过编程方式将片段添加到某个现有 ViewGroup

您可以在 Activity 运行期间随时将片段添加到 Activity 布局中。您只需指定要将片段放入哪个 ViewGroup。

要想在您的 Activity 中执行片段事务（如添加、移除或替换片段），您必须使用 `FragmentManager` 中的 API。您可以像下面这样从 Activity 获取一个 `FragmentTransaction` 实例：

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

然后，您可以使用 `add()` 方法添加一个片段，指定要添加的片段以及将其插入哪个视图。例如：

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

传递到 `add()` 的第一个参数是 `ViewGroup`，即应该放置片段的位置，由资源 ID 指定，第二个参数是要添加的片段。

一旦您通过 `FragmentTransaction` 做出了更改，就必须调用 `commit()` 以使更改生效。

添加没有 UI 的片段

上例展示了如何向您的 Activity 添加片段以提供 UI。不过，您还可以使用片段为 Activity 提供后台行为，而不显示额外 UI。

要想添加没有 UI 的片段，请使用 `add(Fragment, String)` 从 Activity 添加片段（为片段提供一个唯一的字符串“标记”，而不是视图 ID）。这会添加片段，但由于它并不与 Activity 布局中的视图关联，因此不会收到对 `onCreateView()` 的调用。因此，您不需要实现该方法。

并非只能为非 UI 片段提供字符串标记 — 您也可以为具有 UI 的片段提供字符串标记 — 但如果片段没有 UI，则字符串标记将是标识它的唯一方式。如果您想稍后从 Activity 中获取片段，则需要使用 `findFragmentByTag()`。

如需查看将没有 UI 的片段用作后台工作线程的示例 Activity，请参阅 `FragmentRetainInstance.java` 示例，该示例包括在 SDK 示例（通过 Android SDK 管理器提供）中，以

`<sdk_root>/API Demos/app/src/main/java/com/example/android/apis/app/FragmentRetainInstance.java` 形式位于您的系统中。

管理片段

要想管理您的 Activity 中的片段，您需要使用 [FragmentManager](#)。要想获取它，请从您的 Activity 调用 `getFragmentManager()`。

您可以使用 [FragmentManager](#) 执行的操作包括：

- 通过 `findFragmentById()`（对于在 Activity 布局中提供 UI 的片段）或 `findFragmentByTag()`（对于提供或不提供 UI 的片段）获取 Activity 中存在的片段。
- 通过 `popBackStack()`（模拟用户发出的返回命令）将片段从返回栈中弹出。
- 通过 `addOnBackStackChangedListener()` 注册一个侦听返回栈变化的监听器。

如需了解有关这些方法以及其他方法的详细信息，请参阅 [FragmentManager](#) 类文档。

如上文所示，您也可以使用 [FragmentManager](#) 打开一个 [FragmentTransaction](#)，通过它来执行某些事务，如添加和移除片段。

执行片段事务

在 Activity 中使用片段的一大优点是，可以根据用户行为通过它们执行添加、移除、替换以及其他操作。您提交给 Activity 的每组更改都称为事务，您可以使用 [FragmentTransaction](#) 中的 API 来执行一项事务。您也可以将每个事务保存到由 Activity 管理的返回栈内，从而让用户能够回退片段更改（类似于回退 Activity）。

您可以像下面这样从 [FragmentManager](#) 获取一个 [FragmentTransaction](#) 实例：

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

每个事务都是您想要同时执行的一组更改。您可以使用 `add()`、`remove()` 和 `replace()` 等方法为给定事务设置您想要执行的所有更改。然后，要想将事务应用到 Activity，您必须调用 `commit()`。

不过，在您调用 `commit()` 之前，您可能想调用 `addToBackStack()`，以将事务添加到片段事务返回栈。该返回栈由 Activity 管理，允许用户通过按返回按钮返回上一片段状态。

例如，以下示例说明了如何将一个片段替换成另一个片段，以及如何在返回栈中保留先前状态：

```
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

在上例中，`newFragment` 会替换目前在 `R.id.fragment_container` ID 所标识的布局容器中的任何片段（如有）。通过调用 `addToBackStack()` 可将替换事务保存到返回栈，以便用户能够通过按返回按钮撤消事务并回退到上一片段。

如果您向事务添加了多个更改（如又一个 `add()` 或 `remove()`），并且调用了 `addToBackStack()`，则在调用 `commit()` 前应用的所有更改都将作为单一事务添加到返回栈，并且返回按钮会将它们一并撤消。

向 [FragmentTransaction](#) 添加更改的顺序无关紧要，不过：

- 您必须最后调用 `commit()`
- 如果您要向同一容器添加多个片段，则您添加片段的顺序将决定它们在视图层次结构中的出现顺序

如果您没有在执行移除片段的事务时调用 `addToBackStack()`，则事务提交时该片段会被销毁，用户将无法回退到该片段。不过，如果您在删除片段时调用了 `addToBackStack()`，则系统会停止该片段，并在用户回退时将其恢复。

提示：对于每个片段事务，您都可以通过在提交前调用 `setTransition()` 来应用过渡动画。

调用 `commit()` 不会立即执行事务，而是在 Activity 的 UI 线程（“主”线程）可以执行该操作时再安排其在线程上运行。不过，如有必要，您也可以从 UI 线程调用 `executePendingTransactions()` 以立即执行 `commit()` 提交的事务。通常不必这样做，除非其他线程中的作业依赖该事务。

注意：您只能在 Activity 保存其状态（用户离开 Activity）之前使用 `commit()` 提交事务。如果您试图在该时间点后提交，则会引发异常。

这是因为如需恢复 Activity，则提交后的状态可能会丢失。对于丢失提交无关紧要的情况，请使用 `commitAllowingStateLoss()`。

与 Activity 通信

尽管 `Fragment` 是作为独立于 `Activity` 的对象实现，并且可在多个 `Activity` 内使用，但片段的给定实例会直接绑定到包含它的 `Activity`。

具体地说，片段可以通过 `getActivity()` 访问 `Activity` 实例，并轻松地执行在 `Activity` 布局中查找视图等任务。

```
View listView = getActivity().findViewById(R.id.list);
```

同样地，您的 `Activity` 也可以使用 `findFragmentById()` 或 `findFragmentByTag()`，通过从 `FragmentManager` 获取对 `Fragment` 的引用来自调用片段中的方法。例如：

```
ExampleFragment fragment = (ExampleFragment) getSupportFragmentManager().findFragmentById(R.id.example_fragment);
```

创建对 Activity 的事件回调

在某些情况下，您可能需要通过片段与 `Activity` 共享事件。执行此操作的一个好方法是，在片段内定义一个回调接口，并要求宿主 `Activity` 实现它。当 `Activity` 通过该接口收到回调时，可以根据需要与布局中的其他片段共享这些信息。

例如，如果一个新闻应用的 `Activity` 有两个片段 — 一个用于显示文章列表（片段 A），另一个用于显示文章（片段 B）— 那么片段 A 必须在列表项被选定后告知 `Activity`，以便它告知片段 B 显示该文章。在本例中，`OnArticleSelectedListener` 接口在片段 A 内声明：

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

然后，该片段的宿主 `Activity` 会实现 `OnArticleSelectedListener` 接口并替代 `onArticleSelected()`，将来自片段 A 的事件通知片段 B。为确保宿主 `Activity` 实现此接口，片段 A 的 `onAttach()` 回调方法（系统在向 `Activity` 添加片段时调用的方法）会通过转换传递到 `onAttach()` 中的 `Activity` 来实例化 `OnArticleSelectedListener` 的实例：

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement OnArticleSelectedListener");
        }
    }
    ...
}
```

如果 `Activity` 未实现接口，则片段会引发 `ClassCastException`。实现时，`mListener` 成员会保留对 `Activity` 的 `OnArticleSelectedListener` 实现的引用，以便片段 A 可以通过调用 `OnArticleSelectedListener` 接口定义的方法与 `Activity` 共享事

件。例如，如果片段 A 是 `ListFragment` 的一个扩展，则用户每次点击列表项时，系统都会调用片段中的 `onListItemClick()`，然后该方法会调用 `onArticleSelected()` 以与 Activity 共享事件：

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Append the clicked item's row ID with the content provider Uri
        Uri noteUri = ContentUris.withAppendedId(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host activity
        mListener.onArticleSelected(noteUri);
    }
    ...
}
```

传递到 `onListItemClick()` 的 `id` 参数是被点击项的行 ID，即 Activity（或其他片段）用来从应用的 `ContentProvider` 获取文章的 ID。

内容提供程序文档中提供了有关内容提供程序用法的更多详情。

向应用栏添加项目

您的片段可以通过实现 `onCreateOptionsMenu()` 向 Activity 的 [选项菜单](#)（并因此向[应用栏](#)）贡献菜单项。不过，为了使此方法能够收到调用，您必须在 `onCreate()` 期间调用 `setHasOptionsMenu()`，以指示片段想要向选项菜单添加菜单项（否则，片段将不会收到对 `onCreateOptionsMenu()` 的调用）。

您之后从片段添加到选项菜单的任何菜单项都将追加到现有菜单项之后。选定菜单项时，片段还会收到对 `onOptionsItemSelected()` 的回调。

您还可以通过调用 `registerForContextMenu()`，在片段布局中注册一个视图来提供上下文菜单。用户打开上下文菜单时，片段会收到对 `onCreateContextMenu()` 的调用。当用户选择某个菜单项时，片段会收到对 `onContextItemSelected()` 的调用。

注：尽管您的片段会收到与其添加的每个菜单项对应的菜单项选定回调，但当用户选择菜单项时，Activity 会首先收到相应的回调。如果 Activity 对菜单项选定回调的实现不会处理选定的菜单项，则系统会将事件传递到片段的回调。这适用于选项菜单和上下文菜单。

如需了解有关菜单的详细信息，请参阅[菜单开发者指南](#)和[应用栏培训课程](#)。

处理片段生命周期

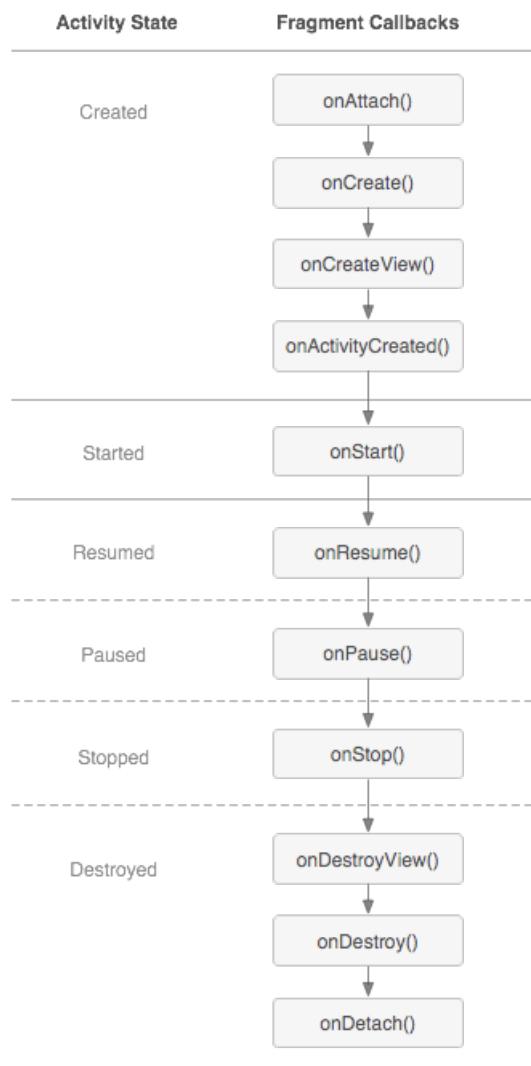


图 3. Activity 生命周期对片段生命周期的影响。

管理片段生命周期与管理 Activity 生命周期很相似。和 Activity 一样，片段也以三种状态存在：

继续

片段在运行中的 Activity 中可见。

暂停

另一个 Activity 位于前台并具有焦点，但此片段所在的 Activity 仍然可见（前台 Activity 部分透明，或未覆盖整个屏幕）。

停止

片段不可见。宿主 Activity 已停止，或片段已从 Activity 中移除，但已添加到返回栈。停止片段仍然处于活动状态（系统会保留所有状态和成员信息）。不过，它对用户不再可见，如果 Activity 被终止，它也会被终止。

同样与 Activity 一样，假使 Activity 的进程被终止，而您需要在重建 Activity 时恢复片段状态，您也可以使用 [Bundle](#) 保留片段的状态。您可以在片段的 `onSaveInstanceState()` 回调期间保存状态，并可在 `onCreate()`、`onCreateView()` 或 `onActivityCreated()` 期间恢复状态。如需了解有关保存状态的详细信息，请参阅 [Activity](#) 文档。

Activity 生命周期与片段生命周期之间的最显著差异在于它们在其各自返回栈中的存储方式。默认情况下，Activity 停止时会被放入由系统管理的 Activity 返回栈（以便用户通过返回按钮回退到 Activity，[任务和返回栈](#)对此做了阐述）。不过，仅当您在移除片段的事务执行期间通过调用 `addToBackStack()` 显式请求保存实例时，系统才会将片段放入由宿主 Activity 管理的返回栈。

在其他方面，管理片段生命周期与管理 Activity 生命周期非常相似。因此，[管理 Activity 生命周期](#)的做法同样适用于片段。但您还需要了解 Activity 的生命周期对片段生命周期的影响。

注意：如需 `Fragment` 内的某个 `Context` 对象，可以调用 `getActivity()`。但要注意，请仅在片段附加到 Activity 时调用

`getActivity()`。如果片段尚未附加，或在其生命周期结束期间分离，则 `getActivity()` 将返回 null。

与 Activity 生命周期协调一致

片段所在的 Activity 的生命周期会直接影响片段的生命周期，其表现为，Activity 的每次生命周期回调都会引发每个片段的类似回调。例如，当 Activity 收到 `onPause()` 时，Activity 中的每个片段也会收到 `onPause()`。

不过，片段还有几个额外的生命周期回调，用于处理与 Activity 的唯一交互，以执行构建和销毁片段 UI 等操作。这些额外的回调方法是：

`onAttach()`

在片段已与 Activity 关联时调用（`Activity` 传递到此方法内）。

`onCreateView()`

调用它可创建与片段关联的视图层次结构。

`onActivityCreated()`

在 Activity 的 `onCreate()` 方法已返回时调用。

`onDestroyView()`

在移除与片段关联的视图层次结构时调用。

`onDetach()`

在取消片段与 Activity 的关联时调用。

图 3 图示说明了受其宿主 Activity 影响的片段生命周期流。在该图中，您可以看到 Activity 的每个连续状态如何决定片段可以收到的回调方法。例如，当 Activity 收到其 `onCreate()` 回调时，Activity 中的片段只会收到 `onActivityCreated()` 回调。

一旦 Activity 达到恢复状态，您就可以随意向 Activity 添加片段和移除其中的片段。因此，只有当 Activity 处于恢复状态时，片段的生命周期才能独立变化。

不过，当 Activity 离开恢复状态时，片段会在 Activity 的推动下再次经历其生命周期。

示例

为了将本文阐述的所有内容融会贯通，以下提供了一个示例，其中的 Activity 使用两个片段来创建一个双窗格布局。下面的 Activity 包括两个片段：一个用于显示莎士比亚戏剧标题列表，另一个用于从列表中选定戏剧时显示其摘要。此外，它还展示了如何根据屏幕配置提供不同的片段配置。

注：`FragmentLayout.java` 中提供了此 Activity 的完整源代码。

主 Activity 会在 `onCreate()` 期间以常规方式应用布局：

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

应用的布局为 `fragment_layout.xml`：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <fragment class="com.example.android.apis.app.FragmentLayout$TitlesFragment"
        android:id="@+id/titles" android:layout_weight="1"
        android:layout_width="0px" android:layout_height="match_parent" />

    <FrameLayout android:id="@+id/details" android:layout_weight="1"
        android:layout_width="0px" android:layout_height="match_parent"
        android:background="?android:attr/detailsElementBackground" />

</LinearLayout>

```

通过使用此布局，系统可在 Activity 加载布局时立即实例化 `TitlesFragment`（列出戏剧标题），而 `FrameLayout`（用于显示戏剧摘要的片段所在位置）则会占用屏幕右侧的空间，但最初处于空白状态。正如您将在下文所见的那样，用户从列表中选择某个项目后，系统才会将片段放入 `FrameLayout`。

不过，并非所有屏幕配置都具有足够的宽度，可以并排显示戏剧列表和摘要。因此，以上布局仅用于横向屏幕配置（布局保存在 `res/layout-land/fragment_layout.xml`）。

因此，当屏幕纵向显示时，系统会应用以下布局（保存在 `res/layout/fragment_layout.xml`）：

```

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <fragment class="com.example.android.apis.app.FragmentLayout$TitlesFragment"
        android:id="@+id/titles"
        android:layout_width="match_parent" android:layout_height="match_parent" />
</FrameLayout>

```

此布局仅包括 `TitlesFragment`。这意味着，当设备纵向显示时，只有戏剧标题列表可见。因此，当用户在此配置中点击某个列表项时，应用会启动一个新 Activity 来显示摘要，而不是加载另一个片段。

接下来，您可以看到如何在片段类中实现此目的。第一个片段是 `TitlesFragment`，它显示莎士比亚戏剧标题列表。该片段扩展了 `ListFragment`，并依靠它来处理大多数列表视图工作。

当您检查此代码时，请注意，用户点击列表项时可能会出现两种行为：系统可能会创建并显示一个新片段，从而在同一 Activity 中显示详细信息（将片段添加到 `FrameLayout`），也可能会启动一个新 Activity（在该 Activity 中可显示片段），具体取决于这两个布局中哪一个处于活动状态。

```

public static class TitlesFragment extends ListFragment {
    boolean mDualPane;
    int mCurCheckPosition = 0;

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Populate list with our static array of titles.
        setListAdapter(new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_activated_1, Shakespeare.TITLES));

        // Check to see if we have a frame in which to embed the details
        // fragment directly in the containing UI.
        View detailsFrame = getActivity().findViewById(R.id.details);
        mDualPane = detailsFrame != null && detailsFrame.getVisibility() == View.VISIBLE;

        if (savedInstanceState != null) {
            // Restore last state for checked position.
            mCurCheckPosition = savedInstanceState.getInt("curChoice", 0);
        }

        if (mDualPane) {
            // In dual-pane mode, the list view highlights the selected item.
            getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
            // Make sure our UI is in the correct state.
            showDetails(mCurCheckPosition);
        }
    }
}

```

```

    }

}

@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("curChoice", mCurCheckPosition);
}

@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    showDetails(position);
}

/**
 * Helper function to show the details of a selected item, either by
 * displaying a fragment in-place in the current UI, or starting a
 * whole new activity in which it is displayed.
 */
void showDetails(int index) {
    mCurCheckPosition = index;

    if (mDualPane) {
        // We can display everything in-place with fragments, so update
        // the list to highlight the selected item and show the data.
        getListView().setItemChecked(index, true);

        // Check what fragment is currently shown, replace if needed.
        DetailsFragment details = (DetailsFragment)
            getFragmentManager().findFragmentById(R.id.details);
        if (details == null || details.getShownIndex() != index) {
            // Make new fragment to show this selection.
            details = DetailsFragment.newInstance(index);

            // Execute a transaction, replacing any existing fragment
            // with this one inside the frame.
            FragmentTransaction ft = getFragmentManager().beginTransaction();
            if (index == 0) {
                ft.replace(R.id.details, details);
            } else {
                ft.replace(R.id.a_item, details);
            }
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        }
    } else {
        // Otherwise we need to launch a new activity to display
        // the dialog fragment with selected text.
        Intent intent = new Intent();
        intent.setClass(getActivity(), DetailsActivity.class);
        intent.putExtra("index", index);
        startActivity(intent);
    }
}
}

```

第二个片段 `DetailsFragment` 显示从 `TitlesFragment` 的列表中选择的项目的戏剧摘要：

```
public static class DetailsFragment extends Fragment {
    /**
     * Create a new instance of DetailsFragment, initialized to
     * show the text at 'index'.
     */
    public static DetailsFragment newInstance(int index) {
        DetailsFragment f = new DetailsFragment();

        // Supply index input as an argument.
        Bundle args = new Bundle();
        args.putInt("index", index);
        f.setArguments(args);

        return f;
    }

    public int getShownIndex() {
        return getArguments().getInt("index", 0);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        if (container == null) {
            // We have different layouts, and in one of them this
            // fragment's containing frame doesn't exist. The fragment
            // may still be created from its saved state, but there is
            // no reason to try to create its view hierarchy because it
            // won't be displayed. Note this is not needed -- we could
            // just run the code below, where we would create and return
            // the view hierarchy; it would just never be used.
            return null;
        }

        ScrollView scroller = new ScrollView(getActivity());
        TextView text = new TextView(getActivity());
        int padding = (int)TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
                                                     4, getActivity().getResources().getDisplayMetrics());
        text.setPadding(padding, padding, padding, padding);
        scroller.addView(text);
        text.setText(Shakespeare.DIALOGUE[getShownIndex()]);
        return scroller;
    }
}
```

从 `TitlesFragment` 类中重新调用，如果用户点击某个列表项，且当前布局“根本不”包括 `R.id.details` 视图（即 `DetailsFragment` 所属视图），则应用会启动 `DetailsActivity` Activity 以显示该项目的内容。

以下是 `DetailsActivity`，它简单地嵌入了 `DetailsFragment`，以在屏幕为纵向时显示所选的戏剧摘要：

```
public static class DetailsActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        if (getResources().getConfiguration().orientation  
            == Configuration.ORIENTATION_LANDSCAPE) {  
            // If the screen is now in landscape mode, we can show the  
            // dialog in-line with the list so we don't need this activity.  
            finish();  
            return;  
        }  
  
        if (savedInstanceState == null) {  
            // During initial setup, plug in the details fragment.  
            DetailsFragment details = new DetailsFragment();  
            details.setArguments(getIntent().getExtras());  
            getFragmentManager().beginTransaction().add(android.R.id.content, details).commit();  
        }  
    }  
}
```

请注意，如果配置为横向，则此 Activity 会自行完成，以便主 Activity 可以接管并沿 `TitlesFragment` 显示 `DetailsFragment`。如果用户在纵向显示时启动 `DetailsActivity`，但随后旋转为横向（这会重启当前 Activity），就可能出现这种情况。

如需查看使用片段的更多示例（以及本示例的完整源文件），请参阅 [ApiDemos](#)（可从[示例 SDK 组件](#)下载）中提供的 API Demos 示例应用。

加载器

本文内容

- › [Loader API 摘要](#)
- › [在应用中使用加载器](#)
 - › [启动加载器](#)
 - › [重启加载器](#)
 - › [使用 LoaderManager 回调](#)
- › [示例](#)
 - › [更多示例](#)

关键类

- › [LoaderManager](#)
- › [Loader](#)

相关示例

- › [LoaderCursor](#)
- › [LoaderThrottle](#)

Android 3.0 中引入了加载器，支持轻松在 Activity 或片段中异步加载数据。加载器具有以下特征：

- 可用于每个 [Activity](#) 和 [Fragment](#)。
- 支持异步加载数据。
- 监控其数据源并在内容变化时传递新结果。
- 在某一配置更改后重建加载器时，会自动重新连接上一个加载器的游标。因此，它们无需重新查询其数据。

Loader API 摘要

在应用中使用加载器时，可能会涉及到多个类和接口。下表汇总了这些类和接口：

类/接口	说明
LoaderManager	一种与 Activity 或 Fragment 相关联的抽象类，用于管理一个或多个 Loader 实例。这有助于应用管理与 Activity 或 Fragment 生命周期相关联的、运行时间较长的操作。它最常见的用法是与 CursorLoader 一起使用，但应用可自由写入其自己的加载器，用于加载其他类型的数据。 每个 Activity 或片段中只有一个 LoaderManager 。但一个 LoaderManager 可以有多个加载器。
LoaderManager.LoaderCallbacks	一种回调接口，用于客户端与 LoaderManager 进行交互。例如，您可使用 onCreateLoader() 回调方法创建新的加载器。
Loader	一种执行异步数据加载的抽象类。这是加载器的基类。您通常会使用 CursorLoader ，但您也可以实现自己的子类。加载器处于活动状态时，应监控其数据源并在内容变化时传递新结果。
AsyncTaskLoader	提供 AsyncTask 来执行工作的抽象加载器。
CursorLoader	AsyncTaskLoader 的子类，它将查询 ContentResolver 并返回一个 Cursor 。此类采用标准方

式为查询游标实现 [Loader](#) 协议。它是以 [AsyncTaskLoader](#) 为基础而构建，在后台线程中执行游标查询，以免阻塞应用的 UI。使用此加载器是从 [ContentProvider](#) 异步加载数据的最佳方式，而不用通过片段或 [Activity](#) 的 API 来执行托管查询。

上表中的类和接口是您在应用中用于实现加载器的基本组件。并非您创建的每个加载器都要用到上述所有类和接口。但是，为了初始化加载器以及实现一个 [Loader](#) 类（如 [CursorLoader](#)），您始终需要要引用 [LoaderManager](#)。下文将为您展示如何在应用中使用这些类和接口。

在应用中使用加载器

此部分描述如何在 Android 应用中使用加载器。使用加载器的应用通常包括：

- [Activity](#) 或 [Fragment](#)。
- [LoaderManager](#) 的实例。
- 一个 [CursorLoader](#)，用于加载由 [ContentProvider](#) 支持的数据。您也可以实现自己的 [Loader](#) 或 [AsyncTaskLoader](#) 子类，从其他源中加载数据。
- 一个 [LoaderManager.LoaderCallbacks](#) 实现。您可以使用它来创建新加载器，并管理对现有加载器的引用。
- 一种显示加载器数据的方法，如 [SimpleCursorAdapter](#)。
- 使用 [CursorLoader](#) 时的数据源，如 [ContentProvider](#)。

启动加载器

[LoaderManager](#) 可在 [Activity](#) 或 [Fragment](#) 内管理一个或多个 [Loader](#) 实例。每个 [Activity](#) 或片段中只有一个 [LoaderManager](#)。

通常，您会在 [Activity](#) 的 [onCreate\(\)](#) 方法或片段的 [onActivityCreated\(\)](#) 方法内初始化 [Loader](#)。您执行操作如下：

```
// Prepare the loader. Either re-connect with an existing one,  
// or start a new one.  
getLoaderManager().initLoader(0, null, this);
```

[initLoader\(\)](#) 方法采用以下参数：

- 用于标识加载器的唯一 ID。在此示例中，ID 为 0。
- 在构建时提供给加载器的可选参数（在此示例中为 `null`）。
- [LoaderManager.LoaderCallbacks](#) 实现，[LoaderManager](#) 将调用此实现来报告加载器事件。在此示例中，本地类实现 [LoaderManager.LoaderCallbacks](#) 接口，因此它会传递对自身的引用 `this`。

[initLoader\(\)](#) 调用确保加载器已初始化且处于活动状态。这可能会出现两种结果：

- 如果 ID 指定的加载器已存在，则将重复使用上次创建的加载器。
- 如果 ID 指定的加载器不存在，则 [initLoader\(\)](#) 将触发 [LoaderManager.LoaderCallbacks](#) 方法 [onCreateLoader\(\)](#)。在此方法中，您可以实现代码以实例化并返回新加载器。有关详细介绍，请参阅 [onCreateLoader](#) 部分。

无论何种情况，给定的 [LoaderManager.LoaderCallbacks](#) 实现均与加载器相关联，且将在加载器状态变化时调用。如果在调用时，调用程序处于启动状态，且请求的加载器已存在并生成了数据，则系统将立即调用 [onLoadFinished\(\)](#)（在 [initLoader\(\)](#) 期间），因此您必须为此做好准备。有关此回调的详细介绍，请参阅 [onLoadFinished](#)。

请注意，[initLoader\(\)](#) 方法将返回已创建的 [Loader](#)，但您不必捕获其引用。[LoaderManager](#) 将自动管理加载器的生命周期。[LoaderManager](#) 将根据需要启动和停止加载，并维护加载器的状态及其相关内容。这意味着您很少直接与加载器进行交互（有关使用加载器方法调整加载器行为的示例，请参阅 [LoaderThrottle](#) 示例）。当特定事件发生时，您通常会使用 [LoaderManager.LoaderCallbacks](#) 方法干预加载进程。有关此主题的详细介绍，请参阅[使用 LoaderManager 回调](#)。

重启加载器

当您使用 [initLoader\(\)](#) 时（如上所述），它将使用含有指定 ID 的现有加载器（如有）。如果没有，则它会创建一个。但有时，您想舍弃这些旧数据并重新开始。

要舍弃旧数据，请使用 `restartLoader()`。例如，当用户的查询更改时，此 `SearchView.OnQueryTextListener` 实现将重启加载器。加载器需要重启，以便它能够使用修订后的搜索过滤器执行新查询：

```
public boolean onQueryTextChanged(String newText) {
    // Called when the action bar search text has changed. Update
    // the search filter, and restart the loader to do a new query
    // with this filter.
    mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
    getLoaderManager().restartLoader(0, null, this);
    return true;
}
```

使用 LoaderManager 回调

`LoaderManager.LoaderCallbacks` 是一个支持客户端与 `LoaderManager` 交互的回调接口。

加载器（特别是 `CursorLoader`）在停止运行后，仍需保留其数据。这样，应用即可保留 Activity 或片段的 `onStop()` 和 `onStart()` 方法中的数据。当用户返回应用时，无需等待它重新加载这些数据。您可使用 `LoaderManager.LoaderCallbacks` 方法了解何时创建新加载器，并告知应用何时停止使用加载器的数据。

`LoaderManager.LoaderCallbacks` 包括以下方法：

- `onCreateLoader()`：针对指定的 ID 进行实例化并返回新的 `Loader`
- `onLoadFinished()`：将在先前创建的加载器完成加载时调用
- `onLoaderReset()`：将在先前创建的加载器重置且其数据因此不可用时调用

下文更详细地描述了这些方法。

onCreateLoader

当您尝试访问加载器时（例如，通过 `initLoader()`），该方法将检查是否已存在由该 ID 指定的加载器。如果没有，它将触发 `LoaderManager.LoaderCallbacks` 方法 `onCreateLoader()`。在此方法中，您可以创建新加载器。通常，这将是 `CursorLoader`，但您也可以实现自己的 `Loader` 子类。

在此示例中，`onCreateLoader()` 回调方法创建了 `CursorLoader`。您必须使用其构造函数方法来构建 `CursorLoader`。该方法需要对 `ContentProvider` 执行查询时所需的一系列完整信息。具体地说，它需要：

- `uri`：用于检索内容的 URI
- `projection`：要返回的列的列表。传递 `null` 时，将返回所有列，这样会导致效率低下
- `selection`：一种用于声明要返回哪些行的过滤器，采用 SQL WHERE 子句格式（WHERE 本身除外）。传递 `null` 时，将为指定的 URI 返回所有行
- `selectionArgs`：您可以在 `selection` 中包含 ?s，它将按照在 `selection` 中显示的顺序替换为 `selectionArgs` 中的值。该值将绑定为字符串
- `sortOrder`：行的排序依据，采用 SQL ORDER BY 子句格式（ORDER BY 自身除外）。传递 `null` 时，将使用默认排序顺序（可能并未排序）

例如：

```

// If non-null, this is the current filter the user has provided.
String mCurFilter;
...
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // This is called when a new Loader needs to be created. This
    // sample only has one Loader, so we don't care about the ID.
    // First, pick the base URI to use depending on whether we are
    // currently filtering.
    Uri baseUri;
    if (mCurFilter != null) {
        baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
            Uri.encode(mCurFilter));
    } else {
        baseUri = Contacts.CONTENT_URI;
    }

    // Now create and return a CursorLoader that will take care of
    // creating a Cursor for the data being displayed.
    String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL) AND (
        + Contacts.HAS_PHONE_NUMBER + "=1) AND (
        + Contacts.DISPLAY_NAME + " != '' ))";
    return new CursorLoader(getActivity(), baseUri,
        CONTACTS_SUMMARY_PROJECTION, select, null,
        Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
}

```

onLoadFinished

当先前创建的加载器完成加载时，将调用此方法。该方法必须在为此加载器提供的最后一个数据释放之前调用。此时，您应移除所有使用的旧数据（因为它们很快会被释放），但不要自行释放这些数据，因为这些数据归其加载器所有，其加载器会处理它们。

当加载器发现应用不再使用这些数据时，即会释放它们。例如，如果数据是来自 `CursorLoader` 的一个游标，则您不应手动对其进行调用 `close()`。如果游标放置在 `CursorAdapter` 中，则应使用 `swapCursor()` 方法，使旧 `Cursor` 不会关闭。例如：

```

// This is the Adapter being used to display the list's data.
SimpleCursorAdapter mAdapter;
...

public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Swap the new cursor in. (The framework will take care of closing the
    // old cursor once we return.)
    mAdapter.swapCursor(data);
}

```

onLoaderReset

此方法将在先前创建的加载器重置且其数据因此不可用时调用。通过此回调，您可以了解何时将释放数据，因而能够及时移除其引用。

此实现调用值为 `null` 的 `swapCursor()`：

```

// This is the Adapter being used to display the list's data.
SimpleCursorAdapter mAdapter;
...

public void onLoaderReset(Loader<Cursor> loader) {
    // This is called when the last Cursor provided to onLoadFinished()
    // above is about to be closed. We need to make sure we are no
    // longer using it.
    mAdapter.swapCursor(null);
}

```

示例

以下是一个 `Fragment` 完整实现示例。它展示了一个 `ListView`，其中包含针对联系人内容提供程序的查询结果。它使用 `CursorLoader` 管理提供程序的查询。

应用如需访问用户联系人（如此示例中所示），其清单文件必须包括权限 `READ_CONTACTS`。

```
public static class CursorLoaderListFragment extends ListFragment
    implements OnQueryTextListener, LoaderManager.LoaderCallbacks<Cursor> {

    // This is the Adapter being used to display the list's data.
    SimpleCursorAdapter mAdapter;

    // If non-null, this is the current filter the user has provided.
    String mCurFilter;

    @Override public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Give some text to display if there is no data. In a real
        // application this would come from a resource.
        setEmptyText("No phone numbers");

        // We have a menu item to show in action bar.
        setHasOptionsMenu(true);

        // Create an empty adapter we will use to display the loaded data.
        mAdapter = new SimpleCursorAdapter(getActivity(),
            android.R.layout.simple_list_item_2, null,
            new String[] { Contacts.DISPLAY_NAME, Contacts.CONTACT_STATUS },
            new int[] { android.R.id.text1, android.R.id.text2 }, 0);
        setListAdapter(mAdapter);

        // Prepare the loader. Either re-connect with an existing one,
        // or start a new one.
        getLoaderManager().initLoader(0, null, this);
    }

    @Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        // Place an action bar item for searching.
        MenuItem item = menu.add("Search");
        item.setIcon(android.R.drawable.ic_menu_search);
        item.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        SearchView sv = new SearchView(getActivity());
        sv.setOnQueryTextListener(this);
        item.setActionView(sv);
    }

    public boolean onQueryTextChange(String newText) {
        // Called when the action bar search text has changed. Update
        // the search filter, and restart the loader to do a new query
        // with this filter.
        mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
        getLoaderManager().restartLoader(0, null, this);
        return true;
    }

    @Override public boolean onQueryTextSubmit(String query) {
        // Don't care about this.
        return true;
    }

    @Override public void onListItemClick(ListView l, View v, int position, long id) {
        // Insert desired behavior here.
        Log.i("FragmentComplexList", "Item clicked: " + id);
    }

    // These are the Contacts rows that we will retrieve.
    static final String[] CONTACTS_SUMMARY_PROJECTION = new String[] {
        Contacts._ID,
        Contacts.DISPLAY_NAME,
        Contacts.CONTACT_STATUS,
        Contacts.CONTACT_PRESENCE,
        Contacts.PHOTO_ID,
        Contacts.LOOKUP_KEY,
    };
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
```

```
// This is called when a new Loader needs to be created. This
// sample only has one Loader, so we don't care about the ID.
// First, pick the base URI to use depending on whether we are
// currently filtering.
Uri baseUri;
if (mCurFilter != null) {
    baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
        Uri.encode(mCurFilter));
} else {
    baseUri = Contacts.CONTENT_URI;
}

// Now create and return a CursorLoader that will take care of
// creating a Cursor for the data being displayed.
String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL) AND (
    + Contacts.HAS_PHONE_NUMBER + "=1) AND (
    + Contacts.DISPLAY_NAME + " != '' ))";
return new CursorLoader(getActivity(), baseUri,
    CONTACTS_SUMMARY_PROJECTION, select, null,
    Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
}

public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Swap the new cursor in. (The framework will take care of closing the
    // old cursor once we return.)
    mAdapter.swapCursor(data);
}

public void onLoaderReset(Loader<Cursor> loader) {
    // This is called when the last Cursor provided to onLoadFinished()
    // above is about to be closed. We need to make sure we are no
    // longer using it.
    mAdapter.swapCursor(null);
}
}
```

更多示例

ApiDemos 中还提供了一些不同的示例，阐述如何使用加载器：

- [LoaderCursor](#)：上述代码段的完整版本
- [LoaderThrottle](#)：此示例显示当数据变化时，如何使用限制来减少内容提供程序的查询次数

有关下载和安装 SDK 示例的信息，请参阅[获取示例](#)。

任务和返回栈

本文内容

- › [保存 Activity 状态](#)
- › [管理任务](#)
- › [定义启动模式](#)
- › [处理关联](#)
- › [清理返回栈](#)
- › [启动任务](#)

文章

- › [Android 多任务运行机制](#)

另请参阅

- › [Android 设计：导航](#)
- › [<activity> 清单文件元素](#)
- › [概览屏幕](#)

应用通常包含多个 **Activity**。每个 Activity 均应围绕用户可以执行的特定操作设计，并且能够启动其他 Activity。例如，电子邮件应用可能有一个 Activity 显示新邮件的列表。用户选择某邮件时，会打开一个新 Activity 以查看该邮件。

一个 Activity 甚至可以启动设备上其他应用中存在的 Activity。例如，如果应用想要发送电子邮件，则可将 Intent 定义为执行“发送”操作并加入一些数据，如电子邮件地址和电子邮件。然后，系统将打开其他应用中声明自己处理此类 Intent 的 Activity。在这种情况下，Intent 是要发送电子邮件，因此将启动电子邮件应用的“撰写”Activity（如果多个 Activity 支持相同 Intent，则系统会让用户选择要使用的 Activity）。发送电子邮件时，Activity 将恢复，看起来好像电子邮件 Activity 是您的应用的一部分。即使这两个 Activity 可能来自不同的应用，但是 Android 仍会将 Activity 保留在相同的任务中，以维护这种无缝的用户体验。

任务是指在执行特定作业时与用户交互的一系列 Activity。这些 Activity 按照各自的打开顺序排列在堆栈（即返回栈）中。

设备主屏幕是大多数任务的起点。当用户触摸应用启动器中的图标（或主屏幕上的快捷方式）时，该应用的任务将出现在前台。如果应用不存在于任务（应用最近未曾使用），则会创建一个新任务，并且该应用的“主”Activity 将作为堆栈中的根 Activity 打开。

当前 Activity 启动另一个 Activity 时，该新 Activity 会被推送到堆栈顶部，成为焦点所在。前一个 Activity 仍保留在堆栈中，但是处于停止状态。Activity 停止时，系统会保持其用户界面的当前状态。用户按“返回”按钮时，当前 Activity 会从堆栈顶部弹出（Activity 被销毁），而前一个 Activity 恢复执行（恢复其 UI 的前一状态）。堆栈中的 Activity 永远不会重新排列，仅推入和弹出堆栈：由当前 Activity 启动时推入堆栈；用户使用“返回”按钮退出时弹出堆栈。因此，返回栈以“后进先出”对象结构运行。图 1 通过时间线显示 Activity 之间的进度以及每个时间点的当前返回栈，直观呈现了这种行为。

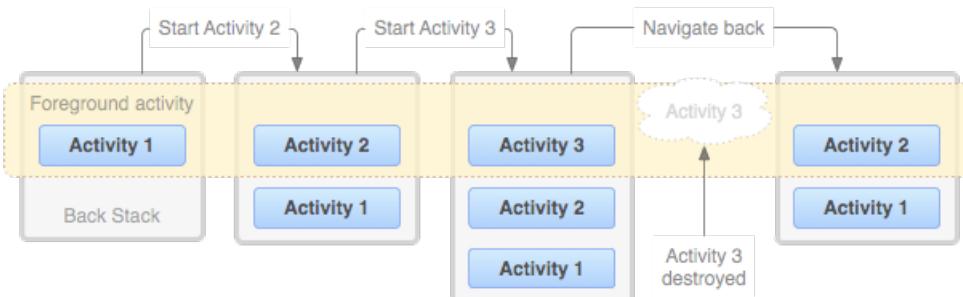


图 1. 显示任务中的每个新 Activity 如何向返回栈添加项目。用户按“返回”按钮时，当前 Activity 随即被销毁，而前一个 Activity 恢复执行。

如果用户继续按“返回”，堆栈中的相应 Activity 就会弹出，以显示前一个 Activity，直到用户返回主屏幕为止（或者，返回任务开始时正在运行

的任意 Activity）。当所有 Activity 均从堆栈中移除后，任务即不复存在。



图 2. 两个任务：任务 B 在前台接收用户交互，而任务 A 则在后台等待恢复。

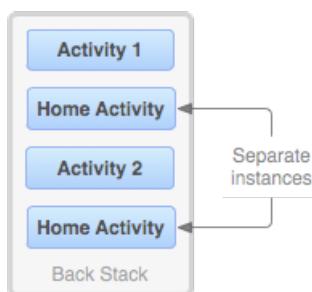


图 3. 一个 Activity 将多次实例化。

任务是一个有机整体，当用户开始新任务或通过“主页”按钮转到主屏幕时，可以移动到“后台”。尽管在后台时，该任务中的所有 Activity 全部停止，但是任务的返回栈仍旧不变，也就是说，当另一个任务发生时，该任务仅仅失去焦点而已，如图 2 中所示。然后，任务可以返回到“前台”，用户就能够回到离开时的状态。例如，假设当前任务（任务 A）的堆栈中有三个 Activity，即当前 Activity 下方还有两个 Activity。用户先按“主页”按钮，然后从应用启动器启动新应用。显示主屏幕时，任务 A 进入后台。新应用启动时，系统会使用自己的 Activity 堆栈为该应用启动一个任务（任务 B）。与该应用交互之后，用户再次返回主屏幕并选择最初启动任务 A 的应用。现在，任务 A 出现在前台，其堆栈中的所有三个 Activity 保持不变，而位于堆栈顶部的 Activity 则会恢复执行。此时，用户还可以通过转到主屏幕并选择启动该任务的应用图标（或者，通过从概览屏幕选择该应用的任务）切换回任务 B。这是 Android 系统中的一个多任务示例。

注：后台可以同时运行多个任务。但是，如果用户同时运行多个后台任务，则系统可能会开始销毁后台 Activity，以回收内存资源，从而导致 Activity 状态丢失。请参阅下面有关 [Activity 状态](#) 的部分。

由于返回栈中的 Activity 永远不会重新排列，因此如果应用允许用户从多个 Activity 中启动特定 Activity，则会创建该 Activity 的新实例并推入堆栈中（而不是将 Activity 的任一先前实例置于顶部）。因此，应用中的一个 Activity 可能会多次实例化（即使 Activity 来自不同的任务），如图 3 所示。因此，如果用户使用“返回”按钮向后导航，则会按 Activity 每个实例的打开顺序显示这些实例（每个实例的 UI 状态各不相同）。但是，如果您不希望 Activity 多次实例化，则可修改此行为。具体操作方法将在后面的[管理任务](#)部分中讨论。

Activity 和任务的默认行为总结如下：

- 当 Activity A 启动 Activity B 时，Activity A 将会停止，但系统会保留其状态（例如，滚动位置和已输入表单中的文本）。如果用户在处于 Activity B 时按“返回”按钮，则 Activity A 将恢复其状态，继续执行。
- 用户通过按“主页”按钮离开任务时，当前 Activity 将停止且其任务会进入后台。系统将保留任务中每个 Activity 的状态。如果用户稍后通过选择开始任务的启动器图标来恢复任务，则任务将出现在前台并恢复执行堆栈顶部的 Activity。
- 如果用户按“返回”按钮，则当前 Activity 会从堆栈弹出并被销毁。堆栈中的前一个 Activity 恢复执行。销毁 Activity 时，系统不会保留该 Activity 的状态。
- 即使来自其他任务，Activity 也可以多次实例化。

导航设计

如需了解有关 Android 应用导航工作方式的详细信息，请阅读 Android 设计的 [导航](#) 指南。

保存 Activity 状态

正如上文所述，当 Activity 停止时，系统的默认行为会保留其状态。这样一来，当用户导航回到上一个 Activity 时，其用户界面与用户离开时一样。但是，在 Activity 被销毁且必须重建时，您可以而且应当主动使用回调方法保留 Activity 的状态。

系统停止您的一个 Activity 时（例如，新 Activity 启动或任务转到前台），如果系统需要回收系统内存资源，则可能会完全销毁该 Activity。发生这种情况时，有关该 Activity 状态的信息将会丢失。如果发生这种情况，系统仍会知道该 Activity 存在于返回栈中，但是当该 Activity 被置于堆栈顶部时，系统一定会重建 Activity（而不是恢复 Activity）。为了避免用户的工作丢失，您应主动通过在 Activity 中实现 `onSaveInstanceState()` 回调方法来保留工作。

如需了解有关如何保存 Activity 状态的详细信息，请参阅 [Activity](#) 文档。

管理任务

Android 管理任务和返回栈的方式（如上所述，即：将所有连续启动的 Activity 放入同一任务和“后进先出”堆栈中）非常适用于大多数应用，而您不必担心 Activity 如何与任务关联或者如何存在于返回栈中。但是，您可能会决定要中断正常行为。也许您希望应用中的 Activity 在启动时开始新任务（而不是放置在当前任务中）；或者，当启动 Activity 时，您希望将其现有实例上移一层（而不是在返回栈的顶部创建新实例）；或者，您希望在用户离开任务时，清除返回栈中除根 Activity 以外的所有其他 Activity。

通过使用 `<activity>` 清单文件元素中的属性和传递给 `startActivity()` 的 Intent 中的标志，您可以执行所有这些操作以及其他操作。

在这一方面，您可以使用的主要 `<activity>` 属性包括：

`taskAffinity`

`launchMode`

`allowTaskReparenting`

`clearTaskOnLaunch`

`alwaysRetainTaskState`

`finishOnTaskLaunch`

您可以使用的主要 Intent 标志包括：

`FLAG_ACTIVITY_NEW_TASK`

`FLAG_ACTIVITY_CLEAR_TOP`

`FLAG_ACTIVITY_SINGLE_TOP`

在下文中，您将了解如何使用这些清单文件属性和 Intent 标志定义 Activity 与任务的关联方式，以及 Activity 在返回栈中的行为方式。

此外，我们还单独介绍了有关如何在概览屏幕中显示和管理任务与 Activity 的注意事项。如需了解详细信息，请参阅 [概览屏幕](#)。通常，您应该允许系统定义任务和 Activity 在概览屏幕中的显示方法，并且无需修改此行为。

注意：大多数应用都不得中断 Activity 和任务的默认行为：如果确定您的 Activity 必须修改默认行为，当使用“返回”按钮从其他 Activity 和任务导航回到该 Activity 时，请务必要谨慎并确保在启动期间测试该 Activity 的可用性。请确保测试导航行为是否有可能与用户的预期行为冲突。

定义启动模式

启动模式允许您定义 Activity 的新实例如何与当前任务关联。您可以通过两种方法定义不同的启动模式：

使用清单文件

在清单文件中声明 Activity 时，您可以指定 Activity 在启动时应该如何与任务关联。

使用 Intent 标志

调用 `startActivity()` 时，可以在 Intent 中加入一个标志，用于声明新 Activity 如何（或是否）与当前任务关联。

因此，如果 Activity A 启动 Activity B，则 Activity B 可以在其清单文件中定义它应该如何与当前任务关联（如果可能），并且 Activity A 还可以请求 Activity B 应该如何与当前任务关联。如果这两个 Activity 均定义 Activity B 应该如何与任务关联，则 Activity A 的请求（如 Intent 中所定义）优先级要高于 Activity B 的请求（如其清单文件中所定义）。

注：某些适用于清单文件的启动模式不可用作 Intent 标志，同样，某些可用作 Intent 标志的启动模式无法在清单文件中定义。

使用清单文件

在清单文件中声明 Activity 时，您可以使用 `<activity>` 元素的 `launchMode` 属性指定 Activity 应该如何与任务关联。

`launchMode` 属性指定有关应如何将 Activity 启动到任务中的指令。您可以分配给 `launchMode` 属性的启动模式共有四种：

"standard"（默认模式）

默认。系统在启动 Activity 的任务中创建 Activity 的新实例并向其传送 Intent。Activity 可以多次实例化，而每个实例均可属于不同的任务，并且一个任务可以拥有多个实例。

"singleTop"

如果当前任务的顶部已存在 Activity 的一个实例，则系统会通过调用该实例的 `onNewIntent()` 方法向其传送 Intent，而不是创建 Activity 的新实例。Activity 可以多次实例化，而每个实例均可属于不同的任务，并且一个任务可以拥有多个实例（但前提是位于返回栈顶部的 Activity 并不是 Activity 的现有实例）。

例如，假设任务的返回栈包含根 Activity A 以及 Activity B、C 和位于顶部的 D（堆栈是 A-B-C-D；D 位于顶部）。收到针对 D 类 Activity 的 Intent。如果 D 具有默认的 "standard" 启动模式，则会启动该类的新实例，且堆栈会变成 A-B-C-D-D。但是，如果 D 的启动模式是 "singleTop"，则 D 的现有实例会通过 `onNewIntent()` 接收 Intent，因为它位于堆栈的顶部；而堆栈仍为 A-B-C-D。但是，如果收到针对 B 类 Activity 的 Intent，则会向堆栈添加 B 的新实例，即便其启动模式为 "singleTop" 也是如此。

注：为某个 Activity 创建新实例时，用户可以按“返回”按钮返回到前一个 Activity。但是，当 Activity 的现有实例处理新 Intent 时，则在新 Intent 到达 `onNewIntent()` 之前，用户无法按“返回”按钮返回到 Activity 的状态。

"singleTask"

系统创建新任务并实例化位于新任务底部的 Activity。但是，如果该 Activity 的一个实例已存在于一个单独的任务中，则系统会通过调用现有实例的 `onNewIntent()` 方法向其传送 Intent，而不是创建新实例。一次只能存在 Activity 的一个实例。

注：尽管 Activity 在新任务中启动，但是用户按“返回”按钮仍会返回到前一个 Activity。

"singleInstance".

与 "singleTask" 相同，只是系统不会将任何其他 Activity 启动到包含实例的任务中。该 Activity 始终是其任务唯一仅有的成员；由此 Activity 启动的任何 Activity 均在单独的任务中打开。

我们再来看另一示例，Android 浏览器应用声明网络浏览器 Activity 应始终在其自己的任务中打开（通过在 `<activity>` 元素中指定 `singleTask` 启动模式）。这意味着，如果您的应用发出打开 Android 浏览器的 Intent，则其 Activity 与您的应用位于不同的任务中。相反，系统会为浏览器启动新任务，或者如果浏览器已有任务正在后台运行，则会将该任务上移一层以处理新 Intent。

无论 Activity 是在新任务中启动，还是在与启动 Activity 相同的任务中启动，用户按“返回”按钮始终会转到前一个 Activity。但是，如果启动指定 `singleTask` 启动模式的 Activity，则当某后台任务中存在该 Activity 的实例时，整个任务都会转移到前台。此时，返回栈包括上移到堆栈顶部的任务中的所有 Activity。图 4 显示了这种情况。

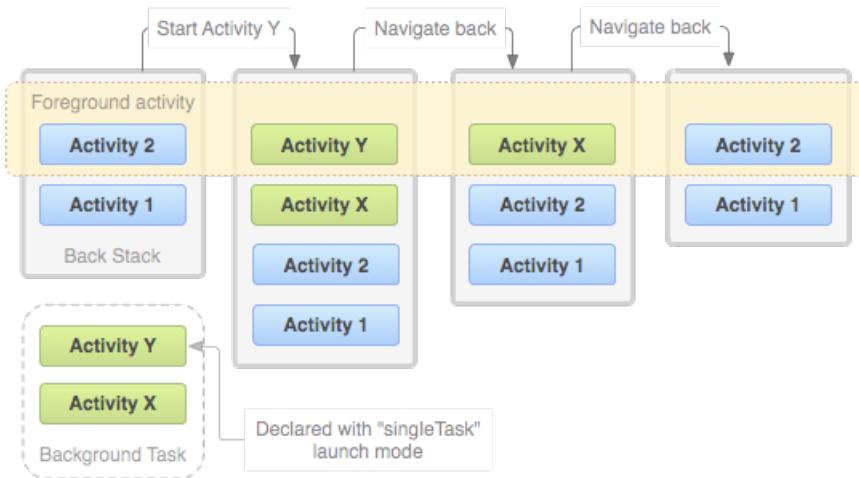


图 4. 显示如何将启动模式为“singleTask”的 Activity 添加到返回栈。如果 Activity 已经是某个拥有自己的返回栈的后台任务的一部分，则整个返回栈也会上移到当前任务的顶部。

如需了解有关在清单文件中使用启动模式的详细信息，请参阅 [<activity>](#) 元素文档，其中更详细地讨论了 `launchMode` 属性和可接受的值。

注：使用 `launchMode` 属性为 Activity 指定的行为可由 Intent 附带的 Activity 启动标志替代，下文将对此进行讨论。

使用 Intent 标志

启动 Activity 时，您可以通过在传递给 `startActivity()` 的 Intent 中加入相应的标志，修改 Activity 与其任务的默认关联方式。可用于修改默认行为的标志包括：

`FLAG_ACTIVITY_NEW_TASK`

在新任务中启动 Activity。如果已为正在启动的 Activity 运行任务，则该任务会转到前台并恢复其最后状态，同时 Activity 会在 `onNewIntent()` 中收到新 Intent。

正如前文所述，这会产生与 “`singleTask`”`launchMode` 值相同的行为。

`FLAG_ACTIVITY_SINGLE_TOP`

如果正在启动的 Activity 是当前 Activity（位于返回栈的顶部），则现有实例会接收对 `onNewIntent()` 的调用，而不是创建 Activity 的新实例。

正如前文所述，这会产生与 “`singleTop`”`launchMode` 值相同的行为。

`FLAG_ACTIVITY_CLEAR_TOP`

如果正在启动的 Activity 已在当前任务中运行，则会销毁当前任务顶部的所有 Activity，并通过 `onNewIntent()` 将此 Intent 传递给 Activity 已恢复的实例（现在位于顶部），而不是启动该 Activity 的新实例。

产生这种行为的 `launchMode` 属性没有值。

`FLAG_ACTIVITY_CLEAR_TOP` 通常与 `FLAG_ACTIVITY_NEW_TASK` 结合使用。一起使用时，通过这些标志，可以找到其他任务中的现有 Activity，并将其放入可从中响应 Intent 的位置。

注：如果指定 Activity 的启动模式为 “`standard`”，则该 Activity 也会从堆栈中移除，并在其位置启动一个新实例，以便处理传入的 Intent。这是因为当启动模式为 “`standard`” 时，将始终为新 Intent 创建新实例。

处理关联

“关联”指示 Activity 优先属于哪个任务。默认情况下，同一应用中的所有 Activity 彼此关联。因此，默认情况下，同一应用中的所有 Activity 优先位于相同任务中。不过，您可以修改 Activity 的默认关联。在不同应用中定义的 Activity 可以共享关联，或者可在同一应用中定义的 Activity 分配不同的任务关联。

可以使用 [<activity>](#) 元素的 `taskAffinity` 属性修改任何给定 Activity 的关联。

`taskAffinity` 属性取字符串值，该值必须不同于在 [<manifest>](#) 元素中声明的默认软件包名称，因为系统使用该名称标识应用的默认任务关

联。

在两种情况下，关联会起作用：

- 启动 Activity 的 Intent 包含 `FLAG_ACTIVITY_NEW_TASK` 标志。

默认情况下，新 Activity 会启动到调用 `startActivity()` 的 Activity 任务中。它将推入与调用方相同的返回栈。但是，如果传递给 `startActivity()` 的 Intent 包含 `FLAG_ACTIVITY_NEW_TASK` 标志，则系统会寻找其他任务来储存新 Activity。这通常是新任务，但未做强制要求。如果现有任务与新 Activity 具有相同关联，则会将 Activity 启动到该任务中。否则，将开始新任务。

如果此标志导致 Activity 开始新任务，且用户按“主页”按钮离开，则必须为用户提供导航回任务的方式。有些实体（如通知管理器）始终在外部任务中启动 Activity，而从不作为其自身的一部分启动 Activity，因此它们始终将 `FLAG_ACTIVITY_NEW_TASK` 放入传递给 `startActivity()` 的 Intent 中。请注意，如果 Activity 能够由可以使用此标志的外部实体调用，则用户可以通过独立方式返回到启动的任务，例如，使用启动器图标（任务的根 Activity 具有 `CATEGORY_LAUNCHER` Intent 过滤器；请参阅下面的[启动任务](#)部分）。

- Activity 将其 `allowTaskReparenting` 属性设置为 `"true"`。

在这种情况下，Activity 可以从其启动的任务移动到与其具有关联的任务（如果该任务出现在前台）。

例如，假设将报告所选城市天气状况的 Activity 定义为旅行应用的一部分。它与同一应用中的其他 Activity 具有相同的关联（默认应用关联），并允许利用此属性重定父级。当您的一个 Activity 启动天气预报 Activity 时，它最初所属的任务与您的 Activity 相同。但是，当旅行应用的任务出现在前台时，系统会将天气预报 Activity 重新分配给该任务并显示在其中。

提示：如果从用户的角度来看，一个 `.apk` 文件包含多个“应用”，则您可能需要使用 `taskAffinity` 属性将不同关联分配给与每个“应用”相关的 Activity。

清理返回栈

如果用户长时间离开任务，则系统会清除所有 Activity 的任务，根 Activity 除外。当用户再次返回到任务时，仅恢复根 Activity。系统这样做的原因是，经过很长一段时间后，用户可能已经放弃之前执行的操作，返回到任务是要开始执行新的操作。

您可以使用下列几个 Activity 属性修改此行为：

`alwaysRetainTaskState`

如果在任务的根 Activity 中将此属性设置为 `"true"`，则不会发生刚才所述的默认行为。即使在很长一段时间后，任务仍将所有 Activity 保留在其堆栈中。

`clearTaskOnLaunch`

如果在任务的根 Activity 中将此属性设置为 `"true"`，则每当用户离开任务然后返回时，系统都会将堆栈清除到只剩下根 Activity。换而言之，它与 `alwaysRetainTaskState` 正好相反。即使只离开任务片刻时间，用户也始终会返回到任务的初始状态。

`finishOnTaskLaunch`

此属性类似于 `clearTaskOnLaunch`，但它对单个 Activity 起作用，而非整个任务。此外，它还可能会导致任何 Activity 停止，包括根 Activity。设置为 `"true"` 时，Activity 仍是任务的一部分，但是仅限于当前会话。如果用户离开然后返回任务，则任务将不复存在。

启动任务

通过为 Activity 提供一个以 `"android.intent.action.MAIN"` 为指定操作、以 `"android.intent.category.LAUNCHER"` 为指定类别的 Intent 过滤器，您可以将 Activity 设置为任务的入口点。例如：

```
<activity ... >
    <intent-filter ... >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    ...
</activity>
```

此类 Intent 过滤器会使 Activity 的图标和标签显示在应用启动器中，让用户能够启动 Activity 并在启动之后随时返回到创建的任务中。

第二个功能非常重要：用户必须能够在离开任务后，再使用此 Activity 启动器返回该任务。因此，只有在 Activity 具有 `ACTION_MAIN` 和 `CATEGORY_LAUNCHER` 过滤器时，才应该使用将 Activity 标记为“始终启动任务”的两种启动模式，即 `"singleTask"` 和 `"singleInstance"`。例如，我们可以想像一下如果缺少过滤器会发生什么情况：Intent 启动一个 `"singleTask"` Activity，从而启动一个新任务，并且用户花了些时间处理该任务。然后，用户按“主页”按钮。任务现已发送到后台，而且不可见。现在，用户无法返回到任务，因为该任务未显示在应用启动器中。

如果您并不想用户能够返回到 Activity，对于这些情况，请将 `<activity>` 元素的 `finishOnTaskLaunch` 设置为 `"true"`（请参阅[清理堆栈](#)）。

有关如何在概览屏幕中显示和管理任务与 Activity 的更多信息，请参阅[概览屏幕](#)。

概览屏幕

本文内容

- › [将任务添加到概览屏幕](#)
- › [使用 Intent 标志添加任务](#)
- › [使用 Activity 属性添加任务](#)
- › [移除任务](#)
- › [使用 AppTask 类移除任务](#)
- › [保留已完成的任务](#)

关键类

- › [ActivityManager.AppTask](#)
- › [Intent](#)

示例代码

- › [以文档为中心的应用](#)

概览屏幕（也称为最新动态屏幕、最近任务列表或最近使用的应用）是一个系统级别 UI，其中列出了最近访问过的 [Activity](#) 和 [任务](#)。用户可以浏览该列表并选择要恢复的任务，也可以通过滑动清除任务将其从列表中移除。对于 Android 5.0 版本（API 级别 21），包含不同文档的同一 Activity 的多个实例可能会以任务的形式显示在概览屏幕上。例如，Google Drive 可能对多个 Google 文档中的每个文档均执行一个任务。每个文档均以任务的形式显示在概览屏幕上。

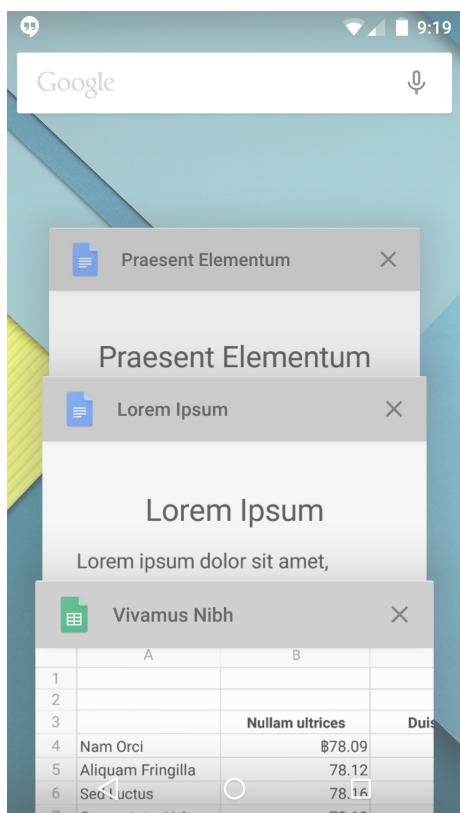


图 1. 显示了三个 Google Drive 文档的概览屏幕，每个文档分别以一个单独的任务表示。

通常，您应该允许系统定义任务和 Activity 在概览屏幕中的显示方法，并且无需修改此行为。不过，应用可以确定 Activity 在概览屏幕中的显示方式和时间。您可以使用 [ActivityManager.AppTask](#) 类来管理任务，使用 [Intent](#) 类的 Activity 标志来指定某 Activity 添加到概览屏幕或从

中移除的时间。此外，您也可以使用 `<activity>` 属性在清单文件中设置该行为。

将任务添加到概览屏幕

通过使用 `Intent` 类的标志添加任务，您可以更好地控制某文档在概览屏幕中打开或重新打开的时间和方式。使用 `<activity>` 属性时，您可以选择始终在新任务中打开文档，或选择对文档重复使用现有任务。

使用 Intent 标志添加任务

为 Activity 创建新文档时，可调用 `ActivityManager.AppTask` 类的 `startActivity()` 方法，以向其传递启动 Activity 的 Intent。要插入逻辑换行符以便系统将 Activity 视为新任务显示在概览屏幕中，可在启动 Activity 的 `Intent` 的 `addFlags()` 方法中传递 `FLAG_ACTIVITY_NEW_DOCUMENT` 标志。

注：`FLAG_ACTIVITY_NEW_DOCUMENT` 标志取代了 `FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET` 标志，后者自 Android 5.0 (API 级别 21) 起已弃用。

如果在创建新文档时设置 `FLAG_ACTIVITY_MULTIPLE_TASK` 标志，则系统始终会以目标 Activity 作为根创建新任务。此设置允许同一文档在多个任务中打开。以下代码演示了主 Activity 如何执行此操作：

`DocumentCentricActivity.java`

```
public void createNewDocument(View view) {
    final Intent newDocumentIntent = newDocumentIntent();
    if (useMultipleTasks) {
        newDocumentIntent.addFlags(Intent.FLAG_ACTIVITY_MULTIPLE_TASK);
    }
    startActivity(newDocumentIntent);
}

private Intent newDocumentIntent() {
    boolean useMultipleTasks = mCheckbox.isChecked();
    final Intent newDocumentIntent = new Intent(this, NewDocumentActivity.class);
    newDocumentIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_DOCUMENT);
    newDocumentIntent.putExtra(KEY_EXTRA_NEW_DOCUMENT_COUNTER, incrementAndGet());
    return newDocumentIntent;
}

private static int incrementAndGet() {
    Log.d(TAG, "incrementAndGet(): " + mDocumentCounter);
    return mDocumentCounter++;
}
```

注：使用 `FLAG_ACTIVITY_NEW_DOCUMENT` 标志启动的 Activity 必须具有在清单文件中设置的 `android:launchMode="standard"` 属性值（默认）。

当主 Activity 启动新 Activity 时，系统会搜遍现有任务，看看是否有任务的 Intent 与 Activity 的 Intent 组件名称和 Intent 数据相匹配。如果未找到任务或者 Intent 包含 `FLAG_ACTIVITY_MULTIPLE_TASK` 标志，则会以该 Activity 作为其根创建新任务。如果找到的话，则会将该任务转到前台并将新 Intent 传递给 `onNewIntent()`。新 Activity 将获得 Intent 并在概览屏幕中创建新文档，如下例所示：

`NewDocumentActivity.java`

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_new_document);
    mDocumentCount = getIntent()
        .getIntExtra(DocumentCentricActivity.KEY_EXTRA_NEW_DOCUMENT_COUNTER, 0);
    mDocumentCounterTextView = (TextView) findViewById(
        R.id.hello_new_document_text_view);
    setDocumentCounterText(R.string.hello_new_document_counter);
}

@Override
protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    /* If FLAG_ACTIVITY_MULTIPLE_TASK has not been used, this activity
    is reused to create a new document.
    */
    setDocumentCounterText(R.string.reusing_document_counter);
}

```

使用 Activity 属性添加任务

此外，Activity 还可以在其清单文件中指定始终通过使用 `<activity>` 属性 `android:documentLaunchMode` 进入新任务。此属性有四个值，会在用户使用该应用打开文档时产生以下效果：

`"intoExisting"`

该 Activity 会对文档重复使用现有任务。这与不设置 `FLAG_ACTIVITY_MULTIPLE_TASK` 标志、但设置 `FLAG_ACTIVITY_NEW_DOCUMENT` 标志所产生的效果相同，如上文的[使用 Intent 标志添加任务](#)中所述。

`"always"`

该 Activity 为文档创建新任务，即便文档已打开也是如此。使用此值与同时设置 `FLAG_ACTIVITY_NEW_DOCUMENT` 和 `FLAG_ACTIVITY_MULTIPLE_TASK` 标志所产生的效果相同。

`"none"`

该 Activity 不会为文档创建新任务。概览屏幕将按其默认方式对待此 Activity：为应用显示单个任务，该任务将从用户上次调用的任意 Activity 开始继续执行。

`"never"`

该 Activity 不会为文档创建新任务。设置此值会替代 `FLAG_ACTIVITY_NEW_DOCUMENT` 和 `FLAG_ACTIVITY_MULTIPLE_TASK` 标志的行为（如果在 Intent 中设置了其中一个标志），并且概览屏幕将为应用显示单个任务，该任务将从用户上次调用的任意 Activity 开始继续执行。

注：对于除 `none` 和 `never` 以外的值，必须使用 `launchMode="standard"` 定义 Activity。如果未指定此属性，则使用 `documentLaunchMode="none"`。

移除任务

默认情况下，在 Activity 结束后，文档任务会从概览屏幕中自动移除。您可以使用 `ActivityManager.AppTask` 类、`Intent` 标志或 `<activity>` 属性替代此行为。

通过将 `<activity>` 属性 `android:excludeFromRecents` 设置为 `true`，您可以始终将任务从概览屏幕中完全排除。

您可以通过将 `<activity>` 属性 `android:maxRecents` 设置为整型值，设置应用能够包括在概览屏幕中的最大任务数。默认值为 16。达到最大任务数后，最近最少使用的任务将从概览屏幕中移除。`android:maxRecents` 的最大值为 50（内存不足的设备上为 25）；小于 1 的值无效。

使用 AppTask 类移除任务

在与概览屏幕创建新任务的 Activity 中，您可以通过调用 `finishAndRemoveTask()` 方法指定何时移除该任务以及结束所有与之相关的 Activity。

NewDocumentActivity.java

```
public void onRemoveFromRecents(View view) {
    // The document is no longer needed; remove its task.
    finishAndRemoveTask();
}
```

注：如下所述，使用 `finishAndRemoveTask()` 方法代替使用 `FLAG_ACTIVITY_RETAIN_IN_RECENTS` 标记。

保留已完成的任务

若要将任务保留在概览屏幕中（即使其 Activity 已完成），可在启动 Activity 的 Intent 的 `addFlags()` 方法中传递 `FLAG_ACTIVITY_RETAIN_IN_RECENTS` 标志。

DocumentCentricActivity.java

```
private Intent newDocumentIntent() {
    final Intent newDocumentIntent = new Intent(this, NewDocumentActivity.class);
    newDocumentIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_DOCUMENT |
        android.content.Intent.FLAG_ACTIVITY_RETAIN_IN_RECENTS);
    newDocumentIntent.putExtra(KEY_EXTRA_NEW_DOCUMENT_COUNTER, incrementAndGet());
    return newDocumentIntent;
}
```

要达到同样的效果，请将 `<activity>` 属性 `android:autoRemoveFromRecents` 设置为 `false`。文档 Activity 的默认值为 `true`，常规 Activity 的默认值为 `false`。如前所述，使用此属性替代 `FLAG_ACTIVITY_RETAIN_IN_RECENTS` 标志。

服务

本文内容

- › [基础知识](#)
- › [使用清单文件声明服务](#)
- › [创建启动服务](#)
 - › [扩展 IntentService 类](#)
 - › [扩展服务类](#)
 - › [启动服务](#)
 - › [停止服务](#)
- › [创建绑定服务](#)
- › [向用户发送通知](#)
- › [在前台运行服务](#)
- › [管理服务生命周期](#)
- › [实现生命周期回调](#)

关键类

- › [Service](#)
- › [IntentService](#)

示例

- › [ServiceStartArguments](#)
- › [LocalService](#)

另请参阅

- › [绑定服务](#)

[Service](#) 是一个可以在后台执行长时间运行操作而不提供用户界面的应用组件。服务可由其他应用组件启动，而且即使用户切换到其他应用，服务仍将在后台继续运行。此外，组件可以绑定到服务，以与之进行交互，甚至是执行进程间通信 (IPC)。例如，服务可以处理网络事务、播放音乐，执行文件 I/O 或与内容提供程序交互，而所有这一切均可在后台进行。

服务基本上分为两种形式：

启动

当应用组件（如 Activity）通过调用 [startService\(\)](#) 启动服务时，服务即处于“启动”状态。一旦启动，服务即可在后台无限期运行，即使启动服务的组件已被销毁也不受影响。已启动的服务通常是执行单一操作，而且不会将结果返回给调用方。例如，它可能通过网络下载或上传文件。操作完成后，服务会自行停止运行。

绑定

当应用组件通过调用 [bindService\(\)](#) 绑定到服务时，服务即处于“绑定”状态。绑定服务提供了一个客户端-服务器接口，允许组件与服务进行交互、发送请求、获取结果，甚至是利用进程间通信 (IPC) 跨进程执行这些操作。仅当与另一个应用组件绑定时，绑定服务才会运行。多个组件可以同时绑定到该服务，但全部取消绑定后，该服务即会被销毁。

虽然本文档是分开概括讨论这两种服务，但是您的服务可以同时以这两种方式运行，也就是说，它既可以是启动服务（以无限期运行），也允许绑定。问题只是在于您是否实现了一组回调方法：[onStartCommand\(\)](#)（允许组件启动服务）和 [onBind\(\)](#)（允许绑定服务）。

无论应用是处于启动状态还是绑定状态，抑或处于启动并且绑定状态，任何应用组件均可像使用 Activity 那样通过调用 [Intent](#) 来使用服务（即使此服务来自另一应用）。不过，您可以通过清单文件将服务声明为私有服务，并阻止其他应用访问。[使用清单文件声明服务](#)部分将对此做更详尽的阐述。

注意：服务在其托管进程的主线程中运行，它既不创建自己的线程，也不在单独的进程中运行（除非另行指定）。这意味着，如果服务将执行任何 CPU 密集型工作或阻止性操作（例如 MP3 播放或联网），则应在服务内创建新线程来完成这项工作。通过使用单独的线程，可以降低发生“应用无响应”(ANR) 错误的风险，而应用的主线程仍可继续专注于运行用户与 Activity 之间的交互。

基础知识

要创建服务，您必须创建 [Service](#) 的子类（或使用它的一个现有子类）。在实现中，您需要重写一些回调方法，以处理服务生命周期的某些关键方面并提供一种机制将组件绑定到服务（如适用）。应重写的最重要的回调方法包括：

[onStartCommand\(\)](#)

当另一个组件（如 Activity）通过调用 [startService\(\)](#) 请求启动服务时，系统将调用此方法。一旦执行此方法，服务即会启动并可在后台无限期运行。如果您实现此方法，则在服务工作完成后，需要由您通过调用 [stopSelf\(\)](#) 或 [stopService\(\)](#) 来停止服务。
(如果您只想提供绑定，则无需实现此方法。)

[onBind\(\)](#)

当另一个组件想通过调用 [bindService\(\)](#) 与服务绑定（例如执行 RPC）时，系统将调用此方法。在此方法的实现中，您必须通过返回 [IBinder](#) 提供一个接口，供客户端用来与服务进行通信。请务必实现此方法，但如果并不希望允许绑定，则应返回 null。

[onCreate\(\)](#)

首次创建服务时，系统将调用此方法来执行一次性设置程序（在调用 [onStartCommand\(\)](#) 或 [onBind\(\)](#) 之前）。如果服务已在运行，则不会调用此方法。

[onDestroy\(\)](#)

当服务不再使用且将被销毁时，系统将调用此方法。服务应该实现此方法来清理所有资源，如线程、注册的侦听器、接收器等。这是服务接收的最后一个调用。

如果组件通过调用 [startService\(\)](#) 启动服务（这会导致对 [onStartCommand\(\)](#) 的调用），则服务将一直运行，直到服务使用 [stopSelf\(\)](#) 自行停止运行，或由其他组件通过调用 [stopService\(\)](#) 停止它为止。

如果组件是通过调用 [bindService\(\)](#) 来创建服务（且未调用 [onStartCommand\(\)](#)），则服务只会在该组件与其绑定时运行。一旦该服务与所有客户端之间的绑定全部取消，系统便会销毁它。

仅当内存过低且必须回收系统资源以供具有用户焦点的 Activity 使用时，Android 系统才会强制停止服务。如果将服务绑定到具有用户焦点的 Activity，则它不太可能会终止；如果将服务声明为[在前台运行](#)（稍后讨论），则它几乎永远不会终止。或者，如果服务已启动并要长时间运行，则系统会随着时间的推移降低服务在后台任务列表中的位置，而服务也将随之变得非常容易被终止；如果服务是启动服务，则您必须将其设计为能够妥善处理系统对它的重启。如果系统终止服务，那么一旦资源变得再次可用，系统便会重启服务（不过这还取决于从 [onStartCommand\(\)](#) 返回的值，本文稍后会对此加以讨论）。如需了解有关系统会在何时销毁服务的详细信息，请参阅[进程和线程](#)文档。

在下文中，您将了解如何创建各类服务以及如何从其他应用组件使用服务。

使用清单文件声明服务

如同 Activity（以及其他组件）一样，您必须在应用的清单文件中声明所有服务。

要声明服务，请添加 [`<service>`](#) 元素作为 [`<application>`](#) 元素的子元素。例如：

您应使用服务还是线程？

简单地说，服务是一种即使用户未与应用交互也可在后台运行的组件。因此，您应仅在必要时才创建服务。

如需在主线程外部执行工作，不过只是在用户正在与应用交互时才有此需要，则应创建新线程而非服务。例如，如果您只是想在 Activity 运行的同时播放一些音乐，则可在 [onCreate\(\)](#) 中创建线程，在 [onStart\(\)](#) 中启动线程，然后在 [onStop\(\)](#) 中停止线程。您还可以考虑使用 [AsyncTask](#) 或 [HandlerThread](#)，而非传统的 [Thread](#) 类。如需了解有关线程的详细信息，请参阅[进程和线程](#)文档。

请记住，如果您确实要使用服务，则默认情况下，它仍会在应用的主线程中运行，因此，如果服务执行的是密集型或阻止性操作，则您仍应在服务内创建新线程。

```
<manifest ... >
...
<application ... >
    <service android:name=".ExampleService" />
    ...
</application>
</manifest>
```

如需了解有关使用清单文件声明服务的详细信息，请参阅 [<service>](#) 元素参考文档。

您还可将其他属性包括在 [<service>](#) 元素中，以定义一些特性，如启动服务及其运行所在进程所需的权限。[android:name](#) 属性是唯一必需的属性，用于指定服务的类名。应用一旦发布，即不应更改此类名，如若不然，可能会存在因依赖显式 Intent 启动或绑定服务而破坏代码的风险（请阅读博客文章[Things That Cannot Change](#)[不能更改的内容]）。

为了确保应用的安全性，请始终使用显式 Intent 启动或绑定 Service，且不要为服务声明 Intent 过滤器。启动哪个服务存在一定的不确定性，而如果对这种不确定性的考量非常有必要，则可为服务提供 Intent 过滤器并从 Intent 中排除相应的组件名称，但随后必须使用 [setPackage\(\)](#) 方法设置 Intent 的软件包，这样可以充分消除目标服务的不确定性。

此外，还可以通过添加 [android:exported](#) 属性并将其设置为 "false"，确保服务仅适用于您的应用。这可以有效阻止其他应用启动您的服务，即便在使用显式 Intent 时也如此。

创建启动服务

启动服务由另一个组件通过调用 [startService\(\)](#) 启动，这会导致调用服务的 [onStartCommand\(\)](#) 方法。

服务启动之后，其生命周期即独立于启动它的组件，并且可以在后台无限期地运行，即使启动服务的组件已被销毁也不受影响。因此，服务应通过调用 [stopSelf\(\)](#) 结束工作来自行停止运行，或者由另一个组件通过调用 [stopService\(\)](#) 来停止它。

应用组件（如 Activity）可以通过调用 [startService\(\)](#) 方法并传递 Intent 对象（指定服务并包含待使用服务的所有数据）来启动服务。服务通过 [onStartCommand\(\)](#) 方法接收此 Intent。

例如，假设某 Activity 需要将一些数据保存到在线数据库中。该 Activity 可以启动一个协同服务，并通过向 [startService\(\)](#) 传递一个 Intent，为该服务提供要保存的数据。服务通过 [onStartCommand\(\)](#) 接收 Intent，连接到互联网并执行数据库事务。事务完成之后，服务会自行停止运行并随即被销毁。

注意：默认情况下，服务与服务声明所在的应用运行于同一进程，而且运行于该应用的主线程中。因此，如果服务在用户与来自同一应用的 Activity 进行交互时执行密集型或阻止性操作，则会降低 Activity 性能。为了避免影响应用性能，您应在服务内启动新线程。

从传统上讲，您可以扩展两个类来创建启动服务：

Service

这是适用于所有服务的基类。扩展此类时，必须创建一个用于执行所有服务工作的新线程，因为默认情况下，服务将使用应用的主线程，这会降低应用正在运行的所有 Activity 的性能。

IntentService

这是 [Service](#) 的子类，它使用工作线程逐一处理所有启动请求。如果您不要求服务同时处理多个请求，这是最好的选择。您只需实现 [onHandleIntent\(\)](#) 方法即可，该方法会接收每个启动请求的 Intent，使您能够执行后台工作。

下文介绍如何使用其中一个类来实现服务。

扩展 IntentService 类

由于大多数启动服务都不必同时处理多个请求（实际上，这种多线程情况可能很危险），因此使用 [IntentService](#) 类实现服务也许是最好的选择。

[IntentService](#) 执行以下操作：

- 创建默认的工作线程，用于在应用的主线程外执行传递给 [onStartCommand\(\)](#) 的所有 Intent。
- 创建工作队列，用于将 Intent 逐一传递给 [onHandleIntent\(\)](#) 实现，这样您就永远不必担心多线程问题。

- 在处理完所有启动请求后停止服务，因此您永远不必调用 `stopSelf()`。
- 提供 `onBind()` 的默认实现（返回 `null`）。
- 提供 `onStartCommand()` 的默认实现，可将 Intent 依次发送到工作队列和 `onHandleIntent()` 实现。

综上所述，您只需实现 `onHandleIntent()` 来完成客户端提供的工作即可。（不过，您还需要为服务提供小型构造函数。）

以下是 `IntentService` 的实现示例：

```
public class HelloIntentService extends IntentService {
    /**
     * A constructor is required, and must call the super IntentService(String)
     * constructor with a name for the worker thread.
     */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /**
     * The IntentService calls this method from the default worker thread with
     * the intent that started the service. When this method returns, IntentService
     * stops the service, as appropriate.
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a file.
        // For our sample, we just sleep for 5 seconds.
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // Restore interrupt status.
            Thread.currentThread().interrupt();
        }
    }
}
```

您只需要一个构造函数和一个 `onHandleIntent()` 实现即可。

如果您决定还重写其他回调方法（如 `onCreate()`、`onStartCommand()` 或 `onDestroy()`），请确保调用超类实现，以便 `IntentService` 能够妥善处理工作线程的生命周期。

例如，`onStartCommand()` 必须返回默认实现（即，如何将 Intent 传递给 `onHandleIntent()`）：

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
    return super.onStartCommand(intent, flags, startId);
}
```

除 `onHandleIntent()` 之外，您无需从中调用超类的唯一方法就是 `onBind()`（仅当服务允许绑定时，才需要实现该方法）。

在下一部分中，您将了解如何在扩展 `Service` 基类时实现同类服务。该基类包含更多代码，但如需同时处理多个启动请求，则更适合使用该基类。

扩展服务类

正如上一部分中所述，使用 `IntentService` 显著简化了启动服务的实现。但是，若要求服务执行多线程（而不是通过工作队列处理启动请求），则可扩展 `Service` 类来处理每个 Intent。

为了便于比较，以下提供了 `Service` 类实现的代码示例，该类执行的工作与上述使用 `IntentService` 的示例完全相同。也就是说，对于每个启动请求，它均使用工作线程执行作业，且每次仅处理一个请求。

```

public class HelloService extends Service {
    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // Handler that receives messages from the thread
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {
            // Normally we would do some work here, like download a file.
            // For our sample, we just sleep for 5 seconds.
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // Restore interrupt status.
                Thread.currentThread().interrupt();
            }
            // Stop the service using the startId, so that we don't stop
            // the service in the middle of handling another job
            stopSelf(msg.arg1);
        }
    }

    @Override
    public void onCreate() {
        // Start up the thread running the service. Note that we create a
        // separate thread because the service normally runs in the process's
        // main thread, which we don't want to block. We also make it
        // background priority so CPU-intensive work will not disrupt our UI.
        HandlerThread thread = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
        thread.start();

        // Get the HandlerThread's Looper and use it for our Handler
        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

        // For each start request, send a message to start a job and deliver the
        // start ID so we know which request we're stopping when we finish the job
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        mServiceHandler.sendMessage(msg);

        // If we get killed, after returning from here, restart
        return START_STICKY;
    }

    @Override
    public IBinder onBind(Intent intent) {
        // We don't provide binding, so return null
        return null;
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
    }
}

```

正如您所见，与使用 `IntentService` 相比，这需要执行更多工作。

但是，因为是由您自己处理对 `onStartCommand()` 的每个调用，因此可以同时执行多个请求。此示例并未这样做，但如果希望如此，则可为每个请求创建一个新线程，然后立即运行这些线程（而不是等待上一个请求完成）。

请注意，`onStartCommand()` 方法必须返回整型数。整型数是一个值，用于描述系统应该如何在服务终止的情况下继续运行服务（如上所述，`IntentService` 的默认实现将为您处理这种情况，不过您可以对其进行修改）。从 `onStartCommand()` 返回的值必须是以下常量之一：

START_NOT_STICKY

如果系统在 `onStartCommand()` 返回后终止服务，则除非有挂起 Intent 要传递，否则系统不会重建服务。这是最安全的选项，可以避免在不必要时以及应用能够轻松重启所有未完成的作业时运行服务。

START_STICKY

如果系统在 `onStartCommand()` 返回后终止服务，则会重建服务并调用 `onStartCommand()`，但不会重新传递最后一个 Intent。相反，除非有挂起 Intent 要启动服务（在这种情况下，将传递这些 Intent），否则系统会通过空 Intent 调用 `onStartCommand()`。这适用于不执行命令、但无限期运行并等待作业的媒体播放器（或类似服务）。

START_REDELIVER_INTENT

如果系统在 `onStartCommand()` 返回后终止服务，则会重建服务，并通过传递给服务的最后一个 Intent 调用 `onStartCommand()`。任何挂起 Intent 均依次传递。这适用于主动执行应该立即恢复的作业（例如下载文件）的服务。

有关这些返回值的更多详细信息，请查阅每个常量链接的参考文档。

启动服务

您可以通过将 `Intent`（指定要启动的服务）传递给 `startService()`，从 `Activity` 或其他应用组件启动服务。Android 系统调用服务的 `onStartCommand()` 方法，并向其传递 `Intent`。（切勿直接调用 `onStartCommand()`。）

例如，`Activity` 可以结合使用显式 `Intent` 与 `startService()`，启动上文中的示例服务 (`HelloService`)：

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

`startService()` 方法将立即返回，且 Android 系统调用服务的 `onStartCommand()` 方法。如果服务尚未运行，则系统会先调用 `onCreate()`，然后再调用 `onStartCommand()`。

如果服务亦未提供绑定，则使用 `startService()` 传递的 `Intent` 是应用组件与服务之间唯一的通信模式。但是，如果您希望服务返回结果，则启动服务的客户端可以为广播创建一个 `PendingIntent`（使用 `getBroadcast()`），并通过启动服务的 `Intent` 传递给服务。然后，服务就可以使用广播传递结果。

多个服务启动请求会导致多次对服务的 `onStartCommand()` 进行相应的调用。但是，要停止服务，只需一个服务停止请求（使用 `stopSelf()` 或 `stopService()`）即可。

停止服务

启动服务必须管理自己的生命周期。也就是说，除非系统必须回收内存资源，否则系统不会停止或销毁服务，而且服务在 `onStartCommand()` 返回后会继续运行。因此，服务必须通过调用 `stopSelf()` 自行停止运行，或者由另一个组件通过调用 `stopService()` 来停止它。

一旦请求使用 `stopSelf()` 或 `stopService()` 停止服务，系统就会尽快销毁服务。

但是，如果服务同时处理多个 `onStartCommand()` 请求，则您不应在处理完一个启动请求之后停止服务，因为您可能已经收到了新的启动请求（在第一个请求结束时停止服务会终止第二个请求）。为了避免这一问题，您可以使用 `stopSelf(int)` 确保服务停止请求始终基于最近的启动请求。也就是说，在调用 `stopSelf(int)` 时，传递与停止请求的 ID 对应的启动请求的 ID（传递给 `onStartCommand()` 的 `startId`）。然后，如果在您能够调用 `stopSelf(int)` 之前服务收到了新的启动请求，ID 就不匹配，服务也就不会停止。

注意：为了避免浪费系统资源和消耗电池电量，应用必须在工作完成之后停止其服务。如有必要，其他组件可以通过调用 `stopService()` 来停止服务。即使为服务启用了绑定，一旦服务收到对 `onStartCommand()` 的调用，您始终仍须亲自停止服务。

如需了解有关服务生命周期的详细信息，请参阅下面有关 [管理服务生命周期](#) 的部分。

创建绑定服务

绑定服务允许应用组件通过调用 `bindService()` 与其绑定，以便创建长期连接（通常不允许组件通过调用 `startService()` 来启动它）。

如需与 Activity 和其他应用组件中的服务进行交互，或者需要通过进程间通信 (IPC) 向其他应用公开某些应用功能，则应创建绑定服务。

要创建绑定服务，必须实现 `onBind()` 回调方法以返回 `IBinder`，用于定义与服务通信的接口。然后，其他应用组件可以调用 `bindService()` 来检索该接口，并开始对服务调用方法。服务只用于与其绑定的应用组件，因此如果没有组件绑定到服务，则系统会销毁服务（您不必按通过 `onStartCommand()` 启动的服务那样来停止绑定服务）。

要创建绑定服务，首先必须定义指定客户端如何与服务通信的接口。服务与客户端之间的这个接口必须是 `IBinder` 的实现，并且服务必须从 `onBind()` 回调方法返回它。一旦客户端收到 `IBinder`，即可开始通过该接口与服务进行交互。

多个客户端可以同时绑定到服务。客户端完成与服务的交互后，会调用 `unbindService()` 取消绑定。一旦没有客户端绑定到该服务，系统就会销毁它。

有多种方法实现绑定服务，其实现比启动服务更为复杂，因此绑定服务将在有关 [绑定服务](#) 的单独文档中专门讨论。

向用户发送通知

一旦运行起来，服务即可使用 [Toast 通知或状态栏通知](#) 来通知用户所发生的事件。

`Toast` 通知是指出现在当前窗口的表面、片刻随即消失不见的消息，而状态栏通知则在状态栏中随消息一起提供图标，用户可以选择该图标来采取操作（例如启动 Activity）。

通常，当某些后台工作已经完成（例如文件下载完成）且用户现在可以对其进行操作时，状态栏通知是最佳方法。当用户从展开视图中选定通知时，通知即可启动 Activity（例如查看已下载的文件）。

如需了解详细信息，请参阅 [Toast 通知或状态栏通知](#) 开发者指南。

在前台运行服务

前台服务被认为是用户主动意识到的一种服务，因此在内存不足时，系统也不会考虑将其终止。前台服务必须为状态栏提供通知，放在“正在进行”标题下方，这意味着除非服务停止或从前台移除，否则不能清除通知。

例如，应该将通过服务播放音乐的音乐播放器设置为在前台运行，这是因为用户明确意识到其操作。状态栏中的通知可能表示正在播放的歌曲，并允许用户启动 Activity 来与音乐播放器进行交互。

要请求让服务运行于前台，请调用 `startForeground()`。此方法采用两个参数：唯一标识通知的整型数和状态栏的 `Notification`。例如：

```
Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker_text),
    System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(this, getText(R.string.notification_title),
    getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

注意：提供给 `startForeground()` 的整型 ID 不得为 0。

要从前台移除服务，请调用 `stopForeground()`。此方法采用一个布尔值，指示是否也移除状态栏通知。此方法不会停止服务。但是，如果您在服务正在前台运行时将其停止，则通知也会被移除。

如需了解有关通知的详细信息，请参阅[创建状态栏通知](#)。

管理服务生命周期

服务的生命周期比 Activity 的生命周期要简单得多。但是，密切关注如何创建和销毁服务反而更加重要，因为服务可以在用户没有意识到的情况下运行于后台。

服务生命周期（从创建到销毁）可以遵循两条不同的路径：

- 启动服务

该服务在其他组件调用 `startService()` 时创建，然后无限期运行，且必须通过调用 `stopSelf()` 来自行停止运行。此外，其他组件也可

以通过调用 `stopService()` 来停止服务。服务停止后，系统会将其销毁。

- 绑定服务

该服务在另一个组件（客户端）调用 `bindService()` 时创建。然后，客户端通过 `IBinder` 接口与服务进行通信。客户端可以通过调用 `unbindService()` 关闭连接。多个客户端可以绑定到相同服务，而且当所有绑定全部取消后，系统即会销毁该服务。（服务不必自行停止运行。）

这两条路径并非完全独立。也就是说，您可以绑定到已经使用 `startService()` 启动的服务。例如，可以通过使用 `Intent`（标识要播放的音乐）调用 `startService()` 来启动后台音乐服务。随后，可能在用户需要稍加控制播放器或获取有关当前播放歌曲的信息时，Activity 可以通过调用 `bindService()` 绑定到服务。在这种情况下，除非所有客户端均取消绑定，否则 `stopService()` 或 `stopSelf()` 不会实际停止服务。

实现生命周期回调

与 Activity 类似，服务也拥有生命周期回调方法，您可以实现这些方法来监控服务状态的变化并适时执行工作。以下框架服务展示了每种生命周期方法：

```
public class ExampleService extends Service {
    int mStartMode;          // indicates how to behave if the service is killed
    IBinder mBinder;         // interface for clients that bind
    boolean mAllowRebind;    // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

注：与 Activity 生命周期回调方法不同，您不需要调用这些回调方法的超类实现。

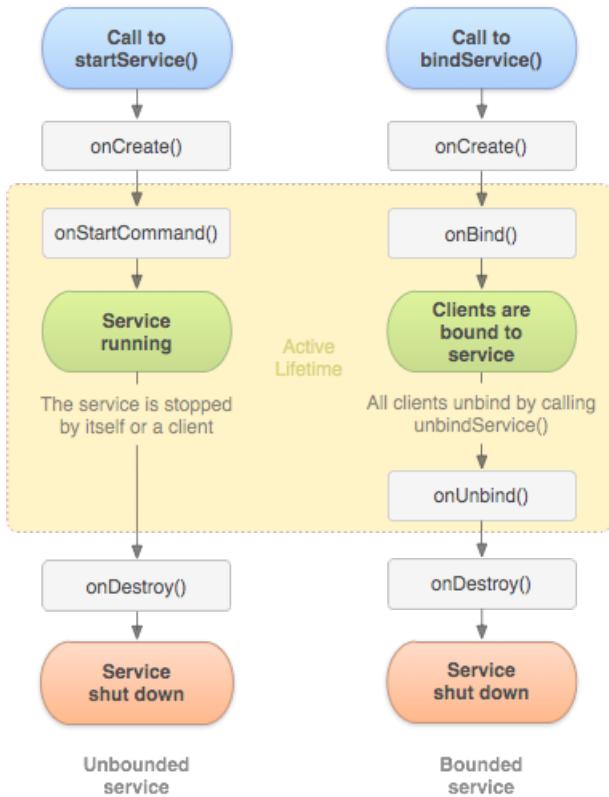


图 2. 服务生命周期。左图显示了使用 `startService()` 所创建的服务的生命周期，右图显示了使用 `bindService()` 所创建的服务的生命周期。

通过实现这些方法，您可以监控服务生命周期的两个嵌套循环：

- 服务的**整个生命周期**从调用 `onCreate()` 开始起，到 `onDestroy()` 返回时结束。与 Activity 类似，服务也在 `onCreate()` 中完成初始设置，并在 `onDestroy()` 中释放所有剩余资源。例如，音乐播放服务可以在 `onCreate()` 中创建用于播放音乐的线程，然后在 `onDestroy()` 中停止该线程。
- 无论服务是通过 `startService()` 还是 `bindService()` 创建，都会为所有服务调用 `onCreate()` 和 `onDestroy()` 方法。
- 服务的**有效生命周期**从调用 `onStartCommand()` 或 `onBind()` 方法开始。每种方法均有 `Intent` 对象，该对象分别传递到 `startService()` 或 `bindService()`。

对于启动服务，有效生命周期与整个生命周期同时结束（即便是在 `onStartCommand()` 返回之后，服务仍然处于活动状态）。对于绑定服务，有效生命周期在 `onUnbind()` 返回时结束。

注：尽管启动服务是通过调用 `stopSelf()` 或 `stopService()` 来停止，但是该服务并无相应的回调（没有 `onStop()` 回调）。因此，除非服务绑定到客户端，否则在服务停止时，系统会将其销毁 — `onDestroy()` 是接收到的唯一回调。

图 2 说明了服务的典型回调方法。尽管该图分开介绍通过 `startService()` 创建的服务和通过 `bindService()` 创建的服务，但是请记住，不管启动方式如何，任何服务均有可能允许客户端与其绑定。因此，最初使用 `onStartCommand()`（通过客户端调用 `startService()`）启动的服务仍可接收对 `onBind()` 的调用（当客户端调用 `bindService()` 时）。

如需了解有关创建提供绑定的服务的详细信息，请参阅[绑定服务](#)文档，该文档的[管理绑定服务的生命周期](#)部分提供了有关 `onRebind()` 回调方法的更多信息。

绑定服务

本文内容

- › [基础知识](#)
- › [创建绑定服务](#)
 - › [扩展 Binder 类](#)
 - › [使用 Messenger](#)
- › [绑定到服务](#)
 - › [附加说明](#)
- › [管理绑定服务的生命周期](#)

关键类

- › [Service](#)
- › [ServiceConnection](#)
- › [IBinder](#)

示例

- › [RemoteService](#)
- › [LocalService](#)

另请参阅

- › [服务](#)

绑定服务是客户端-服务器接口中的服务器。绑定服务可让组件（例如 Activity）绑定到服务、发送请求、接收响应，甚至执行进程间通信 (IPC)。绑定服务通常只在为其他应用组件服务时处于活动状态，不会无限期在后台运行。

本文向您介绍如何创建绑定服务，包括如何绑定到来自其他应用组件的服务。不过，您还应参阅[服务](#)文档，了解有关一般服务的更多信息，例如：如何利用服务传送通知、如何将服务设置为在前台运行等等。

基础知识

绑定服务是 [Service](#) 类的实现，可让其他应用与其绑定和交互。要提供服务绑定，您必须实现 [onBind\(\)](#) 回调方法。该方法返回的 [IBinder](#) 对象定义了客户端用来与服务进行交互的编程接口。

客户端可通过调用 [bindService\(\)](#) 绑定到服务。调用时，它必须提供 [ServiceConnection](#) 的实现，后者会监控与服务的连接。[bindService\(\)](#) 方法会立即无值返回，但当 Android 系统创建客户端与服务之间的连接时，会对 [ServiceConnection](#) 调用 [onServiceConnected\(\)](#)，向客户端传递用来与服务通信的 [IBinder](#)。

多个客户端可同时连接到一个服务。不过，只有在第一个客户端绑定时，系统才会调用服务的 [onBind\(\)](#) 方法来检索 [IBinder](#)。系统随后无需再次调用 [onBind\(\)](#)，便可将同一 [IBinder](#) 传递至任何其他绑定的客户端。

当最后一个客户端取消与服务的绑定时，系统会将服务销毁（除非 [startService\(\)](#) 也启动了该服务）。

当您实现绑定服务时，最重要的环节是定义您的 [onBind\(\)](#) 回调方法返回的接口。您可以通过几种不同的方法定义服务的 [IBinder](#) 接口，下文对这些方法逐一做了阐述。

绑定到已启动服务

正如[服务](#)文档中所述，您可以创建同时具有已启动和绑定两种状态的服务。也就是说，可通过调用 [startService\(\)](#) 启动该服务，让服务无限期运行；此外，还可通过调用 [bindService\(\)](#) 使客户端绑定到服务。

如果您确实允许服务同时具有已启动和绑定状态，则服务启动后，系统“不会”在所有客户端都取消绑定时销毁服务。为此，您必须通过调用 [stopSelf\(\)](#) 或 [stopService\(\)](#) 显式停止服务。

创建绑定服务

创建提供绑定的服务时，您必须提供 [IBinder](#)，用以提供客户端用来与服务进行交互的编程接口。您可以通过三种方法定义接口：

扩展 Binder 类

如果服务是供您的自有应用专用，并且在与客户端相同的进程中运行（常见情况），则应通过扩展 [Binder](#) 类并从 [onBind\(\)](#) 返回它的一个实例来创建接口。客户端收到 [Binder](#) 后，可利用它直接访问 [Binder](#) 实现中乃至 [Service](#) 中可用的公共方法。

如果服务只是您的自有应用的后台工作线程，则优先采用这种方法。不以这种方式创建接口的唯一原因是，您的服务被其他应用或不同的进程占用。

使用 Messenger

如需让接口跨不同的进程工作，则可使用 [Messenger](#) 为服务创建接口。服务可以这种方式定义对应于不同类型 [Message](#) 对象的 [Handler](#)。此 [Handler](#) 是 [Messenger](#) 的基础，后者随后可与客户端分享一个 [IBinder](#)，从而让客户端能利用 [Message](#) 对象向服务发送命令。此外，客户端还可定义自有 [Messenger](#)，以便服务回传消息。

这是执行进程间通信 (IPC) 的最简单方法，因为 [Messenger](#) 会在单一线程中创建包含所有请求的队列，这样您就不必对服务进行线程安全设计。

使用 AIDL

AIDL（Android 接口定义语言）执行所有将对象分解成原语的工作，操作系统可以识别这些原语并将它们编组到各进程中，以执行 IPC。之前采用 [Messenger](#) 的方法实际上是以 AIDL 作为其底层结构。如上所述，[Messenger](#) 会在单一线程中创建包含所有客户端请求的队列，以便服务一次接收一个请求。不过，如果您想让服务同时处理多个请求，则可直接使用 AIDL。在此情况下，您的服务必须具备多线程处理能力，并采用线程安全式设计。

如需直接使用 AIDL，您必须创建一个定义编程接口的 `.aidl` 文件。Android SDK 工具利用该文件生成一个实现接口并处理 IPC 的抽象类，您随后可在服务内对其进行扩展。

注：大多数应用“都不会”使用 AIDL 来创建绑定服务，因为它可能要求具备多线程处理能力，并可能导致实现的复杂性增加。因此，AIDL 并不适合大多数应用，本文也不会阐述如何将其用于您的服务。如果您确定自己需要直接使用 AIDL，请参阅 [AIDL 文档](#)。

扩展 Binder 类

如果您的服务仅供本地应用使用，不需要跨进程工作，则可以实现自有 [Binder](#) 类，让您的客户端通过该类直接访问服务中的公共方法。

注：此方法只有在客户端和服务位于同一应用和进程内这一最常见的情况下才有效。例如，对于需要将 Activity 绑定到在后台播放音乐的自有服务的音乐应用，此方法非常有效。

以下是具体的设置方法：

1. 在您的服务中，创建一个可满足下列任一要求的 [Binder](#) 实例：
 - 包含客户端可调用的公共方法
 - 返回当前 [Service](#) 实例，其中包含客户端可调用的公共方法
 - 或返回由服务承载的其他类的实例，其中包含客户端可调用的公共方法
2. 从 [onBind\(\)](#) 回调方法返回此 [Binder](#) 实例。
3. 在客户端中，从 [onServiceConnected\(\)](#) 回调方法接收 [Binder](#)，并使用提供的方法调用绑定服务。

注：之所以要求服务和客户端必须在同一应用内，是为了便于客户端转换返回的对象和正确调用其 API。服务和客户端还必须在同一进程中，因为此方法不执行任何跨进程编组。

例如，以下这个服务可让客户端通过 [Binder](#) 实现访问服务中的方法：

尽管您通常应该实现 [onBind\(\)](#) 或 [onStartCommand\(\)](#)，但有时需要同时实现这两者。例如，音乐播放器可能发现让其服务无限期运行并同时提供绑定很有用处。这样一来，Activity 便可启动服务进行音乐播放，即使用户离开应用，音乐播放也不会停止。然后，当用户返回应用时，Activity 可绑定到服务，重新获得回放控制权。

请务必阅读[管理绑定服务的生命周期](#)部分，详细了解有关为已启动服务添加绑定时该服务的生命周期信息。

```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder. Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC.
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Return this instance of LocalService so clients can call public methods
            return LocalService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }

    /** method for clients */
    public int getRandomNumber() {
        return mGenerator.nextInt(100);
    }
}
```

LocalBinder 为客户端提供 `getService()` 方法，以检索 `LocalService` 的当前实例。这样，客户端便可调用服务中的公共方法。例如，客户端可调用服务中的 `getRandomNumber()`。

点击按钮时，以下这个 Activity 会绑定到 `LocalService` 并调用 `getRandomNumber()`：

```

public class BindingActivity extends Activity {
    LocalService mService;
    boolean mBound = false;

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

    @Override
    protected void onStart() {
        super.onStart();
        // Bind to LocalService
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Unbind from the service
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }

    /** Called when a button is clicked (the button in the layout file attaches to
     * this method with the android:onClick attribute) */
    public void onClick(View v) {
        if (mBound) {
            // Call a method from the LocalService.
            // However, if this call were something that might hang, then this request should
            // occur in a separate thread to avoid slowing down the activity performance.
            int num = mService.getRandomNumber();
            Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show();
        }
    }

    /** Defines callbacks for service binding, passed to bindService() */
    private ServiceConnection mConnection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName className,
                                      IBinder service) {
            // We've bound to LocalService, cast the IBinder and get LocalService instance
            LocalBinder binder = (LocalBinder) service;
            mService = binder.getService();
            mBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            mBound = false;
        }
    };
}

```

上例说明了客户端如何使用 `ServiceConnection` 的实现和 `onServiceConnected()` 回调绑定到服务。下文更详细介绍了绑定到服务的过程。

注：在上例中，`onStop()` 方法将客户端与服务取消绑定。客户端应在适当时机与服务取消绑定，如[附加说明](#)中所述。

如需查看更多示例代码，请参见 `ApiDemos` 中的 `LocalService.java` 类和 `LocalServiceActivities.java` 类。

使用 Messenger

如需让服务与远程进程通信，则可使用 `Messenger` 为您的服务提供接口。利用此方法，您无需使用 AIDL 便可执行进程间通信 (IPC)。

以下是 `Messenger` 的使用方法摘要：

- 服务实现一个 `Handler`，由其接收来自客户端的每个调用的回调
- `Handler` 用于创建 `Messenger` 对象（对 `Handler` 的引用）
- `Messenger` 创建一个 `IBinder`，服务通过 `onBind()` 使其返回客户端
- 客户端使用 `IBinder` 将 `Messenger`（引用服务的 `Handler`）实例化，然后使用后者将 `Message` 对象发送给服务
- 服务在其 `Handler` 中（具体地讲，是在 `handleMessage()` 方法中）接收每个 `Message`。

这样，客户端并没有调用服务的“方法”。而客户端传递的“消息”（`Message` 对象）是服务在其 `Handler` 中接收的。

以下是一个使用 `Messenger` 接口的简单服务示例：

```
public class MessengerService extends Service {
    /** Command to the service to display a message */
    static final int MSG_SAY_HELLO = 1;

    /**
     * Handler of incoming messages from clients.
     */
    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_SAY_HELLO:
                    Toast.makeText(getApplicationContext(), "hello!", Toast.LENGTH_SHORT).show();
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    /**
     * Target we publish for clients to send messages to IncomingHandler.
     */
    final Messenger mMessenger = new Messenger(new IncomingHandler());

    /**
     * When binding to the service, we return an interface to our messenger
     * for sending messages to the service.
     */
    @Override
    public IBinder onBind(Intent intent) {
        Toast.makeText(getApplicationContext(), "binding", Toast.LENGTH_SHORT).show();
        return mMessenger.getBinder();
    }
}
```

请注意，服务就是在 `Handler` 的 `handleMessage()` 方法中接收传入的 `Message`，并根据 `what` 成员决定下一步操作。

客户端只需根据服务返回的 `IBinder` 创建一个 `Messenger`，然后利用 `send()` 发送一条消息。例如，以下就是一个绑定到服务并向服务传递 `MSG_SAY_HELLO` 消息的简单 Activity：

与 AIDL 比较

当您需要执行 IPC 时，为您的接口使用 `Messenger` 要比使用 AIDL 实现它更加简单，因为 `Messenger` 会将所有服务调用排入队列，而纯粹的 AIDL 接口会同时向服务发送多个请求，服务随后必须应对多线程处理。

对于大多数应用，服务不需要执行多线程处理，因此使用 `Messenger` 可让服务一次处理一个调用。如果您的服务必须执行多线程处理，则应使用 AIDL 来定义接口。

```

public class ActivityMessenger extends Activity {
    /** Messenger for communicating with the service. */
    Messenger mService = null;

    /** Flag indicating whether we have called bind on the service. */
    boolean mBound;

    /**
     * Class for interacting with the main interface of the service.
     */
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            // This is called when the connection with the service has been
            // established, giving us the object we can use to
            // interact with the service. We are communicating with the
            // service using a Messenger, so here we get a client-side
            // representation of that from the raw IBinder object.
            mService = new Messenger(service);
            mBound = true;
        }

        public void onServiceDisconnected(ComponentName className) {
            // This is called when the connection with the service has been
            // unexpectedly disconnected -- that is, its process crashed.
            mService = null;
            mBound = false;
        }
    };

    public void sayHello(View v) {
        if (!mBound) return;
        // Create and send a message to the service, using a supported 'what' value
        Message msg = Message.obtain(null, MessengerService.MSG_SAY_HELLO, 0, 0);
        try {
            mService.send(msg);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

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

    @Override
    protected void onStart() {
        super.onStart();
        // Bind to the service
        bindService(new Intent(this, MessengerService.class), mConnection,
                   Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Unbind from the service
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }
}

```

请注意，此示例并未说明服务如何对客户端作出响应。如果您想让服务作出响应，则还需要在客户端中创建一个 `Messenger`。然后，当客户端收到 `onServiceConnected()` 回调时，会向服务发送一条 `Message`，并在其 `send()` 方法的 `replyTo` 参数中包含客户端的 `Messenger`。

如需查看如何提供双向消息传递的示例，请参阅 `MessengerService.java`（服务）和 `MessengerServiceActivities.java`（客户端）示例。

绑定到服务

应用组件（客户端）可通过调用 `bindService()` 绑定到服务。Android 系统随后调用服务的 `onBind()` 方法，该方法返回用于与服务交互的 `IBinder`。

绑定是异步的。`bindService()` 会立即返回，“不会”使 `IBinder` 返回客户端。要接收 `IBinder`，客户端必须创建一个 `ServiceConnection` 实例，并将其传递给 `bindService()`。`ServiceConnection` 包括一个回调方法，系统通过调用它来传递 `IBinder`。

注：只有 Activity、服务和内容提供程序可以绑定到服务 — 您无法从广播接收器绑定到服务。

因此，要想从您的客户端绑定到服务，您必须：

1. 实现 `ServiceConnection`。

您的实现必须重写两个回调方法：

`onServiceConnected()`

系统会调用该方法以传递服务的 `onBind()` 方法返回的 `IBinder`。

`onServiceDisconnected()`

Android 系统会在与服务的连接意外中断时（例如当服务崩溃或被终止时）调用该方法。当客户端取消绑定时，系统“不会”调用该方法。

2. 调用 `bindService()`，传递 `ServiceConnection` 实现。

3. 当系统调用您的 `onServiceConnected()` 回调方法时，您可以使用接口定义的方法开始调用服务。

4. 要断开与服务的连接，请调用 `unbindService()`。

如果应用在客户端仍绑定到服务时销毁客户端，则销毁会导致客户端取消绑定。更好的做法是在客户端与服务交互完成后立即取消绑定客户端。这样可以关闭空闲服务。如需了解有关绑定和取消绑定的适当时机的详细信息，请参阅[附加说明](#)。

例如，以下代码段通过[扩展 Binder 类](#)将客户端与上面创建的服务相连，因此它只需将返回的 `IBinder` 转换为 `LocalService` 类并请求 `LocalService` 实例：

```
LocalService mService;
private ServiceConnection mConnection = new ServiceConnection() {
    // Called when the connection with the service is established
    public void onServiceConnected(ComponentName className, IBinder service) {
        // Because we have bound to an explicit
        // service that is running in our own process, we can
        // cast its IBinder to a concrete class and directly access it.
        LocalBinder binder = (LocalBinder) service;
        mService = binder.getService();
        mBound = true;
    }

    // Called when the connection with the service disconnects unexpectedly
    public void onServiceDisconnected(ComponentName className) {
        Log.e(TAG, "onServiceDisconnected");
        mBound = false;
    }
};
```

客户端可通过将此 `ServiceConnection` 传递至 `bindService()` 绑定到服务。例如：

```
Intent intent = new Intent(this, LocalService.class);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

- `bindService()` 的第一个参数是一个 `Intent`，用于显式命名要绑定的服务（但 `Intent` 可能是隐式的）
- 第二个参数是 `ServiceConnection` 对象

- 第三个参数是一个指示绑定选项的标志。它通常应该是 `BIND_AUTO_CREATE`，以便创建尚未激活的服务。其他可能的值为 `BIND_DEBUG_UNBIND` 和 `BIND_NOT_FOREGROUND`，或 `0`（表示无）。

附加说明

以下是一些有关绑定到服务的重要说明：

- 您应该始终捕获 `DeadObjectException` 异常，它们是在连接中断时引发的。这是远程方法引发的唯一异常。
- 对象是跨进程计数的引用。
- 您通常应该在客户端生命周期的匹配引入 (bring-up) 和退出 (tear-down) 时刻期间配对绑定和取消绑定。例如：
 - 如果您只需要在 Activity 可见时与服务交互，则应在 `onStart()` 期间绑定，在 `onStop()` 期间取消绑定。
 - 如果您希望 Activity 在后台停止运行状态下仍可接响应，则可在 `onCreate()` 期间绑定，在 `onDestroy()` 期间取消绑定。请注意，这意味着您的 Activity 在其整个运行过程中（甚至包括后台运行期间）都需要使用服务，因此如果服务位于其他进程中，那么当您提高该进程的权重时，系统终止该进程的可能性会增加。

注：通常情况下，**切勿**在 Activity 的 `onResume()` 和 `onPause()` 期间绑定和取消绑定，因为每一次生命周期转换都会发生这些回调，您应该使发生在这些转换期间的处理保持在最低水平。此外，如果您的应用内的多个 Activity 绑定到同一服务，并且其中两个 Activity 之间发生了转换，则如果当前 Activity 在下一个 Activity 绑定（恢复期间）之前取消绑定（暂停期间），系统可能会销毁服务并重建服务。
(Activity 文档中介绍了这种有关 Activity 如何协调其生命周期的 Activity 转换。)

如需查看更多显示如何绑定到服务的示例代码，请参阅 `ApiDemos` 中的 `RemoteService.java` 类。

管理绑定服务的生命周期

当服务与所有客户端之间的绑定全部取消时，Android 系统便会销毁服务（除非还使用 `onStartCommand()` 启动了该服务）。因此，如果您的服务是纯粹的绑定服务，则无需对其生命周期进行管理 — Android 系统会根据它是否绑定到任何客户端代您管理。

不过，如果您选择实现 `onStartCommand()` 回调方法，则您必须显式停止服务，因为系统现在已将服务视为已启动。在此情况下，服务将一直运行到其通过 `stopSelf()` 自行停止，或其他组件调用 `stopService()` 为止，无论其是否绑定到任何客户端。

此外，如果您的服务已启动并接受绑定，则当系统调用您的 `onUnbind()` 方法时，如果您想在客户端下一次绑定到服务时接收 `onRebind()` 调用，则可选择返回 `true`。`onRebind()` 返回空值，但客户端仍在其 `onServiceConnected()` 回调中接收 `IBinder`。下文图 1 说明了这种生命周期的逻辑。

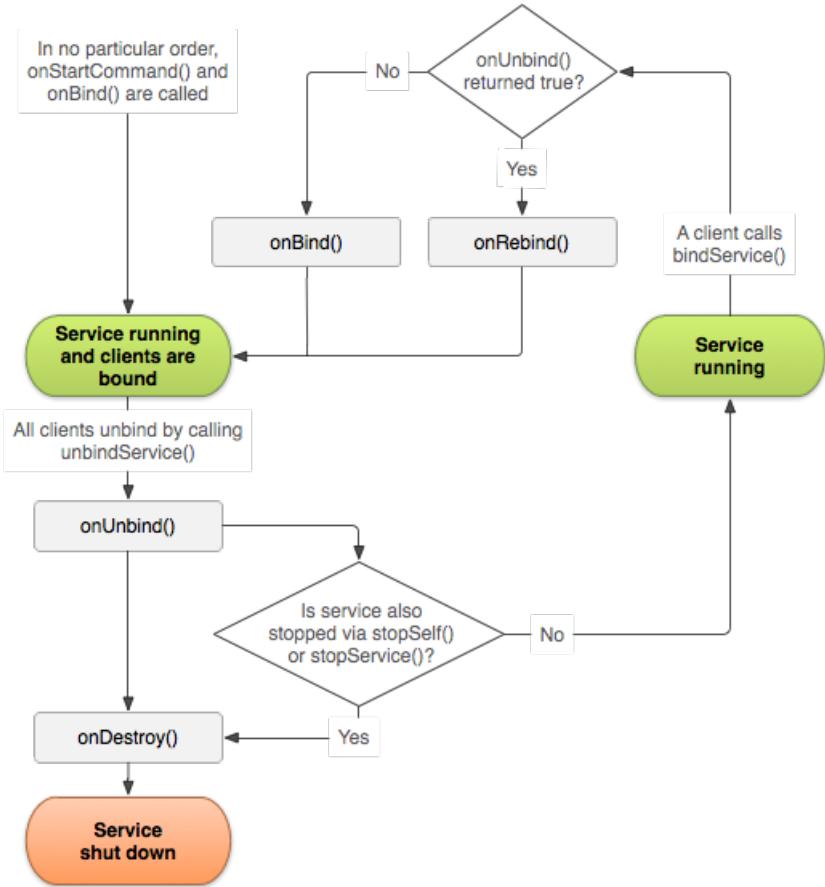


图 1. 允许绑定的已启动服务的生命周期。

如需了解有关已启动服务生命周期的详细信息，请参阅[服务](#)文档。



Android 接口定义语言 (AIDL)

本文内容

- › [定义 AIDL 接口](#)
 - › [创建 .aidl 文件](#)
 - › [实现接口](#)
 - › [向客户端公开该接口](#)
- › [通过 IPC 传递对象](#)
- › [调用 IPC 方法](#)

另请参阅

- › [绑定服务](#)

AIDL（Android 接口定义语言）与您可能使用过的其他 IDL 类似。您可以利用它定义客户端与服务使用进程间通信 (IPC) 进行相互通信时都认可的编程接口。在 Android 上，一个进程通常无法访问另一个进程的内存。尽管如此，进程需要将其对象分解成操作系统能够识别的原语，并将对象编组成跨越边界的对象。编写执行这一编组操作的代码是一项繁琐的工作，因此 Android 会使用 AIDL 来处理。

注：只有允许不同应用的客户端用 IPC 方式访问服务，并且想要在服务中处理多线程时，才有必要使用 AIDL。如果您不需要执行跨越不同应用的并发 IPC，就应该通过[实现一个 Binder 创建接口](#)；或者，如果您想执行 IPC，但根本不需要处理多线程，则[使用 Messenger 类来实现接口](#)。无论如何，在实现 AIDL 之前，请您务必理解[绑定服务](#)。

在您开始设计 AIDL 接口之前，要注意 AIDL 接口的调用是直接函数调用。您不应该假设发生调用的线程。视调用来自本地进程还是远程进程中的线程，实际情况会有所差异。具体而言：

- 来自本地进程的调用在发起调用的同一线程内执行。如果该线程是您的主 UI 线程，则该线程继续在 AIDL 接口中执行。如果该线程是其他线程，则其便是在服务中执行您的代码的线程。因此，只有在本地线程访问服务时，您才能完全控制哪些线程在服务中执行（但如果真是这种情况，您根本不应该使用 AIDL，而是应该通过[实现 Binder 类创建接口](#)）。
- 来自远程进程的调用分派自平台在您的自有进程内部维护的线程池。您必须为来自未知线程的多次并发传入调用做好准备。换言之，AIDL 接口的实现必须是完全线程安全实现。
- `oneway` 关键字用于修改远程调用的行为。使用该关键字时，远程调用不会阻塞；它只是发送事务数据并立即返回。接口的实现最终接收此调用时，是以正常远程调用形式将其作为来自 [Binder](#) 线程池的常规调用进行接收。如果 `oneway` 用于本地调用，则不会有任何影响，调用仍是同步调用。

定义 AIDL 接口

您必须使用 Java 编程语言语法在 `.aidl` 文件中定义 AIDL 接口，然后将它保存在托管服务的应用以及任何其他绑定到服务的应用的源代码 (`src/` 目录) 内。

您开发每个包含 `.aidl` 文件的应用时，Android SDK 工具都会生成一个基于该 `.aidl` 文件的 `IBinder` 接口，并将其保存在项目的 `gen/` 目录中。服务必须视情况实现 `IBinder` 接口。然后客户端应用便可绑定到该服务，并调用 `IBinder` 中的方法来执行 IPC。

如需使用 AIDL 创建绑定服务，请执行以下步骤：

1. 创建 `.aidl` 文件

此文件定义带有方法签名的编程接口。

2. 实现接口

Android SDK 工具基于您的 `.aidl` 文件，使用 Java 编程语言生成一个接口。此接口具有一个名为 `Stub` 的内部抽象类，用于扩展 `Binder` 类并实现 AIDL 接口中的方法。您必须扩展 `Stub` 类并实现方法。

3. 向客户端公开该接口

实现 `Service` 并重写 `onBind()` 以返回 `Stub` 类的实现。

注意：在 AIDL 接口首次发布后对其进行的任何更改都必须保持向后兼容性，以避免中断其他应用对您的服务的使用。也就是说，因为必须将您的 `.aidl` 文件复制到其他应用，才能让这些应用访问您的服务的接口，因此您必须保留对原始接口的支持。

1. 创建 `.aidl` 文件

AIDL 使用简单语法，使您能通过可带参数和返回值的一个或多个方法来声明接口。参数和返回值可以是任意类型，甚至可以是其他 AIDL 生成的接口。

您必须使用 Java 编程语言构建 `.aidl` 文件。每个 `.aidl` 文件都必须定义单个接口，并且只需包含接口声明和方法签名。

默认情况下，AIDL 支持下列数据类型：

- Java 编程语言中的所有原语类型（如 `int`、`long`、`char`、`boolean` 等等）
- `String`
- `CharSequence`
- `List`

`List` 中的所有元素都必须是以上列表中支持的数据类型、其他 AIDL 生成的接口或您声明的可打包类型。可选择将 `List` 用作“通用”类（例如，`List<String>`）。另一端实际接收的具体类始终是 `ArrayList`，但生成的方法使用的是 `List` 接口。

- `Map`
- `Map` 中的所有元素都必须是以上列表中支持的数据类型、其他 AIDL 生成的接口或您声明的可打包类型。不支持通用 `Map`（如 `Map<String, Integer>` 形式的 `Map`）。另一端实际接收的具体类始终是 `HashMap`，但生成的方法使用的是 `Map` 接口。

您必须为以上未列出的每个附加类型加入一个 `import` 语句，即使这些类型是在与您的接口相同的软件包中定义。

定义服务接口时，请注意：

- 方法可带零个或多个参数，返回值或空值。
- 所有非原语参数都需要指示数据走向的方向标记。可以是 `in`、`out` 或 `inout`（见以下示例）。

原语默认为 `in`，不能是其他方向。

注意：您应该将方向限定为真正需要的方向，因为编组参数的开销极大。

- `.aidl` 文件中包括的所有代码注释都包含在生成的 `IBinder` 接口中（`import` 和 `package` 语句之前的注释除外）
- 只支持方法；您不能公开 AIDL 中的静态字段。

以下是一个 `.aidl` 文件示例：

```

// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import statements

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service, to do evil things with it. */
    int getPid();

    /** Demonstrates some basic types that you can use as parameters
     * and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat,
                    double aDouble, String aString);
}

```

只需将您的 `.aidl` 文件保存在项目的 `src/` 目录内，当您开发应用时，SDK 工具会在项目的 `gen/` 目录中生成 `IBinder` 接口文件。生成的文件名与 `.aidl` 文件名一致，只是使用了 `.java` 扩展名（例如，`IRemoteService.aidl` 生成的文件名是 `IRemoteService.java`）。

如果您使用 Android Studio，增量编译几乎会立即生成 Binder 类。如果您不使用 Android Studio，则 Gradle 工具会在您下一次开发应用时生成 Binder 类 — 您应该在编写完 `.aidl` 文件后立即用 `gradle assembleDebug`（或 `gradle assembleRelease`）编译项目，以便您的代码能够链接到生成的类。

2. 实现接口

当您开发应用时，Android SDK 工具会生成一个以 `.aidl` 文件命名的 `.java` 接口文件。生成的接口包括一个名为 `Stub` 的子类，这个子类是其父接口（例如，`YourInterface.Stub`）的抽象实现，用于声明 `.aidl` 文件中的所有方法。

注：`Stub` 还定义了几个帮助程序方法，其中最引人关注的是 `asInterface()`，该方法带 `IBinder`（通常便是传递给客户端 `onServiceConnected()` 回调方法的参数）并返回存根接口实例。如需了解如何进行这种转换的更多详细信息，请参见[调用 IPC 方法](#)一节。

如需实现 `.aidl` 生成的接口，请扩展生成的 `Binder` 接口（例如，`YourInterface.Stub`）并实现从 `.aidl` 文件继承的方法。

以下是一个使用匿名实例实现名为 `IRemoteService` 的接口（由以上 `IRemoteService.aidl` 示例定义）的示例：

```

private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
    public int getPid(){
        return Process.myPid();
    }
    public void basicTypes(int anInt, long aLong, boolean aBoolean,
                          float aFloat, double aDouble, String aString) {
        // Does nothing
    }
};

```

现在，`mBinder` 是 `Stub` 类的一个实例（一个 `Binder`），用于定义服务的 RPC 接口。在下一步中，将向客户端公开该实例，以便客户端能与服务进行交互。

在实现 AIDL 接口时应注意遵守以下这几个规则：

- 由于不能保证在主线程上执行传入调用，因此您一开始就需要做好多线程处理准备，并将您的服务正确地编译为线程安全服务。
- 默认情况下，RPC 调用是同步调用。如果您明知服务完成请求的时间不止几毫秒，就不应该从 Activity 的主线程调用服务，因为这样做可能会使应用挂起（Android 可能会显示“Application is Not Responding”对话框）— 您通常应该从客户端内的单独线程调用服务。
- 您引发的任何异常都不会回传给调用方。

3. 向客户端公开该接口

您为服务实现该接口后，就需要向客户端公开该接口，以便客户端进行绑定。要为您的服务公开该接口，请扩展 `Service` 并实现 `onBind()`，以返回一个类实例，这个类实现了生成的 `Stub`（见前文所述）。以下是一个向客户端公开 `IRemoteService` 示例接口的服务示例。

```

public class RemoteService extends Service {
    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // Return the interface
        return mBinder;
    }

    private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
        public int getPid(){
            return Process.myPid();
        }
        public void basicTypes(int anInt, long aLong, boolean aBoolean,
            float aFloat, double aDouble, String aString) {
            // Does nothing
        }
    };
}

```

现在，当客户端（如 Activity）调用 `bindService()` 以连接此服务时，客户端的 `onServiceConnected()` 回调会接收服务的 `onBind()` 方法返回的 `mBinder` 实例。

客户端还必须具有对 interface 类的访问权限，因此如果客户端和服务在不同的应用内，则客户端的应用 `src/` 目录内必须包含 `.aidl` 文件（它生成 `android.os.Binder` 接口 — 为客户端提供对 AIDL 方法的访问权限）的副本。

当客户端在 `onServiceConnected()` 回调中收到 `IBinder` 时，它必须调用 `YourServiceInterface.Stub.asInterface(service)` 以将返回的参数转换成 `YourServiceInterface` 类型。例如：

```

IRemoteService mIRemoteService;
private ServiceConnection mConnection = new ServiceConnection() {
    // Called when the connection with the service is established
    public void onServiceConnected(ComponentName className, IBinder service) {
        // Following the example above for an AIDL interface,
        // this gets an instance of the IRemoteInterface, which we can use to call on the service
        mIRemoteService = IRemoteService.Stub.asInterface(service);
    }

    // Called when the connection with the service disconnects unexpectedly
    public void onServiceDisconnected(ComponentName className) {
        Log.e(TAG, "Service has unexpectedly disconnected");
        mIRemoteService = null;
    }
};

```

如需查看更多示例代码，请参见 `ApiDemos` 中的 `RemoteService.java` 类。

通过 IPC 传递对象

通过 IPC 接口把某个类从一个进程发送到另一个进程是可以实现的。不过，您必须确保该类的代码对 IPC 通道的另一端可用，并且该类必须支持 `Parcelable` 接口。支持 `Parcelable` 接口很重要，因为 Android 系统可通过它将对象分解成可编组到各进程的原语。

如需创建支持 `Parcelable` 协议的类，您必须执行以下操作：

1. 让您的类实现 `Parcelable` 接口。
2. 实现 `writeToParcel`，它会获取对象的当前状态并将其写入 `Parcel`。
3. 为您的类添加一个名为 `CREATOR` 的静态字段，这个字段是一个实现 `Parcelable.Creator` 接口的对象。
4. 最后，创建一个声明可打包类的 `.aidl` 文件（按照下文 `Rect.aidl` 文件所示步骤）。

如果您使用的是自定义编译进程，切勿在您的编译中添加 `.aidl` 文件。此 `.aidl` 文件与 C 语言中的头文件类似，并未编译。

AIDL 在它生成的代码中使用这些方法和字段将您的对象编组和取消编组。

例如，以下这个 `Rect.aidl` 文件可创建一个可打包的 `Rect` 类：

```
package android.graphics;

// Declare Rect so AIDL can find it and knows that it implements
// the parcelable protocol.
parcelable Rect;
```

以下示例展示了 `Rect` 类如何实现 `Parcelable` 协议。

```
import android.os.Parcel;
import android.os.Parcelable;

public final class Rect implements Parcelable {
    public int left;
    public int top;
    public int right;
    public int bottom;

    public static final Parcelable.Creator<Rect> CREATOR = new
Parcelable.Creator<Rect>() {
        public Rect createFromParcel(Parcel in) {
            return new Rect(in);
        }

        public Rect[] newArray(int size) {
            return new Rect[size];
        }
    };

    public Rect() {
    }

    private Rect(Parcel in) {
        readFromParcel(in);
    }

    public void writeToParcel(Parcel out) {
        out.writeInt(left);
        out.writeInt(top);
        out.writeInt(right);
        out.writeInt(bottom);
    }

    public void readFromParcel(Parcel in) {
        left = in.readInt();
        top = in.readInt();
        right = in.readInt();
        bottom = in.readInt();
    }
}
```

`Rect` 类中的编组相当简单。看一看 `Parcel` 上的其他方法，了解您可以向 `Parcel` 写入哪些其他类型的值。

警告：别忘记从其他进程接收数据的安全影响。在本例中，`Rect` 从 `Parcel` 读取四个数字，但要由您来确保无论调用方目的为何这些数字都在相应的可接受值范围内。如需了解有关如何防止应用受到恶意软件侵害、保证应用安全的更多信息，请参见[安全与权限](#)。

调用 IPC 方法

调用类必须执行以下步骤，才能调用使用 AIDL 定义的远程接口：

1. 在项目 `src/` 目录中加入 `.aidl` 文件。
2. 声明一个 `IBinder` 接口实例（基于 AIDL 生成）。

3. 实现 `ServiceConnection`。
4. 调用 `Context.bindService()`，以传入您的 `ServiceConnection` 实现。
5. 在您的 `onServiceConnected()` 实现中，您将收到一个 `IBinder` 实例（名为 `service`）。调用 `YourInterfaceName.Stub.asInterface((IBinder)service)`，以将返回的参数转换为 `YourInterface` 类型。
6. 调用您在接口上定义的方法。您应该始终捕获 `DeadObjectException` 异常，它们是在连接中断时引发的；这将是远程方法引发的唯一异常。
7. 如需断开连接，请使用您的接口实例调用 `Context.unbindService()`。

有关调用 IPC 服务的几点说明：

- 对象是跨进程计数的引用。
- 您可以将匿名对象作为方法参数发送。

如需了解有关绑定到服务的详细信息，请阅读[绑定服务](#)文档。

以下这些示例代码摘自 ApiDemos 项目的远程服务示例代码，展示了如何调用 AIDL 创建的服务。

```
public static class Binding extends Activity {  
    /** The primary interface we will be calling on the service. */  
    IRemoteService mService = null;  
    /** Another interface we use on the service. */  
    ISecondary mSecondaryService = null;  
  
    Button mKillButton;  
    TextView mCallbackText;  
  
    private boolean mIsBound;  
  
    /**  
     * Standard initialization of this activity. Set up the UI, then wait  
     * for the user to poke it before doing anything.  
     */  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.remote_service_binding);  
  
        // Watch for button clicks.  
        Button button = (Button) findViewById(R.id.bind);  
        button.setOnClickListener(mBindListener);  
        button = (Button) findViewById(R.id.unbind);  
        button.setOnClickListener(mUnbindListener);  
        mKillButton = (Button) findViewById(R.id.kill);  
        mKillButton.setOnClickListener(mKillListener);  
        mKillButton.setEnabled(false);  
  
        mCallbackText = (TextView) findViewById(R.id.callback);  
        mCallbackText.setText("Not attached.");  
    }  
  
    /**  
     * Class for interacting with the main interface of the service.  
     */  
    private ServiceConnection mConnection = new ServiceConnection() {  
        public void onServiceConnected(ComponentName className,  
                                      IBinder service) {  
            // This is called when the connection with the service has been  
            // established, giving us the service object we can use to  
            // interact with the service. We are communicating with our  
            // service through an IDL interface, so get a client-side  
            // representation of that from the raw service object.  
            mService = IRemoteService.Stub.asInterface(service);  
            mKillButton.setEnabled(true);  
            mCallbackText.setText("Attached.");  
        }  
    };
```

```

// We want to monitor the service for as long as we are
// connected to it.
try {
    mService.registerCallback(mCallback);
} catch (RemoteException e) {
    // In this case the service has crashed before we could even
    // do anything with it; we can count on soon being
    // disconnected (and then reconnected if it can be restarted)
    // so there is no need to do anything here.
}

// As part of the sample, tell the user what happened.
Toast.makeText(Binding.this, R.string.remote_service_connected,
    Toast.LENGTH_SHORT).show();
}

public void onServiceDisconnected(ComponentName className) {
    // This is called when the connection with the service has been
    // unexpectedly disconnected -- that is, its process crashed.
    mService = null;
    mKillButton.setEnabled(false);
    mCallbackText.setText("Disconnected.");

    // As part of the sample, tell the user what happened.
    Toast.makeText(Binding.this, R.string.remote_service_disconnected,
        Toast.LENGTH_SHORT).show();
}
};

/***
 * Class for interacting with the secondary interface of the service.
 */
private ServiceConnection mSecondaryConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
        IBinder service) {
        // Connecting to a secondary interface is the same as any
        // other interface.
        mSecondaryService = ISecondary.Stub.asInterface(service);
        mKillButton.setEnabled(true);
    }

    public void onServiceDisconnected(ComponentName className) {
        mSecondaryService = null;
        mKillButton.setEnabled(false);
    }
};

private OnClickListener mBindListener = new OnClickListener() {
    public void onClick(View v) {
        // Establish a couple connections with the service, binding
        // by interface names. This allows other applications to be
        // installed that replace the remote service by implementing
        // the same interface.
        Intent intent = new Intent(Binding.this, RemoteService.class);
        intent.setAction(IRemoteService.class.getName());
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
        intent.setAction(ISecondary.class.getName());
        bindService(intent, mSecondaryConnection, Context.BIND_AUTO_CREATE);
        mIsBound = true;
        mCallbackText.setText("Binding.");
    }
};

private OnClickListener mUnbindListener = new OnClickListener() {
    public void onClick(View v) {
        if (mIsBound) {
            // If we have received the service, and hence registered with
            // it, then now is the time to unregister.
            if (mService != null) {
                try {
                    mService.unregisterCallback(mCallback);
                } catch (RemoteException e) {
                    // There is nothing special we need to do if the service

```

```

        // There is nothing special we need to do if the service
        // has crashed.
    }

    // Detach our existing connection.
    unbindService(mConnection);
    unbindService(mSecondaryConnection);
    mKillButton.setEnabled(false);
    mIsBound = false;
    mCallbackText.setText("Unbinding.");
}
}

private OnClickListener mKillListener = new OnClickListener() {
    public void onClick(View v) {
        // To kill the process hosting our service, we need to know its
        // PID. Conveniently our service has a call that will return
        // to us that information.
        if (mSecondaryService != null) {
            try {
                int pid = mSecondaryService.getPid();
                // Note that, though this API allows us to request to
                // kill any process based on its PID, the kernel will
                // still impose standard restrictions on which PIDs you
                // are actually able to kill. Typically this means only
                // the process running your application and any additional
                // processes created by that app as shown here; packages
                // sharing a common UID will also be able to kill each
                // other's processes.
                Process.killProcess(pid);
                mCallbackText.setText("Killed service process.");
            } catch (RemoteException ex) {
                // Recover gracefully from the process hosting the
                // server dying.
                // Just for purposes of the sample, put up a notification.
                Toast.makeText(Binding.this,
                    R.string.remote_call_failed,
                    Toast.LENGTH_SHORT).show();
            }
        }
    }
};

// -----
// Code showing how to deal with callbacks.
// -----


/**
 * This implementation is used to receive callbacks from the remote
 * service.
 */
private IRemoteServiceCallback mCallback = new IRemoteServiceCallback.Stub() {
    /**
     * This is called by the remote service regularly to tell us about
     * new values. Note that IPC calls are dispatched through a thread
     * pool running in each process, so the code executing here will
     * NOT be running in our main thread like most other things -- so,
     * to update the UI, we need to use a Handler to hop over there.
     */
    public void valueChanged(int value) {
        mHandler.sendMessage(mHandler.obtainMessage(BUMP_MSG, value, 0));
    }
};

private static final int BUMP_MSG = 1;

private Handler mHandler = new Handler() {
    @Override public void handleMessage(Message msg) {
        switch (msg.what) {
            case BUMP_MSG:
                mCallbackText.setText("Received from service: " + msg.arg1);
        }
    }
};

```

```
        break;
    default:
        super.handleMessage(msg);
    }
}
};
```



内容提供程序

主题

- › [内容提供程序基础知识](#)
- › [创建内容提供程序](#)
- › [日历提供程序](#)
- › [联系人提供程序](#)

相关示例

- › [联系人管理器应用](#)
- › [“游标（联系人）”](#)
- › [“游标（电话）”](#)
- › [示例同步适配器](#)

内容提供程序管理对结构化数据集的访问。它们封装数据，并提供用于定义数据安全性的机制。内容提供程序是连接一个进程中的数据与另一个进程中运行的代码的标准界面。

如果您想要访问内容提供程序中的数据，可以将应用的 `Context` 中的 `ContentResolver` 对象用作客户端来与提供程序通信。

`ContentResolver` 对象会与提供程序对象（即实现 `ContentProvider` 的类实例）通信。提供程序对象从客户端接收数据请求，执行请求的操作并返回结果。

如果您不打算与其他应用共享数据，则无需开发自己的提供程序。不过，您需要通过自己的提供程序在您自己的应用中提供自定义搜索建议。如果您想将复杂的数据或文件从您的应用复制并粘贴到其他应用中，也需要创建您自己的提供程序。

Android 本身包括的内容提供程序可管理音频、视频、图像和个人联系信息等数据。`android.provider` 软件包参考文档中列出了部分提供程序。任何 Android 应用都可以访问这些提供程序，但会受到某些限制。

以下主题对内容提供程序做了更详尽的描述：

[内容提供程序基础知识](#)

如何访问内容提供程序中以表形式组织的数据。

[创建内容提供程序](#)

如何创建您自己的内容提供程序。

[日历提供程序](#)

如何访问作为 Android 平台一部分的日历提供程序。

[联系人提供程序](#)

如何访问作为 Android 平台一部分的联系人提供程序。

内容提供程序基础知识

本文内容

- › [概览](#)
- › [访问提供程序](#)
- › [内容 URI](#)
- › [从提供程序检索数据](#)
- › [请求读取访问权限](#)
- › [构建查询](#)
- › [显示查询结果](#)
- › [从查询结果中获取数据](#)
- › [内容提供程序权限](#)
- › [插入、更新和删除数据](#)
- › [插入数据](#)
- › [更新数据](#)
- › [删除数据](#)
- › [提供程序数据类型](#)
- › [提供程序访问的替代形式](#)
- › [批量访问](#)
- › [通过 Intent 访问数据](#)
- › [协定类](#)
- › [MIME 类型引用](#)

关键类

- › [ContentProvider](#)
- › [ContentResolver](#)
- › [Cursor](#)
- › [Uri](#)

相关示例

- › [游标（联系人）](#)
- › [游标（电话）](#)

另请参阅

- › [创建内容提供程序](#)
- › [日历提供程序](#)

内容提供程序管理对中央数据存储区的访问。提供程序是 Android 应用的一部分，通常提供自己的 UI 来使用数据。但是，内容提供程序主要旨在供其他应用使用，这些应用使用提供程序客户端对象来访问提供程序。提供程序与提供程序客户端共同提供一致的标准数据界面，该界面还可处理跨进程通信并保护数据访问的安全性。

本主题介绍了以下基础知识：

- 内容提供程序的工作方式。
- 用于从内容提供程序检索数据的 API。
- 用于在内容提供程序中插入、更新或删除数据的 API。

- 其他有助于使用提供程序的 API 功能。

概览

内容提供程序以一个或多个表（与在关系型数据库中找到的表类似）的形式将数据呈现给外部应用。行表示提供程序收集的某种数据类型的实例，行中的每个列表示为实例收集的每条数据。

例如，Android 平台的内置提供程序之一是用户字典，它会存储用户想要保存的非标准字词的拼写。表 1 描述了数据在此提供程序表中的显示情况：

表 1. 用户字典示例表格。

字词	应用 id	频率	语言区域	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

在表 1 中，每行表示可能无法在标准词典中找到的字词实例。每列表示该字词的某些数据，如该字词首次出现时的语言区域。列标题是存储在提供程序中的列名称。要引用行的语言区域，需要引用其 `locale` 列。对于此提供程序，`_ID` 列充当由提供程序自动维护的“主键”列。

注：提供程序无需具有主键，也无需将 `_ID` 用作其主键的列名称（如果存在主键）。但是，如果您要将来自提供程序的数据与 `ListView` 绑定，则其中一个列名称必须是 `_ID`。[显示查询结果](#)部分详细说明了此要求。

访问提供程序

应用从具有 `ContentResolver` 客户端对象的内容提供程序访问数据。此对象具有调用提供程序对象（`ContentProvider` 的某个具体子类的实例）中同名方法的方法。`ContentResolver` 方法可提供持续存储的基本“CRUD”（创建、检索、更新和删除）功能。

客户端应用进程中的 `ContentResolver` 对象和拥有提供程序的应用中的 `ContentProvider` 对象可自动处理跨进程通信。`ContentProvider` 还可充当其数据存储区和表格形式的数据外部显示之间的抽象层。

注：要访问提供程序，您的应用通常需要在其清单文件中请求特定权限。[内容提供程序权限](#)部分详细介绍了此内容。

例如，要从用户字典提供程序中获取字词及其语言区域的列表，则需调用 `ContentResolver.query()`。`query()` 方法会调用用户字典提供程序所定义的 `ContentProvider.query()` 方法。以下代码行显示了 `ContentResolver.query()` 调用：

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the words table
    mProjection, // The columns to return for each row
    mSelectionClause, // Selection criteria
    mSelectionArgs, // Selection criteria
    mSortOrder); // The sort order for the returned rows
```

表 2 显示了 `query(Uri,projection,selection,selectionArgs,sortOrder)` 的参数如何匹配 SQL SELECT 语句：

表 2. `Query()` 与 SQL 查询对比。

query() 参数	SELECT 关键字/参数	说明
<code>Uri</code>	<code>FROM table_name</code>	<code>Uri</code> 映射至提供程序中名为 <code>table_name</code> 的表。
<code>projection</code>	<code>col,col,col,...</code>	<code>projection</code> 是应该为检索到的每个行包含的列的数组。
<code>selection</code>	<code>WHERE col = value</code>	<code>selection</code> 会指定选择行的条件。
<code>selectionArgs</code>	(没有完全等效项。)	

	选择参数会替换选择子句中 ? 占位符。)	
sortOrder	ORDER BY col, col, ...	sortOrder 指定行在返回的 Cursor 中的显示顺序。

内容 URI

内容 URI 是用于在提供程序中标识数据的 URI。内容 URI 包括整个提供程序的符号名称（其授权）和一个指向表的名称（路径）。当您调用客户端方法来访问提供程序中的表时，该表的内容 URI 将是其参数之一。

在前面的代码行中，常量 `CONTENT_URI` 包含用户字典的“字词”表的内容 URI。`ContentResolver` 对象会分析出 URI 的授权，并通过将该授权与已知提供程序的系统表进行比较，来“解析”提供程序。然后，`ContentResolver` 可以将查询参数分派给正确的提供程序。

`ContentProvider` 使用内容 URI 的路径部分来选择要访问的表。提供程序通常会为其公开的每个表显示一条路径。

在前面的代码行中，“字词”表的完整 URI 是：

```
content://user_dictionary/words
```

其中，`user_dictionary` 字符串是提供程序的授权，`words` 字符串是表的路径。字符串 `content://`（架构）始终显示，并将此标识为内容 URI。

许多提供程序都允许您通过将 ID 值追加到 URI 末尾来访问表中的单个行。例如，要从用户字典中检索 `_ID` 为 4 的行，则可使用此内容 URI：

```
Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI, 4);
```

在检索到一组行后想要更新或删除其中某一行时通常会用到 ID 值。

注：`Uri` 和 `Uri.Builder` 类包含根据字符串构建格式规范的 URI 对象的便利方法。`ContentUris` 包含一些可以将 ID 值轻松追加到 URI 后的方法。前面的代码段就是使用 `withAppendedId()` 将 ID 追加到 UserDictionary 内容 URI 后。

从提供程序检索数据

本节将以用户字典提供程序为例，介绍如何从提供程序中检索数据。

为了明确进行说明，本节中的代码段将在“UI 线程”上调用 `ContentResolver.query()`。但在实际代码中，您应该在单独线程上异步执行查询。执行此操作的方式之一是使用 `CursorLoader` 类，[加载器](#)指南中对此有更为详细的介绍。此外，前述代码行只是片段；它们不会显示整个应用。

要从提供程序中检索数据，请按照以下基本步骤执行操作：

1. 请求对提供程序的读取访问权限。
2. 定义将查询发送至提供程序的代码。

请求读取访问权限

要从提供程序检索数据，您的应用需要具备对提供程序的“读取访问”权限。您无法在运行时请求此权限；相反，您需要使用`<uses-permission>`元素和提供程序定义的准确权限名称，在清单文件中指明您需要此权限。在您的清单文件中指定此元素后，您将有效地为应用“请求”此权限。用户安装您的应用时，会隐式授予允许此请求。

要找出您正在使用的提供程序的读取访问权限的准确名称，以及提供程序使用的其他访问权限的名称，请查看提供程序的文档。

[内容提供程序权限](#)部分详细介绍了权限在访问提供程序过程中的作用。

用户字典提供程序在其清单文件中定义了权限 `android.permission.READ_USER_DICTIONARY`，因此希望从提供程序中进行读取的应用必需请求此权限。

构建查询

从提供程序中检索数据的下一步是构建查询。第一个代码段定义某些用于访问用户字典提供程序的变量：

```
// A "projection" defines the columns that will be returned for each row
String[] mProjection =
{
    UserDictionary.Words._ID,      // Contract class constant for the _ID column name
    UserDictionary.Words.WORD,     // Contract class constant for the word column name
    UserDictionary.Words.LOCALE   // Contract class constant for the locale column name
};

// Defines a string to contain the selection clause
String mSelectionClause = null;

// Initializes an array to contain selection arguments
String[] mSelectionArgs = {"");
```

下一个代码段以用户字典提供程序为例，显示了如何使用 `ContentResolver.query()`。 提供程序客户端查询与 SQL 查询类似，并且包含一组要返回的列、一组选择条件和排序顺序。

查询应该返回的列集被称为**投影**（变量 `mProjection`）。

用于指定要检索的行的表达式分割为选择子句和选择参数。 选择子句是逻辑和布尔表达式、列名称和值（变量 `mSelectionClause`）的组合。 如果您指定了可替换参数 `?` 而非值，则查询方法会从选择参数数组（变量 `mSelectionArgs`）中检索值。

在下一个代码段中，如果用户未输入字词，则选择子句将设置为 `null`，而且查询会返回提供程序中的所有字词。 如果用户输入了字词，选择子句将设置为 `UserDictionary.Words.WORD + " = ?"` 且选择参数数组的第一个元素将设置为用户输入的字词。

```

/*
 * This defines a one-element String array to contain the selection argument.
 */
String[] mSelectionArgs = {"\""}; // Gets a word from the UI
mSearchString = mSearchWord.getText().toString(); // Remember to insert code here to check for invalid or malicious input.

// If the word is the empty string, gets everything
if (TextUtils.isEmpty(mSearchString)) {
    // Setting the selection clause to null will return all words
    mSelectionClause = null;
    mSelectionArgs[0] = "";
}

} else {
    // Constructs a selection clause that matches the word that the user entered.
    mSelectionClause = UserDictionary.Words.WORD + " = ?"; // Moves the user's input string to the selection arguments.
    mSelectionArgs[0] = mSearchString;
}

// Does a query against the table and returns a Cursor object
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the words table
    mProjection, // The columns to return for each row
    mSelectionClause, // Either null, or the word the user entered
    mSelectionArgs, // Either empty, or the string the user entered
    mSortOrder); // The sort order for the returned rows

// Some providers return null if an error occurs, others throw an exception
if (null == mCursor) {
    /*
     * Insert code here to handle the error. Be sure not to use the cursor! You may want to
     * call android.util.Log.e() to log this error.
     */
}

// If the Cursor is empty, the provider found no matches
} else if (mCursor.getCount() < 1) {

    /*
     * Insert code here to notify the user that the search was unsuccessful. This isn't necessarily
     * an error. You may want to offer the user the option to insert a new row, or re-type the
     * search term.
     */
}

} else {
    // Insert code here to do something with the results
}

```

此查询与 SQL 语句相似：

```
SELECT _ID, word, locale FROM words WHERE word = <userinput> ORDER BY word ASC;
```

在此 SQL 语句中，会使用实际的列名称而非协定类常量。

防止恶意输入

如果内容提供程序管理的数据位于 SQL 数据库中，将不受信任的外部数据包括在原始 SQL 语句中可能会导致 SQL 注入。

考虑此选择子句：

```
// Constructs a selection clause by concatenating the user's input to the column name
String mSelectionClause = "var = " + mUserInput;
```

如果您执行此操作，则会允许用户将恶意 SQL 串连到 SQL 语句上。例如，用户可以为 `mUserInput` 输入“`nothing; DROP TABLE *;`”，这会生成选择子句 `var = nothing; DROP TABLE *;`。由于选择子句是作为 SQL 语句处理，因此这可能会导致提供程序擦除基础 SQLite 数据库中的所有表（除非提供程序设置为可捕获 [SQL 注入尝试](#)）。

要避免此问题，可使用一个用于将 `?` 作为可替换参数的选择子句以及一个单独的选择参数数组。执行此操作时，用户输入直接受查询约束，而不解释为 SQL 语句的一部分。由于用户输入未作为 SQL 处理，因此无法注入恶意 SQL。请使用此选择子句，而不要使用串连来包括用户输入：

```
// Constructs a selection clause with a replaceable parameter
String mSelectionClause = "var = ?";
```

按如下所示设置选择参数数组：

```
// Defines an array to contain the selection arguments
String[] selectionArgs = {"");
```

按如下所示将值置于选择参数数组中：

```
// Sets the selection argument to the user's input
selectionArgs[0] = mUserInput;
```

一个用于将 `?` 用作可替换参数的选择子句和一个选择参数数组是指定选择的首选方式，即使提供程序并未基于 SQL 数据库。

显示查询结果

`ContentResolver.query()` 客户端方法始终会返回符合以下条件的 `Cursor`：包含查询的投影为匹配查询选择条件的行指定的列。`Cursor` 对象为其包含的行和列提供随机读取访问权限。通过使用 `Cursor` 方法，您可以循环访问结果中的行、确定每个列的数据类型、从列中获取数据，并检查结果的其他属性。某些 `Cursor` 实现会在提供程序的数据发生更改时自动更新对象和/或在 `Cursor` 更改时触发观察程序对象中的方法。

注：提供程序可能会根据发出查询的对象的性质来限制对列的访问。例如，联系人提供程序会限定只有同步适配器才能访问某些列，因此不会将它们返回至 `Activity` 或服务。

如果没有与选择条件匹配的行，则提供程序会返回 `Cursor.getCount()` 为 0（空游标）的 `cursor` 对象。

如果出现内部错误，查询结果将取决于具体的提供程序。它可能会选择返回 `null`，或引发 `Exception`。

由于 `Cursor` 是行“列表”，因此显示 `Cursor` 内容的良好方式是通过 `SimpleCursorAdapter` 将其与 `ListView` 关联。

以下代码段将延续上一代码段的代码。它会创建一个包含由查询检索到的 `Cursor` 的 `SimpleCursorAdapter` 对象，并将此对象设置为 `ListView` 的适配器：

```

// Defines a list of columns to retrieve from the Cursor and load into an output row
String[] mWordListColumns =
{
    UserDictionary.Words.WORD, // Contract class constant containing the word column name
    UserDictionary.Words.LOCALE // Contract class constant containing the locale column name
};

// Defines a list of View IDs that will receive the Cursor columns for each row
int[] mWordListItems = { R.id.dictWord, R.id.locale};

// Creates a new SimpleCursorAdapter
mCursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(), // The application's Context object
    R.layout.wordlistrow, // A layout in XML for one row in the ListView
    mCursor, // The result from the query
    mWordListColumns, // A string array of column names in the cursor
    mWordListItems, // An integer array of view IDs in the row layout
    0); // Flags (usually none are needed)

// Sets the adapter for the ListView
mWordList.setAdapter(mCursorAdapter);

```

注：要通过 `Cursor` 支持 `ListView`，游标必需包含名为 `_ID` 的列。正因如此，前文显示的查询会为“字词”表检索 `_ID` 列，即使 `ListView` 未显示该列。此限制也解释了为什么大多数提供程序的每个表都具有 `_ID` 列。

从查询结果中获取数据

您可以将查询结果用于其他任务，而不是仅显示它们。例如，您可以从用户字典中检索拼写，然后在其他提供程序中查找它们。要执行此操作，您需要在 `Cursor` 中循环访问行：

```

// Determine the column index of the column named "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
 * Only executes if the cursor is valid. The User Dictionary Provider returns null if
 * an internal error occurs. Other providers may throw an Exception instead of returning null.
 */

if (mCursor != null) {
    /*
     * Moves to the next row in the cursor. Before the first movement in the cursor, the
     * "row pointer" is -1, and if you try to retrieve data at that position you will get an
     * exception.
     */
    while (mCursor.moveToNext()) {

        // Gets the value from the column.
        newWord = mCursor.getString(index);

        // Insert code here to process the retrieved word.

        ...

        // end of while loop
    }
} else {
    // Insert code here to report an error if the cursor is null or the provider threw an exception.
}

```

`Cursor` 实现包含多个用于从对象中检索不同类型的数据的“获取”方法。例如，上一个代码段使用 `getString()`。它们还具有 `getType()` 方法，该方法会返回指示列的数据类型的值。

内容提供程序权限

提供程序的应用可以指定其他应用访问提供程序的数据所必需的权限。这些权限可确保用户了解应用将尝试访问的数据。根据提供程序的要求，其他应用会请求它们访问提供程序所需的权限。最终用户会在安装应用时看到所请求的权限。

如果提供程序的应用未指定任何权限，则其他应用将无权访问提供程序的数据。但是，无论指定权限为何，提供程序的应用中的组件始终具有完整的读取和写入访问权限。

如前所述，用户字典提供程序需要 `android.permission.READ_USER_DICTIONARY` 权限才能从中检索数据。提供程序具有用于插入、更新或删除数据的单独 `android.permission.WRITE_USER_DICTIONARY` 权限。

要获取访问提供程序所需的权限，应用将通过其清单文件中的 `<uses-permission>` 元素来请求这些权限。Android 软件包管理器安装应用时，用户必须批准该应用请求的所有权限。如果用户批准所有权限，软件包管理器将继续安装；如果用户未批准这些权限，软件包管理器将中止安装。

以下 `<uses-permission>` 元素会请求对用户字典提供程序的读取访问权限：

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

[安全与权限](#)指南中详细介绍了权限对提供程序访问的影响。

插入、更新和删除数据

与从提供程序检索数据的方式相同，也可以通过提供程序客户端和提供程序 `ContentProvider` 之间的交互来修改数据。您通过传递到 `ContentProvider` 的对应方法的参数来调用 `ContentResolver` 方法。提供程序和提供程序客户端会自动处理安全性和跨进程通信。

插入数据

要将数据插入提供程序，可调用 `ContentResolver.insert()` 方法。此方法会在提供程序中插入新行并为该行返回内容 URI。此代码段显示如何将新字词插入用户字典提供程序：

```
// Defines a new Uri object that receives the result of the insertion
Uri mNewUri;

...

// Defines an object to contain the new values to insert
ContentValues mNewValues = new ContentValues();

/*
 * Sets the values of each column and inserts the word. The arguments to the "put"
 * method are "column name" and "value"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI, // the user dictionary content URI
    mNewValues // the values to insert
);
```

新行的数据会进入单个 `ContentValues` 对象中，该对象在形式上与单行游标类似。此对象中的列不需要具有相同的数据类型，如果您不想指定值，则可以使用 `ContentValues.putNull()` 将列设置为 `null`。

代码段不会添加 `_ID` 列，因为系统会自动维护此列。提供程序会向添加的每个行分配唯一的 `_ID` 值。通常，提供程序会将此值用作表的主键。

`newUri` 中返回的内容 URI 会按照以下格式标识新添加的行：

```
content://user_dictionary/words/<id_value>
```

`<id_value>` 是新行的 `_ID` 内容。大多数提供程序都能自动检测这种格式的内容 URI，然后在该特定行上执行请求的操作。

要从返回的 `Uri` 中获取 `_ID` 的值，请调用 `ContentUris.parseLong()`。

更新数据

要更新行，请按照执行插入的方式使用具有更新值的 `ContentValues` 对象，并按照执行查询的方式使用选择条件。您使用的客户端方法是 `ContentResolver.update()`。您只需将值添加至您要更新的列的 `ContentValues` 对象。如果您要清除列的内容，请将值设置为 `null`。

以下代码段会将语言区域具有语言“en”的所有行的语言区域更改为 `null`。返回值是已更新的行数：

```
// Defines an object to contain the updated values
ContentValues mUpdateValues = new ContentValues();

// Defines selection criteria for the rows you want to update
String mSelectionClause = UserDictionary.Words.LOCALE + " LIKE ?";
String[] mSelectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int mRowsUpdated = 0;

...

/*
 * Sets the updated value and updates the selected words.
 */
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content URI
    mUpdateValues, // the columns to update
    mSelectionClause, // the column to select on
    mSelectionArgs // the value to compare to
);
```

您还应该在调用 `ContentResolver.update()` 时检查用户输入。如需了解有关此内容的更多详情，请阅读[防止恶意输入部分](#)。

删除数据

删除行与检索行数据类似：为要删除的行指定选择条件，客户端方法会返回已删除的行数。以下代码段会删除应用 ID 与“用户”匹配的行。该方法会返回已删除的行数。

```
// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};

// Defines a variable to contain the number of rows deleted
int mRowsDeleted = 0;

...

// Deletes the words that match the selection criteria
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content URI
    mSelectionClause, // the column to select on
    mSelectionArgs // the value to compare to
);
```

您还应该在调用 `ContentResolver.delete()` 时检查用户输入。如需了解有关此内容的更多详情，请阅读[防止恶意输入部分](#)。

提供程序数据类型

内容提供程序可以提供多种不同的数据类型。用户字典提供程序仅提供文本，但提供程序也能提供以下格式：

- 整型

- 长整型（长）
- 浮点型
- 长浮点型（双倍）

提供程序经常使用的另一种数据类型是作为 64KB 字节的数组实施的二进制大型对象 (BLOB)。您可以通过查看 [Cursor](#) 类“获取”方法看到可用数据类型。

提供程序文档中通常都列出了其每个列的数据类型。用户字典提供程序的数据类型列在其协定类 [UserDictionary.Words](#) 参考文档中（[协定类](#)部分对协定类进行了介绍）。您也可以通过调用 [Cursor.getType\(\)](#) 来确定数据类型。

提供程序还会维护其定义的每个内容 URI 的 MIME（多用途互联网邮件扩展）数据类型信息。您可以使用 MIME 类型信息查明应用是否可以处理提供程序提供的数据，或根据 MIME 类型选择处理类型。在使用包含复杂数据结构或文件的提供程序时，通常需要 MIME 类型。例如，联系人提供程序中的 [ContactsContract.Data](#) 表会使用 MIME 类型来标记每行中存储的联系人数据类型。要获取与内容 URI 对应的 MIME 类型，请调用 [ContentResolver.getType\(\)](#)。

[MIME 类型引用](#)部分介绍了标准和自定义 MIME 类型的语法。

提供程序访问的替代形式

提供程序访问的三种替代形式在应用开发过程中十分重要：

- **批量访问**：您可以通过 [ContentProviderOperation](#) 类中的方法创建一批访问调用，然后通过 [ContentResolver.applyBatch\(\)](#) 应用它们。
- **异步查询**：您应该在单独线程中执行查询。执行此操作的方式之一是使用 [CursorLoader](#) 对象。[加载器](#)指南中的示例展示了如何执行此操作。
- **通过 Intent 访问数据**：尽管您无法直接向提供程序发送 Intent，但可以向提供程序的应用发送 Intent，后者通常具有修改提供程序数据的最佳配置。

下文将介绍通过 Intent 进行的批量访问和修改。

批量访问

批量访问提供程序适用于插入大量行，或通过同一方法调用在多个表中插入行，或者通常用于跨进程界限将一组操作作为事务处理（原子操作）执行。

要在“批量模式”下访问提供程序，您可以创建 [ContentProviderOperation](#) 对象数组，然后使用 [ContentResolver.applyBatch\(\)](#) 将其分派给内容提供程序。您需将内容提供程序的授权传递给此方法，而不是特定内容 URI。这样可使数组中的每个 [ContentProviderOperation](#) 对象都能适用于其他表。调用 [ContentResolver.applyBatch\(\)](#) 会返回结果数组。

[ContactsContract.RawContacts](#) 协定类的说明包括展示批量注入的代码段。[联系人管理器](#)示例应用包含在其 [ContactAdder.java](#) 源文件中进行批量访问的示例。

通过 Intent 访问数据

Intent 可以提供对内容提供程序的间接访问。即使您的应用不具备访问权限，您也可以通过以下方式允许用户访问提供程序中的数据：从具有权限的应用中获取回结果 Intent，或者通过激活具有权限的应用，然后让用户在其中工作。

通过临时权限获取访问权限

即使您没有适当的访问权限，也可以通过以下方式访问内容提供程序中的数据：将 Intent 发送至具有权限的应用，然后接收回包含“URI”权限的结果 Intent。这些是特定内容 URI 的权限，将持续至接收该权限的 Activity 结束。具有永久权限的应用将通过在结果 Intent 中设置标志来授予临时权限：

- **读取权限**：[FLAG_GRANT_READ_URI_PERMISSION](#)
- **写入权限**：[FLAG_GRANT_WRITE_URI_PERMISSION](#)

使用帮助程序应用显示数据

如果您的应用具有访问权限，您可能仍想使用 Intent 对象在其他应用中显示数据。例如，日历应用接受

[ACTION_VIEW](#) Intent 对象，用于显示特定的日期或事件。这样，您可以在不创建自己的 UI 的情况下显示日历信息。如需了解有关此功能的更多信息，请参见[日历提供程序](#)指南。

您向其发送 Intent 对象的应用不必是与提供程序关联的应用。例如，您可以从联系人提供程序中检索联系人，然后将包含联系人图像的内容 URI 的 [ACTION_VIEW](#) Intent 发送至图像查看器。

注：这些标志不会为其授权包含在内容 URI 中的提供程序提供常规的读取或写入访问权限。访问权限仅适用于 URI 本身。

提供程序使用 `<provider>` 元素的 `android:grantUriPermission` 属性以及 `<provider>` 元素的 `<grant-uri-permission>` 子元素在其清单文件中定义内容 URI 的 URI 权限。[安全与权限](#)指南中“URI 权限”部分更加详细地说明了 URI 权限机制。

例如，即使您没有 `READ_CONTACTS` 权限，也可以在联系人提供程序中检索联系人的数据。您可能希望在向联系人发送电子生日祝福的应用中执行此操作。您更愿意让用户控制应用所使用的联系人，而不是请求 `READ_CONTACTS`，让您能够访问用户的所有联系人及其信息。要执行此操作，您需要使用以下进程：

1. 您的应用会使用方法 `startActivityForResult()` 发送包含操作 `ACTION_PICK` 和“联系人”MIME 类型 `CONTENT_ITEM_TYPE` 的 Intent 对象。
2. 由于此 Intent 与“联系人”应用的“选择”Activity 的 Intent 过滤器相匹配，因此 Activity 会显示在前台。
3. 在选择 Activity 中，用户选择要更新的联系人。发生此情况时，选择 Activity 会调用 `setResult(resultcode, intent)` 以设置用于返回至应用的 Intent。Intent 包含用户选择的联系人的内容 URI，以及“extra”标志 `FLAG_GRANT_READ_URI_PERMISSION`。这些标志会为您的应用授予读取内容 URI 所指向联系人数据的 URI 权限。然后，选择 Activity 会调用 `finish()`，将控制权交还给您的应用。
4. 您的 Activity 会返回至前台，系统会调用您的 Activity 的 `onActivityResult()` 方法。此方法会收到“联系人”应用中选择 Activity 所创建的结果 Intent。
5. 通过来自结果 Intent 的内容 URI，您可以读取来自联系人提供程序的联系人数据，即使您未在清单文件中请求对该提供程序的永久读取访问权限。您可以获取联系人的生日信息或其电子邮件地址，然后发送电子祝福。

使用其他应用

允许用户修改您无权访问的数据的简单方法是激活具有权限的应用，让用户在其中执行工作。

例如，日历应用接受 `ACTION_INSERT` Intent 对象，这让您可以在应用的插入 UI。您可以在此 Intent（应用将使用该 Intent 来预先填充 UI）中传递“extra”数据。由于定期事件具有复杂的语法，因此将事件插入日历提供程序的首选方式是激活具有 `ACTION_INSERT` 的日历应用，然后让用户在其中插入事件。

协定类

协定类帮助应用使用内容 URI、列名称、Intent 操作以及内容提供程序的其他功能的常量。协定类未自动包含在提供程序中；提供程序的开发者需要定义它们，然后使其可用于其他开发者。Android 平台中包含的许多提供程序都在软件包 `android.provider` 中具有对应的协定类。

例如，用户字典提供程序具有包含内容 URI 和列名称常量的协定类 `UserDictionary`。“字词”表的内容 URI 在常量 `UserDictionary.Words.CONTENT_URI` 中定义。`UserDictionary.Words` 类也包含列名称常量，本指南的示例代码段中就使用了该常量。例如，查询投影可以定义为：

```
String[] mProjection =
{
    UserDictionary.Words._ID,
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
};
```

联系人提供程序的 `ContactsContract` 也是一个协定类。此类的参考文档包括示例代码段。其子类之一 `ContactsContract.Intents.Insert` 是包含 Intent 和 Intent 数据的协定类。

MIME 类型引用

内容提供程序可以返回标准 MIME 媒体类型和/或自定义 MIME 类型字符串。

MIME 类型具有格式

`type/subtype`

例如，众所周知的 MIME 类型 `text/html` 具有 `text` 类型和 `html` 子类型。如果提供程序为 URI 返回此类型，则意味着使用该 URI 的查询会

返回包含 HTML 标记的文本。

自定义 MIME 类型字符串（也称为“特定于供应商”的 MIME 类型）具有更加复杂的类型和子类型值。类型值始终为

```
vnd.android.cursor.dir
```

(多行) 或

```
vnd.android.cursor.item
```

(单行)。

子类型特定于提供程序。Android 内置提供程序通常具有简单的子类型。例如，当“通讯录”应用为电话号码创建行时，它会在行中设置以下 MIME 类型：

```
vnd.android.cursor.item/phone_v2
```

请注意，子类型值只是 phone_v2。

其他提供程序开发者可能会根据提供程序的授权和表名称创建自己的子类型模式。例如，假设提供程序包含列车时刻表。提供程序的授权是 com.example.trains，并包含表 Line1、Line2 和 Line3。在响应表 Line1 的内容 URI

```
content://com.example.trains/Line1
```

时，提供程序会返回 MIME 类型

```
vnd.android.cursor.dir/vnd.example.line1
```

在响应表 Line2 中第 5 行的内容 URI

```
content://com.example.trains/Line2/5
```

时，提供程序会返回 MIME 类型

```
vnd.android.cursor.item/vnd.example.line2
```

大多数内容提供程序都会为其使用的 MIME 类型定义协定类常量。例如，联系人提供程序协定类 [ContactsContract.RawContacts](#) 会为单个原始联系人行的 MIME 类型定义常量 [CONTENT_ITEM_TYPE](#)。

[内容 URI](#) 部分介绍了单个行的内容 URI。



创建内容提供程序

本文内容

- › [设计数据存储](#)
- › [设计内容 URI](#)
- › [实现 ContentProvider 类](#)
 - › [必需方法](#)
 - › [实现 query\(\) 方法](#)
 - › [实现 insert\(\) 方法](#)
 - › [实现 delete\(\) 方法](#)
 - › [实现 update\(\) 方法](#)
 - › [实现 onCreate\(\) 方法](#)
- › [实现内容提供程序 MIME 类型](#)
 - › [表的 MIME 类型](#)
 - › [文件的 MIME 类型](#)
- › [实现协定类](#)
- › [实现内容提供程序权限](#)
- › [<Provider> 元素](#)
- › [Intent 和数据访问](#)

关键类

- › [ContentProvider](#)
- › [Cursor](#)
- › [Uri](#)

相关示例

- › [记事本示例应用](#)

另请参阅

- › [内容提供程序基础知识](#)
- › [日历提供程序](#)

内容提供程序管理对中央数据存储区的访问。您将提供程序作为 Android 应用中的一个或多个类（连同清单文件中的元素）实现。其中一个类会实现子类 [ContentProvider](#)，即您的提供程序与其他应用之间的接口。尽管内容提供程序旨在向其他应用提供数据，但您的应用中必定有这样一些 Activity，它们允许用户查询和修改由提供程序管理的数据。

本主题的其余部分列出了开发内容提供程序的基本步骤和需要使用的 API。

着手开发前的准备工作

请在着手开发提供程序之前执行以下操作：

1. **决定您是否需要内容提供程序。**如果您想提供下列一项或多项功能，则需要开发内容提供程序：

- 您想为其他应用提供复杂的数据或文件
- 您想允许用户将复杂的数据从您的应用复制到其他应用中
- 您想使用搜索框架提供自定义搜索建议

如果完全是在您自己的应用中使用，则不需要提供程序即可使用 SQLite 数据库。

2. 如果您尚未完成此项操作，请阅读[内容提供程序基础知识](#)主题，了解有关提供程序的详情。

接下来，请按照以下步骤开发您的提供程序：

1. 为您的数据设计原始存储。内容提供程序以两种方式提供数据：

文件数据

通常存储在文件中的数据，如照片、音频或视频。将文件存储在您的应用的私有空间内。您的提供程序可以应其他应用发出的文件请求提供文件句柄。

“结构化”数据

通常存储在数据库、数组或类似结构中的数据。以兼容行列表的形式存储数据。行表示实体，如人员或库存项目。列表示实体的某项数据，如人员的姓名或商品的价格。此类数据通常存储在 SQLite 数据库中，但您可以使用任何类型的持久存储。如需了解有关 Android 系统中提供的存储类型的更多信息，请参阅[设计数据存储](#)部分。

2. 定义 [ContentProvider](#) 类及其所需方法的具体实现。此类是您的数据与 Android 系统其余部分之间的接口。如需了解有关此类的详细信息，请参阅[实现 ContentProvider 类](#)部分。
3. 定义提供程序的授权字符串、其内容 URI 以及列名称。如果您想让提供程序的应用处理 Intent，则还要定义 Intent 操作、Extra 数据以及标志。此外，还要定义想要访问您的数据的应用必须具备的权限。您应该考虑在一个单独的协定类中将所有这些值定义为常量；以后您可以将此类公开给其他开发者。如需了解有关内容 URI 的详细信息，请参阅[设计内容 URI](#) 部分。如需了解有关 Intent 的详细信息，请参阅[Intent 和数据访问](#)部分。
4. 添加其他可选部分，如示例数据或可以在提供程序与云数据之间同步数据的 [AbstractThreadingSyncAdapter](#) 实现。

设计数据存储

内容提供程序是用于访问以结构化格式保存的数据的接口。在您创建该接口之前，必须决定如何存储数据。您可以按自己的喜好以任何形式存储数据，然后根据需要设计读写数据的接口。

以下是 Android 中提供的一些数据存储技术：

- Android 系统包括一个 SQLite 数据库 API，Android 自己的提供程序使用它来存储面向表的数据。[SQLiteOpenHelper](#) 类可帮助您创建数据库，[SQLiteDatabase](#) 类是用于访问数据库的基类。
请记住，您不必使用数据库来实现存储区。提供程序在外部表现为一组表，与关系型数据库类似，但这并不是对提供程序内部实现的要求；
- 为了存储文件数据，Android 提供了各种面向文件的 API。如需了解有关文件存储的更多信息，请阅读[数据存储](#)主题。如果您要设计提供媒体相关数据（如音乐或视频）的提供程序，则可开发一个合并了表数据和文件的提供程序。
- 要想使用基于网络的数据，请使用 [java.net](#) 和 [android.net](#) 中的类。您也可以将基于网络的数据与本地数据存储（如数据库）同步，然后以表或文件的形式提供数据。[示例同步适配器](#)示例应用展示了这类同步。

数据设计考虑事项

以下是一些设计提供程序数据结构的技巧：

- 表数据应始终具有一个“主键”列，提供程序将其作为与每行对应的唯一数字值加以维护。您可以使用此值将该行链接到其他表中的相关行（将其用作“外键”）。尽管您可以为此列使用任何名称，但使用 [BaseColumns._ID](#) 是最佳选择，因为将提供程序查询的结果链接到 [ListView](#) 的条件是，检索到的其中一个列的名称必须是 [_ID](#)；
- 如果您想提供位图图像或其他非常庞大的文件导向型数据，请将数据存储在一个文件中，然后间接提供这些数据，而不是直接将其存储在表中。如果您执行了此操作，则需要告知提供程序的用户，他们需要使用 [ContentResolver](#) 文件方法来访问数据；
- 使用二进制大型对象 (BLOB) 数据类型存储大小或结构会发生变化的数据。例如，您可以使用 BLOB 列来存储[协议缓冲区](#)或[JSON 结构](#)。您也可以使用 BLOB 来实现独立于架构的表。在这类表中，您需要以 BLOB 形式定义一个主键列、一个 MIME 类型列以及一个或多个通用列。这些 BLOB 列中数据的含义通过 MIME 类型列中的值指示。这样一来，您就可以在同一个表中存储不同类型的行。举例来说，联系人提供程序的“数据”表 [ContactsContract.Data](#) 便是一个独立于架构的表。

设计内容 URI

内容 URI 是用于在提供程序中标识数据的 URI。内容 URI 包括整个提供程序的符号名称（**其授权**）和一个指向表或文件的名称（**路径**）。可选 ID 部分指向表中的单个行。**ContentProvider** 的每一个数据访问方法都将内容 URI 作为参数；您可以利用这一点确定要访问的表、行或文件。

[内容提供程序基础知识](#) 主题中描述了内容 URI 的基础知识。

设计授权

提供程序通常具有单一授权，该授权充当其 Android 内部名称。为避免与其他提供程序发生冲突，您应该使用互联网网域所有权（反向）作为提供程序授权的基础。由于此建议也适用于 Android 软件包名称，因此您可以将提供程序授权定义为包含该提供程序的软件包名称的扩展名。例如，如果您的 Android 软件包名称为 `com.example.<appname>`，则应为提供程序提供 `com.example.<appname>.provider` 授权。

设计路径结构

开发者通常通过追加指向单个表的路径来根据权限创建内容 URI。例如，如果您有两个表：`table1` 和 `table2`，则可以通过合并上一示例中的权限来生成内容 URI `com.example.<appname>.provider/table1` 和 `com.example.<appname>.provider/table2`。路径并不限于单个片段，也无需为每一级路径都创建一个表。

处理内容 URI ID

按照惯例，提供程序通过接受末尾具有行所对应 ID 值的内容 URI 来提供对表中单个行的访问。同样按照惯例，提供程序会将该 ID 值与表的 `_ID` 列进行匹配，并对匹配的行执行请求的访问。

这一惯例为访问提供程序的应用的常见设计模式提供了便利。应用会对提供程序执行查询，并使用 `CursorAdapter` 以 `ListView` 显示生成的 `Cursor`。定义 `CursorAdapter` 的条件是，`Cursor` 中的其中一个列必须是 `_ID`

用户随后从 UI 上显示的行中选取其中一行，以查看或修改数据。应用会从支持 `ListView` 的 `Cursor` 中获取对应行，获取该行的 `_ID` 值，将其追加到内容 URI，然后向提供程序发送访问请求。然后，提供程序便可对用户选取的特定行执行查询或修改。

内容 URI 模式

为帮助您选择对传入的内容 URI 执行的操作，提供程序 API 加入了实用类 `UriMatcher`，它会将内容 URI“模式”映射到整型值。您可以在一个 `switch` 语句中使用这些整型值，为匹配特定模式的一个或多个内容 URI 选择所需操作。

内容 URI 模式使用通配符匹配内容 URI：

- `*`：匹配由任意长度的任何有效字符组成的字符串
- `#`：匹配由任意长度的数字字符组成的字符串

以设计和编码内容 URI 处理为例，假设一个具有授权 `com.example.app.provider` 的提供程序能识别以下指向表的内容 URI：

- `content://com.example.app.provider/table1`：一个名为 `table1` 的表
- `content://com.example.app.provider/table2/dataset1`：一个名为 `dataset1` 的表
- `content://com.example.app.provider/table2/dataset2`：一个名为 `dataset2` 的表
- `content://com.example.app.provider/table3`：一个名为 `table3` 的表

提供程序也能识别追加了行 ID 的内容 URI，例如，`content://com.example.app.provider/table3/1` 对应由 `table3` 中 `1` 标识的行的内容 URI。

可以使用以下内容 URI 模式：

```
content://com.example.app.provider/*
```

匹配提供程序中的任何内容 URI。

```
content://com.example.app.provider/table2/* :
```

匹配表 `dataset1` 和表 `dataset2` 的内容 URI，但不匹配 `table1` 或 `table3` 的内容 URI。

```
content://com.example.app.provider/table3/# : 匹配 table3 中单个行的内容 URI，如
```

```
content://com.example.app.provider/table3/6 对应由 6 标识的行的内容 URI。
```

以下代码段演示了 `UriMatcher` 中方法的工作方式。此代码采用不同方式处理整个表的 URI 与单个行的 URI，它为表使用的内容 URI 模式是 `content://<authority>/<path>`，为单个行使用的内容 URI 模式则是 `content://<authority>/<path>/<id>`。

方法 `addURI()` 会将授权和路径映射到一个整型值。方法 `match()` 会返回 URI 的整型值。`switch` 语句会在查询整个表与查询单个记录之间进行选择：

```

public class ExampleProvider extends ContentProvider {
...
    // Creates a UriMatcher object.
    private static final UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

    static {
        /*
         * The calls to addURI() go here, for all of the content URI patterns that the provider
         * should recognize. For this snippet, only the calls for table 3 are shown.
         */

        /*
         * Sets the integer value for multiple rows in table 3 to 1. Notice that no wildcard is used
         * in the path
         */
        sUriMatcher.addURI("com.example.app.provider", "table3", 1);

        /*
         * Sets the code for a single row to 2. In this case, the "#" wildcard is
         * used. "content://com.example.app.provider/table3/3" matches, but
         * "content://com.example.app.provider/table3" doesn't.
         */
        sUriMatcher.addURI("com.example.app.provider", "table3/#", 2);
    }
...
    // Implements ContentProvider.query()
    public Cursor query(
        Uri uri,
        String[] projection,
        String selection,
        String[] selectionArgs,
        String sortOrder) {
...
        /*
         * Choose the table to query and a sort order based on the code returned for the incoming
         * URI. Here, too, only the statements for table 3 are shown.
         */
        switch (sUriMatcher.match(uri)) {

            // If the incoming URI was for all of table3
            case 1:

                if (TextUtils.isEmpty(sortOrder)) sortOrder = "_ID ASC";
                break;

            // If the incoming URI was for a single row
            case 2:

                /*
                 * Because this URI was for a single row, the _ID value part is
                 * present. Get the last path segment from the URI; this is the _ID value.
                 * Then, append the value to the WHERE clause for the query
                 */
                selection = selection + "_ID = " + uri.getLastPathSegment();
                break;

            default:
...
                // If the URI is not recognized, you should do some error handling here.
        }
        // call the code to actually do the query
    }
}

```

另一个类 [ContentUris](#) 会提供一些工具方法，用于处理内容 URI 的 `id` 部分。[Uri](#) 类和 [Uri.Builder](#) 类包括一些工具方法，用于解析现有 [Uri](#) 对象和构建新对象。

实现 ContentProvider 类

`ContentProvider` 实例通过处理来自其他应用的请求来管理对结构化数据集的访问。所有形式的访问最终都会调用 `ContentResolver`，后者接着调用 `ContentProvider` 的具体方法来获取访问权限。

必需方法

抽象类 `ContentProvider` 定义了六个抽象方法，您必须将这些方法作为自己具体子类的一部分加以实现。所有这些方法（`onCreate()` 除外）都由一个尝试访问您的内容提供程序的客户端应用调用：

`query()`

从您的提供程序检索数据。使用参数选择要查询的表、要返回的行和列以及结果的排序顺序。将数据作为 `Cursor` 对象返回。

`insert()`

在您的提供程序中插入一个新行。使用参数选择目标表并获取要使用的列值。返回新插入行的内容 URI。

`update()`

更新您提供程序中的现有行。使用参数选择要更新的表和行，并获取更新后的列值。返回已更新的行数。

`delete()`

从您的提供程序中删除行。使用参数选择要删除的表和行。返回已删除的行数。

`getType()`

返回内容 URI 对应的 MIME 类型。[实现内容提供程序 MIME 类型](#)部分对此方法做了更详尽的描述。

`onCreate()`

初始化您的提供程序。Android 系统会在创建您的提供程序后立即调用此方法。请注意，`ContentResolver` 对象尝试访问您的提供程序时，系统才会创建它。

请注意，这些方法的签名与同名的 `ContentResolver` 方法相同。

您在实现这些方法时应考虑以下事项：

- 所有这些方法（`onCreate()` 除外）都可由多个线程同时调用，因此它们必须是线程安全方法。如需了解有关多个线程的更多信息，请参阅[进程和线程](#)主题；
- 避免在 `onCreate()` 中执行长时间操作。将初始化任务推迟到实际需要时进行。[实现 onCreate\(\) 方法](#)部分对此做了更详尽的描述；
- 尽管您必须实现这些方法，但您的代码只需返回要求的数据类型，无需执行任何其他操作。例如，您可能想防止其他应用向某些表插入数据。要实现此目的，您可以忽略 `insert()` 调用并返回 0。

实现 `query()` 方法

`ContentProvider.query()` 方法必须返回 `Cursor` 对象。如果失败，则会引发 `Exception`。如果您使用 SQLite 数据库作为数据存储，则只需返回由 `SQLiteDatabase` 类的其中一个 `query()` 方法返回的 `Cursor`。如果查询不匹配任何行，您应该返回一个 `Cursor` 实例（其 `getCount()` 方法返回 0）。只有当查询过程中出现内部错误时，您才应该返回 `null`。

如果您不使用 SQLite 数据库作为数据存储，请使用 `Cursor` 的其中一个具体子类。例如，在 `MatrixCursor` 类实现的游标中，每一行都是一个 `Object` 数组。对于此类，请使用 `addRow()` 来添加新行。

请记住，Android 系统必须能够跨进程边界传播 `Exception`。Android 可以为以下异常执行此操作，这些异常可能有助于处理查询错误：

- `IllegalArgumentException`（您可以选择在提供程序收到无效的内容 URI 时引发此异常）
- `NullPointerException`

实现 `insert()` 方法

`insert()` 方法会使用 `ContentValues` 参数中的值向相应表中添加新行。如果 `ContentValues` 参数中未包含列名称，您可能想在您的提供程序代码或数据库架构中提供其默认值。

此方法应该返回新行的内容 URI。要想构建此方法，请使用 `withAppendedId()` 向表的内容 URI 追加新行的 `_ID`（或其他主键）值。

实现 `delete()` 方法

`delete()` 方法不需要从您的数据存储中实际删除行。如果您将同步适配器与提供程序一起使用，应该考虑为已删除的行添加“删除”标志，而不是将行整个移除。同步适配器可以检查是否存在已删除的行，并将它们从服务器中移除，然后再将它们从提供程序中删除。

实现 `update()` 方法

`update()` 方法采用 `insert()` 所使用的相同 `ContentValues` 参数，以及 `delete()` 和 `ContentProvider.query()` 所使用的相同 `selection` 和 `selectionArgs` 参数。这样一来，您就可以在这些方法之间重复使用代码。

实现 `onCreate()` 方法

Android 系统会在启动提供程序时调用 `onCreate()`。您只应在此方法中执行运行快速的初始化任务，并将数据库创建和数据加载推迟到提供程序实际收到数据请求时进行。如果您在 `onCreate()` 中执行长时间的任务，则会减慢提供程序的启动速度，进而减慢提供程序对其他应用的响应速度。

例如，如果您使用 SQLite 数据库，可以在 `ContentProvider.onCreate()` 中创建一个新的 `SQLiteOpenHelper` 对象，然后在首次打开数据库时创建 SQL 表。为简化这一过程，在您首次调用 `getWritableDatabase()` 时，它会自动调用 `SQLiteOpenHelper.onCreate()` 方法。

以下两个代码段展示了 `ContentProvider.onCreate()` 与 `SQLiteOpenHelper.onCreate()` 之间的交互。第一个代码段是 `ContentProvider.onCreate()` 的实现：

```
public class ExampleProvider extends ContentProvider {

    /*
     * Defines a handle to the database helper object. The MainDatabaseHelper class is defined
     * in a following snippet.
     */
    private MainDatabaseHelper mOpenHelper;

    // Defines the database name
    private static final String DBNAME = "mydb";

    // Holds the database object
    private SQLiteDatabase db;

    public boolean onCreate() {

        /*
         * Creates a new helper object. This method always returns quickly.
         * Notice that the database itself isn't created or opened
         * until SQLiteOpenHelper.getWritableDatabase is called
         */
        mOpenHelper = new MainDatabaseHelper(
            getContext(),           // the application context
            DBNAME,                 // the name of the database)
            null,                   // uses the default SQLite cursor
            1                      // the version number
        );

        return true;
    }

    ...

    // Implements the provider's insert method
    public Cursor insert(Uri uri, ContentValues values) {
        // Insert code here to determine which table to open, handle error-checking, and so forth

        ...

        /*
         * Gets a writeable database. This will trigger its creation if it doesn't already exist.
         *
         */
        db = mOpenHelper.getWritableDatabase();
    }
}
```

下一个代码段是 `SQLiteOpenHelper.onCreate()` 的实现，其中包括一个帮助程序类：

```

...
// A string that defines the SQL statement for creating a table
private static final String SQL_CREATE_MAIN = "CREATE TABLE " +
    "main " +                               // Table's name
    "(" +                                     // The columns in the table
    " _ID INTEGER PRIMARY KEY, " +
    " WORD TEXT"
    " FREQUENCY INTEGER " +
    " LOCALE TEXT )";
...
/***
 * Helper class that actually creates and manages the provider's underlying data repository.
 */
protected static final class MainDatabaseHelper extends SQLiteOpenHelper {

    /*
     * Instantiates an open helper for the provider's SQLite data repository
     * Do not do database creation and upgrade here.
     */
    MainDatabaseHelper(Context context) {
        super(context, DBNAME, null, 1);
    }

    /*
     * Creates the data repository. This is called when the provider attempts to open the
     * repository and SQLite reports that it doesn't exist.
     */
    public void onCreate(SQLiteDatabase db) {

        // Creates the main table
        db.execSQL(SQL_CREATE_MAIN);
    }
}

```

实现内容提供程序 MIME 类型

`ContentProvider` 类具有两个返回 MIME 类型的方法：

`getType()`

您必须为任何提供程序实现的必需方法之一。

`getStreamTypes()`

系统在您的提供程序提供文件时要求实现的方法。

表的 MIME 类型

`getType()` 方法会返回一个 MIME 格式的 `String`，后者描述内容 URI 参数返回的数据类型。`Uri` 参数可以是模式，而不是特定 URI；在这种情况下，您应该返回与匹配该模式的内容 URI 关联的数据类型。

对于文本、HTML 或 JPEG 等常见数据类型，`getType()` 应该为该数据返回标准 MIME 类型。[IANA MIME Media Types](#) 网站上提供了这些标准类型的完整列表。

对于指向一个或多个表数据行的内容 URI，`getType()` 应该以 Android 供应商特有 MIME 格式返回 MIME 类型：

- 类型部分：`vnd`
- 子类型部分：
 - 如果 URI 模式用于单个行：`android.cursor.item/`
 - 如果 URI 模式用于多个行：`android.cursor.dir/`
- 提供程序特有部分：`vnd.<name>.<type>`

您提供 `<name>` 和 `<type>`。`<name>` 值应具有全局唯一性，`<type>` 值应在对应的 URI 模式中具有唯一性。适合选择贵公司的名称或您的

应用 Android 软件包名称的某个部分作为 `<name>`。适合选择 URI 关联表的标识字符串作为 `<type>`。

例如，如果提供程序的授权是 `com.example.app.provider`，并且它公开了一个名为 `table1` 的表，则 `table1` 中多个行的 MIME 类型是：

```
vnd.android.cursor.dir/vnd.com.example.provider.table1
```

对于 `table1` 的单个行，MIME 类型是：

```
vnd.android.cursor.item/vnd.com.example.provider.table1
```

文件的 MIME 类型

如果您的提供程序提供文件，请实现 `getStreamTypes()`。该方法会为您的提供程序可以为给定内容 URI 返回的文件返回一个 MIME 类型 `String` 数组。您应该通过 MIME 类型过滤器参数过滤您提供的 MIME 类型，以便只返回客户端想处理的那些 MIME 类型。

例如，假设提供程序以 `.jpg`、`.png` 和 `.gif` 格式文件形式提供照片图像。如果应用调用 `ContentResolver.getStreamTypes()` 时使用了过滤器字符串 `image/*`（任何是“图像”的内容），则 `ContentProvider.getStreamTypes()` 方法应返回数组：

```
{ "image/jpeg", "image/png", "image/gif"}
```

如果应用只对 `.jpg` 文件感兴趣，则可以在调用 `ContentResolver.getStreamTypes()` 时使用过滤器字符串 `*\\jpeg`，`ContentProvider.getStreamTypes()` 应返回：

```
{"image/jpeg"}
```

如果您的提供程序未提供过滤器字符串中请求的任何 MIME 类型，则 `getStreamTypes()` 应返回 `null`。

实现协定类

协定类是一种 `public final` 类，其中包含对 URI、列名称、MIME 类型以及其他与提供程序有关的元数据的常量定义。该类可确保即使 URI、列名称等数据的实际值发生变化，也可以正确访问提供程序，从而在提供程序与其他应用之间建立协定。

协定类对开发者也有帮助，因为其常量通常采用助记名称，因此可以降低开发者为列名称或 URI 使用错误值的可能性。由于它是一种类，因此可以包含 Javadoc 文档。集成开发环境（如 Android Studio）可以根据协定类自动完成常量名称，并为常量显示 Javadoc。

开发者无法从您的应用访问协定类的类文件，但他们可以通过您提供的 `.jar` 文件将其静态编译到其应用内。

举例来说，`ContactsContract` 类及其嵌套类便属于协定类。

实现内容提供程序权限

[安全与权限](#) 主题中详细描述了 Android 系统各个方面的权限和访问。[数据存储](#) 主题也描述了各类存储实行中的安全与权限。其中的要点简述如下：

- 默认情况下，存储在设备内部存储上的数据文件是您的应用和提供程序的私有数据文件；
- 您创建的 `SQLiteDatabase` 数据库是您的应用和提供程序的私有数据库；
- 默认情况下，您保存到外部存储的数据文件是公用并可全局读取的数据文件。您无法使用内容提供程序来限制对外部存储内文件的访问，因为其他应用可以使用其他 API 调用来对它们执行读取和写入操作；
- 用于在您的设备的内部存储上打开或创建文件或 SQLite 数据库的方法调用可能会为所有其他应用同时授予读取和写入访问权限。如果您将内部文件或数据库用作提供程序的存储区，并向其授予“可全局读取”或“可全局写入”访问权限，则您在清单文件中为提供程序设置的权限不会保护您的数据。内部存储中文件和数据库的默认访问权限是“私有”，对于提供程序的存储区，您不应更改此权限。

如果您想使用内容提供程序权限来控制对数据的访问，则应将数据存储在内部文件、SQLite 数据库或“云”中（例如，远程服务器上），而且您应该保持文件和数据库为您的应用所私有。

实现权限

即使底层数据为私有数据，所有应用仍可从您的提供程序读取数据或向其写入数据，因为在默认情况下，您的提供程序未设置权限。要想改变这种情况，请使用属性或 `<provider>` 元素的子元素在您的清单文件中为您的提供程序设置权限。您可以设置适用于整个提供程序、特定表甚至特定记录的权限，或者设置同时适用于这三者的权限。

您可以通过清单文件中的一个或多个 `<permission>` 元素为您的提供程序定义权限。要使权限对您的提供程序具有唯一性，请为 `android:name` 属性使用 Java 风格作用域。例如，将读取权限命名为 `com.example.app.provider.permission.READ_PROVIDER`。

以下列表描述了提供程序权限的作用域，从适用于整个提供程序的权限开始，然后逐渐细化。更细化的权限优先于作用域较大的权限：

统一读写提供程序级别权限

一个同时控制对整个提供程序读取和写入访问的权限，通过 `<provider>` 元素的 `android:permission` 属性指定。

单独的读取和写入提供程序级别权限

针对整个提供程序的读取权限和写入权限。您可以通过 `<provider>` 元素的 `android:readPermission` 属性和 `android:writePermission` 属性指定它们。它们优先于 `android:permission` 所需的权限。

路径级别权限

针对提供程序中内容 URI 的读取、写入或读取/写入权限。您可以通过 `<provider>` 元素的 `<path-permission>` 子元素指定您想控制的每个 URI。对于您指定的每个内容 URI，您都可以指定读取/写入权限、读取权限或写入权限，或同时指定所有三种权限。读取权限和写入权限优先于读取/写入权限。此外，路径级别权限优先于提供程序级别权限。

临时权限

一种权限级别，即使应用不具备通常需要的权限，该级别也能授予对应用的临时访问权限。临时访问功能可减少应用需要在其清单文件中请求的权限数量。当您启用临时权限时，只有持续访问您的所有数据的应用才需要“永久性”提供程序访问权限。

假设您需要实现电子邮件提供程序和应用的权限，如果您想允许外部图像查看器应用显示您的提供程序中的照片附件，为了在不请求权限的情况下为图像查看器提供必要的访问权限，可以为照片的内容 URI 设置临时权限。对您的电子邮件应用进行相应设计，使应用能够在用户想要显示照片时向图像查看器发送一个 Intent，其中包含照片的内容 URI 以及权限标志。图像查看器可随后查询您的电子邮件提供程序以检索照片，即使查看器不具备对您提供程序的正常读取权限，也不受影响。

要想启用临时权限，请设置 `<provider>` 元素的 `android:grantUriPermissions` 属性，或者向您的 `<provider>` 元素添加一个或多个 `<grant-uri-permission>` 子元素。如果您使用了临时权限，则每当您从提供程序中移除对某个内容 URI 的支持，并且该内容 URI 关联了临时权限时，都需要调用 `Context.revokeUriPermission()`。

该属性的值决定可访问的提供程序范围。如果该属性设置为 `true`，则系统会向整个提供程序授予临时权限，该权限将替代您的提供程序级别或路径级别权限所需的任何其他权限。

如果此标志设置为 `false`，则您必须向 `<provider>` 元素添加 `<grant-uri-permission>` 子元素。每个子元素都指定授予的临时权限所对应的一个或多个内容 URI。

要向应用授予临时访问权限，Intent 必须包含 `FLAG_GRANT_READ_URI_PERMISSION` 和/或 `FLAG_GRANT_WRITE_URI_PERMISSION` 标志。它们通过 `setFlags()` 方法进行设置。

如果 `android:grantUriPermissions` 属性不存在，则假设其为 `false`。

<Provider> 元素

与 `Activity` 和 `Service` 组件类似，必须使用 `<provider>` 元素在清单文件中为其应用定义 `ContentProvider` 的子类。Android 系统会从该元素获取以下信息：

授权 (`android:authorities`)

用于在系统内标识整个提供程序的符号名称。[设计内容 URI](#) 部分对此属性做了更详尽的描述。

提供程序类名 (`android:name`)

实现 `ContentProvider` 的类。实现 `ContentProvider` 类中对此类做了更详尽的描述。

权限

指定其他应用访问提供程序的数据所必须具备权限的属性：

- `android:grantUriPermissions`：临时权限标志
- `android:permission`：统一提供程序范围读取/写入权限
- `android:readPermission`：提供程序范围读取权限
- `android:writePermission`：提供程序范围写入权限

实现 `内容提供程序权限` 部分对权限及其对应属性做了更详尽的描述。

启动和控制属性

这些属性决定 Android 系统如何以及何时启动提供程序、提供程序的进程特性以及其他运行时设置：

- `android:enabled`：允许系统启动提供程序的标志。
- `android:exported`：允许其他应用使用此提供程序的标志。
- `android:initOrder`：此提供程序相对于同一进程中其他提供程序的启动顺序。
- `android:multiProcess`：允许系统在与调用客户端相同的进程中启动提供程序的标志。
- `android:process`：应在其中运行提供程序的进程的名称。
- `android:syncable`：指示提供程序的数据将与服务器上的数据同步的标志。

开发指南中针对 `<provider>` 元素的主题提供了这些属性的完整资料。

信息属性

提供程序的可选图标和标签：

- `android:icon`：包含提供程序图标的可绘制对象资源。该图标出现在 `Settings > Apps > All` 中应用列表内的提供程序标签旁；
- `android:label`：描述提供程序和/或其数据的信息标签。该标签出现在 `Settings > Apps > All` 中的应用列表内。

开发指南中针对 `<provider>` 元素的主题提供了这些属性的完整资料。

Intent 和数据访问

应用可以通过 `Intent` 间接访问内容提供程序。应用不会调用 `ContentResolver` 或 `ContentProvider` 的任何方法，它并不会直接提供，而是会发送一个启动某个 `Activity` 的 `Intent`，该 `Activity` 通常是提供程序自身应用的一部分。目标 `Activity` 负责检索和显示其 UI 中的数据。视 `Intent` 中的操作而定，目标 `Activity` 可能还会提示用户对提供程序的数据进行修改。`Intent` 可能还包含目标 `Activity` 在 UI 中显示的“extra”数据；用户随后可以选择更改此数据，然后使用它来修改提供程序中的数据。

您可能想使用 `Intent` 访问权限来帮助确保数据完整性。您的提供程序可能依赖于根据严格定义的业务逻辑插入、更新和删除数据。如果是这种情况，则允许其他应用直接修改您的数据可能会导致无效的数据。如果您想让开发者使用 `Intent` 访问权限，请务必为其提供详尽的参考资料。向他们解释为什么使用自身应用 UI 的 `Intent` 访问比尝试通过代码修改数据更好。

处理想要修改您的提供程序数据的传入 `Intent` 与处理其他 `Intent` 没有区别。您可以通过阅读 `Intent` 和 `Intent 过滤器` 主题了解有关 `Intent` 用法的更多信息。

日历提供程序

本文内容

- › [基础知识](#)
- › [用户权限](#)
- › [日历表](#)
 - › [查询日历](#)
 - › [修改日历](#)
 - › [插入日历](#)
- › [事件表](#)
 - › [添加事件](#)
 - › [更新事件](#)
 - › [删除事件](#)
- › [参加者表](#)
 - › [添加参加者](#)
- › [提醒表](#)
 - › [添加提醒](#)
- › [实例表](#)
 - › [查询实例表](#)
- › [日历 Intent](#)
 - › [使用 Intent 插入事件](#)
 - › [使用 Intent 编辑事件](#)
 - › [使用 Intent 查看日历数据](#)
- › [同步适配器](#)

关键类

- › [CalendarContract.Calendars](#)
- › [CalendarContract.Events](#)
- › [CalendarContract.Attendees](#)
- › [CalendarContract.Reminders](#)

日历提供程序是用户日历事件的存储区。您可以利用 Calendar Provider API 对日历、事件、参加者、提醒等执行查询、插入、更新和删除操作。

Calender Provider API 可供应用和同步适配器使用。规则因进行调用的程序类型而异。本文主要侧重于介绍使用 Calendar Provider API 作为应用的情况。如需了解对各类同步适配器差异的阐述，请参阅[同步适配器](#)。

正常情况下，要想读取或写入日历数据，应用的清单文件必须包括[用户权限](#)中所述的适当权限。为简化常见操作的执行，日历提供程序提供了一组 Intent，[日历 Intent](#)中对这些 Intent 做了说明。这些 Intent 会将用户转到日历应用，执行插入事件、查看事件和编辑事件。用户与日历应用交互，然后返回原来的应用。因此，您的应用不需要请求权限，也不需要提供用于查看事件或创建事件的用户界面。

基础知识

[内容提供程序](#)存储数据并使其可供应用访问。Android 平台提供的内容提供程序（包括日历提供程序）通常以一组基于关系型数据库模型的表格形式公开数据，在这个表格中，每一行都是一条记录，每一列都是特定类型和含义的数据。应用和同步适配器可以通过 Calendar Provider API 获得对储存用户日历数据的数据库表的读取/写入权限。

每一个内容提供程序都会公开一个对其数据集进行唯一标识的公共 URI（包装成一个 [Uri](#) 对象）。控制多个数据集（多个表）的内容提供程序会为每个数据集公开单独的 URI。所有提供程序 URI 都以字符串“content://”开头。这表示数据受内容提供程序的控制。日历提供程序会为其每个类（表）定义 URI 常量。这些 URI 的格式为 `<class>.CONTENT_URI`。例如，[Events.CONTENT_URI](#)。

图 1 是对日历提供程序数据模型的图形化表示。它显示了将彼此链接在一起的主要表和字段。

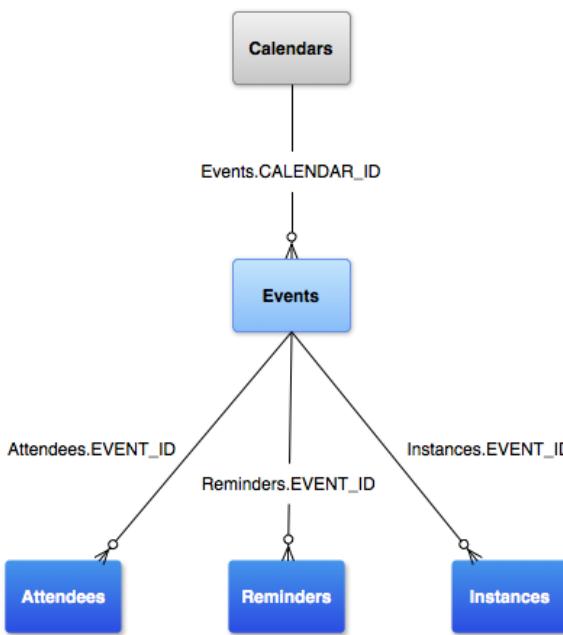


图 1. 日历提供程序数据模型。

用户可以有多个日历，可将不同类型的日历与不同类型的帐户（Google 日历、Exchange 等）关联。

[CalendarContract](#) 定义了日历和事件相关信息的数据模型。这些数据存储在以下所列的若干表中。

表 (类)	说明
CalendarContract.Calendars	此表储存日历特定信息。此表中的每一行都包含一个日历的详细信息，例如名称、颜色、同步信息等。
CalendarContract.Events	此表储存事件特定信息。此表中的每一行都包含一个事件的信息 — 例如事件标题、地点、开始时间、结束时间等。事件可一次性发生，也可多次重复发生。参加者、提醒和扩展属性存储在单独的表内。它们各自具有一个 <code>EVENT_ID</code> ，用于引用 <code>Events</code> 表中的 <code>_ID</code> 。
CalendarContract.Instances	此表储存每个事件实例的开始时间和结束时间。此表中的每一行都表示一个事件实例。对于一次性事件，实例与事件为 1:1 映射。对于重复事件，会自动生成多个行，分别对应多个事件实例。
CalendarContract.Attendees	此表储存事件参加者（来宾）信息。每一行都表示事件的一位来宾。它指定来宾的类型以及事件的来宾出席响应。
CalendarContract.Reminders	此表储存提醒/通知数据。每一行都表示事件的一个提醒。一个事件可以有多个提醒。每个事件的最大提醒数量在 <code>MAX_REMINDERS</code> 中指定，后者由拥有给定日历的同步适配器设置。提醒以事件发生前的分钟数形式指定，其具有一个可决定用户提醒方式的方法。

Calendar Provider API 以灵活、强大为设计宗旨。提供良好的最终用户体验以及保护日历及其数据的完整性也同样重要。因此，请在使用该 API 时牢记以下要点：

- **插入、更新和查看日历事件。** 要想直接从日历提供程序插入事件、修改事件以及读取事件，您需要具备相应[权限](#)。不过，如果您开发的并不是完备的日历应用或同步适配器，则无需请求这些权限。您可以改用 Android 的日历应用支持的 Intent 将读取操作和写入操作转到该应用执行。当您使用 Intent 时，您的应用会将用户转到日历应用，在一个预填充表单中执行所需操作。完成操作后，用户将返回您的应用。通过将您的应用设计为通过日历执行常见操作，可以为用户提供一致、可靠的用户界面。这是推荐您采用的方法。如需了解详细信息，请参阅[日历 Intent](#)。
- **同步适配器。** 同步适配器用于将用户设备上的日历数据与其他服务器或数据源同步。在 [CalendarContract.Calendars](#) 和 [CalendarContract.Events](#) 表中，预留了一些供同步适配器使用的列。提供程序和应用不应修改它们。实际上，除非以同步适配器形式进行访问，否则它们处于不可见状态。如需了解有关同步适配器的详细信息，请参阅[同步适配器](#)。

用户权限

如需读取日历数据，应用必须在其清单文件中加入 `READ_CALENDAR` 权限。文件中必须包括用于删除、插入或更新日历数据的 `WRITE_CALENDAR` 权限：

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>
    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_CALENDAR" />
    ...
</manifest>

```

日历表

`CalendarContract.Calendars` 表包含各日历的详细信息。应用和[同步适配器](#)均可写入下列日历列。如需查看所支持字段的完整列表，请参阅 `CalendarContract.Calendars` 参考资料。

常量	说明
<code>NAME</code>	日历的名称。
<code>CALENDAR_DISPLAY_NAME</code>	该日历显示给用户时使用的名称。
<code>VISIBLE</code>	表示是否选择显示该日历的布尔值。值为 0 表示不应显示与该日历关联的事件。值为 1 表示应该显示与该日历关联的事件。此值影响 <code>CalendarContract.Instances</code> 表中行的生成。
<code>SYNC_EVENTS</code>	一个布尔值，表示是否应同步日历并将其事件存储在设备上。值为 0 表示不同步该日历，也不将其事件存储在设备上。值为 1 表示同步该日历的事件，并将其事件存储在设备上。

查询日历

以下示例说明了如何获取特定用户拥有的日历。为了简便起见，在此示例中，查询操作显示在用户界面线程（“主线程”）中。实际上，此操作应该在一个异步线程而非主线程中完成。如需查看更详细的介绍，请参阅[加载器](#)。如果您的目的不只是读取数据，还要修改数据，请参阅 `AsyncQueryHandler`。

```
// Projection array. Creating indices for this array instead of doing
// dynamic lookups improves performance.
public static final String[] EVENT_PROJECTION = new String[] {
    Calendars._ID, // 0
    Calendars.ACCOUNT_NAME, // 1
    Calendars.CALENDAR_DISPLAY_NAME, // 2
    Calendars.OWNER_ACCOUNT // 3
};

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_ACCOUNT_NAME_INDEX = 1;
private static final int PROJECTION_DISPLAY_NAME_INDEX = 2;
private static final int PROJECTION_OWNER_ACCOUNT_INDEX = 3;
```

在示例的下一部分，您需要构建查询。选定范围指定查询的条件。在此示例中，查询寻找的是 `ACCOUNT_NAME` 为“sampleuser@google.com”、`ACCOUNT_TYPE` 为“com.google”、`OWNER_ACCOUNT` 为“sampleuser@google.com”的日历。如果您想查看用户查看过的所有日历，而不只是用户拥有的日历，请省略 `OWNER_ACCOUNT`。您可以利用查询返回的 `Cursor` 对象遍历数据库查询返回的结果集。如需查看有关在内容提供程序中使用查询的详细介绍，请参阅[内容提供程序](#)。

为何必须加入 ACCOUNT_TYPE？

如果您查询 `Calendars.ACCOUNT_NAME`，还必须将 `Calendars.ACCOUNT_TYPE` 加入选定范围。这是因为，对于给定帐户，只有在同时指定其 `ACCOUNT_NAME` 及其 `ACCOUNT_TYPE` 的情况下，才能将其视为唯一帐户。`ACCOUNT_TYPE` 字符串对应于在 `AccountManager` 处注册帐户时使用的帐户验证器。还有一种名为 `ACCOUNT_TYPE_LOCAL` 的特殊帐户类型，用于未关联设备帐户的日历。`ACCOUNT_TYPE_LOCAL` 帐户不会进行同步。

```
// Run query
Cursor cur = null;
ContentResolver cr = getContentResolver();
Uri uri = Calendars.CONTENT_URI;
String selection = "(" + Calendars.ACCOUNT_NAME + " = ?) AND (" +
    + Calendars.ACCOUNT_TYPE + " = ?) AND (" +
    + Calendars.OWNER_ACCOUNT + " = ?))";
String[] selectionArgs = new String[] {"sampleuser@gmail.com", "com.google",
    "sampleuser@gmail.com"};
// Submit the query and get a Cursor object back.
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs, null);
```

以下后续部分使用游标单步调试结果集。它使用在示例开头设置的常量来返回每个字段的值。

```

// Use the cursor to step through the returned records
while (cur.moveToFirst()) {
    long calID = 0;
    String displayName = null;
    String accountName = null;
    String ownerName = null;

    // Get the field values
    calID = cur.getLong(PROJECTION_ID_INDEX);
    displayName = cur.getString(PROJECTION_DISPLAY_NAME_INDEX);
    accountName = cur.getString(PROJECTION_ACCOUNT_NAME_INDEX);
    ownerName = cur.getString(PROJECTION_OWNER_ACCOUNT_INDEX);

    // Do something with the values...

    ...
}

```

修改日历

如需执行日历更新，您可以通过 URI 追加 ID (`withAppendedId()`) 或第一个选定项形式提供日历的 `_ID`。选定范围应以 "`_id=?`" 开头，并且第一个 `selectionArg` 应为事件的 `_ID`。您还可以通过在 URI 中编码 ID 来执行更新。下例使用 (`withAppendedId()`) 方法更改日历的显示名称：

```

private static final String DEBUG_TAG = "MyActivity";
...
long calID = 2;
ContentValues values = new ContentValues();
// The new display name for the calendar
values.put(CalendarContract.CALENDAR_DISPLAY_NAME, "Trevor's Calendar");
Uri updateUri = ContentUris.withAppendedId(CalendarContract.CONTENT_URI, calID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);

```

插入日历

日历设计为主要由同步适配器进行管理，因此您只应以同步适配器形式插入新日历。在大多数情况下，应用只能对日历进行一些表面更改，如更改显示名称。如果应用需要创建本地日历，可以利用 `ACCOUNT_TYPE_LOCAL` 的 `ACCOUNT_TYPE`，通过以同步适配器形式执行日历插入来实现目的。`ACCOUNT_TYPE_LOCAL` 是一种特殊的帐户类型，用于未关联设备帐户的日历。这种类型的日历不与服务器同步。如需了解对同步适配器的阐述，请参阅[同步适配器](#)。

事件表

`CalendarContract.Events` 表包含各事件的详细信息。要想添加、更新或删除事件，应用必须在其[清单文件](#)中加入 `WRITE_CALENDAR` 权限。

应用和同步适配器均可写入下列事件列。如需查看所支持字段的完整列表，请参阅[CalendarContract.Events](#) 参考资料。

常量	说明
<code>CALENDAR_ID</code>	事件所属日历的 <code>_ID</code> 。
<code>ORGANIZER</code>	事件组织者（所有者）的电子邮件。
<code>TITLE</code>	事件的标题。
<code>EVENT_LOCATION</code>	事件的发生地点。
<code>DESCRIPTION</code>	事件的描述。
<code>DTSTART</code>	事件开始时间，以从公元纪年开始计算的协调世界时毫秒数表示。
<code>DTEND</code>	事件结束时间，以从公元纪年开始计算的协调世界时毫秒数表示。
<code>EVENT_TIMEZONE</code>	事件的时区。
<code>EVENT_END_TIMEZONE</code>	事件结束时间的时区。
<code>DURATION</code>	<code>RFC5545</code> 格式的事件持续时间。例如，值为 <code>"PT1H"</code> 表示事件应持续一小时，值为 <code>"P2W"</code> 表示持续 2 周。
<code>ALL_DAY</code>	值为 1 表示此事件占用一整天（按照本地时区的定义）。值为 0 表示它是常规事件，可在一天内的任何时间开始和结束。
<code>RRULE</code>	事件的重复发生规则格式。例如， <code>"FREQ=WEEKLY;COUNT=10;WKST=SU"</code> 。您可以在 此处 找到更多示例。
<code>RDATE</code>	事件的重复发生日期。 <code>RDATE</code> 与 <code>RRULE</code> 通常联合用于定义一组聚合重复实例。如需查看更详细的介绍，请参阅

	RFC5545 规范。
AVAILABILITY	将此事件视为忙碌时间还是可调度的空闲时间。
GUESTS_CAN MODIFY	来宾是否可修改事件。
GUESTS_CAN_INVITE_OTHERS	来宾是否可邀请其他来宾。
GUESTS_CAN_SEE_GUESTS	来宾是否可查看参加者列表。

添加事件

当您的应用插入新事件时，我们建议您按照[使用 Intent 插入事件](#)中所述使用 `INSERT` Intent。不过，您可以在需要时直接插入事件。本节描述如何执行此操作。

以下是插入新事件的规则：

- 您必须加入 `CALENDAR_ID` 和 `DTSTART`。
- 您必须加入 `EVENT_TIMEZONE`。如需获取系统中已安装时区 ID 的列表，请使用 `getAvailableIDs()`。请注意，如果您按[使用 Intent 插入事件](#)中所述通过 `INSERT` Intent 插入事件，则此规则不适用 — 在该情形下，系统会提供默认时区。
- 对于非重复事件，您必须加入 `DTEND`。
- 对于重复事件，您必须加入 `DURATION` 以及 `RRULE` 或 `RDATE`。请注意，如果您按[使用 Intent 插入事件](#)中所述通过 `INSERT` Intent 插入事件，则此规则不适用 — 在该情形下，您可以将 `RRULE` 与 `DTSTART` 和 `DTEND` 结合使用，日历应用会自动将其转换为持续时间。

以下是一个插入事件的示例。为了简便起见，此操作是在 UI 线程内执行的。实际上，应该在异步线程中完成插入和更新，以便将操作移入后台线程。如需了解详细信息，请参阅 [AsyncQueryHandler](#)。

```
long calID = 3;
long startMillis = 0;
long endMillis = 0;
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 9, 14, 7, 30);
startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 9, 14, 8, 45);
endMillis = endTime.getTimeInMillis();
...

ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Events.DTSTART, startMillis);
values.put(Events.DTEND, endMillis);
values.put(Events.TITLE, "Jazzercise");
values.put(Events.DESCRIPTION, "Group workout");
values.put(Events.CALENDAR_ID, calID);
values.put(Events.EVENT_TIMEZONE, "America/Los_Angeles");
Uri uri = cr.insert(Events.CONTENT_URI, values);

// get the event ID that is the last element in the Uri
long eventId = Long.parseLong(uri.getLastPathSegment());
//
// ... do something with event ID
//
//
```

注：请注意以上示例如何在事件创建后捕获事件 ID。这是获取事件 ID 的最简单方法。您经常需要使用事件 ID 来执行其他日历操作 — 例如，向事件添加参加者或提醒。

更新事件

当您的应用想允许用户编辑事件时，我们建议您按照[使用 Intent 编辑事件](#)中所述使用 `EDIT` Intent。不过，您可以在需要时直接编辑事件。如需执行事件更新，您可以通过 URI 追加 ID (`withAppendedId()`) 或第一个选定项形式提供事件的 `_ID`。选定范围应以 `"_id=?"` 开头，并且第一个 `selectionArg` 应为事件的 `_ID`。您还可以使用不含 ID 的选定范围执行更新。以下是一个更新事件的示例。它使用 `withAppendedId()` 方法更改事件的标题：

```

private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 188;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri updateUri = null;
// The new title for the event
values.put(Events.TITLE, "Kickboxing");
updateUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);

```

删除事件

您可以通过将事件 `_ID` 作为 URI 追加 ID 或通过使用标准选定范围来删除事件。如果您使用追加 ID，则将无法同时使用选定范围。共有两个版本的删除：应用删除和同步适配器删除。应用删除将 `deleted` 列设置为 1。此标志告知同步适配器该行已删除，并且应将此删除操作传播至服务器。同步适配器删除会将事件连同其所有关联数据从数据库中移除。以下是一个应用通过事件 `_ID` 删除事件的示例：

```

private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 201;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri deleteUri = null;
deleteUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().delete(deleteUri, null, null);
Log.i(DEBUG_TAG, "Rows deleted: " + rows);

```

参加者表

`CalendarContract.Attendees` 表的每一行都表示事件的一位参加者或来宾。调用 `query()` 会返回一个参加者列表，其中包含具有给定 `EVENT_ID` 的事件的参加者。此 `EVENT_ID` 必须匹配特定事件的 `_ID`。

下表列出了可写入的字段。插入新参加者时，您必须加入除 `ATTENDEE_NAME` 之外的所有字段。

常量	说明
<code>EVENT_ID</code>	事件的 ID。
<code>ATTENDEE_NAME</code>	参加者的姓名。
<code>ATTENDEE_EMAIL</code>	参加者的电子邮件地址。
<code>ATTENDEE_RELATIONSHIP</code>	参加者与事件的关系。下列值之一： <ul style="list-style-type: none">• <code>RELATIONSHIP_ATTENDEE</code>• <code>RELATIONSHIP_NONE</code>• <code>RELATIONSHIP_ORGANIZER</code>• <code>RELATIONSHIP_PERFORMER</code>• <code>RELATIONSHIP_SPEAKER</code>
<code>ATTENDEE_TYPE</code>	参加者的类型。下列值之一： <ul style="list-style-type: none">• <code>TYPE_REQUIRED</code>• <code>TYPE_OPTIONAL</code>
<code>ATTENDEE_STATUS</code>	参加者的出席状态。下列值之一： <ul style="list-style-type: none">• <code>ATTENDEE_STATUS_ACCEPTED</code>• <code>ATTENDEE_STATUS_DECLINED</code>• <code>ATTENDEE_STATUS_INVITED</code>• <code>ATTENDEE_STATUS_NONE</code>• <code>ATTENDEE_STATUS_TENTATIVE</code>

添加参加者

以下是一个为事件添加一位参加者的示例。请注意，`EVENT_ID` 是必填项：

```
long eventID = 202;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Attendees.ATTENDEE_NAME, "Trevor");
values.put(Attendees.ATTENDEE_EMAIL, "trevor@example.com");
values.put(Attendees.ATTENDEE_RELATIONSHIP, Attendees.RELATIONSHIP_ATTENDEE);
values.put(Attendees.ATTENDEE_TYPE, Attendees.TYPE_OPTIONAL);
values.put(Attendees.ATTENDEE_STATUS, Attendees.ATTENDEE_STATUS_INVITED);
values.put(Attendees.EVENT_ID, eventID);
Uri uri = cr.insert(Attendees.CONTENT_URI, values);
```

提醒表

`CalendarContract.Reminders` 表的每一行都表示事件的一个提醒。调用 `query()` 会返回一个提醒列表，其中包含具有给定 `EVENT_ID` 的事件的提醒。

下表列出了提醒的可写入字段。插入新提醒时，必须加入所有字段。请注意，同步适配器指定它们在 `CalendarContract.Calendars` 表中支持的提醒类型。详情请参阅 [ALLOWED_Reminders](#)。

常量	说明
<code>EVENT_ID</code>	事件的 ID。
<code>MINUTES</code>	事件发生前的分钟数，应在达到该时间时发出提醒。
<code>METHOD</code>	服务器上设置的提醒方法。下列值之一： <ul style="list-style-type: none">• <code>METHOD_ALERT</code>• <code>METHOD_DEFAULT</code>• <code>METHOD_EMAIL</code>• <code>METHOD_SMS</code>

添加提醒

下例显示如何为事件添加提醒。提醒在事件发生前 15 分钟发出。

```
long eventID = 221;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Reminders.MINUTES, 15);
values.put(Reminders.EVENT_ID, eventID);
values.put(Reminders.METHOD, Reminders.METHOD_ALERT);
Uri uri = cr.insert(Reminders.CONTENT_URI, values);
```

实例表

`CalendarContract.Instances` 表储存事件实例的开始时间和结束时间。此表中的每一行都表示一个事件实例。实例表无法写入，只提供查询事件实例的途径。

下表列出了一些您可以执行实例查询的字段。请注意，时区由 `KEY_TIMEZONE_TYPE` 和 `KEY_TIMEZONE_INSTANCES` 定义。

常量	说明
<code>BEGIN</code>	实例的开始时间，以协调世界时毫秒数表示。
<code>END</code>	实例的结束时间，以协调世界时毫秒数表示。
<code>END_DAY</code>	与日历时区相应的实例儒略历结束日。
<code>END_MINUTE</code>	从日历时区午夜开始计算的实例结束时间（分钟）。
<code>EVENT_ID</code>	该实例对应事件的 <code>ID</code> 。

<code>START_DAY</code>	与日历时区相应的实例儒略历开始日。
<code>START_MINUTE</code>	从日历时区午夜开始计算的实例开始时间（分钟）。

查询实例表

如需查询实例表，您需要在 URI 中指定查询的时间范围。在以下示例中，`CalendarContract.Instances` 通过其 `CalendarContract.EventsColumns` 接口实现获得对 `TITLE` 字段的访问权限。换言之，`TITLE` 是通过数据库视图，而不是通过查询原始 `CalendarContract.Instances` 表返回的。

```

private static final String DEBUG_TAG = "MyActivity";
public static final String[] INSTANCE_PROJECTION = new String[] {
    Instances.EVENT_ID,           // 0
    Instances.BEGIN,              // 1
    Instances.TITLE               // 2
};

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_BEGIN_INDEX = 1;
private static final int PROJECTION_TITLE_INDEX = 2;
...

// Specify the date range you want to search for recurring
// event instances
Calendar beginTime = Calendar.getInstance();
beginTime.set(2011, 9, 23, 8, 0);
long startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2011, 10, 24, 8, 0);
long endMillis = endTime.getTimeInMillis();

Cursor cur = null;
ContentResolver cr = getContentResolver();

// The ID of the recurring event whose instances you are searching
// for in the Instances table
String selection = Instances.EVENT_ID + " = ?";
String[] selectionArgs = new String[] {"207"};

// Construct the query with the desired date range.
Uri.Builder builder = Instances.CONTENT_URI.buildUpon();
ContentUris.appendId(builder, startMillis);
ContentUris.appendId(builder, endMillis);

// Submit the query
cur = cr.query(builder.build(),
    INSTANCE_PROJECTION,
    selection,
    selectionArgs,
    null);

while (cur.moveToFirst()) {
    String title = null;
    long eventID = 0;
    long beginVal = 0;

    // Get the field values
    eventID = cur.getLong(PROJECTION_ID_INDEX);
    beginVal = cur.getLong(PROJECTION_BEGIN_INDEX);
    title = cur.getString(PROJECTION_TITLE_INDEX);

    // Do something with the values.
    Log.i(DEBUG_TAG, "Event: " + title);
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(beginVal);
    DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
    Log.i(DEBUG_TAG, "Date: " + formatter.format(calendar.getTime()));
}
}

```

日历 Intent

您的应用不需要读取和写入日历数据的权限。它可以改用 Android 的日历应用支持的 Intent 将读取和写入操作转到该应用执行。下表列出了日历提供程序支持的 Intent：

操作	URI	说明	Extra
VIEW	<code>content://com.android.calendar/time/<ms_since_epoch></code> 您还可以通过 <code>CalendarContract.CONTENT_URI</code> 引用 URI。如需查看使用该 Intent 的示例，请参阅 使用 Intent 查看日历数据 。	打开日历后定位到 <code><ms_since_epoch></code> 指定的时间。	无。
VIEW	<code>content://com.android.calendar/events/<event_id></code> 您还可以通过 <code>Events.CONTENT_URI</code> 引用 URI。如需查看使用该 Intent 的示例，请参阅 使用 Intent 查看日历数据 。	查看 <code><event_id></code> 指定的事件。	<code>CalendarContract.EXTRA_EVENT_BEGIN_TIME</code> <code>CalendarContract.EXTRA_EVENT_END_TIME</code>
EDIT	<code>content://com.android.calendar/events/<event_id></code> 您还可以通过 <code>Events.CONTENT_URI</code> 引用 URI。如需查看使用该 Intent 的示例，请参阅 使用 Intent 编辑事件 。	编辑 <code><event_id></code> 指定的事件。	<code>CalendarContract.EXTRA_EVENT_BEGIN_TIME</code> <code>CalendarContract.EXTRA_EVENT_END_TIME</code>
EDIT INSERT	<code>content://com.android.calendar/events</code> 您还可以通过 <code>Events.CONTENT_URI</code> 引用 URI。如需查看使用该 Intent 的示例，请参阅 使用 Intent 插入事件 。	创建事件。	下表列出的任一 Extra。

下表列出了日历提供程序支持的 Intent Extra：

Intent Extra	说明
<code>Events.TITLE</code>	事件的名称。
<code>CalendarContract.EXTRA_EVENT_BEGIN_TIME</code>	事件开始时间，以从公元纪年开始计算的毫秒数表示。
<code>CalendarContract.EXTRA_EVENT_END_TIME</code>	事件结束时间，以从公元纪年开始计算的毫秒数表示。
<code>CalendarContract.EXTRA_EVENT_ALL_DAY</code>	一个布尔值，表示事件属于全天事件。值可以是 <code>true</code> 或 <code>false</code> 。
<code>Events.EVENT_LOCATION</code>	事件的地点。
<code>Events.DESCRIPTION</code>	事件描述。
<code>Intent.EXTRA_EMAIL</code>	逗号分隔值形式的受邀者电子邮件地址列表。
<code>Events.RRULE</code>	事件的重复发生规则。
<code>Events.ACCESS_LEVEL</code>	事件是私人性质还是公共性质。
<code>Events.AVAILABILITY</code>	将此事件视为忙碌时间还是可调度的空闲时间。

下文描述如何使用这些 Intent。

使用 Intent 插入事件

您的应用可以利用 `INSERT` Intent 将事件插入任务转到日历应用执行。使用此方法时，您的应用甚至不需要在其[清单文件](#)中加入 `WRITE_CALENDAR` 权限。

当用户运行使用此方法的应用时，应用会将其转到日历来完成事件添加操作。`INSERT` Intent 利用 extra 字段为表单预填充日历中事件的详细信息。用户随后可取消事件、根据需要编辑表单或将事件保存到日历中。

以下是一个代码段，用于安排一个在 2012 年 1 月 19 日上午 7:30 开始、8:30 结束的事件。请注意该代码段中的以下内容：

- 它将 `Events.CONTENT_URI` 指定为 URI。
- 它使用 `CalendarContract.EXTRA_EVENT_BEGIN_TIME` 和 `CalendarContract.EXTRA_EVENT_END_TIME` extra 字段为表单预填充事件的时间。这些时间的值必须以从公元纪年开始计算的协调世界时毫秒数表示。
- 它使用 `Intent.EXTRA_EMAIL` extra 字段提供以逗号分隔的受邀者电子邮件地址列表。

```
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 0, 19, 8, 30);
Intent intent = new Intent(Intent.ACTION_INSERT)
    .setData(Events.CONTENT_URI)
    .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis())
    .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis())
    .putExtra(Events.TITLE, "Yoga")
    .putExtra(Events.DESCRIPTION, "Group class")
    .putExtra(Events.EVENT_LOCATION, "The gym")
    .putExtra(Events.AVAILABILITY, Events.AVAILABILITY_BUSY)
    .putExtra(Intent.EXTRA_EMAIL, "rowan@example.com,trevor@example.com");
startActivity(intent);
```

使用 Intent 编辑事件

您可以按[更新事件](#)中所述直接更新事件。但使用 `EDIT` Intent 可以让不具有事件编辑权限的应用将事件编辑操作转到日历应用执行。当用户在日历中完成事件编辑后，将会返回原来的应用。

以下是一个 Intent 的示例，它为指定事件设置新名称，并允许用户在日历中编辑事件。

```
long eventID = 208;
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent = new Intent(Intent.ACTION_EDIT)
    .setData(uri)
    .putExtra(Events.TITLE, "My New Title");
startActivity(intent);
```

使用 Intent 查看日历数据

日历提供程序提供了两种不同的 `VIEW` Intent 使用方法：

- 打开日历并定位到特定日期。
- 查看事件。

下例显示如何打开日历并定位到特定日期：

```
// A date-time specified in milliseconds since the epoch.
long startMillis;
...
Uri.Builder builder = CalendarContract.CONTENT_URI.buildUpon();
builder.appendPath("time");
ContentUris.appendId(builder, startMillis);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(builder.build());
startActivity(intent);
```

下例显示如何打开事件进行查看：

```
long eventID = 208;
...
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(uri);
startActivity(intent);
```

同步适配器

应用和同步适配器在访问日历提供程序的方式上只存在微小差异：

- 同步适配器需要通过将 `CALLER_IS_SYNCADAPTER` 设置为 `true` 来表明它是同步适配器。
- 同步适配器需要提供 `ACCOUNT_NAME` 和 `ACCOUNT_TYPE` 作为 URI 中的查询参数。
- 与应用或小部件相比，同步适配器拥有写入权限的列更多。例如，应用只能修改日历的少数几种特性，例如其名称、显示名称、能见度设置以及是否同步日历。相比之下，同步适配器不仅可以访问这些列，还能访问许多其他列，例如日历颜色、时区、访问级别、地点等等。不过，同步适配器受限于它指定的 `ACCOUNT_NAME` 和 `ACCOUNT_TYPE`。

您可以利用以下帮助程序方法返回供与同步适配器一起使用的 URI：

```
static Uri asSyncAdapter(Uri uri, String account, String accountType) {
    return uri.buildUpon()
        .appendQueryParameter(android.provider.CalendarContract.CALLER_IS_SYNCADAPTER, "true")
        .appendQueryParameter(CalendarContract Calendars.ACCOUNT_NAME, account)
        .appendQueryParameter(CalendarContract Calendars.ACCOUNT_TYPE, accountType).build();
}
```

如需查看同步适配器的实现示例（并非仅限与日历有关的实现），请参阅 [SampleSyncAdapter](#)。

联系人提供程序

内容快览

- › Android 的联系人相关信息存储区。
- › 与网页同步。
- › 集成社交交流数据。

本文内容

- › [联系人提供程序组织](#)
- › [原始联系人](#)
- › [数据](#)
- › [联系人](#)
- › [来自同步适配器的数据](#)
- › [所需权限](#)
- › [用户个人资料](#)
- › [联系人提供程序元数据](#)
- › [联系人提供程序访问](#)
- › [联系人提供程序同步适配器](#)
- › [社交交流数据](#)
- › [其他联系人提供程序功能](#)

关键类

- › [ContactsContract.Contacts](#)
- › [ContactsContract.RawContacts](#)
- › [ContactsContract.Data](#)
- › [android.provider.ContactsContract.StreamItems](#)

相关示例

- › [联系人管理器](#)
- › [示例同步适配器](#)

另请参阅

- › [内容提供程序基础知识](#)

联系人提供程序是一个强大而又灵活的 Android 组件，用于管理设备上联系人相关数据的中央存储区。联系人提供程序是您在设备的联系人应用中看到的数据源，您也可以在自己的应用中访问其数据，并可在设备与在线服务之间传送数据。提供程序储存有多种数据源，由于它会试图为每个联系人管理尽可能多的数据，因此造成其组织结构非常复杂。为此，该提供程序的 API 包含丰富的协定类和接口，为数据检索和修改提供便利。

本指南介绍下列内容：

- 提供程序基本结构
- 如何从提供程序检索数据
- 如何修改提供程序中的数据
- 如何编写用于同步服务器数据与联系人提供程序数据的同步适配器。

本指南假定您了解 Android 内容提供程序的基础知识。如需了解有关 Android 内容提供程序的更多信息，请阅读 [内容提供程序基础知识](#) 指南。[示例同步适配器](#) 示例应用是一个示例，展示如何使用同步适配器在联系人提供程序与 Google 网络服务托管的一个示例应用之间传送数据。

联系人提供程序组织

联系人提供程序是 Android 内容提供程序的一个组件。它保留了三种类型的联系人数据，每一种数据都对应提供程序提供的一个表，如图 1 所示：

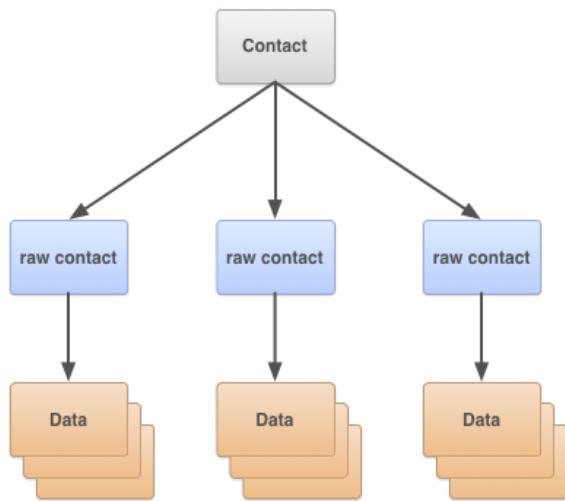


图 1. 联系人提供程序表结构。

这三个表通常以其协定类的名称命名。这些类定义表所使用的内容 URI、列名称及列值相应的常量：

[ContactsContract.Contacts](#) 表

表示不同联系人的行，基于聚合的原始联系人行。

[ContactsContract.RawContacts](#) 表

包含联系人数据摘要的行，针对特定用户帐户和类型。

[ContactsContract.Data](#) 表

包含原始联系人详细信息（例如电子邮件地址或电话号码）的行。

由 [ContactsContract](#) 中的协定类表示的其他表是辅助表，联系人提供程序利用它们来管理其操作，或为设备的联系人或电话应用中的特定功能提供支持。

原始联系人

一个原始联系人表示来自某一帐户类型和帐户名称、有关某个联系人的数据。由于联系人提供程序允许将多个在线服务作为某一联系人的数据源，因此它允许同一联系人对应多个原始联系人。借助支持多个原始联系人的特性，用户还可以将某一联系人在帐户类型相同的多个帐户中的数据进行合并。

原始联系人的大部分数据并不存储在 [ContactsContract.RawContacts](#) 表内，而是存储在 [ContactsContract.Data](#) 表中的一行或多行内。每个数据行都有一个 `Data.RAW_CONTACT_ID` 列，其中包含其父级 [ContactsContract.RawContacts](#) 行的 `RawContacts._ID` 值。

重要的原始联系人列

表 1 列出了 [ContactsContract.RawContacts](#) 表中的重要列。请阅读表后的说明：

表 1. 重要的原始联系人列。

列名称	用途	说明
<code>ACCOUNT_NAME</code>	作为该原始联系人来源的帐户类型的帐户名称。例如，Google 帐户的帐户名称是设备所有者的某个 Gmail 地址。如需了解详细信息，请参阅有关 <code>ACCOUNT_TYPE</code> 的下一条目。	此名称的格式专用于其帐户类型。它不一定是电子邮件地址。
<code>ACCOUNT_TYPE</code>	作为该原始联系人来源的帐户类型。例如，Google 帐户的帐户类型是 <code>com.google</code> 。请务必使用您拥有或控制的域的域标识符限定您的帐户类型。这可以确保您的帐户类型具有唯一性。	提供联系人数据的帐户类型通常关联有同步适配器，用于与联系人提供程序进行同步。
<code>DELETED</code>	原始联系人的“已删除”标志。	此标志让联系人提供程序能够在内部保留该行，直至同步适配器能够从服务器删除该行，然后再从存储区中最终删除该行。

说明

以下是关于 `ContactsContract.RawContacts` 表的重要说明：

- 原始联系人的姓名并不存储其在 `ContactsContract.RawContacts` 中的行内，而是存储在 `ContactsContract.Data` 表的 `ContactsContract.CommonDataKinds.StructuredName` 行内。一个原始联系人在 `ContactsContract.Data` 表中只有一个该类型的行。
- 注意：**要想在原始联系人行中使用您自己的帐户数据，必须先在 `AccountManager` 中注册帐户。为此，请提示用户将帐户类型及其帐户名称添加到帐户列表。如果您不这样做，联系人提供程序将自动删除您的原始联系人行。

例如，如果您想让您的应用为您域名为 `com.example.dataservice`、基于网络的服务保留联系人数据，并且您的服务的用户帐户是 `becky.sharp@dataservice.example.com`，则用户必须先添加帐户“类型”(`com.example.dataservice`) 和帐户“名称”(`becky.smart@dataservice.example.com`)，然后您的应用才能添加原始联系人行。您可以在文档中向用户解释这项要求，也可以提示用户添加类型和名称，或者同时采用这两种措施。下文对帐户类型和帐户名称做了更详尽的描述。

原始联系人数据来源

为理解原始联系人的工作方式，假设有一位用户“Emily Dickinson”，她的设备上定义了以下三个用户帐户：

- `emily.dickinson@gmail.com`
- `emilyd@gmail.com`
- Twitter 帐户“`belle_of_amherst`”

该用户已在“Accounts”设置中为全部三个帐户启用了“Sync Contacts”。

假定 Emily Dickinson 打开一个浏览器窗口，以 `emily.dickinson@gmail.com` 身份登录 Gmail，然后打开“联系人”，并添加“Thomas Higginson”。后来，她以 `emilyd@gmail.com` 身份登录 Gmail，并向“Thomas Higginson”发送一封电子邮件，此操作会自动将他添加为联系人。她还在 Twitter 上关注了“colonel_tom”(Thomas Higginson 的 Twitter ID)。

以上操作的结果是，联系人提供程序会创建以下这三个原始联系人：

- 第一个原始联系人对应“Thomas Higginson”，关联帐户 `emily.dickinson@gmail.com`。用户帐户类型是 Google。
- 第二个原始联系人对应“Thomas Higginson”，关联帐户 `emilyd@gmail.com`。用户帐户类型也是 Google。由于添加的联系人对应的用户帐户不同，因此尽管名称与前一名称完全相同，也只能作为第二个原始联系人。
- 第三个原始联系人对应“Thomas Higginson”，关联帐户“`belle_of_amherst`”。用户帐户类型是 Twitter。

数据

所前所述，原始联系人的数据存储在一个 `ContactsContract.Data` 行中，该行链接到原始联系人的 `_ID` 值。这使一位原始联系人可以拥有多个具有相同数据类型的实例，例如电子邮件地址或电话号码。例如，如果对应 `emilyd@gmail.com` 的“Thomas Higginson”(关联 Google 帐户 `emilyd@gmail.com` 的 Thomas Higginson 的原始联系人行) 的住宅电子邮件地址为 `thigg@gmail.com`，办公电子邮件地址为 `thomas.higginson@gmail.com`，则联系人提供程序会存储这两个电子邮件地址行，并将它们都链接到原始联系人。

请注意，这个表中存储了不同类型的数据。显示姓名、电话号码、电子邮件、邮政地址、照片以及网站明细行都可以在 `ContactsContract.Data` 表中找到。为便于管理这些数据，`ContactsContract.Data` 表为一些列使用了描述性名称，为其他列使用了通用名称。使用描述性名称的列的内容具有相同的含义，与行中数据的类型无关，而使用通用名称的列的内容则会随数据类型的不同而具有不同的含义。

描述性列名称

以下是一些描述性列名称的示例：

RAW_CONTACT_ID

该数据对应的原始联系人 `_ID` 列的值。

MIMETYPE

该行中存储的数据类型，以自定义 MIME (多用途互联网邮件扩展) 类型表示。联系人提供程序使用了 `ContactsContract.CommonDataKinds` 子类中定义的 MIME 类型。这些 MIME 类型为开源类型，可供与联系人提供程序协作的任何应用或同步适配器使用。

IS_PRIMARY

如果一个原始联系人可能具有多个这种类型的数据行，`IS_PRIMARY` 列会标记包含该类型主要数据的数据行。例如，如果用户长按某个联系人的电话号码，并选择 `Set default`，则包含该号码的 `ContactsContract.Data` 行会将其 `IS_PRIMARY` 列设置为一个非零值。

通用列名称

有 15 个通用列命名为 **DATA1** 至 **DATA15**，可普遍适用；还有四个通用列命名为 **SYNC1** 至 **SYNC4**，只应由同步适配器使用。通用列名称常量始终有效，与行包含的数据类型无关。

DATA1 列为索引列。联系人提供程序总是在此列中存储其预期会成为最频繁查询目标的数据。例如，在一个电子邮件行中，此列包含实际电子邮件地址。

按照惯例，DATA15 为预留列，用于存储照片缩略图等二进制大型对象 (BLOB) 数据。

类型专用列名称

为便于处理特定类型行的列，联系人提供程序还提供了 `ContactsContract.CommonDataKinds` 子类中定义的类型专用列名称常量。这些常量只是为同一列名称提供不同的常量名称，这有助于您访问特定类型行中的数据。

例如，`ContactsContract.CommonDataKinds.Email` 类为 `ContactsContract.Data` 行定义类型专用列名称常量，该行的 MIME 类型为 `Email.CONTENT_ITEM_TYPE`。该类包含电子邮件地址列的 `ADDRESS` 常量。`ADDRESS` 的实际值为“`data1`”，这与列的通用名称相同。

注意：请勿使用具有提供程序某个预定义 MIME 类型的行向 `contactsContract.Data` 表中添加您自己的自定义数据。否则您可能会丢失数据，或导致提供程序发生故障。例如，如果某一行具有 MIME 类型 `Email.CONTENT_ITEM_TYPE`，并且 `DATA1` 列包含的是用户名而不是电子邮件地址，您就不应添加该行。如果您为该行使用自定义的 MIME 类型，则可自由定义您的自定义类型专用的列名称，并随心所欲地使用这些列。

图 2 显示的是描述性列和数据列在 `ContactsContract.Data` 行中的显示情况，以及类型专用列名称“覆盖”通用列名称的情况

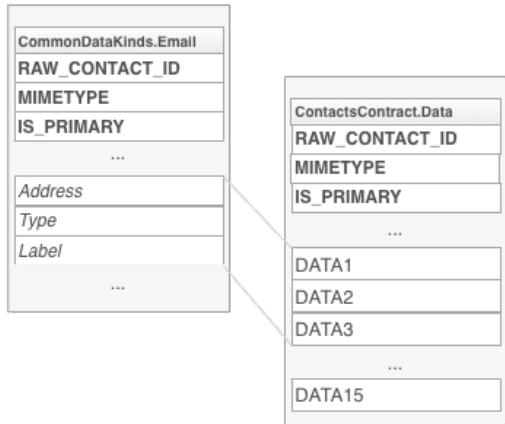


图 2. 类型专用列名称和通用列名称。

类型专用列名称类

表 2 列出了最常用的类型专用列名称类：

表 2. 类型专用列名称类

映射类	数据类型	说明
<code>ContactsContract.CommonDataKinds.StructuredName</code>	与该数据行关联的原始联系人的姓名数据。	一位原始联系人只有其中一行。
<code>ContactsContract.CommonDataKinds.Photo</code>	与该数据行关联的原始联系人的主要照片。	一位原始联系人只有其中一行。
<code>ContactsContract.CommonDataKinds.Email</code>	与该数据行关联的原始联系人的电子邮件地址。	一位原始联系人可有多个电子邮件地址。
<code>ContactsContract.CommonDataKinds.StructuredPostal</code>	与该数据行关联的原始联系人的邮政地址。	一位原始联系人可有多个邮政地址。
<code>ContactsContract.CommonDataKinds.GroupMembership</code>	将原始联系人链接到联系人提供程序内其中一组的标识符。	组是帐户类型和帐户名称的一项可选功能。 联系人组 部分对其做了更详尽的描述。

联系人

联系人提供程序通过将所有帐户类型和帐户名称的原始联系人行合并来形成**联系人**。这可以为显示和修改用户针对某一联系人收集的所有数据提供便利。联系人提供程序管理新联系人行的创建，以及原始联系人与现有联系人行的合并。系统不允许应用或同步适配器添加联系人，并且联系人行中的某些列是只读列。

注：如果您试图通过 `insert()` 向联系人提供程序添加联系人，会引发一个 `UnsupportedOperationException` 异常。如果您试图更新一个列为“只读”的列，更新会被忽略。

如果添加的新原始联系人不匹配任何现有联系人，联系人提供程序会相应地创建新联系人。如果某个现有原始联系人的数据发生了变化，不再匹配其之前关联的联系人，则提供程序也会执行此操作。如果应用或同步适配器创建的新原始联系人的确匹配某位现有联系人，则新原始联系人将与现有联系人合并。

联系人提供程序通过 `Contacts` 表中联系人行的 `_ID` 列将联系人行与其各原始联系人行链接起来。原始联系人表 `ContactsContract.RawContacts` 的 `CONTACT_ID` 列包含对应于每个原始联系人行所关联联系人行的 `_ID` 值。

`ContactsContract.Contacts` 表还有一个 `LOOKUP_KEY` 列，它是一个指向联系人行的“永久性”链接。由于联系人提供程序会自动维护联系人，因此可能会在合并或同步时相应地更改联系人行的 `_ID` 值。即使发生这种情况，合并了联系人 `LOOKUP_KEY` 的内容 URI `CONTENT_LOOKUP_URI` 仍将指向联系人行，这样，您就能使用 `LOOKUP_KEY` 保持指向“最喜爱”联系人的链接，以及执行其他操作。该列具有其自己的格式，与 `_ID` 列的格式无关。

图 3 显示的是这三个主要表的相互关系。

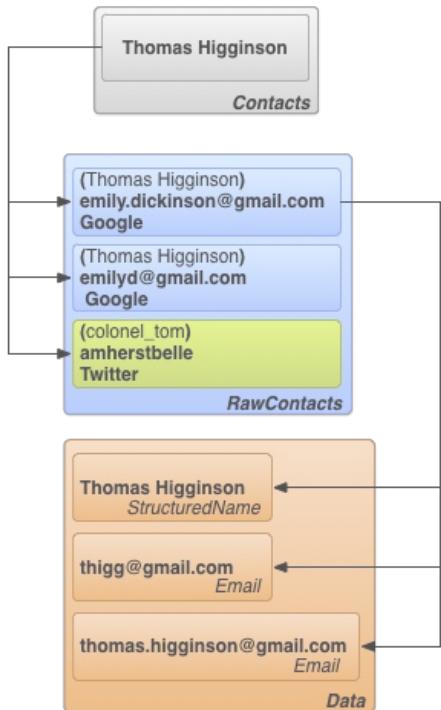


图 3. 联系人表、原始联系人表与详细信息表之间的关系。

来自同步适配器的数据

虽然用户是直接将联系人数据输入到设备中，但这些数据也会通过**同步适配器**从网络服务流入联系人提供程序中，这些同步适配器可自动化设备与服务之间的数据传送。同步适配器在系统控制下在后台运行，它们会调用 `ContentResolver` 方法来管理数据。

在 Android 中，与同步适配器协作的网络服务通过帐户类型加以标识。每个同步适配器都与一个帐户类型协作，但它可以支持该类型的多个帐户名称。[原始联系人数据来源](#)部分对帐户类型和帐户名称做了简要描述。下列定义提供了更多详细信息，并描述了帐户类型及帐户名称与同步适配器及服务之间的关系。

帐户类型

表示用户在其中存储数据的服务。在大多数时候，用户需要向服务验证身份。例如，Google 通讯录是一个以代码 `google.com` 标识的帐户类型。该值对应于 `AccountManager` 使用的帐户类型。

帐户名称

表示某个帐户类型的特定帐户或登录名。Google 通讯录帐户与 Google 帐户相同，都是以电子邮件地址作为帐户名称。其他服务可能使用一个单词的用户名或数字 ID。

帐户类型不必具有唯一性。用户可以配置多个 Google 通讯录帐户并将它们的数据下载到联系人提供程序；如果用户为个人帐户名称和工作帐户名称分别设置了一组联系人，就可能发生这种情况。帐户名称通常具有唯一性。它们共同标识联系人提供程序与外部服务之间的特定数据流。

如果您想将服务的数据传送到联系人提供程序，则需编写您自己的同步适配器。[联系人提供程序同步适配器](#)部分对此做了更详尽的描述。

图 4 显示的是联系人提供程序如何融入联系人数据的流动。在名为“同步适配器”的方框中，每个适配器都以其帐户类型命名。

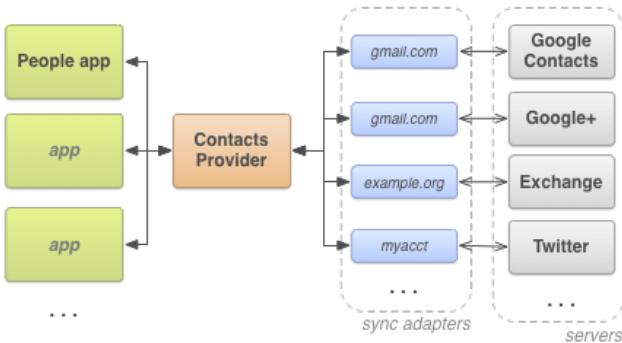


图 4. 联系人提供程序数据流。

所需权限

想要访问联系人提供程序的应用必须请求以下权限：

对一个或多个表的读取权限

`READ_CONTACTS`, 在 `AndroidManifest.xml` 中指定, 使用 `<uses-permission>` 元素作为 `<uses-permission android:name="android.permission.READ_CONTACTS">`。

对一个或多个表的写入权限

`WRITE_CONTACTS`, 在 `AndroidManifest.xml` 中指定, 使用 `<uses-permission>` 元素作为 `<uses-permission android:name="android.permission.WRITE_CONTACTS">`。

这些权限不适用于用户个人资料数据。下面的[用户个人资料](#)部分对用户个人资料及其所需权限做了阐述。

请切记, 用户的联系人数据属于个人敏感数据。用户关心其隐私权, 因此不希望应用收集有关其自身的数据或其联系人的数据。如需权限来访问其联系人数据的理由并不充分, 用户可能给您的应用作出差评或干脆拒绝安装。

用户个人资料

`ContactsContract.Contacts` 表有一行包含设备用户的个人资料数据。这些数据描述设备的 `user` 而不是用户的其中一位联系人。对于每个使用个人资料的系统, 该个人资料联系人行都链接到某个原始联系人行。每个个人资料原始联系人行可具有多个数据行。`ContactsContract.Profile` 类中提供了用于访问用户个人资料的常量。

访问用户个人资料需要特殊权限。除了进行读取和写入所需的 `READ_CONTACTS` 和 `WRITE_CONTACTS` 权限外, 如果想访问用户个人资料, 还分别需要 `android.Manifest.permission#READ_PROFILE` 和 `android.Manifest.permission#WRITE_PROFILE` 权限进行读取和写入访问。

请切记, 您应该将用户的个人资料视为敏感数据。权限 `android.Manifest.permission#READ_PROFILE` 让您可以访问设备用户的个人身份识别数据。请务必在您的应用的描述中告知用户您需要用户个人资料访问权限的原因。

要检索包含用户个人资料的联系人行, 请调用 `ContentResolver.query()`。将内容 URI 设置为 `CONTENT_URI` 并且不要提供任何选择条件。您还可以使用该内容 URI 作为检索原始联系人或个人资料数据的基本 URI。例如, 以下代码段用于检索个人资料数据 :

```

// Sets the columns to retrieve for the user profile
mProjection = new String[]
{
    Profile._ID,
    Profile.DISPLAY_NAME_PRIMARY,
    Profile.LOOKUP_KEY,
    Profile.PHOTO_THUMBNAIL_URI
};

// Retrieves the profile from the Contacts Provider
mProfileCursor =
    getContentResolver().query(
        Profile.CONTENT_URI,
        mProjection ,
        null,
        null,
        null);

```

注：如果您要检索多个联系人行并想要确定其中一个是否为用户个人资料, 请测试该行的 `IS_USER_PROFILE` 列。如果该联系人是用户个人资料, 则此列设置为“1”。

联系人提供程序元数据

联系人提供程序管理用于追踪存储区中联系人数据状态的数据。这些有关存储区的元数据存储在各处，其中包括原始联系人表行、数据表行和联系人表行、`ContactsContract.Settings` 表以及 `ContactsContract.SyncState` 表。下表显示的是每一部分元数据的作用：

表 3. 联系人提供程序中的元数据

表	列	值	含义
<code>ContactsContract.RawContacts</code>	<code>DIRTY</code>	“0”：上次同步以来未发生变化。	标记设备上因发生变化而需要同步回服务器的原始联系人。当 Android 应用更新行时，联系人提供程序会自动设置该值。 修改原始联系人表或数据表的同步适配器应始终向他们使用的内容 URI 追加字符串 <code>CALLER_IS_SYNCADAPTER</code> 。这可以防止提供程序将行标记为已更新。否则，即使服务器是修改的来源，同步适配器修改仍显示为本地修改，并会发送到服务器。
		“1”：上次同步以来发生了变化，需要同步回服务器。	
<code>ContactsContract.RawContacts</code>	<code>VERSION</code>	此行的版本号。	每当行或其相关数据发生变化时，联系人提供程序都会自动增加此值。
<code>ContactsContract.Data</code>	<code>DATA_VERSION</code>	此行的版本号。	每当数据行发生变化时，联系人提供程序都会自动增加此值。
<code>ContactsContract.RawContacts</code>	<code>SOURCE_ID</code>	一个字符串值，用于在创建此原始联系人的帐户中对该联系人进行唯一标识。	当同步适配器创建新原始联系人时，此列应设置为该原始联系人在服务器中的唯一 ID。当 Android 应用创建新原始联系人时，应将此列留空。这是为了向同步适配器表明，它应该在服务器上创建新原始联系人，并获取 <code>SOURCE_ID</code> 的值。 具体地讲，对于每个帐户类型，该源 ID 都必须是唯一的，并且应在所有同步中保持稳定： <ul style="list-style-type: none">唯一：帐户的每个原始联系人都必须有自己的源 ID。如果您不强制执行此要求，会在联系人应用中引发问题。请注意，帐户类型相同的两个原始联系人可以具有相同的源 ID。例如，允许帐户 <code>emily.dickinson@gmail.com</code> 的原始联系人“Thomas Higginson”与帐户 <code>emilyd@gmail.com</code> 的原始联系人“Thomas Higginson”具有相同的源 ID。稳定：源 ID 是该原始联系人在在线服务中的数据的永久性组成部分。例如，如果用户从应用设置中清除存储的联系人数据并重新同步，则恢复的原始联系人的源 ID 应与以前相同。如果您不强制执行此要求，快捷方式将停止工作。
<code>ContactsContract.Groups</code>	<code>GROUP_VISIBLE</code>	“0”：此组中的联系人在 Android 应用 UI 中不应处于可见状态。	此列用于兼容那些允许用户隐藏特定组中联系人的服务器。
		“1”：系统允许此组中的联系人在应用 UI 中处于可见状态。	
<code>ContactsContract.Settings</code>	<code>UNGROUPED_VISIBLE</code>	“0”：对于此帐户和帐户类型，未归入组的联系人在 Android 应用 UI 中处	默认情况下，如果联系人的所有原始联系人都未归入组，则它们将处于不可见状态（原始联系人的组成员身份通过 <code>ContactsContract.Data</code> 表中的一个或多个 <code>ContactsContract.CommonDataKinds.GroupMembership</code> 行指示）。通过在 <code>ContactsContract.Settings</code> 表行中为帐户类型和帐户设置此标志，您可以强制未归入组的联系人处于可见状态。此标志的一个用途是显示不使用组的服务器上的联系人。

		于不可见状态。 “1”：对于此帐户和帐户类型，未归属组的联系人在应用 UI 中处于可见状态。	
ContactsContract.SyncState	(all)	此表用于存储同步适配器的元数据。	利用此表，您可以将同步状态及其他同步相关数据持久地存储在设备中。

联系人提供程序访问

本节描述访问联系人提供程序中数据的准则，侧重于阐述以下内容：

- 实体查询。
- 批量修改。
- 通过 Intent 执行检索和修改。
- 数据完整性。

[联系人提供程序同步适配器](#)部分也对通过同步适配器进行修改做了更详尽的阐述。

查询实体

由于联系人提供程序表是以层级形式组织，因此对于检索某一行以及与其链接的所有“子”行，往往很有帮助。例如，要想显示某位联系人的所有信息，您可能需要检索某个 [ContactsContract.Contacts](#) 行的所有 [ContactsContract.RawContacts](#) 行，或者检索某个 [ContactsContract.RawContacts](#) 行的所有 [ContactsContract.CommonDataKinds.Email](#) 行。为便于执行此操作，联系人提供程序提供了**实体**构造，其作用类似于表间的数据库连接。

实体类似于一个表，由父表及其子表中的选定列组成。当您查询实体时，需要根据实体中的可用列提供投影和搜索条件。结果会得到一个 [Cursor](#)，检索的每个子表行在其中都有一行与之对应。例如，如果您在 [ContactsContract.Contacts.Entity](#) 中查询某个联系人姓名以及该姓名所有原始联系人的所有 [ContactsContract.CommonDataKinds.Email](#) 行，您会获得一个 [Cursor](#)，每个 [ContactsContract.CommonDataKinds.Email](#) 行在其中都有一行与之对应。

实体简化了查询。使用实体时，您可以一次性检索联系人或原始联系人的所有联系人数据，而不必先通过查询父表获得 ID，然后通过该 ID 查询子表。此外，联系人提供程序可通过单一事务处理实体查询，这确保了所检索数据的内部一致性。

注：实体通常不包含父表和子表的所有列。如果您试图使用的列名称并未出现在实体的列名称常量列表中，则会引发一个 [Exception](#)。

以下代码段说明如何检索某位联系人的所有原始联系人行。该代码段是一个大型应用的组成部分，包含“主”和“详”两个 Activity。主 Activity 显示一个联系人行列表；当用户选择一行时，该 Activity 会将其 ID 发送至详 Activity。详 Activity 使用 [ContactsContract.Contacts.Entity](#) 显示与所选联系人关联的所有原始联系人中的所有数据行。

以下代码段摘自“详”Activity：

```

...
/*
 * Appends the entity path to the URI. In the case of the Contacts Provider, the
 * expected URI is content://com.google.contacts/#/entity (# is the ID value).
 */
mContactUri = Uri.withAppendedPath(
    mContactUri,
    ContactsContract.Contacts.Entity.CONTENT_DIRECTORY);

// Initializes the loader identified by LOADER_ID.
getLoaderManager().initLoader(
    LOADER_ID, // The identifier of the loader to initialize
    null, // Arguments for the loader (in this case, none)
    this); // The context of the activity

// Creates a new cursor adapter to attach to the list view
mCursorAdapter = new SimpleCursorAdapter(
    this, // the context of the activity
    R.layout.detail_list_item, // the view item containing the detail widgets
    mCursor, // the backing cursor
    mFromColumns, // the columns in the cursor that provide the data
    mToViews, // the views in the view item that display the data
    0); // flags

// Sets the ListView's backing adapter.
mRawContactList.setAdapter(mCursorAdapter);
...
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    /*
     * Sets the columns to retrieve.
     * RAW_CONTACT_ID is included to identify the raw contact associated with the data row.
     * DATA1 contains the first column in the data row (usually the most important one).
     * MIMETYPE indicates the type of data in the data row.
     */
    String[] projection =
    {
        ContactsContract.Contacts.Entity.RAW_CONTACT_ID,
        ContactsContract.Contacts.Entity.DATA1,
        ContactsContract.Contacts.Entity.MIMETYPE
    };

    /*
     * Sorts the retrieved cursor by raw contact id, to keep all data rows for a single raw
     * contact collated together.
     */
    String sortOrder =
        ContactsContract.Contacts.Entity.RAW_CONTACT_ID +
        " ASC";

    /*
     * Returns a new CursorLoader. The arguments are similar to
     * ContentResolver.query(), except for the Context argument, which supplies the location of
     * the ContentResolver to use.
     */
    return new CursorLoader(
        getApplicationContext(), // The activity's context
        mContactUri, // The entity content URI for a single contact
        projection, // The columns to retrieve
        null, // Retrieve all the raw contacts and their data rows.
        null, // Sort by the raw contact ID.
        sortOrder);
}

```

加载完成时，`LoaderManager` 会调用一个 `onLoadFinished()` 回调。此方法的传入参数之一是一个 `Cursor`，其中包含查询的结果。在您自己的应用中，您可以从该 `Cursor` 获取数据，以进行显示或做进一步处理。

批量修改

您应尽可能地通过创建一个 `ContentProviderOperation` 对象 `ArrayList` 并调用 `applyBatch()`，以“批处理模式”在联系人提供程序中插入、更新和删除数据。由于联系人提供程序是在 `applyBatch()` 中通过单一事务执行所有操作，因此您的修改绝不会使联系人存储区出现不一致问题。此外，批量修改还有便于同时插入原始联系人及其明细数据。

注：要修改单个原始联系人，可以考虑向设备的联系人应用发送一个 Intent，而不是在您的应用中处理修改。通过 Intent 执行检索和修改部分对此操

作做了更详尽的描述。

屈服点

一个包含大量操作的批量修改可能会阻断其他进程，导致糟糕的总体用户体验。要将您想执行的所有修改组织到尽可能少的单独列表中，同时防止它们阻断系统，则应为一项或多项操作设置**屈服点**。屈服点是一个 `ContentProviderOperation` 对象，其 `isYieldAllowed()` 值设置为 `true`。当联系人提供程序遇到屈服点时，它会暂停其工作，让其他进程运行，并关闭当前事务。当提供程序再次启动时，它会继续执行 `ArrayList` 中的下一项操作，并启动一个新的事务。

屈服点会导致每次调用 `applyBatch()` 会产生多个事务。因此，您应该为针对一组相关行的最后一项操作设置屈服点。例如，您应该为一组操作中添加原始联系人行及其关联数据行的最后一项操作，或者针对一组与一位联系人相关的行的最后一项操作设置屈服点。

屈服点也是一个原子操作单元。两个屈服点之间所有访问的成功或失败都将作为一个单元的形式出现。如果您不设置任何屈服点，则最小的原子操作是整个批量操作。如果您使用了屈服点，则可以防止操作降低系统性能，还可确保一部分操作是原子操作。

修改向后引用

当您将一个新原始联系人行及其关联的数据行作为一组 `ContentProviderOperation` 对象插入时，需要通过将原始联系人的 `_ID` 值作为 `RAW_CONTACT_ID` 值插入，将数据行链接到原始联系人行。不过，当您为数据行创建 `ContentProviderOperation` 时，该值不可用，因为您尚未对原始联系人行应用 `ContentProviderOperation`。为解决此问题，`ContentProviderOperation.Builder` 类使用了 `WithValueBackReference()` 方法。该方法让您可以在插入或修改包含上一操作结果的列。

`WithValueBackReference()` 方法具有两个参数：

`key`

键-值对的键。此参数的值应为您要修改的表中某一列的名称。

`previousResult`

`applyBatch()` 中 `ContentProviderResult` 对象数组内某一值以 0 开始的索引。应用批处理操作时，每个操作的结果都存储在一个中间结果数组内。`previousResult` 值是其中一个结果的索引，它通过 `key` 值进行检索和存储。这样，您就可以插入一条新的原始联系人记录，并取回其 `_ID` 值，然后在添加 `ContactsContract.Data` 行时“向后引用”该值。

系统会在您首次调用 `applyBatch()` 时创建整个结果数组，其大小与您提供的 `ContentProviderOperation` 对象的 `ArrayList` 大小相等。不过，结果数组中的所有元素都设置为 `null`，如果您试图向后引用某个尚未应用的操作的结果，`WithValueBackReference()` 会引发一个 `Exception`。

以下代码段说明如何批量插入新原始联系人和数据。代码段中包括用于建立屈服点和使用向后引用的代码。这些代码段是扩展版本的 `createContactEntry()` 方法，该方法是 `Contact Manager` 示例应用中 `ContactAdder` 类的组成部分。

第一个代码段用于检索 UI 中的联系人数据。此时，用户已经选择了应添加新原始联系人的帐户。

```
// Creates a contact entry from the current UI values, using the currently-selected account.
protected void createContactEntry() {
    /*
     * Gets values from the UI
     */
    String name = mContactNameEditText.getText().toString();
    String phone = mContactPhoneEditText.getText().toString();
    String email = mContactEmailEditText.getText().toString();

    int phoneType = mContactPhoneTypes.get(
        mContactPhoneTypeSpinner.getSelectedItemPosition());

    int emailType = mContactEmailTypes.get(
        mContactEmailTypeSpinner.getSelectedItemPosition());
```

下一个代码段用于创建将该原始联系人行插入 `ContactsContract.RawContacts` 表的操作：

```
/*
 * Prepares the batch operation for inserting a new raw contact and its data. Even if
 * the Contacts Provider does not have any data for this person, you can't add a Contact,
 * only a raw contact. The Contacts Provider will then add a Contact automatically.
 */

// Creates a new array of ContentProviderOperation objects.
ArrayList<ContentProviderOperation> ops =
    new ArrayList<ContentProviderOperation>();

/*
 * Creates a new raw contact with its account type (server type) and account name
 * (user's account). Remember that the display name is not stored in this row, but in a
 * StructuredName data row. No other data is required.
 */
ContentProviderOperation.Builder op =
    ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_URI)
        .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, mSelectedAccount.getType())
        .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, mSelectedAccount.getName());

// Builds the operation and adds it to the array of operations
ops.add(op.build());
```

接着，代码会创建显示姓名行、电话行和电子邮件行的数据行。

每个操作生成器对象都使用 `WithValueBackReference()` 来获取 `RAW_CONTACT_ID`。引用指回来自第一次操作的 `ContentProviderResult` 对象，第一次操作就是添加原始联系人行并返回其新 `_ID` 值。结果是，每个数据行都通过其 `RAW_CONTACT_ID` 自动链接到其所属的 `ContactsContract.RawContacts` 行。

添加电子邮件行的 `ContentProviderOperation.Builder` 对象带有 `withYieldAllowed()` 标志，用于设置屈服点：

```

// Creates the display name for the new raw contact, as a StructuredName data row.
op =
    ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
    /*
     *WithValueBackReference sets the value of the first argument to the value of
     *the ContentProviderResult indexed by the second argument. In this particular
     *call, the raw contact ID column of the StructuredName data row is set to the
     *value of the result returned by the first operation, which is the one that
     *actually adds the raw contact row.
     */
    .WithValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

    // Sets the data row's MIME type to StructuredName
    .WithValue(ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)

    // Sets the data row's display name to the name in the UI.
    .WithValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME, name);

// Builds the operation and adds it to the array of operations
ops.add(op.build());

// Inserts the specified phone number and type as a Phone data row
op =
    ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
    /*
     * Sets the value of the raw contact id column to the new raw contact ID returned
     * by the first operation in the batch.
     */
    .WithValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

    // Sets the data row's MIME type to Phone
    .WithValue(ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)

    // Sets the phone number and type
    .WithValue(ContactsContract.CommonDataKinds.Phone.NUMBER, phone)
    .WithValue(ContactsContract.CommonDataKinds.Phone.TYPE, phoneType);

// Builds the operation and adds it to the array of operations
ops.add(op.build());

// Inserts the specified email and type as a Phone data row
op =
    ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
    /*
     * Sets the value of the raw contact id column to the new raw contact ID returned
     * by the first operation in the batch.
     */
    .WithValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

    // Sets the data row's MIME type to Email
    .WithValue(ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)

    // Sets the email address and type
    .WithValue(ContactsContract.CommonDataKinds.Email.ADDRESS, email)
    .WithValue(ContactsContract.CommonDataKinds.Email.TYPE, emailType);

/*
 * Demonstrates a yield point. At the end of this insert, the batch operation's thread
 * will yield priority to other threads. Use after every set of operations that affect a
 * single contact, to avoid degrading performance.
 */
op.withYieldAllowed(true);

// Builds the operation and adds it to the array of operations
ops.add(op.build());

```

最后一个代码段显示的是 `applyBatch()` 调用，用于插入新原始联系人行和数据行。

```

// Ask the Contacts Provider to create a new contact
Log.d(TAG, "Selected account: " + mSelectedAccount.getName() + " (" +
    mSelectedAccount.getType() + ")");
Log.d(TAG, "Creating contact: " + name);

/*
 * Applies the array of ContentProviderOperation objects in batch. The results are
 * discarded.
 */
try {
    getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
} catch (Exception e) {

    // Display a warning
    Context ctx = getApplicationContext();

    CharSequence txt = getString(R.string.contactCreationFailure);
    int duration = Toast.LENGTH_SHORT;
    Toast toast = Toast.makeText(ctx, txt, duration);
    toast.show();

    // Log exception
    Log.e(TAG, "Exception encountered while inserting contact: " + e);
}
}

```

此外，您还可以利用批处理操作实现**乐观并发控制**，这是一种无需锁定底层存储区便可应用修改事务的控制方法。要使用此方法，您需要应用事务，然后检查是否存在可能已同时做出的其他修改。如果您发现了不一致的修改，请回滚事务并重试。

乐观并发控制对于移动设备很有用，因为在移动设备上，同一时间只有一位用户，并且同时访问数据存储区的情况很少见。由于未使用锁定功能，因此不用浪费时间设置锁定或等待其他事务解除锁定。

要在更新某个 `ContactsContract.RawContacts` 行时使用乐观并发控制，请按以下步骤操作：

1. 检索原始联系人的 `VERSION` 列以及要检索的其他数据。
2. 创建一个适合使用 `newAssertQuery(Uri)` 方法强制执行约束的 `ContentProviderOperation.Builder` 对象。对于内容 URI，请使用追加有原始联系人 `_ID` 的 `RawContacts.CONTENT_URI`。
3. 对于 `ContentProviderOperation.Builder` 对象，请调用 `WithValue()`，对 `VERSION` 列与您刚检索的版本号进行比较。
4. 对于同一 `ContentProviderOperation.Builder`，请调用 `WithExpectedCount()`，确保此断言只对一行进行测试。
5. 调用 `build()` 创建 `ContentProviderOperation` 对象，然后将此对象添加为要传递至 `applyBatch()` 的 `ArrayList` 中的第一个对象。
6. 应用批处理事务。

如果在您读取原始联系人行到您试图对其进行修改这段时间有另一项操作更新了该行，“断言”`ContentProviderOperation` 将会失败，系统将终止整个批处理操作。此情况下，您可以选择重新执行批处理操作，或执行其他某操作。

以下代码段演示如何在使用 `CursorLoader` 查询一位原始联系人后创建一个“断言”`ContentProviderOperation`：

```

/*
 * The application uses CursorLoader to query the raw contacts table. The system calls this method
 * when the load is finished.
 */
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    // Gets the raw contact's _ID and VERSION values
    mRawContactID = cursor.getLong(cursor.getColumnIndex(BaseColumns._ID));
    mVersion = cursor.getInt(cursor.getColumnIndex(SyncColumns.VERSION));
}

...
// Sets up a Uri for the assert operation
Uri rawContactUri = ContentUris.withAppendedId(RawContacts.CONTENT_URI, mRawContactID);

// Creates a builder for the assert operation
ContentProviderOperation.Builder assertOp = ContentProviderOperation.netAssertQuery(rawContactUri);

// Adds the assertions to the assert operation: checks the version and count of rows tested
assertOp.withValue(SyncColumns.VERSION, mVersion);
assertOp.withExpectedCount(1);

// Creates an ArrayList to hold the ContentProviderOperation objects
ArrayList ops = new ArrayList<ContentProviderOperation>;

ops.add(assertOp.build());

// You would add the rest of your batch operations to "ops" here
...

// Applies the batch. If the assert fails, an Exception is thrown
try {
    ContentProviderResult[] results =
        getContentResolver().applyBatch(AUTHORITY, ops);

} catch (OperationApplicationException e) {

    // Actions you want to take if the assert operation fails go here
}

```

通过 Intent 执行检索和修改

通过向设备的联系人应用发送 Intent，您可以间接访问联系人提供程序。Intent 会启动设备的联系人应用 UI，用户可以在其中执行与联系人有关的操作。通过这种访问方式，用户可以：

- 从列表中选取一位联系人并将其返回给您的应用以执行进一步操作。
- 编辑现有联系人的数据。
- 为其任一帐户插入新原始联系人。
- 删除联系人或联系人数据。

如果用户要插入或更新数据，您可以先收集数据，然后将其作为 Intent 的一部分发送。

当您使用 Intent 通过设备的联系人应用访问联系人提供程序时，您无需自行编写用于访问该提供程序的 UI 或代码。您也无需请求对提供程序的读取或写入权限。设备的联系人应用可以将联系人读取权限授予给您，而且您是通过另一个应用对该提供程序进行修改，不需要拥有写入权限。

[内容提供程序基础知识](#)指南的“通过 Intent 访问数据”部分详细描述了通过发送 Intent 来访问某提供程序的一般过程。表 4 汇总了您为可用任务使用的操作、MIME 类型以及数据值，[ContactsContract.Intents.Insert](#) 参考文档列出了您可用于 [putExtra\(\)](#) 的 Extra 值：

表 4. 联系人提供程序 Intent。

任务	操作	数据	MIME 类型	说明
从列表中选取一位联系人	ACTION_PICK	下列值之一： <ul style="list-style-type: none"> Contacts.CONTENT_URI，显示联系人列表。 Phone.CONTENT_URI，显示原始联系人的电话号码列表。 	未使用	显示原始联系人列表或一位原始联系人的数据列表，具体取决于您提供的内容 URI 类型。调用 startActivityForResult() 方法，该方法返回所进行的内

		<ul style="list-style-type: none"> • <code>StructuredPostal.CONTENT_URI</code>, 显示原始联系人的邮政地址列表。 • <code>Email.CONTENT_URI</code>, 显示原始联系人的电子邮件地址列表。 		容 URI。该 URI 的形式为：追加有该行 <code>LOOKUP_ID</code> 的表的内容 URI。设备的联系人应用会在 Activity 的生命周期内将读取和写入权限授予给此内容 URI。如需了解更多详细信息, 请参阅 内容提供程序基础知识指南 。
插入新原始联系人	<code>Insert.ACTION</code>	不适用	<code>RawContacts.CONTENT_TYPE</code> , 用于一组原始联系人的 MIME 类型。	显示设备“通讯录”应用的 <code>Add Contact</code> 屏幕。系统会显示您添加到 Intent 中的 Extra 值。如果是随 <code>startActivityForResult()</code> 发送, 系统会将新添加的原始联系人的内容 URI 传回给 Activity 的 <code>onActivityResult()</code> 回调方法并作为后者 Intent 参数的“data”字段。要获取该值, 请调用 <code>getData()</code> 。
编辑联系人	<code>ACTION_EDIT</code>	该联系人的 <code>CONTENT_LOOKUP_URI</code> 。该编辑器 Activity 让用户能够对任何与该联系人关联的数据进行编辑。	<code>Contacts.CONTENT_ITEM_TYPE</code> , 一位联系人。	显示“通讯录”应用中的“Edit Contact”屏幕。系统会显示您添加到 Intent 中的 Extra 值。当用户点击 <code>Done</code> 保存编辑时, 您的 Activity 会返回前台。
显示一个同样可以添加数据的选取器。	<code>ACTION_INSERT_OR_EDIT</code>	不适用	<code>CONTENT_ITEM_TYPE</code>	此 Intent 始终显示“通讯录”应用的选取器屏幕。用户可以选择要编辑的联系人, 或添加新联系人。根据您的选择, 系统会显示编辑屏幕或添加屏幕, 还会显示您使用 Intent 传递的 Extra 数据。如果您的应用显示电子邮件或电话号码等联系人数据, 请使用此 Intent 来允许用户向现有联系人添加数据。

注：不需要通过此 Intent 的 Extra 发送姓名值, 因为用户总是会选取现有姓名或添加新姓名。此外, 如果您发送姓名, 并且用户选择执行编辑操作, 则联系人应用将显示您发送的姓名, 该姓名将覆盖以前的值。如果用户未注意这一情况便保存了编辑, 原有值将会丢失。

设备的联系人应用不允许您使用 Intent 删除原始联系人或其任何数据。因此, 要删除原始联系人, 请使用 `ContentResolver.delete()` 或 `ContentProviderOperation.newDelete()`。

以下代码段说明如何构建和发送一个插入新原始联系人和数据的 Intent :

```
// Gets values from the UI
String name = mContactNameEditText.getText().toString();
String phone = mContactPhoneEditText.getText().toString();
String email = mContactEmailEditText.getText().toString();
```

```
String company = mCompanyName.getText().toString();
String jobtitle = mJobTitle.getText().toString();

// Creates a new intent for sending to the device's contacts application
Intent insertIntent = new Intent(ContactsContract.Inserts.ACTION);

// Sets the MIME type to the one expected by the insertion activity
insertIntent.setType(ContactsContract.RawContacts.CONTENT_TYPE);

// Sets the new contact name
insertIntent.putExtra(ContactsContract.Inserts.NAME, name);

// Sets the new company and job title
insertIntent.putExtra(ContactsContract.Inserts.COMPANY, company);
insertIntent.putExtra(ContactsContract.Inserts.JOB_TITLE, jobtitle);

/*
 * Demonstrates adding data rows as an array list associated with the DATA key
 */

// Defines an array list to contain the ContentValues objects for each row
ArrayList<ContentValues> contactData = new ArrayList<ContentValues>();

/*
 * Defines the raw contact row
 */

// Sets up the row as a ContentValues object
ContentValues rawContactRow = new ContentValues();

// Adds the account type and name to the row
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_TYPE, mSelectedAccount.getType());
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_NAME, mSelectedAccount.getName());

// Adds the row to the array
contactData.add(rawContactRow);

/*
 * Sets up the phone number data row
 */

// Sets up the row as a ContentValues object
ContentValues phoneRow = new ContentValues();

// Specifies the MIME type for this data row (all data rows must be marked by their type)
phoneRow.put(
    ContactsContract.Data.MIMETYPE,
    ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
);

// Adds the phone number and its type to the row
phoneRow.put(ContactsContract.CommonDataKinds.Phone.NUMBER, phone);

// Adds the row to the array
contactData.add(phoneRow);

/*
 * Sets up the email data row
 */

// Sets up the row as a ContentValues object
ContentValues emailRow = new ContentValues();

// Specifies the MIME type for this data row (all data rows must be marked by their type)
emailRow.put(
    ContactsContract.Data.MIMETYPE,
    ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
);

// Adds the email address and its type to the row
emailRow.put(ContactsContract.CommonDataKinds.Email.ADDRESS, email);

// Adds the row to the array
contactData.add(emailRow);

/*
 * Adds the array to the intent's extras. It must be a parcelable object in order to
 * travel between processes. The device's contacts app expects its key to be
 * Intents.Insert.DATA
*/
```

```
/*
insertIntent.putParcelableArrayListExtra(ContactsContract.Inserts.Insert.DATA, contactData);

// Send out the intent to start the device's contacts app in its add contact activity.
startActivity(insertIntent);
```

数据完整性

联系人存储区包含用户认为是正确且是最新的重要敏感数据，因此联系人提供程序具有规定清晰的数据完整性规则。您有责任在修改联系人数据时遵守这些规则。以下列出了其中的重要规则：

务必为您添加的每个 `ContactsContract.RawContacts` 行添加一个 `ContactsContract.CommonDataKinds.StructuredName` 行。

如果 `ContactsContract.Data` 表中的 `ContactsContract.RawContacts` 行没有 `ContactsContract.CommonDataKinds.StructuredName` 行，可能会在聚合时引发问题。

务必将新 `ContactsContract.Data` 行链接到其父 `ContactsContract.RawContacts` 行。

如果 `ContactsContract.Data` 行未链接到 `ContactsContract.RawContacts`，则其在设备的联系人应用中将处于不可见状态，而且这可能会导致同步适配器出现问题。

请仅更改您拥有的那些原始联系人的数据。

请切记，联系人提供程序所管理的数据通常来自多个不同帐户类型/在线服务。您需要确保您的应用仅修改或删除归您所有的行的数据，并且仅通过您控制的帐户类型和帐户名称插入数据。

务必使用在 `ContactsContract` 及其子类中为权限、内容 URI、URI 路径、列名称、MIME 类型以及 `TYPE` 值定义的常量。

使用这些常量有助于您避免错误。如有任何常量被弃用，您还会从编译器警告收到通知。

自定义数据行

通过创建和使用自己的自定义 MIME 类型，您可以在 `ContactsContract.Data` 表中插入、编辑、删除和检索您的自有数据行。这些行仅限使用 `ContactsContract.DataColumns` 中定义的列，但您可以将您自己的类型专用列名称映射到默认列名称。在设备的联系人应用中，会显示这些行的数据，但无法对其进行编辑或删除，用户也无法添加其他数据。要允许用户修改您的自定义数据行，您必须在自己的应用中提供编辑器 Activity。

要显示您的自定义数据，请提供一个 `contacts.xml` 文件，其中须包含一个 `<ContactsAccountType>` 元素，及其一个或多个 `<ContactsDataKind>` 子元素。`<ContactsDataKind> element` 部分对此做了更详尽的描述。

如需了解有关自定义 MIME 类型的更多信息，请阅读[创建内容提供程序](#)指南。

联系人提供程序同步适配器

联系人提供程序专门设计用于处理设备与在线服务之间的联系人数据**同步**。借助同步功能，用户可以将现有数据下载到新设备，以及将现有数据上传到新帐户。此外，同步还能确保用户掌握最新数据，无需考虑数据增加和更改的来源。同步的另一个优点是，即使设备未连接网络，联系人数据同样可用。

虽然您可以通过各种方式实现同步，不过 Android 系统提供了一个插件同步框架，可自动化完成下列任务：

- 检查网络可用性。
- 根据用户偏好安排和执行同步。
- 重启已停止的同步。

要使用此框架，您需要提供一个同步适配器插件。每个同步适配器都专用于某个服务和内容提供程序，但可以处理同一服务的多个帐户名称。该框架还允许同一服务和提供程序具有多个同步适配器。

同步适配器类和文件

您需要将同步适配器作为 `AbstractThreadedSyncAdapter` 的子类进行实现，并作为 Android 应用的一部分进行安装。系统通过您的应用清单文件中的元素以及由清单文件指向的一个特殊 XML 文件了解有关同步适配器的信息。该 XML 文件定义在线服务的帐户类型和内容提供程序的权限，它们共同对适配器进行唯一标识。用户为同步适配器的帐户类型添加一个帐户，并为与同步适配器同步的内容提供程序启用同步后，同步适配器才会激活。激活后，系统将开始管理适配器，并在必要时调用它，以在内容提供程序与服务器之间同步数据。

注：将帐户类型用作同步适配器标识的一部分让系统可以发现从同一组织访问不同服务的同步适配器，并将它们组合在一起。例如，Google 在线服

务的同步适配器都具有相同的帐户类型 `com.google`。当用户向其设备添加 Google 帐户时，已安装的所有 Google 服务同步适配器将一起列出；列出的每个同步适配器都与设备上不同的内容提供程序同步。

大多数服务都要求用户验证身份后才能访问数据，为此，Android 系统提供了一个身份验证框架，该框架与同步适配器框架类似，并且经常与其联用。该身份验证框架使用的插件身份验证器是 `AbstractAccountAuthenticator` 的子类。身份验证器通过下列步骤验证用户的身份：

1. 收集用户名、用户密码或类似信息（用户的凭据）。
2. 将凭据发送给服务
3. 检查服务的回复。

如果服务接受了凭据，身份验证器便可存储凭据以供日后使用。由于插件身份验证器框架的存在，`AccountManager` 可以提供对身份验证器支持并选择公开的任何身份验证令牌（例如 OAuth2 身份验证令牌）的访问。

尽管身份验证并非必需，但大多数联系人服务都会使用它。不过，您不一定要使用 Android 身份验证框架进行身份验证。

同步适配器实现

要为联系人提供程序实现同步适配器，您首先要创建一个包含以下内容的 Android 应用：

一个 `Service` 组件，用于响应系统发出的绑定到同步适配器的请求。

当系统想要运行同步时，它会调用服务的 `onBind()` 方法，为同步适配器获取一个 `IBinder`。这样，系统便可跨进程调用适配器的方法。

在[示例同步适配器示例应用](#)中，该服务的类名是 `com.example.android.samplesync.syncadapter.SyncService`。

作为 `AbstractThreadedSyncAdapter` 具体子类实现的实际同步适配器。

此类的作用是从服务器下载数据、从设备上传数据以及解决冲突。适配器的主要工作是在方法 `onPerformSync()` 中完成的。必须将此类实例化为单一实例。

在[示例同步适配器示例应用](#)中，同步适配器是在 `com.example.android.samplesync.syncadapter.SyncAdapter` 类中定义的。

`Application` 的子类。

此类充当同步适配器单一实例的工厂。使用 `onCreate()` 方法实例化同步适配器，并提供一个静态“getter”方法，使单一实例返回同步适配器服务的 `onBind()` 方法。

可选：一个 `Service` 组件，用于响应系统发出的用户身份验证请求。

`AccountManager` 会启动此服务以开始身份验证流程。该服务的 `onCreate()` 方法会将一个身份验证器对象实例化。当系统想要对应用同步适配器的用户帐户进行身份验证时，它会调用该服务的 `onBind()` 方法，为该身份验证器获取一个 `IBinder`。这样，系统便可跨进程调用身份验证器的方法。

在[示例同步适配器示例应用](#)中，该服务的类名是 `com.example.android.samplesync.authenticator.AuthenticationService`。

可选：一个用于处理身份验证请求的 `AbstractAccountAuthenticator` 具体子类。

`AccountManager` 就是调用此类所提供的方法向服务器验证用户的凭据。详细的身份验证过程会因服务器所采用技术的不同而有很大差异。您应该参阅服务器软件的文档，了解有关身份验证的更多信息。

在[示例同步适配器示例应用](#)中，身份验证器是在 `com.example.android.samplesync.authenticator.Authentication` 类中定义的。

用于定义系统同步适配器和身份验证器的 XML 文件。

之前描述的同步适配器和身份验证器服务组件都是在应用清单文件中的 `<service>` 元素内定义的。这些元素包含以下用于向系统提供特定数据的 `<meta-data>` 子元素：

- 同步适配器服务的 `<meta-data>` 元素指向 XML 文件 `res/xml/syncadapter.xml`。而该文件则指定将与联系人提供程序同步的网络服务的 URI，以及指定该 Web 服务的帐户类型。
- **可选：**身份验证器的 `<meta-data>` 元素指向 XML 文件 `res/xml/authenticator.xml`。而该文件则指定此身份验证器所支持的帐户类型，以及指定身份验证过程中出现的 UI 资源。在此元素中指定的帐户类型必须与为同步适配器指定的帐户类型相同。

社交流数据

`android.provider.ContactsContract.StreamItems` 表和 `android.provider.ContactsContract.StreamItemPhotos` 表管理来自社交网络的传入数据。您可以编写一个同步适配器，用其将您自己社交网络中的流数据添加到这些表中，也可以从这些表读取流数据并将其显示在您的自有应用中，或者同时采用这两种

方法。利用这些功能，可以将您的社交网络服务和应用集成到 Android 的社交网络体验之中。

社交流文本

流项目始终与原始联系人关联。`android.provider.ContactsContract.StreamItemsColumns#RAW_CONTACT_ID` 链接到原始联系人的 `_ID` 值。原始联系人的帐户类型和帐户名称也存储在流项目行中。

将您的流数据存储在以下列中：

`android.provider.ContactsContract.StreamItemsColumns#ACCOUNT_TYPE`

必备。与该流项目关联的原始联系人对应的用户帐户类型。请记得在插入流项目时设置此值。

`android.provider.ContactsContract.StreamItemsColumns#ACCOUNT_NAME`

必备。与该流项目关联的原始联系人对应的用户帐户名称。请记得在插入流项目时设置此值。

标识符列

必备。您必须在插入流项目时插入下列标识符列：

- `android.provider.ContactsContract.StreamItemsColumns#CONTACT_ID`：此流项目关联的联系人的 `android.provider.BaseColumns#_ID` 值。
- `android.provider.ContactsContract.StreamItemsColumns#CONTACT_LOOKUP_KEY`：此流项目关联的联系人的 `android.provider.ContactsContract.ContactsColumns#LOOKUP_KEY` 值。
- `android.provider.ContactsContract.StreamItemsColumns#RAW_CONTACT_ID`：此流项目关联的原始联系人的 `android.provider.BaseColumns#_ID` 值。

`android.provider.ContactsContract.StreamItemsColumns#COMMENTS`

可选。存储可在流项目开头显示的摘要信息。

`android.provider.ContactsContract.StreamItemsColumns#TEXT`

流项目的文本，或为项目来源发布的内容，或是对生成流项目的某项操作的描述。此列可包含可由 `fromHtml()` 渲染的任何格式设置和嵌入式资源图像。提供程序可能会截断或省略较长内容，但它会尽力避免破坏标记。

`android.provider.ContactsContract.StreamItemsColumns#TIMESTAMP`

一个包含流项目插入时间或更新时间的文本字符串，以从公元纪年开始计算的毫秒数形式表示。此列由插入或更新流项目的应用负责维护；联系人提供程序不会自动对其进行维护。

要显示您的流项目的标识信息，请使用 `android.provider.ContactsContract.StreamItemsColumns#RES_ICON`、`android.provider.ContactsContract.StreamItemsColumns#RES_LABEL` 和 `android.provider.ContactsContract.StreamItemsColumns#RES_PACKAGE` 链接到您的应用中的资源。

`android.provider.ContactsContract.StreamItems` 表还包含供同步适配器专用的列 `android.provider.ContactsContract.StreamItemsColumns#SYNC1` 至 `android.provider.ContactsContract.StreamItemsColumns#SYNC4`。

社交流照片

`android.provider.ContactsContract.StreamItemPhotos` 表存储与流项目关联的照片。该表的

`android.provider.ContactsContract.StreamItemPhotosColumns#STREAM_ITEM_ID` 列链接到 `android.provider.ContactsContract.StreamItems` 表的 `_ID` 列中的值。照片引用存储在表中的以下列：

`android.provider.ContactsContract.StreamItemPhotos#PHOTO` 列（一个二进制大型对象）。

照片的二进制表示，为便于存储和显示，由提供程序调整了尺寸。此列可用于向后兼容使用它来存储照片的旧版本联系人提供程序。不过，在当前版本中，您不应使用此列来存储照片，而应使用 `android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_FILE_ID` 或 `android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_URI`（下文对两者都做了描述）将照片存储在一个文件内。此列现在包含可用于读取的照片缩略图。

`android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_FILE_ID`

原始联系人照片的数字标识符。将此值追加到常量 `DisplayPhoto.CONTENT_URI`，获取指向单一照片文件的内容 URI，然后调用 `openAssetFileDescriptor()` 来获取照片文件的句柄。

`android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_URI`

一个内容 URI，直接指向此行所表示的照片的照片文件。通过此 URI 调用 `openAssetFileDescriptor()` 以获得照片文件的句柄。

使用社交流表

这些表的工作方式与联系人提供程序中的其他主表基本相同，不同的是：

- 这些表需要额外的访问权限。要读取它们的数据，您的应用必须具有 `android.Manifest.permission#READ_SOCIAL_STREAM` 权限。要修改它们，您的应用必须具有 `android.Manifest.permission#WRITE_SOCIAL_STREAM` 权限。
- 对于 `android.provider.ContactsContract.StreamItems` 表，为每一位原始联系人存储的行数有限。一旦达到该限制，联系人提供程序即会自动删除 `android.provider.ContactsContract.StreamItemsColumns#TIMESTAMP` 最早的行，为新流项目腾出空间。要获取该限制，请发出对内容 URI `android.provider.ContactsContract.StreamItems#CONTENT_LIMIT_URI` 的查询。您可以将内容 URI 以外的所有其他参数保持设置为 `null`。查询会返回一个 Cursor，其中包含一行，并且只有 `android.provider.ContactsContract.StreamItems#MAX_ITEMS` 一列。

`android.provider.ContactsContract.StreamItems.StreamItemPhotos` 类定义了 `android.provider.ContactsContract.StreamItemPhotos` 的一个子表，其中包含某个流项目的照片行。

社交流交互

通过将联系人提供程序管理的社交流数据与设备的联系人应用相结合，可以在您的社交网络系统与现有联系人之间建立起有效的连接。这种结合实现了下列功能：

- 您可以通过同步适配器让您的社交网络服务与联系人提供程序同步，检索用户联系人的近期 Activity，并将其存储在 `android.provider.ContactsContract.StreamItems` 表和 `android.provider.ContactsContract.StreamItemPhotos` 表中，以供日后使用。
- 除了定期同步外，您还可以在用户选择某位联系人进行查看时触发您的同步适配器以检索更多数据。这样，您的同步适配器便可检索该联系人的高分辨率照片和最近流项目。
- 通过在设备的联系人应用以及联系人提供程序中注册通知功能，您可以在用户查看联系人时收到一个 Intent，并在那时通过您的服务更新联系人的状态。与通过同步适配器执行完全同步相比，此方法可能更快速，占用的带宽也更少。
- 用户可以在查看设备联系人应用中的联系人时，将其添加到您的社交网络服务。您可以通过“邀请联系人”功能实现此目的，而该功能则是通过将 Activity 与 XML 文件结合使用来实现的，前者将现有联系人添加到您的社交网络，后者为设备的联系人应用以及联系人提供程序提供有关您的应用的详细信息。

流项目与联系人提供程序的定期同步与其他同步相同。如需了解有关同步的更多信息，请参阅 [联系人提供程序同步适配器](#) 部分。接下来的两节介绍如何注册通知和邀请联系人。

通过注册处理社交网络查看

要注册您的同步适配器，以便在用户查看由您的同步适配器管理的联系人时收到通知，请执行以下步骤：

1. 在您项目的 `res/xml/` 目录中创建一个名为 `contacts.xml` 的文件。如果您已有该文件，可跳过此步骤。
2. 在该文件中添加元素 `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`。如果该元素已存在，可跳过此步骤。
3. 要注册一项服务，以便在用户于设备的联系人应用中打开某位联系人的详细信息页面时通知该服务，请为该元素添加 `viewContactNotifyService="serviceclass"` 属性，其中 `serviceclass` 是该服务的完全限定类名，应由该服务接收来自设备联系人应用的 Intent。对于这个通知程序服务，请使用一个扩展 `IntentService` 的类，以让该服务能够接收 Intent。传入 Intent 中的数据包含用户点击的原始联系人的内容 URI。您可以通过通知程序服务绑定到您的同步适配器，然后调用同步适配器来更新原始联系人的数据。

要注册需要在用户点击流项目或照片（或同时点击这两者）时调用的 Activity，请执行以下步骤：

1. 在您项目的 `res/xml/` 目录中创建一个名为 `contacts.xml` 的文件。如果您已有该文件，可跳过此步骤。
2. 在该文件中添加元素 `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`。如果该元素已存在，可跳过此步骤。
3. 要注册某个 Activity，以处理用户在设备联系人应用中点击某个流项目的操作，请为该元素添加 `viewStreamItemActivity="activityclass"` 属性，其中 `activityclass` 是该 Activity 的完全限定类名，应由该 Activity 接收来自设备联系人应用的 Intent。
4. 要注册某个 Activity，以处理用户在设备联系人应用中点击某个流照片的操作，请为该元素添加 `viewStreamItemPhotoActivity="activityclass"` 属性，其中 `activityclass` 是该 Activity 的完全限定类名，应由该 Activity 接收来自设备联系人应用的 Intent。

`<ContactsAccountType>` 元素部分对 `<ContactsAccountType>` 元素做了更详尽的描述。

传入 Intent 包含用户点击的项目或照片的内容 URI。要让文本项目和照片具有独立的 Activity，请在同一文件中使用这两个属性。

与您的社交网络服务交互

用户不必为了邀请联系人到您的社交网络网站而离开设备的联系人应用。取而代之是，您可以让设备的联系人应用发送一个 Intent，将联系人邀请到您的 Activity 之一。要设置此功能，请执行以下步骤：

- 在您项目的 res/xml/ 目录中创建一个名为 contacts.xml 的文件。如果您已有该文件，可跳过此步骤。
- 在该文件中添加元素 <ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">。如果该元素已存在，可跳过此步骤。
- 添加以下属性：

- inviteContactActivity="activityclass"
- inviteContactActionLabel="@string/invite_action_label"

activityclass 值是应该接收该 Intent 的 Activity 的完全限定类名。invite_action_label 值是一个文本字符串，将显示在设备联系人应用的 Add Connection 菜单中。

注：ContactsSource 是 ContactsAccountType 的一个已弃用的标记名称。

contacts.xml 引用

文件 contacts.xml 包含一些 XML 元素，这些元素控制您的同步适配器和应用与联系人应用及联系人提供程序的交互。下文对这些元素做了描述。

<ContactsAccountType> 元素

<ContactsAccountType> 元素控制您的应用与联系人应用的交互。它采用了以下语法：

```
<ContactsAccountType
    xmlns:android="http://schemas.android.com/apk/res/android"
    inviteContactActivity="activity_name"
    inviteContactActionLabel="invite_command_text"
    viewContactNotifyService="view_notify_service"
    viewGroupActivity="group_view_activity"
    viewGroupActionLabel="group_action_text"
    viewStreamItemActivity="viewstream_activity_name"
    viewStreamItemPhotoActivity="viewphotostream_activity_name">
```

包含它的文件：

res/xml/contacts.xml

可包含：

<ContactsDataKind>

说明：

声明 Android 组件和 UI 标签，让用户能够邀请他们的一位联系人加入社交网络，在他们的某个社交网络流更新时通知用户，以及执行其他操作。

请注意，对 <ContactsAccountType> 的属性而言，属性前缀 android: 并非必需的。

属性：

inviteContactActivity

您的应用中某个 Activity 的完全限定类名，您想要在用户于设备的联系人应用中选择 Add connection 时激活该 Activity。

inviteContactActionLabel

Add connection 菜单中为 inviteContactActivity 中指定的 Activity 显示的文本字符串。例如，您可以使用字符串“Follow in my network”。您可以为此标签使用字符串资源标识符。

viewContactNotifyService

您的应用中某项服务的完全限定类名，当用户查看联系人时，应由该服务接收通知。此通知由设备的联系人应用发送；您的应用可以根据通知将数据密集型操作推迟到必要时再执行。例如，您的应用对此通知的响应可以是：读入并显示联系人的高分辨率照片和最近的社交流项目。[社交流互](#)部分对此功能做了更详尽的描述。您可以在 SampleSyncAdapter 示例应用的 NotifierService.java 文件中查看通知服务的示例。

viewGroupActivity

您的应用中某个可显示组信息的 Activity 的完全限定类名。当用户点击设备联系人应用中的组标签时，将显示此 Activity 的 UI。

viewGroupActionLabel

联系人应用为某个 UI 控件显示的标签，用户可通过该控件查看您的应用中的组。

例如，如果您在设备上安装了 Google+ 应用，并将 Google+ 与联系人应用同步，就会看到 Google+ 圈子以组的形式出现在您的联系人应用的 **Groups** 选项卡内。如果您点击某个 Google+ 圈子，就会看到该圈子内的联系人以“组”的形式列出。在该显示页面的顶部，您会看到一个 Google+ 图标；如果您点击它，控制权将切换给 Google+ 应用。“通讯录”应用以 Google+ 图标作为 `viewGroupActionLabel` 的值，通过 `viewGroupActivity` 来实现此目的。

允许使用字符串资源标识符作为该属性的值。

viewStreamItemActivity

您的应用中某个 Activity 的完全限定类名，设备的联系人应用会在用户点击原始联系人的流项目时启动该 Activity。

viewStreamItemPhotoActivity

您的应用中某个 Activity 的完全限定类名，设备的联系人应用会在用户点击原始联系人流项目中的照片时启动该 Activity。

<ContactsDataKind> 元素

<ContactsDataKind> 元素控制您的应用的自定义数据行在联系人应用 UI 中的显示。它采用了以下语法：

```
<ContactsDataKind
    android:mimeType="MIMETYPE"
    android:icon="icon_resources"
    android:summaryColumn="column_name"
    android:detailColumn="column_name">
```

包含它的文件：

<ContactsAccountType>

说明：

此元素用于让联系人应用将自定义数据行的内容显示为原始联系人详细信息的一部分。<ContactsAccountType> 的每个 <ContactsDataKind> 子元素都代表您的同步适配器向 `ContactsContract.Data` 表添加的某个自定义数据行类型。请为您使用的每个自定义 MIME 类型添加一个 <ContactsDataKind> 元素。如果您不想显示任何自定义数据行的数据，则无需添加该元素。

属性：

android:mimeType

您为 `ContactsContract.Data` 表中某个自定义数据行类型定义的自定义 MIME 类型。例如，可将值 `vnd.android.cursor.item/vnd.example.locationstatus` 作为记录联系人最后已知位置的数据行的自定义 MIME 类型。

android:icon

联系人应用在您的数据旁显示的 Android [Drawable 资源](#)。它用于向用户指示数据来自您的服务。

android:summaryColumn

从数据行检索的两个值中第一个值的列名。该值显示为该数据行的第一个输入行。第一行专用作数据摘要，不过它是可选项。另请参阅 `android:detailColumn`。

android:detailColumn

从数据行检索的两个值中第二个值的列名。该值显示为该数据行的第二个输入行。另请参阅 `android:summaryColumn`。

其他联系人提供程序功能

除了上文描述的主要功能外，联系人提供程序还为处理联系人数据提供了下列有用的功能：

- 联系人组
- 照片功能

帐户八组

联系人提供程序可以选择性地为相关联系人集合添加组数据标签。如果与某个用户帐户关联的服务器想要维护组，则与该帐户的帐户类型对应的同步适配器应在联系人提供程序与服务器之间传送组数据。当用户向服务器添加一个新联系人，然后将该联系人放入一个新组时，同步适配器必须将这个新组添加到 `ContactsContract.Groups` 表中。原始联系人所属的一个或多个组使用 `ContactsContract.CommonDataKinds.GroupMembership` MIME 类型存储在 `ContactsContract.Data` 表内。

如果您设计的同步适配器会将服务器中的原始联系人数据添加到联系人提供程序，并且您不使用组，则需要指示提供程序让您的数据可见。在用户向设备添加帐户时执行的代码中，更新联系人提供程序为该帐户添加的 `ContactsContract.Settings` 行。在该行中，将 `Settings.UNGROUPED_VISIBLE` 列的值设置为 1。执行此操作后，即使您不使用组，联系人提供程序也会让您的联系人数据始终可见。

联系人照片

`ContactsContract.Data` 表通过 MIME 类型 `Photo.CONTENT_ITEM_TYPE` 以行的形式存储照片。该行的 `CONTACT_ID` 列链接到其所属原始联系人的 `_ID` 列。`ContactsContract.Contacts.Photo` 类定义了一个 `ContactsContract.Contacts` 子表，其中包含联系人主要照片（联系人的主要原始联系人的主要照片）的照片信息。同样，`ContactsContract.RawContacts.DisplayPhoto` 类定义了一个 `ContactsContract.RawContacts` 子表，其中包含原始联系人主要照片的照片信息。

`ContactsContract.Contacts.Photo` 和 `ContactsContract.RawContacts.DisplayPhoto` 参考文档包含检索照片信息的示例。并没有可用来检索原始联系人主要缩略图的实用类，但您可以向 `ContactsContract.Data` 表发送查询，从而通过选定原始联系人的 `_ID`、`Photo.CONTENT_ITEM_TYPE` 以及 `IS_PRIMARY` 列，找到原始联系人的主要照片行。

联系人的社交交流数据也可能包含照片。这些照片存储在 `android.provider.ContactsContract.StreamItemPhotos` 表中，[社交流照片](#) 部分对该表做了更详尽的描述。



存储访问框架

本文内容

[显示详细信息](#)

- › [概览](#)
- › [控制流](#)
- › [编写客户端应用](#)
- › [编写自定义文档提供程序](#)

关键类

- › [DocumentsProvider](#)
- › [DocumentsContract](#)

视频

- › [DevBytes：Android 4.4 存储访问框架：提供程序](#)
- › [DevBytes：Android 4.4 存储访问框架：客户端](#)

代码示例

- › [存储提供程序](#)
- › [StorageClient](#)

另请参阅

- › [内容提供程序基础知识](#)

Android 4.4 (API 级别 19) 引入了存储访问框架 (SAF)。SAF 让用户能够在其所有首选文档存储提供程序中方便地浏览并打开文档、图像以及其他文件。 用户可以通过易用的标准 UI，以统一方式在所有应用和提供程序中浏览文件和访问最近使用的文件。

云存储服务或本地存储服务可以通过实现封装其服务的 [DocumentsProvider](#) 参与此生态系统。只需几行代码，便可将需要访问提供程序文档的客户端应用与 SAF 集成。

SAF 包括以下内容：

- **文档提供程序** — 一种内容提供程序，允许存储服务（如 Google Drive）显示其管理的文件。 文档提供程序作为 [DocumentsProvider](#) 类的子类实现。 文档提供程序的架构基于传统文件层次结构，但其实际数据存储方式由您决定。 Android 平台包括若干内置文档提供程序，如 Downloads、Images 和 Videos。
- **客户端应用** — 一种自定义应用，它调用 [ACTION_OPEN_DOCUMENT](#) 和/或 [ACTION_CREATE_DOCUMENT](#) Intent 并接收文档提供程序返回的文件；
- **选取器** — 一种系统 UI，允许用户访问所有满足客户端应用搜索条件的文档提供程序内的文档。

SAF 提供的部分功能如下：

- 允许用户浏览所有文档提供程序而不仅仅是单个应用中的内容；
- 让您的应用获得对文档提供程序所拥有文档的长期、持久性访问权限。 用户可以通过此访问权限添加、编辑、保存和删除提供程序上的文件；
- 支持多个用户帐户和临时根目录，如只有在插入驱动器后才会出现的 USB 存储提供程序。

SAF 围绕的内容提供程序是 [DocumentsProvider](#) 类的一个子类。在文档提供程序内，数据结构采用传统的文件层次结构：

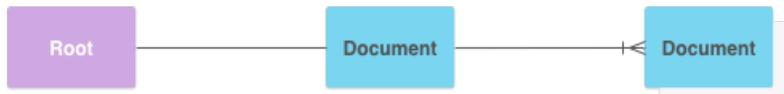


图 1. 文档提供程序数据模型。根目录指向单个文档，后者随即启动整个结构树的扇出。

请注意以下事项：

- 每个文档提供程序都会报告一个或多个作为探索文档结构树起点的“根目录”。每个根目录都有一个唯一的 `COLUMN_ROOT_ID`，并且指向表示该根目录下内容的文档（目录）。根目录采用动态设计，以支持多个帐户、临时 USB 存储设备或用户登录/注销等用例；
- 每个根目录下都有一个文档。该文档指向 1 至 N 个文档，而其中每个文档又可指向 1 至 N 个文档；
- 每个存储后端都会通过使用唯一的 `COLUMN_DOCUMENT_ID` 引用各个文件和目录来显示它们。文档 ID 必须具有唯一性，一旦发放便不得更改，因为它们用于所有设备重启过程中的永久性 URI 授权；
- 文档可以是可打开的文件（具有特定 MIME 类型）或包含附加文档的目录（具有 `MIME_TYPE_DIR` MIME 类型）；
- 每个文档都可以具有不同的功能，如 `COLUMN_FLAGS` 所述。例如，`FLAG_SUPPORTS_WRITE`、`FLAG_SUPPORTS_DELETE` 和 `FLAG_SUPPORTS_THUMBNAIL`。多个目录中可以包含相同的 `COLUMN_DOCUMENT_ID`。

控制流

如前文所述，文档提供程序数据模型基于传统文件层次结构。不过，只要可以通过 [DocumentsProvider](#) API 访问数据，您实际上可以按照自己喜好的任何方式存储数据。例如，您可以使用基于标记的云存储来存储数据。

图 2 中的示例展示的是照片应用如何利用 SAF 访问存储的数据：

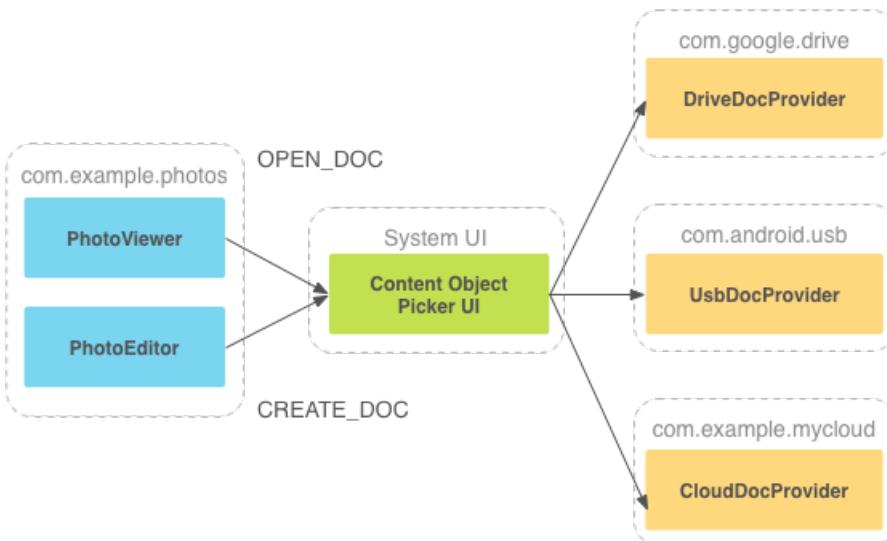


图 2. 存储访问框架流

请注意以下事项：

- 在 SAF 中，提供程序和客户端并不直接交互。客户端请求与文件交互（即读取、编辑、创建或删除文件）的权限；
- 交互在应用（在本示例中为照片应用）触发 Intent `ACTION_OPEN_DOCUMENT` 或 `ACTION_CREATE_DOCUMENT` 后开始。Intent 可能包括进一步细化条件的过滤器 — 例如，“为我提供所有 MIME 类型为‘图像’的可打开文件”；
- Intent 触发后，系统选取器将检索每个已注册的提供程序，并向用户显示匹配的内容根目录；
- 选取器会为用户提供一个标准的文档访问界面，但底层文档提供程序可能与其差异很大。例如，图 2 显示了一个 Google Drive 提供程序、一个 USB 提供程序和一个云提供程序。

图 3 显示了一个选取器，一位搜索图像的用户在其中选择了一个 Google Drive 帐户：

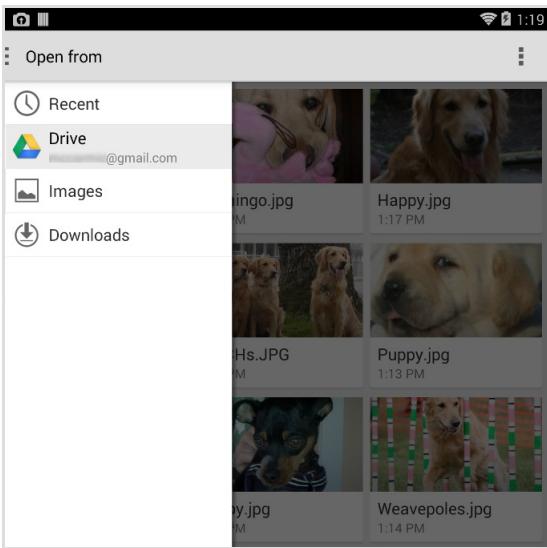


图 3. 选取器

当用户选择 Google Drive 时，系统会显示图像，如图 4 所示。从这时起，用户就可以通过提供程序和客户端应用支持的任何方式与它们进行交互。

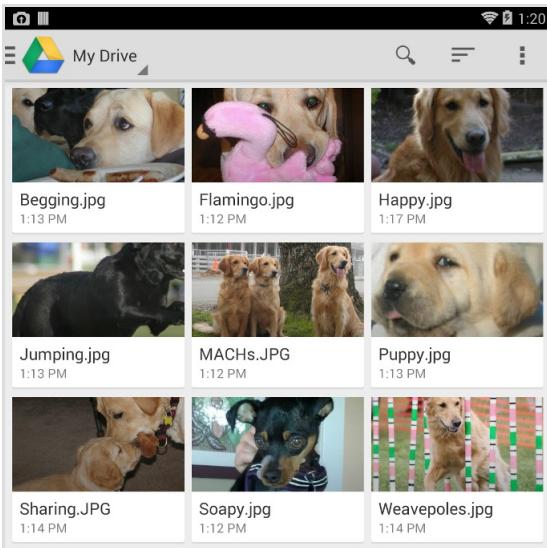


图 4. 图像

编写客户端应用

对于 Android 4.3 及更低版本，如果您想让应用从其他应用中检索文件，它必须调用 `ACTION_PICK` 或 `ACTION_GET_CONTENT` 等 Intent。然后，用户必须选择一个要从中选取文件的应用，并且所选应用必须提供一个用户界面，以便用户浏览和选取可用文件。

对于 Android 4.4 及更高版本，您还可以选择使用 `ACTION_OPEN_DOCUMENT` Intent，后者会显示一个由系统控制的选取器 UI，用户可以通过它浏览其他应用提供的所有文件。用户只需通过这一个 UI 便可从任何受支持的应用中选取文件。

`ACTION_OPEN_DOCUMENT` 并非设计用于替代 `ACTION_GET_CONTENT`。应使用的 Intent 取决于应用的需要：

- 如果您只想让应用读取/导入数据，请使用 `ACTION_GET_CONTENT`。使用此方法时，应用会导入数据（如图像文件）的副本；
- 如果您想让应用获得对文档提供程序所拥有文档的长期、持久性访问权限，请使用 `ACTION_OPEN_DOCUMENT`。例如，允许用户编辑存储在文档提供程序中的图像的照片编辑应用。

本节描述如何编写基于 `ACTION_OPEN_DOCUMENT` 和 `ACTION_CREATE_DOCUMENT` Intent 的客户端应用。

搜索文档

以下代码段使用 `ACTION_OPEN_DOCUMENT` 来搜索包含图像文件的文档提供程序：

```

private static final int READ_REQUEST_CODE = 42;
...
/**
 * Fires an intent to spin up the "file chooser" UI and select an image.
 */
public void performFileSearch() {

    // ACTION_OPEN_DOCUMENT is the intent to choose a file via the system's file
    // browser.
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    // Filter to only show results that can be "opened", such as a
    // file (as opposed to a list of contacts or timezones)
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Filter to show only images, using the image MIME data type.
    // If one wanted to search for ogg vorbis files, the type would be "audio/ogg".
    // To search for all documents available via installed storage providers,
    // it would be "*/*".
    intent.setType("image/*");

    startActivityForResult(intent, READ_REQUEST_CODE);
}

```

请注意以下事项：

- 当应用触发 `ACTION_OPEN_DOCUMENT` Intent 时，后者会启动一个选取器来显示所有匹配的文档提供程序
- 在 Intent 中添加类别 `CATEGORY_OPENABLE` 可对结果进行过滤，以仅显示可以打开的文档（如图像文件）
- 语句 `intent.setType("image/*")` 可做进一步过滤，以仅显示 MIME 数据类型为图像的文档

处理结果

用户在选取器中选择文档后，系统就会调用 `onActivityResult()`。指向所选文档的 URI 包含在 `resultData` 参数中。使用 `getData()` 提取 URI。获得 URI 后，即可使用它来检索用户想要的文档。例如：

```

@Override
public void onActivityResult(int requestCode, int resultCode,
    Intent resultData) {

    // The ACTION_OPEN_DOCUMENT intent was sent with the request code
    // READ_REQUEST_CODE. If the request code seen here doesn't match, it's the
    // response to some other intent, and the code below shouldn't run at all.

    if (requestCode == READ_REQUEST_CODE && resultCode == Activity.RESULT_OK) {
        // The document selected by the user won't be returned in the intent.
        // Instead, a URI to that document will be contained in the return intent
        // provided to this method as a parameter.
        // Pull that URI using resultData.getData().
        Uri uri = null;
        if (resultData != null) {
            uri = resultData.getData();
            Log.i(TAG, "Uri: " + uri.toString());
            showImage(uri);
        }
    }
}

```

检查文档元数据

获得文档的 URI 后，即可获得对其元数据的访问权限。以下代码段用于获取 URI 所指定文档的元数据并将其记入日志：

```
public void dumpImageMetaData(Uri uri) {  
    // The query, since it only applies to a single document, will only return  
    // one row. There's no need to filter, sort, or select fields, since we want  
    // all fields for one document.  
    Cursor cursor = getActivity().getContentResolver()  
        .query(uri, null, null, null, null);  
  
    try {  
        // moveToFirst() returns false if the cursor has 0 rows. Very handy for  
        // "if there's anything to look at, look at it" conditionals.  
        if (cursor != null && cursor.moveToFirst()) {  
  
            // Note it's called "Display Name". This is  
            // provider-specific, and might not necessarily be the file name.  
            String displayName = cursor.getString(  
                cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME));  
            Log.i(TAG, "Display Name: " + displayName);  
  
            int sizeIndex = cursor.getColumnIndex(OpenableColumns.SIZE);  
            // If the size is unknown, the value stored is null. But since an  
            // int can't be null in Java, the behavior is implementation-specific,  
            // which is just a fancy term for "unpredictable". So as  
            // a rule, check if it's null before assigning to an int. This will  
            // happen often: The storage API allows for remote files, whose  
            // size might not be locally known.  
            String size = null;  
            if (!cursor.isNull(sizeIndex)) {  
                // Technically the column stores an int, but cursor.getString()  
                // will do the conversion automatically.  
                size = cursor.getString(sizeIndex);  
            } else {  
                size = "Unknown";  
            }  
            Log.i(TAG, "Size: " + size);  
        }  
    } finally {  
        cursor.close();  
    }  
}
```

打开文档

获得文档的 URI 后，即可打开文档或对其执行任何其他您想要执行的操作。

位图

以下示例展示了如何打开 [Bitmap](#)：

```
private Bitmap getBitmapFromUri(Uri uri) throws IOException {  
    ParcelFileDescriptor parcelFileDescriptor =  
        getContentResolver().openFileDescriptor(uri, "r");  
    FileDescriptor fileDescriptor = parcelFileDescriptor.getFileDescriptor();  
    Bitmap image = BitmapFactory.decodeFileDescriptor(fileDescriptor);  
    parcelFileDescriptor.close();  
    return image;  
}
```

请注意，您不应在 UI 线程上执行此操作。请使用 [AsyncTask](#) 在后台执行此操作。打开位图后，即可在 [ImageView](#) 中显示它。

获取 InputStream

以下示例展示了如何从 URI 中获取 [InputStream](#)。在此代码段中，系统将文件行读取到一个字符串中：

```
private String readTextFromUri(Uri uri) throws IOException {
    InputStream inputStream = getContentResolver().openInputStream(uri);
    BufferedReader reader = new BufferedReader(new InputStreamReader(
        inputStream));
    StringBuilder stringBuilder = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        stringBuilder.append(line);
    }
    fileInputStream.close();
    parcelFileDescriptor.close();
    return stringBuilder.toString();
}
```

创建新文档

您的应用可以使用 `ACTION_CREATE_DOCUMENT` Intent 在文档提供程序中创建新文档。要想创建文件，请为您的 Intent 提供一个 MIME 类型和文件名，然后通过唯一的请求代码启动它。系统会为您执行其余操作：

```
// Here are some examples of how you might call this method.
// The first parameter is the MIME type, and the second parameter is the name
// of the file you are creating:
//
// createFile("text/plain", "foobar.txt");
// createFile("image/png", "mypicture.png");

// Unique request code.
private static final int WRITE_REQUEST_CODE = 43;
...
private void createFile(String mimeType, String fileName) {
    Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);

    // Filter to only show results that can be "opened", such as
    // a file (as opposed to a list of contacts or timezones).
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Create a file with the requested MIME type.
    intent.setType(mimeType);
    intent.putExtra(Intent.EXTRA_TITLE, fileName);
    startActivityForResult(intent, WRITE_REQUEST_CODE);
}
```

创建新文档后，即可在 `onActivityResult()` 中获取其 URI，以便继续向其写入内容。

删除文档

如果您获得了文档的 URI，并且文档的 `Document.COLUMN_FLAGS` 包含 `SUPPORTS_DELETE`，便可以删除该文档。例如：

```
DocumentsContract.deleteDocument(getContentResolver(), uri);
```

编辑文档

您可以使用 SAF 就地编辑文本文档。以下代码段会触发 `ACTION_OPEN_DOCUMENT` Intent 并使用类别 `CATEGORY_OPENABLE` 以仅显示可以打开的文档。它会进一步过滤以仅显示文本文件：

```

private static final int EDIT_REQUEST_CODE = 44;
/**
 * Open a file for writing and append some text to it.
 */
private void editDocument() {
    // ACTION_OPEN_DOCUMENT is the intent to choose a file via the system's
    // file browser.
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    // Filter to only show results that can be "opened", such as a
    // file (as opposed to a list of contacts or timezones).
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Filter to show only text files.
    intent.setType("text/plain");

    startActivityForResult(intent, EDIT_REQUEST_CODE);
}

```

接下来，您可以从 `onActivityResult()`（请参阅[处理结果](#)）调用代码以执行编辑。以下代码段可从 `ContentResolver` 获取 `FileOutputStream`。默认情况下，它使用“写入”模式。最佳做法是请求获得所需的最低限度访问权限，因此如果您只需要写入权限，就不要请求获得读取/写入权限。

```

private void alterDocument(Uri uri) {
    try {
        ParcelFileDescriptor pfd = getActivity().getContentResolver().
            openFileDescriptor(uri, "w");
        FileOutputStream fileOutputStream =
            new FileOutputStream(pfd.getFileDescriptor());
        fileOutputStream.write(("Overwritten by MyCloud at " +
            System.currentTimeMillis() + "\n").getBytes());
        // Let the document provider know you're done by closing the stream.
        fileOutputStream.close();
        pfd.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

保留权限

当您的应用打开文件进行读取或写入时，系统会为您的应用提供针对该文件的 URI 授权。该授权将一直持续到用户设备重启时。但假定您的应用是图像编辑应用，而且您希望用户能够直接从应用中访问他们编辑的最后 5 张图像。如果用户的设备已经重启，您就需要将用户转回系统选取器以查找这些文件，这显然不是理想的做法。

为防止出现这种情况，您可以保留系统为您的应用授予的权限。您的应用实际上是“获取”了系统提供的持久 URI 授权。这使用户能够通过您的应用持续访问文件，即使设备已重启也不受影响：

```

final int takeFlags = intent.getFlags()
    & (Intent.FLAG_GRANT_READ_URI_PERMISSION
    | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
// Check for the freshest data.
getContentResolver().takePersistableUriPermission(uri, takeFlags);

```

还有最后一个步骤。您可能已经保存了应用最近访问的 URI，但它们可能不再有效 — 另一个应用可能已删除或修改了文档。因此，您应该始终调用 `getContentResolver().takePersistableUriPermission()` 以检查有无最新数据。

编写自定义文档提供程序

如果您要开发为文件提供存储服务（如云保存服务）的应用，可以通过编写自定义文档提供程序，通过 SAF 提供您的文件。本节描述如何执行此操作。

清单文件

要想实现自定义文档提供程序，请将以下内容添加到您的应用的清单文件：

- 一个 API 级别 19 或更高级别的目标；
- 一个声明自定义存储提供程序的 `<provider>` 元素；
- 提供程序的名称（即其类名），包括软件包名称。例如：`com.example.android.storageprovider.MyCloudProvider`；
- 授权的名称，即您的软件包名称（在本例中为 `com.example.android.storageprovider`）加内容提供程序的类型 (`documents`)。例如，`com.example.android.storageprovider.documents`。
- 属性 `android:exported` 设置为 "true"。您必须导出提供程序，以便其他应用可以看到；
- 属性 `android:grantUriPermissions` 设置为 "true"。此设置允许系统向其他应用授予对提供程序中内容的访问权限。如需查看有关保留对特定文档授权的阐述，请参阅[保留权限](#)；
- `MANAGE_DOCUMENTS` 权限。默认情况下，提供程序对所有人可用。添加此权限将限定您的提供程序只能供系统使用。此限制具有重要的安全意义；
- `android:enabled` 属性设置为在资源文件中定义的一个布尔值。此属性的用途是，在运行 Android 4.3 或更低版本的设备上停用提供程序。例如，`android:enabled="@bool/atLeastKitKat"`。除了在清单文件中加入此属性外，您还需要执行以下操作：
 - 在 `res/values/` 下的 `bool.xml` 资源文件中，添加以下行：

```
<bool name="atLeastKitKat">false</bool>
```
 - 在 `res/values-v19/` 下的 `bool.xml` 资源文件中，添加以下行：

```
<bool name="atLeastKitKat">true</bool>
```

- 一个包括 `android.content.action.DOCUMENTS_PROVIDER` 操作的 Intent 过滤器，以便在系统搜索提供程序时让您的提供程序出现在选取器中。

以下是从一个包括提供程序的示例清单文件中摘录的内容：

```
<manifest... >
    ...
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />
    ...
    <provider
        android:name="com.example.android.storageprovider.MyCloudProvider"
        android:authorities="com.example.android.storageprovider.documents"
        android:grantUriPermissions="true"
        android:exported="true"
        android:permission="android.permission.MANAGE_DOCUMENTS"
        android:enabled="@bool/atLeastKitKat">
        <intent-filter>
            <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
        </intent-filter>
    </provider>
</application>

</manifest>
```

支持运行 Android 4.3 及更低版本的设备

`ACTION_OPEN_DOCUMENT` Intent 仅可用于运行 Android 4.4 及更高版本的设备。如果您想让应用支持 `ACTION_GET_CONTENT` 以适应运行 Android 4.3 及更低版本的设备，则应在您的清单文件中为运行 Android 4.4 或更高版本的设备停用 `ACTION_GET_CONTENT` Intent 过滤器。应将文档提供程序和 `ACTION_GET_CONTENT` 视为具有互斥性。如果您同时支持这两者，您的应用将在系统选取器 UI 中出现两次，提供两种不同的方式来访问您存储的数据。这会给用户造成困惑。

建议按照以下步骤为运行 Android 4.4 版或更高版本的设备停用 `ACTION_GET_CONTENT` Intent 过滤器：

1. 在 `res/values/` 下的 `bool.xml` 资源文件中，添加以下行：

```
<bool name="atMostJellyBeanMR2">true</bool>
```

2. 在 `res/values-v19/` 下的 `bool.xml` 资源文件中，添加以下行：

```
<bool name="atMostJellyBeanMR2">false</bool>
```

3. 添加一个 `Activity 别名`，为 4.4 版 (API 级别 19) 或更高版本停用 `ACTION_GET_CONTENT Intent` 过滤器。例如：

```
<!-- This activity alias is added so that GET_CONTENT intent-filter
     can be disabled for builds on API level 19 and higher. -->
<activity-alias android:name="com.android.example.app.MyPicker"
    android:targetActivity="com.android.example.app.MyActivity"
    ...
    android:enabled="@bool/atMostJellyBeanMR2">
    <intent-filter>
        <action android:name="android.intent.action.GET_CONTENT" />
        <category android:name="android.intent.category.OPENABLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
        <data android:mimeType="video/*" />
    </intent-filter>
</activity-alias>
```

协定类

通常，当您编写自定义内容提供程序时，其中一项任务是实现协定类，如[内容提供程序开发者指南](#)中所述。协定类是一种 `public final` 类，它包含对 URI、列名称、MIME 类型以及其他与提供程序有关的元数据的常量定义。SAF 会为您提供这些协定类，因此您无需自行编写：

- `DocumentsContract.Document`
- `DocumentsContract.Root`

例如，当系统在您的文档提供程序中查询文档或根目录时，您可能会在游标中返回以下列：

```
private static final String[] DEFAULT_ROOT_PROJECTION =
    new String[]{Root.COLUMN_ROOT_ID, Root.COLUMN_MIME_TYPES,
    Root.COLUMN_FLAGS, Root.COLUMN_ICON, Root.COLUMN_TITLE,
    Root.COLUMN_SUMMARY, Root.COLUMN_DOCUMENT_ID,
    Root.COLUMN_AVAILABLE_BYTES,};
private static final String[] DEFAULT_DOCUMENT_PROJECTION = new
    String[]{Document.COLUMN_DOCUMENT_ID, Document.COLUMN_MIME_TYPE,
    Document.COLUMN_DISPLAY_NAME, Document.COLUMN_LAST_MODIFIED,
    Document.COLUMN_FLAGS, Document.COLUMN_SIZE,};
```

为 `DocumentsProvider` 创建子类

编写自定义文档提供程序的下一步是为抽象类 `DocumentsProvider` 创建子类。您至少需要实现以下方法：

- `queryRoots()`
- `queryChildDocuments()`
- `queryDocument()`
- `openDocument()`

这些只是您需要严格实现的方法，但您可能还想实现许多其他方法。详情请参阅 [DocumentsProvider](#)。

实现 `queryRoots`

您实现的 `queryRoots()` 必须使用在 `DocumentsContract.Root` 中定义的列返回一个指向文档提供程序所有根目录的 `Cursor`。

在以下代码段中，`projection` 参数表示调用方想要返回的特定字段。代码段会创建一个新游标，并为其添加一行 — 一个根目录，如

Downloads 或 Images 等顶层目录。大多数提供程序只有一个根目录。有时您可能有多个根目录，例如，当您具有多个用户帐户时。在这种情况下，只需再为游标添加一行。

```
@Override
public Cursor queryRoots(String[] projection) throws FileNotFoundException {
    // Create a cursor with either the requested fields, or the default
    // projection if "projection" is null.
    final MatrixCursor result =
        new MatrixCursor(resolveRootProjection(projection));

    // If user is not logged in, return an empty root cursor. This removes our
    // provider from the list entirely.
    if (!isUserLoggedIn()) {
        return result;
    }

    // It's possible to have multiple roots (e.g. for multiple accounts in the
    // same app) -- just add multiple cursor rows.
    // Construct one row for a root called "MyCloud".
    final MatrixCursor.RowBuilder row = result.newRow();
    row.add(Root.COLUMN_ROOT_ID, ROOT);
    row.add(Root.COLUMN_SUMMARY, getContext().getString(R.string.root_summary));

    // FLAG_SUPPORTS_CREATE means at least one directory under the root supports
    // creating documents. FLAG_SUPPORTS_RECENTS means your application's most
    // recently used documents will show up in the "Recents" category.
    // FLAG_SUPPORTS_SEARCH allows users to search all documents the application
    // shares.
    row.add(Root.COLUMN_FLAGS, Root.FLAG_SUPPORTS_CREATE |
        Root.FLAG_SUPPORTS_RECENTS |
        Root.FLAG_SUPPORTS_SEARCH);

    // COLUMN_TITLE is the root title (e.g. Gallery, Drive).
    row.add(Root.COLUMN_TITLE, getContext().getString(R.string.title));

    // This document id cannot change once it's shared.
    row.add(Root.COLUMN_DOCUMENT_ID, getDocIdForFile(mBaseDir));

    // The child MIME types are used to filter the roots and only present to the
    // user roots that contain the desired type somewhere in their file hierarchy.
    row.add(Root.COLUMN_MIME_TYPES, getChildMimeTypes(mBaseDir));
    row.add(Root.COLUMN_AVAILABLE_BYTES, mBaseDir.getFreeSpace());
    row.add(Root.COLUMN_ICON, R.drawable.ic_launcher);

    return result;
}
```

实现 queryChildDocuments

您实现的 `queryChildDocuments()` 必须使用在 `DocumentsContract.Document` 中定义的列返回一个指向指定目录中所有文件的 `Cursor`。

当您在选取器 UI 中选择应用时，系统会调用此方法。它会获取根目录下某个目录内的子文档。可以在文件层次结构的任何级别调用此方法，并非只能从根目录调用。以下代码段可创建一个包含所请求列的新游标，然后向游标添加父目录中每个直接子目录的相关信息。子目录可以是图像、另一个目录乃至任何文件：

```
@Override
public Cursor queryChildDocuments(String parentDocumentId, String[] projection,
                                    String sortOrder) throws FileNotFoundException {

    final MatrixCursor result = new
        MatrixCursor(resolveDocumentProjection(projection));
    final File parent = getFileForDocId(parentDocumentId);
    for (File file : parent.listFiles()) {
        // Adds the file's display name, MIME type, size, and so on.
        includeFile(result, null, file);
    }
    return result;
}
```

实现 queryDocument

您实现的 `queryDocument()` 必须使用在 `DocumentsContract.Document` 中定义的列返回一个指向指定文件的 `Cursor`。

除了特定文件的信息外，`queryDocument()` 方法返回的信息与 `queryChildDocuments()` 中传递的信息相同：

```
@Override
public Cursor queryDocument(String documentId, String[] projection) throws
    FileNotFoundException {

    // Create a cursor with the requested projection, or the default projection.
    final MatrixCursor result = new
        MatrixCursor(resolveDocumentProjection(projection));
    includeFile(result, documentId, null);
    return result;
}
```

实现 queryDocument

您必须实现 `openDocument()` 以返回表示指定文件的 `ParcelFileDescriptor`。其他应用可以使用返回的 `ParcelFileDescriptor` 来流式传输数据。用户选择了文件，并且客户端应用通过调用 `openFileDescriptor()` 来请求对文件的访问权限后，系统便会调用此方法。例如：

```

@Override
public ParcelFileDescriptor openDocument(final String documentId,
                                         final String mode,
                                         CancellationSignal signal) throws
    FileNotFoundException {
    Log.v(TAG, "openDocument, mode: " + mode);
    // It's OK to do network operations in this method to download the document,
    // as long as you periodically check the CancellationSignal. If you have an
    // extremely large file to transfer from the network, a better solution may
    // be pipes or sockets (see ParcelFileDescriptor for helper methods).

    final File file = getFileForDocId(documentId);

    final boolean isWrite = (mode.indexOf('w') != -1);
    if(isWrite) {
        // Attach a close listener if the document is opened in write mode.
        try {
            Handler handler = new Handler(getContext().getMainLooper());
            return ParcelFileDescriptor.open(file, accessMode, handler,
                new ParcelFileDescriptor.OnCloseListener() {
                    @Override
                    public void onClose(IOException e) {

                        // Update the file with the cloud server. The client is done
                        // writing.
                        Log.i(TAG, "A file with id " +
                            documentId + " has been closed!
                            Time to " +
                            "update the server.");
                    }
                });
        } catch (IOException e) {
            throw new FileNotFoundException("Failed to open document with id "
                + documentId + " and mode " + mode);
        }
    } else {
        return ParcelFileDescriptor.open(file, accessMode);
    }
}

```

安全性

假设您的文档提供程序是受密码保护的云存储服务，并且您想在开始共享用户的文件之前确保其已登录。如果用户未登录，您的应用应该执行哪些操作呢？解决方案是在您实现的 `queryRoots()` 中返回零个根目录。也就是空的根目录游标：

```

public Cursor queryRoots(String[] projection) throws FileNotFoundException {
    ...
    // If user is not logged in, return an empty root cursor. This removes our
    // provider from the list entirely.
    if (!isUserLoggedIn()) {
        return result;
    }
}

```

另一个步骤是调用 `getContentResolver().notifyChange()`。还记得 `DocumentsContract` 吗？我们将使用它来创建此 URI。以下代码段会在每次用户的登录状态发生变化时指示系统查询文档提供程序的根目录。如果用户未登录，则调用 `queryRoots()` 会返回一个空游标，如上文所示。这可以确保只有在用户登录提供程序后其中的文档才可用。

```

private void onLoginButtonClick() {
    loginOrLogout();
    getContentResolver().notifyChange(DocumentsContract
        .buildRootsUri(AUTHORITY), null);
}

```

App Widgets

In this document

- › [The Basics](#)
- › [Declaring an App Widget in the Manifest](#)
- › [Adding the AppWidgetProviderInfo Metadata](#)
- › [Creating the App Widget Layout](#)
- › [Using the AppWidgetProvider Class](#)
 - › [Receiving App Widget broadcast Intents](#)
 - › [Creating an App Widget Configuration Activity](#)
 - › [Updating the App Widget from the Configuration Activity](#)
- › [Setting a Preview Image](#)
- › [Using App Widgets with Collections](#)
 - › [Sample application](#)
 - › [Implementing app widgets with collections](#)
 - › [Keeping Collection Data Fresh](#)

Key classes

- › [AppWidgetProvider](#)
- › [AppWidgetProviderInfo](#)
- › [AppWidgetManager](#)

App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates. These views are referred to as Widgets in the user interface, and you can publish one with an App Widget provider. An application component that is able to hold other App Widgets is called an App Widget host. The screenshot below shows the Music App Widget.



This document describes how to publish an App Widget using an App Widget provider. For a discussion of creating your own [AppWidgetHost](#) to host app widgets, see [App Widget Host](#).

Widget Design

For information about how to design your app widget, read the [Widgets](#) design guide.

The Basics

To create an App Widget, you need the following:

[AppWidgetProviderInfo](#) object

Describes the metadata for an App Widget, such as the App Widget's layout, update frequency, and the AppWidgetProvider class. This should be defined in XML.

[AppWidgetProvider](#) class implementation

Defines the basic methods that allow you to programmatically interface with the App Widget, based on broadcast events. Through it, you will receive broadcasts when the App Widget is updated, enabled, disabled and deleted.

View layout

Defines the initial layout for the App Widget, defined in XML.

Additionally, you can implement an App Widget configuration Activity. This is an optional [Activity](#) that launches when the user adds your App Widget and allows him or her to modify App Widget settings at create-time.

The following sections describe how to set up each of these components.

Declaring an App Widget in the Manifest

First, declare the [AppWidgetProvider](#) class in your application's [AndroidManifest.xml](#) file. For example:

```
<receiver android:name="ExampleAppWidgetProvider" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data android:name="android.appwidget.provider"
              android:resource="@xml/example_appwidget_info" />
</receiver>
```

The `<receiver>` element requires the `android:name` attribute, which specifies the [AppWidgetProvider](#) used by the App Widget.

The `<intent-filter>` element must include an `<action>` element with the `android:name` attribute. This attribute specifies that the [AppWidgetProvider](#) accepts the `ACTION_APPWIDGET_UPDATE` broadcast. This is the only broadcast that you must explicitly declare. The [AppWidgetManager](#) automatically sends all other App Widget broadcasts to the AppWidgetProvider as necessary.

The `<meta-data>` element specifies the [AppWidgetProviderInfo](#) resource and requires the following attributes:

- `android:name` - Specifies the metadata name. Use `android.appwidget.provider` to identify the data as the [AppWidgetProviderInfo](#) descriptor.
- `android:resource` - Specifies the [AppWidgetProviderInfo](#) resource location.

Adding the AppWidgetProviderInfo Metadata

The [AppWidgetProviderInfo](#) defines the essential qualities of an App Widget, such as its minimum layout dimensions, its initial layout resource, how often to update the App Widget, and (optionally) a configuration Activity to launch at create-time. Define the [AppWidgetProviderInfo](#) object in an XML resource using a single `<appwidget-provider>` element and save it in the project's `res/xml/` folder.

For example:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen">
</appwidget-provider>
```

Here's a summary of the `<appwidget-provider>` attributes:

- The values for the `minWidth` and `minHeight` attributes specify the minimum amount of space the App Widget consumes by default. The

default Home screen positions App Widgets in its window based on a grid of cells that have a defined height and width. If the values for an App Widget's minimum width or height don't match the dimensions of the cells, then the App Widget dimensions round *up* to the nearest cell size.

See the [App Widget Design Guidelines](#) for more information on sizing your App Widgets.

Note: To make your app widget portable across devices, your app widget's minimum size should never be larger than 4 x 4 cells.

- The `minResizeWidth` and `minResizeHeight` attributes specify the App Widget's absolute minimum size. These values should specify the size below which the App Widget would be illegible or otherwise unusable. Using these attributes allows the user to resize the widget to a size that may be smaller than the default widget size defined by the `minWidth` and `minHeight` attributes. Introduced in Android 3.1.

See the [App Widget Design Guidelines](#) for more information on sizing your App Widgets.

- The `updatePeriodMillis` attribute defines how often the App Widget framework should request an update from the `AppWidgetProvider` by calling the `onUpdate()` callback method. The actual update is not guaranteed to occur exactly on time with this value and we suggest updating as infrequently as possible—perhaps no more than once an hour to conserve the battery. You might also allow the user to adjust the frequency in a configuration—some people might want a stock ticker to update every 15 minutes, or maybe only four times a day.

Note: If the device is asleep when it is time for an update (as defined by `updatePeriodMillis`), then the device will wake up in order to perform the update. If you don't update more than once per hour, this probably won't cause significant problems for the battery life. If, however, you need to update more frequently and/or you do not need to update while the device is asleep, then you can instead perform updates based on an alarm that will not wake the device. To do so, set an alarm with an Intent that your `AppWidgetProvider` receives, using the `AlarmManager`. Set the alarm type to either `ELAPSED_REALTIME` or `RTC`, which will only deliver the alarm when the device is awake. Then set `updatePeriodMillis` to zero ("0").

- The `initialLayout` attribute points to the layout resource that defines the App Widget layout.
- The `configure` attribute defines the `Activity` to launch when the user adds the App Widget, in order for him or her to configure App Widget properties. This is optional (read [Creating an App Widget Configuration Activity](#) below).
- The `previewImage` attribute specifies a preview of what the app widget will look like after it's configured, which the user sees when selecting the app widget. If not supplied, the user instead sees your application's launcher icon. This field corresponds to the `android:previewImage` attribute in the `<receiver>` element in the `AndroidManifest.xml` file. For more discussion of using `previewImage`, see [Setting a Preview Image](#). Introduced in Android 3.0.
- The `autoAdvanceViewId` attribute specifies the view ID of the app widget subview that should be auto-advanced by the widget's host. Introduced in Android 3.0.
- The `resizeMode` attribute specifies the rules by which a widget can be resized. You use this attribute to make homescreen widgets resizable—horizontally, vertically, or on both axes. Users touch-hold a widget to show its resize handles, then drag the horizontal and/or vertical handles to change the size on the layout grid. Values for the `resizeMode` attribute include "horizontal", "vertical", and "none". To declare a widget as resizable horizontally and vertically, supply the value "horizontal|vertical". Introduced in Android 3.1.
- The `minResizeHeight` attribute specifies the minimum height (in dps) to which the widget can be resized. This field has no effect if it is greater than `minHeight` or if vertical resizing isn't enabled (see `resizeMode`). Introduced in Android 4.0.
- The `minResizeWidth` attribute specifies the minimum width (in dps) to which the widget can be resized. This field has no effect if it is greater than `minWidth` or if horizontal resizing isn't enabled (see `resizeMode`). Introduced in Android 4.0.
- The `widgetCategory` attribute declares whether your App Widget can be displayed on the home screen (`home_screen`), the lock screen (`keyguard`), or both. Only Android versions lower than 5.0 support lock-screen widgets. For Android 5.0 and higher, only `home_screen` is valid.

See the `AppWidgetProviderInfo` class for more information on the attributes accepted by the `<appwidget-provider>` element.

Creating the App Widget Layout

You must define an initial layout for your App Widget in XML and save it in the project's `res/layout/` directory. You can design your App Widget using the View objects listed below, but before you begin designing your App Widget, please read and understand the [App Widget](#)

[Design Guidelines](#).

Creating the App Widget layout is simple if you're familiar with [Layouts](#). However, you must be aware that App Widget layouts are based on [RemoteViews](#), which do not support every kind of layout or view widget.

A `RemoteViews` object (and, consequently, an App Widget) can support the following layout classes:

[FrameLayout](#)

[LinearLayout](#)

[RelativeLayout](#)

[GridLayout](#)

And the following widget classes:

[AnalogClock](#)

[Button](#)

[Chronometer](#)

[ImageButton](#)

[ImageView](#)

[ProgressBar](#)

[TextView](#)

[ViewFlipper](#)

[ListView](#)

[GridView](#)

[StackView](#)

[AdapterViewFlipper](#)

Descendants of these classes are not supported.

RemoteViews also supports [ViewStub](#), which is an invisible, zero-sized View you can use to lazily inflate layout resources at runtime.

Adding margins to App Widgets

Widgets should not generally extend to screen edges and should not visually be flush with other widgets, so you should add margins on all sides around your widget frame.

As of Android 4.0, app widgets are automatically given padding between the widget frame and the app widget's bounding box to provide better alignment with other widgets and icons on the user's home screen. To take advantage of this strongly recommended behavior, set your application's `targetSdkVersion` to 14 or greater.

It's easy to write a single layout that has custom margins applied for earlier versions of the platform, and has no extra margins for Android 4.0 and greater:

1. Set your application's `targetSdkVersion` to 14 or greater.
2. Create a layout such as the one below, that references a [dimension resource](#) for its margins:

```
<FrameLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:padding="@dimen/widget_margin">  
  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:orientation="horizontal"  
        android:background="@drawable/my_widget_background">  
        ...  
    </LinearLayout>  
  
</FrameLayout>
```

3. Create two dimensions resources, one in `res/values/` to provide the pre-Android 4.0 custom margins, and one in `res/values-v14/` to provide no extra padding for Android 4.0 widgets:

`res/values/dimens.xml`:

```
<dimen name="widget_margin">8dp</dimen>
```

`res/values-v14/dimens.xml`:

```
<dimen name="widget_margin">0dp</dimen>
```

Another option is to simply build extra margins into your [nine-patch](#) background assets by default, and provide different nine-patches with no margins for API level 14 or later.

Using the AppWidgetProvider Class

The [AppWidgetProvider](#) class extends BroadcastReceiver as a convenience class to handle the App Widget broadcasts. The AppWidgetProvider receives only the event broadcasts that are relevant to the App Widget, such as when the App Widget is updated, deleted, enabled, and disabled. When these broadcast events occur, the AppWidgetProvider receives the following method calls:

`onUpdate()`

This is called to update the App Widget at intervals defined by the `updatePeriodMillis` attribute in the AppWidgetProviderInfo (see [Adding the AppWidgetProviderInfo Metadata](#) above). This method is also called when the user adds the App Widget, so it should perform the essential setup, such as define event handlers for Views and start a temporary [Service](#), if necessary. However, if you have declared a configuration Activity, **this method is not called** when the user adds the App Widget, but is called for the subsequent updates. It is the responsibility of the configuration Activity to perform the first update when configuration is done. (See [Creating an App Widget Configuration Activity](#) below.)

`onAppWidgetOptionsChanged()`

This is called when the widget is first placed and any time the widget is resized. You can use this callback to show or hide content based on the widget's size ranges. You get the size ranges by calling `getAppWidgetOptions()`, which returns a `Bundle` that includes the following:

- `OPTION_APPWIDGET_MIN_WIDTH`—Contains the lower bound on the current width, in dp units, of a widget instance.
- `OPTION_APPWIDGET_MIN_HEIGHT`—Contains the lower bound on the current height, in dp units, of a widget instance.
- `OPTION_APPWIDGET_MAX_WIDTH`—Contains the upper bound on the current width, in dp units, of a widget instance.
- `OPTION_APPWIDGET_MAX_HEIGHT`—Contains the upper bound on the current width, in dp units, of a widget instance.

You must declare your AppWidgetProvider class implementation as a broadcast receiver using the `<receiver>` element in the AndroidManifest (see [Declaring an App Widget in the Manifest](#) above).

This callback was introduced in API Level 16 (Android 4.1). If you implement this callback, make sure that your app doesn't depend on

it since it won't be called on older devices.

`onDeleted(Context, int[])`

This is called every time an App Widget is deleted from the App Widget host.

`onEnabled(Context)`

This is called when an instance the App Widget is created for the first time. For example, if the user adds two instances of your App Widget, this is only called the first time. If you need to open a new database or perform other setup that only needs to occur once for all App Widget instances, then this is a good place to do it.

`onDisabled(Context)`

This is called when the last instance of your App Widget is deleted from the App Widget host. This is where you should clean up any work done in `onEnabled(Context)`, such as delete a temporary database.

`onReceive(Context, Intent)`

This is called for every broadcast and before each of the above callback methods. You normally don't need to implement this method because the default AppWidgetProvider implementation filters all App Widget broadcasts and calls the above methods as appropriate.

The most important AppWidgetProvider callback is `onUpdate()` because it is called when each App Widget is added to a host (unless you use a configuration Activity). If your App Widget accepts any user interaction events, then you need to register the event handlers in this callback. If your App Widget doesn't create temporary files or databases, or perform other work that requires clean-up, then `onUpdate()` may be the only callback method you need to define. For example, if you want an App Widget with a button that launches an Activity when clicked, you could use the following implementation of AppWidgetProvider:

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {

    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        final int N = appWidgetIds.length;

        // Perform this loop procedure for each App Widget that belongs to this provider
        for (int i=0; i<N; i++) {
            int appWidgetId = appWidgetIds[i];

            // Create an Intent to launch ExampleActivity
            Intent intent = new Intent(context, ExampleActivity.class);
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);

            // Get the layout for the App Widget and attach an on-click listener
            // to the button
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.appwidget_provider_layout);
            views.setOnClickPendingIntent(R.id.button, pendingIntent);

            // Tell the AppWidgetManager to perform an update on the current app widget
            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```

This AppWidgetProvider defines only the `onUpdate()` method for the purpose of defining a `PendingIntent` that launches an `Activity` and attaching it to the App Widget's button with `setOnClickPendingIntent(int, PendingIntent)`. Notice that it includes a loop that iterates through each entry in `appWidgetIds`, which is an array of IDs that identify each App Widget created by this provider. In this way, if the user creates more than one instance of the App Widget, then they are all updated simultaneously. However, only one `updatePeriodMillis` schedule will be managed for all instances of the App Widget. For example, if the update schedule is defined to be every two hours, and a second instance of the App Widget is added one hour after the first one, then they will both be updated on the period defined by the first one and the second update period will be ignored (they'll both be updated every two hours, not every hour).

Note: Because `AppWidgetProvider` is an extension of `BroadcastReceiver`, your process is not guaranteed to keep running after the callback methods return (see `BroadcastReceiver` for information about the broadcast lifecycle). If your App Widget setup process can

take several seconds (perhaps while performing web requests) and you require that your process continues, consider starting a [Service](#) in the `onUpdate()` method. From within the Service, you can perform your own updates to the App Widget without worrying about the AppWidgetProvider closing down due to an [Application Not Responding](#) (ANR) error. See the [Wiktionary sample's AppWidgetProvider](#) for an example of an App Widget running a [Service](#).

Also see the [ExampleAppWidgetProvider.java](#) sample class.

Receiving App Widget broadcast Intents

[AppWidgetProvider](#) is just a convenience class. If you would like to receive the App Widget broadcasts directly, you can implement your own [BroadcastReceiver](#) or override the `onReceive(Context, Intent)` callback. The Intents you need to care about are as follows:

- `ACTION_APPWIDGET_UPDATE`
- `ACTION_APPWIDGET_DELETED`
- `ACTION_APPWIDGET_ENABLED`
- `ACTION_APPWIDGET_DISABLED`
- `ACTION_APPWIDGET_OPTIONS_CHANGED`

Creating an App Widget Configuration Activity

If you would like the user to configure settings when he or she adds a new App Widget, you can create an App Widget configuration Activity. This [Activity](#) will be automatically launched by the App Widget host and allows the user to configure available settings for the App Widget at create-time, such as the App Widget color, size, update period or other functionality settings.

The configuration Activity should be declared as a normal Activity in the Android manifest file. However, it will be launched by the App Widget host with the `ACTION_APPWIDGET_CONFIGURE` action, so the Activity needs to accept this Intent. For example:

```
<activity android:name=".ExampleAppWidgetConfigure">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
    </intent-filter>
</activity>
```

Also, the Activity must be declared in the `AppWidgetProviderInfo` XML file, with the `android:configure` attribute (see [Adding the AppWidgetProviderInfo Metadata](#) above). For example, the configuration Activity can be declared like this:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    ... >
</appwidget-provider>
```

Notice that the Activity is declared with a fully-qualified namespace, because it will be referenced from outside your package scope.

That's all you need to get started with a configuration Activity. Now all you need is the actual Activity. There are, however, two important things to remember when you implement the Activity:

- The App Widget host calls the configuration Activity and the configuration Activity should always return a result. The result should include the App Widget ID passed by the Intent that launched the Activity (saved in the Intent extras as `EXTRA_APPWIDGET_ID`).
- The `onUpdate()` method **will not be called** when the App Widget is created (the system will not send the `ACTION_APPWIDGET_UPDATE` broadcast when a configuration Activity is launched). It is the responsibility of the configuration Activity to request an update from the `AppWidgetManager` when the App Widget is first created. However, `onUpdate()` will be called for subsequent updates—it is only skipped the first time.

See the code snippets in the following section for an example of how to return a result from the configuration and update the App Widget.

Updating the App Widget from the configuration Activity

When an App Widget uses a configuration Activity, it is the responsibility of the Activity to update the App Widget when configuration is complete. You can do so by requesting an update directly from the [AppWidgetManager](#).

Here's a summary of the procedure to properly update the App Widget and close the configuration Activity:

1. First, get the App Widget ID from the Intent that launched the Activity:

```
Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
    mAppWidgetId = extras.getInt(
        AppWidgetManager.EXTRA_APPWIDGET_ID,
        AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

2. Perform your App Widget configuration.

3. When the configuration is complete, get an instance of the AppWidgetManager by calling `getInstance(Context)`:

```
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

4. Update the App Widget with a `RemoteViews` layout by calling `updateAppWidget(int, RemoteViews)`:

```
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.example_appwidget);
appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

5. Finally, create the return Intent, set it with the Activity result, and finish the Activity:

```
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);
finish();
```

Tip: When your configuration Activity first opens, set the Activity result to `RESULT_CANCELED`, along with `EXTRA_APPWIDGET_ID`, as shown in step 5 above. This way, if the user backs-out of the Activity before reaching the end, the App Widget host is notified that the configuration was cancelled and the App Widget will not be added.

See the [ExampleAppWidgetConfigure.java](#) sample class in ApiDemos for an example.

Setting a Preview Image

Android 3.0 introduces the `previewImage` field, which specifies a preview of what the app widget looks like. This preview is shown to the user from the widget picker. If this field is not supplied, the app widget's icon is used for the preview.

This is how you specify this setting in XML:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
...
    android:previewImage="@drawable/preview"
</appwidget-provider>
```

To help create a preview image for your app widget (to specify in the `previewImage` field), the Android emulator includes an application called "Widget Preview." To create a preview image, launch this application, select the app widget for your application and set it up how you'd like your preview image to appear, then save it and place it in your application's drawable resources.

Using App Widgets with Collections

Android 3.0 introduces app widgets with collections. These kinds of App Widgets use the `RemoteViewsService` to display collections that are backed by remote data, such as from a `content provider`. The data provided by the `RemoteViewsService` is presented in the app widget

using one of the following view types, which we'll refer to as "collection views:"

[ListView](#)

A view that shows items in a vertically scrolling list. For an example, see the Gmail app widget.

[GridView](#)

A view that shows items in two-dimensional scrolling grid. For an example, see the Bookmarks app widget.

[StackView](#)

A stacked card view (kind of like a rolodex), where the user can flick the front card up/down to see the previous/next card, respectively. Examples include the YouTube and Books app widgets.

[AdapterViewFlipper](#)

An adapter-backed simple [ViewAnimator](#) that animates between two or more views. Only one child is shown at a time.

As stated above, these collection views display collections backed by remote data. This means that they use an [Adapter](#) to bind their user interface to their data. An [Adapter](#) binds individual items from a set of data into individual [View](#) objects. Because these collection views are backed by adapters, the Android framework must include extra architecture to support their use in app widgets. In the context of an app widget, the [Adapter](#) is replaced by a [RemoteViewsFactory](#), which is simply a thin wrapper around the [Adapter](#) interface. When requested for a specific item in the collection, the [RemoteViewsFactory](#) creates and returns the item for the collection as a [RemoteViews](#) object. In order to include a collection view in your app widget, you must implement [RemoteViewsService](#) and [RemoteViewsFactory](#).

[RemoteViewsService](#) is a service that allows a remote adapter to request [RemoteViews](#) objects. [RemoteViewsFactory](#) is an interface for an adapter between a collection view (such as [ListView](#), [GridView](#), and so on) and the underlying data for that view. From the [StackView Widget sample](#), here is an example of the boilerplate code you use to implement this service and interface:

```
public class StackWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new StackRemoteViewsFactory(this.getApplicationContext(), intent);
    }
}

class StackRemoteViewsFactory implements RemoteViewsService.RemoteViewsFactory {
    //... include adapter-like methods here. See the StackView Widget sample.
}
```

Sample application

The code excerpts in this section are drawn from the [StackView Widget sample](#):



This sample consists of a stack of 10 views, which display the values "0!" through "9!". The sample app widget has these primary behaviors:

- The user can vertically flick the top view in the app widget to display the next or previous view. This is a built-in StackView behavior.
- Without any user interaction, the app widget automatically advances through its views in sequence, like a slide show. This is due to the

setting `android:autoAdvanceViewId="@+id/stack_view"` in the `res/xml/stackwidgetinfo.xml` file. This setting applies to the view ID, which in this case is the view ID of the stack view.

- If the user touches the top view, the app widget displays the `Toast` message "Touched view *n*," where *n* is the index (position) of the touched view. For more discussion of how this is implemented, see [Adding behavior to individual items](#).

Implementing app widgets with collections

To implement an app widget with collections, you follow the same basic steps you would use to implement any app widget. The following sections describe the additional steps you need to perform to implement an app widget with collections.

Manifest for app widgets with collections

In addition to the requirements listed in [Declaring an app widget in the Manifest](#), to make it possible for app widgets with collections to bind to your `RemoteViewsService`, you must declare the service in your manifest file with the permission `BIND_REMOTEVIEWS`. This prevents other applications from freely accessing your app widget's data. For example, when creating an App Widget that uses `RemoteViewsService` to populate a collection view, the manifest entry may look like this:

```
<service android:name="MyWidgetService"
...
    android:permission="android.permission.BIND_REMOTEVIEWS" />
```

The line `android:name="MyWidgetService"` refers to your subclass of `RemoteViewsService`.

Layout for app widgets with collections

The main requirement for your app widget layout XML file is that it include one of the collection views: `ListView`, `GridView`, `StackView`, or `AdapterViewFlipper`. Here is the `widget_layout.xml` for the [StackView Widget sample](#):

```
<?xml version="1.0" encoding="utf-8"?>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <StackView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/stack_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:loopViews="true" />
    <TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/empty_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:background="@drawable/widget_item_background"
        android:textColor="#ffffffff"
        android:textStyle="bold"
        android:text="@string/empty_view_text"
        android:textSize="20sp" />
</FrameLayout>
```

Note that empty views must be siblings of the collection view for which the empty view represents empty state.

In addition to the layout file for your entire app widget, you must create another layout file that defines the layout for each item in the collection (for example, a layout for each book in a collection of books). For example, the [StackView Widget sample](#) only has one layout file, `widget_item.xml`, since all items use the same layout. But the [WeatherListWidget sample](#) has two layout files: `dark_widget_item.xml` and `light_widget_item.xml`.

AppWidgetProvider class for app widgets with collections

As with a regular app widget, the bulk of your code in your `AppWidgetProvider` subclass typically goes in `onUpdate()`. The major difference in your implementation for `onUpdate()` when creating an app widget with collections is that you must call `setRemoteAdapter()`. This tells the collection view where to get its data. The `RemoteViewsService` can then return your implementation of `RemoteViewsFactory`, and the

widget can serve up the appropriate data. When you call this method, you must pass an intent that points to your implementation of [RemoteViewsService](#) and the app widget ID that specifies the app widget to update.

For example, here's how the StackView Widget sample implements the `onUpdate()` callback method to set the [RemoteViewsService](#) as the remote adapter for the app widget collection:

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
    // update each of the app widgets with the remote adapter
    for (int i = 0; i < appWidgetIds.length; ++i) {

        // Set up the intent that starts the StackViewService, which will
        // provide the views for this collection.
        Intent intent = new Intent(context, StackWidgetService.class);
        // Add the app widget ID to the intent extras.
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetIds[i]);
        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
        // Instantiate the RemoteViews object for the app widget layout.
        RemoteViews rv = new RemoteViews(context.getPackageName(), R.layout.widget_layout);
        // Set up the RemoteViews object to use a RemoteViews adapter.
        // This adapter connects
        // to a RemoteViewsService through the specified intent.
        // This is how you populate the data.
        rv.setRemoteAdapter(appWidgetIds[i], R.id.stack_view, intent);

        // The empty view is displayed when the collection has no items.
        // It should be in the same layout used to instantiate the RemoteViews
        // object above.
        rv.setEmptyView(R.id.stack_view, R.id.empty_view);

        //
        // Do additional processing specific to this app widget...
        //

        appWidgetManager.updateAppWidget(appWidgetIds[i], rv);
    }
    super.onUpdate(context, appWidgetManager, appWidgetIds);
}
```

RemoteViewsService class

As described above, your [RemoteViewsService](#) subclass provides the [RemoteViewsFactory](#) used to populate the remote collection view.

Specifically, you need to perform these steps:

1. Subclass [RemoteViewsService](#). [RemoteViewsService](#) is the service through which a remote adapter can request [RemoteViews](#).
2. In your [RemoteViewsService](#) subclass, include a class that implements the [RemoteViewsFactory](#) interface. [RemoteViewsFactory](#) is an interface for an adapter between a remote collection view (such as [ListView](#), [GridView](#), and so on) and the underlying data for that view. Your implementation is responsible for making a [RemoteViews](#) object for each item in the data set. This interface is a thin wrapper around [Adapter](#).

The primary contents of the [RemoteViewsService](#) implementation is its [RemoteViewsFactory](#), described below.

RemoteViewsFactory interface

Your custom class that implements the [RemoteViewsFactory](#) interface provides the app widget with the data for the items in its collection. To do this, it combines your app widget item XML layout file with a source of data. This source of data could be anything from a database to a simple array. In the [StackView Widget sample](#), the data source is an array of [WidgetItem](#)s. The [RemoteViewsFactory](#) functions as an adapter to glue the data to the remote collection view.

The two most important methods you need to implement for your [RemoteViewsFactory](#) subclass are `onCreate()` and `getViewAt()`.

The system calls `onCreate()` when creating your factory for the first time. This is where you set up any connections and/or cursors to your

Persisting data

You can't rely on a single instance of your service, or any data it contains, to persist. You should therefore not store any data in your [RemoteViewsService](#) (unless it is static). If you want your app widget's data to persist, the best approach is to use a [ContentProvider](#) whose data persists beyond the process lifecycle.

data source. For example, the [StackView Widget sample](#) uses `onCreate()` to initialize an array of `WidgetItem` objects. When your app widget is active, the system accesses these objects using their index position in the array and the text they contain is displayed.

Here is an excerpt from the [StackView Widget sample](#)'s `RemoteViewsFactory` implementation that shows portions of the `onCreate()` method:

```
class StackRemoteViewsFactory implements
RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 10;
    private List<WidgetItem> mWidgetItems = new ArrayList<WidgetItem>();
    private Context mContext;
    private int mAppWidgetId;

    public StackRemoteViewsFactory(Context context, Intent intent) {
        mContext = context;
        mAppWidgetId = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                         AppWidgetManager.INVALID_APPWIDGET_ID);
    }

    public void onCreate() {
        // In onCreate() you setup any connections / cursors to your data source. Heavy lifting,
        // for example downloading or creating content etc, should be deferred to onDataSetChanged()
        // or getViewAt(). Taking more than 20 seconds in this call will result in an ANR.
        for (int i = 0; i < mCount; i++) {
            mWidgetItems.add(new WidgetItem(i + "!="));
        }
        ...
    }
}
```

The `RemoteViewsFactory` method `getViewAt()` returns a `RemoteViews` object corresponding to the data at the specified `position` in the data set. Here is an excerpt from the [StackView Widget sample](#)'s `RemoteViewsFactory` implementation:

```
public RemoteViews getViewAt(int position) {
    // Construct a remote views item based on the app widget item XML file,
    // and set the text based on the position.
    RemoteViews rv = new RemoteViews(mContext.getPackageName(), R.layout.widget_item);
    rv.setTextViewText(R.id.widget_item, mWidgetItems.get(position).text);

    ...
    // Return the remote views object.
    return rv;
}
```

Adding behavior to individual items

The above sections show you how to bind your data to your app widget collection. But what if you want to add dynamic behavior to the individual items in your collection view?

As described in [Using the AppWidgetProvider Class](#), you normally use `setOnClickPendingIntent()` to set an object's click behavior—such as to cause a button to launch an [Activity](#). But this approach is not allowed for child views in an individual collection item (to clarify, you could use `setOnClickPendingIntent()` to set up a global button in the Gmail app widget that launches the app, for example, but not on the individual list items). Instead, to add click behavior to individual items in a collection, you use `setOnClickFillInIntent()`. This entails setting up a pending intent template for your collection view, and then setting a fill-in intent on each item in the collection via your `RemoteViewsFactory`.

This section uses the [StackView Widget sample](#) to describe how to add behavior to individual items. In the [StackView Widget sample](#), if the user touches the top view, the app widget displays the [Toast](#) message "Touched view n," where *n* is the index (position) of the touched view. This is how it works:

- The `StackWidgetProvider` (an `AppWidgetProvider` subclass) creates a pending intent that has a custom action called `TOAST_ACTION`.
- When the user touches a view, the intent is fired and it broadcasts `TOAST_ACTION`.
- This broadcast is intercepted by the `StackWidgetProvider`'s `onReceive()` method, and the app widget displays the [Toast](#) message for

the touched view. The data for the collection items is provided by the [RemoteViewsFactory](#), via the [RemoteViewsService](#).

Note: The [StackView Widget sample](#) uses a broadcast, but typically an app widget would simply launch an activity in a scenario like this one.

Setting up the pending intent template

The [StackWidgetProvider](#) ([AppWidgetProvider](#) subclass) sets up a pending intent. Individual items of a collection cannot set up their own pending intents. Instead, the collection as a whole sets up a pending intent template, and the individual items set a fill-in intent to create unique behavior on an item-by-item basis.

This class also receives the broadcast that is sent when the user touches a view. It processes this event in its [onReceive\(\)](#) method. If the intent's action is [TOAST_ACTION](#), the app widget displays a [Toast](#) message for the current view.

```

public class StackWidgetProvider extends AppWidgetProvider {
    public static final String TOAST_ACTION = "com.example.android.stackwidget.TOAST_ACTION";
    public static final String EXTRA_ITEM = "com.example.android.stackwidget.EXTRA_ITEM";

    ...

    // Called when the BroadcastReceiver receives an Intent broadcast.
    // Checks to see whether the intent's action is TOAST_ACTION. If it is, the app widget
    // displays a Toast message for the current item.
    @Override
    public void onReceive(Context context, Intent intent) {
        AppWidgetManager mgr = AppWidgetManager.getInstance(context);
        if (intent.getAction().equals(TOAST_ACTION)) {
            int appWidgetId = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                AppWidgetManager.INVALID_APPWIDGET_ID);
            int viewIndex = intent.getIntExtra(EXTRA_ITEM, 0);
            Toast.makeText(context, "Touched view " + viewIndex, Toast.LENGTH_SHORT).show();
        }
        super.onReceive(context, intent);
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        // update each of the app widgets with the remote adapter
        for (int i = 0; i < appWidgetIds.length; ++i) {

            // Sets up the intent that points to the StackWidgetService that will
            // provide the views for this collection.
            Intent intent = new Intent(context, StackWidgetService.class);
            intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetIds[i]);
            // When intents are compared, the extras are ignored, so we need to embed the extras
            // into the data so that the extras will not be ignored.
            intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
            RemoteViews rv = new RemoteViews(context.getPackageName(), R.layout.widget_layout);
            rv.setRemoteAdapter(appWidgetIds[i], R.id.stack_view, intent);

            // The empty view is displayed when the collection has no items. It should be a sibling
            // of the collection view.
            rv.setEmptyView(R.id.stack_view, R.id.empty_view);

            // This section makes it possible for items to have individualized behavior.
            // It does this by setting up a pending intent template. Individuals items of a collection
            // cannot set up their own pending intents. Instead, the collection as a whole sets
            // up a pending intent template, and the individual items set a fillInIntent
            // to create unique behavior on an item-by-item basis.
            Intent toastIntent = new Intent(context, StackWidgetProvider.class);
            // Set the action for the intent.
            // When the user touches a particular view, it will have the effect of
            // broadcasting TOAST_ACTION.
            toastIntent.setAction(StackWidgetProvider.TOAST_ACTION);
            toastIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetIds[i]);
            intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
            PendingIntent toastPendingIntent = PendingIntent.getBroadcast(context, 0, toastIntent,
                PendingIntent.FLAG_UPDATE_CURRENT);
            rv.setPendingIntentTemplate(R.id.stack_view, toastPendingIntent);

            appWidgetManager.updateAppWidget(appWidgetIds[i], rv);
        }
        super.onUpdate(context, appWidgetManager, appWidgetIds);
    }
}

```

Setting the fill-in Intent

Your `RemoteViewsFactory` must set a fill-in intent on each item in the collection. This makes it possible to distinguish the individual on-click action of a given item. The fill-in intent is then combined with the `PendingIntent` template in order to determine the final intent that will be executed when the item is clicked.

```

public class StackWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new StackRemoteViewsFactory(this.getApplicationContext(), intent);
    }
}

class StackRemoteViewsFactory implements RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 10;
    private List<WidgetItem> mWidgetItemList = new ArrayList<WidgetItem>();
    private Context mContext;
    private int mAppWidgetId;

    public StackRemoteViewsFactory(Context context, Intent intent) {
        mContext = context;
        mAppWidgetId = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                         AppWidgetManager.INVALID_APPWIDGET_ID);
    }

    // Initialize the data set.
    public void onCreate() {
        // In onCreate() you set up any connections / cursors to your data source. Heavy lifting,
        // for example downloading or creating content etc, should be deferred to onDataSetChanged()
        // or getViewAt(). Taking more than 20 seconds in this call will result in an ANR.
        for (int i = 0; i < mCount; i++) {
            mWidgetItemList.add(new WidgetItem(i + "!!"));
        }
        ...
    }
    ...

    // Given the position (index) of a WidgetItem in the array, use the item's text value in
    // combination with the app widget item XML file to construct a RemoteViews object.
    public RemoteViews getViewAt(int position) {
        // position will always range from 0 to getCount() - 1.

        // Construct a RemoteViews item based on the app widget item XML file, and set the
        // text based on the position.
        RemoteViews rv = new RemoteViews(mContext.getPackageName(), R.layout.widget_item);
        rv.setTextViewText(R.id.widget_item, mWidgetItemList.get(position).text);

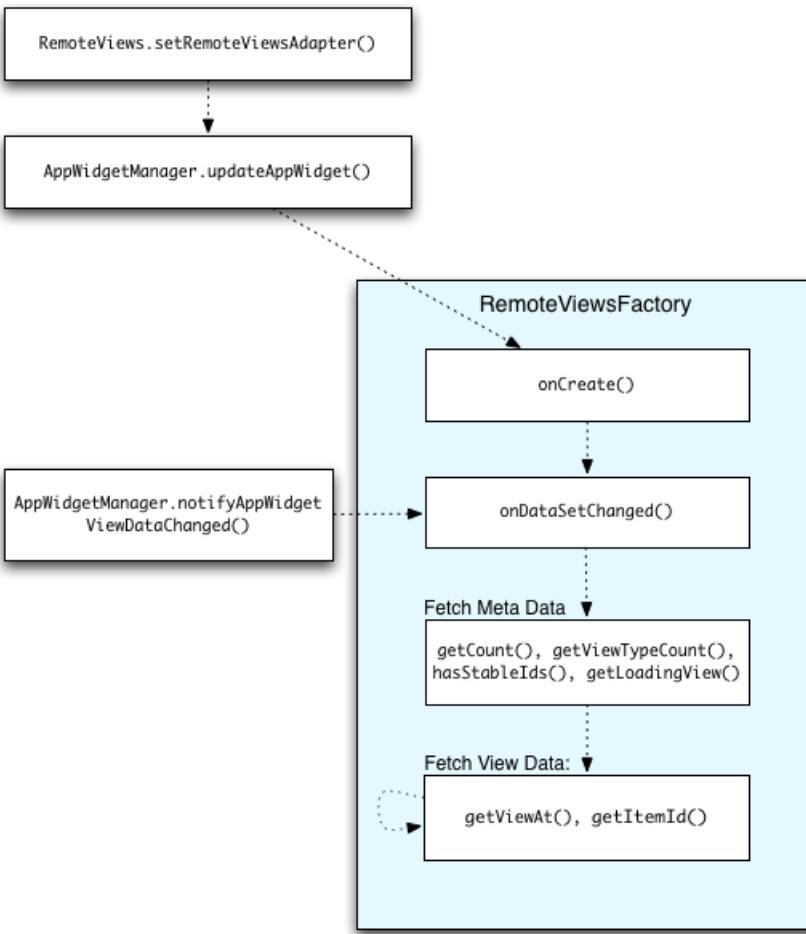
        // Next, set a fill-intent, which will be used to fill in the pending intent template
        // that is set on the collection view in StackWidgetProvider.
        Bundle extras = new Bundle();
        extras.putInt(StackWidgetProvider.EXTRA_ITEM, position);
        Intent fillInIntent = new Intent();
        fillInIntent.putExtras(extras);
        // Make it possible to distinguish the individual on-click
        // action of a given item
        rv.setOnClickFillInIntent(R.id.widget_item, fillInIntent);

        ...
        // Return the RemoteViews object.
        return rv;
    }
}

```

Keeping Collection Data Fresh

The following figure illustrates the flow that occurs in an app widget that uses collections when updates occur. It shows how the app widget code interacts with the `RemoteViewsFactory`, and how you can trigger updates:



One feature of app widgets that use collections is the ability to provide users with up-to-date content. For example, consider the Android 3.0 Gmail app widget, which provides users with a snapshot of their inbox. To make this possible, you need to be able to trigger your `RemoteViewsFactory` and collection view to fetch and display new data. You achieve this with the `AppWidgetManager` call `notifyAppWidgetViewDataChanged()`. This call results in a callback to your `RemoteViewsFactory`'s `onDataSetChanged()` method, which gives you the opportunity to fetch any new data. Note that you can perform processing-intensive operations synchronously within the `onDataSetChanged()` callback. You are guaranteed that this call will be completed before the metadata or view data is fetched from the `RemoteViewsFactory`. In addition, you can perform processing-intensive operations within the `getViewAt()` method. If this call takes a long time, the loading view (specified by the `RemoteViewsFactory`'s `getLoadingView()` method) will be displayed in the corresponding position of the collection view until it returns.



App Widget Host

In this document

- › [Binding App Widgets](#)
 - › [Binding app widgets on Android 4.0 and lower](#)
 - › [Binding app widgets on Android 4.1 and higher](#)
- › [Host Responsibilities](#)
 - › [Android 3.0](#)
 - › [Android 3.1](#)
 - › [Android 4.0](#)
 - › [Android 4.1](#)
 - › [Android 4.2](#)

The Android Home screen available on most Android devices allows the user to embed [app widgets](#) for quick access to content. If you're building a Home replacement or a similar app, you can also allow the user to embed app widgets by implementing an [AppWidgetHost](#). This is not something that most apps will ever need to do, but if you are creating your own host, it's important to understand the contractual obligations a host implicitly agrees to.

This document focuses on the responsibilities involved in implementing a custom [AppWidgetHost](#). For an example of how to implement an [AppWidgetHost](#), see the source code for the Android Home screen [Launcher](#).

Here is an overview of key classes and concepts involved in implementing a custom [AppWidgetHost](#):

- **App Widget Host**— The [AppWidgetHost](#) provides the interaction with the AppWidget service for apps, like the home screen, that want to embed app widgets in their UI. An [AppWidgetHost](#) must have an ID that is unique within the host's own package. This ID remains persistent across all uses of the host. The ID is typically a hard-coded value that you assign in your application.
- **App Widget ID**— Each app widget instance is assigned a unique ID at the time of binding (see [bindAppWidgetIdIfAllowed\(\)](#), discussed in more detail in [Binding app widgets](#)). The unique ID is obtained by the host using [allocateAppWidgetId\(\)](#). This ID is persistent across the lifetime of the widget, that is, until it is deleted from the host. Any host-specific state (such as the size and location of the widget) should be persisted by the hosting package and associated with the app widget ID.
- **App Widget Host View**— [AppWidgetHostView](#) can be thought of as a frame that the widget is wrapped in whenever it needs to be displayed. An app widget is assigned to an [AppWidgetHostView](#) every time the widget is inflated by the host.
- **Options Bundle**— The [AppWidgetHost](#) uses the options bundle to communicate information to the [AppWidgetProvider](#) about how the widget is being displayed (for example, size range, and whether the widget is on a lockscreen or the home screen). This information allows the [AppWidgetProvider](#) to tailor the widget's contents and appearance based on how and where it is displayed. You use [updateAppWidgetOptions\(\)](#) and [updateAppWidgetSize\(\)](#) to modify an app widget's bundle. Both of these methods trigger a callback to the [AppWidgetProvider](#).

Binding App Widgets

When a user adds an app widget to a host, a process called *binding* occurs. *Binding* refers to associating a particular app widget ID to a specific host and to a specific [AppWidgetProvider](#). There are different ways of achieving this, depending on what version of Android your app is running on.

Binding app widgets on Android 4.0 and lower

On devices running Android version 4.0 and lower, users add app widgets via a system activity that allows users to select a widget. This implicitly does a permission check—that is, by adding the app widget, the user is implicitly granting permission to your app to add app widgets to the host. Here is an example that illustrates this approach, taken from the original [Launcher](#). In this snippet, an event handler invokes `startActivityForResult()` with the request code `REQUEST_PICK_APPWIDGET` in response to a user action:

```
private static final int REQUEST_CREATE_APPWIDGET = 5;
private static final int REQUEST_PICK_APPWIDGET = 9;
...
public void onClick(DialogInterface dialog, int which) {
    switch (which) {
        ...
        case AddAdapter.ITEM_APPWIDGET: {
            ...
            int appWidgetId =
                Launcher.this.mAppWidgetHost.allocateAppWidgetId();
            Intent pickIntent =
                new Intent(AppWidgetManager.ACTION_APPWIDGET_PICK);
            pickIntent.putExtra
                (AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
            ...
            startActivityForResult(pickIntent, REQUEST_PICK_APPWIDGET);
            break;
        }
        ...
    }
}
```

When the system activity finishes, it returns a result with the user's chosen app widget to your activity. In the following example, the activity responds by calling `addAppWidget()` to add the app widget:

```
public final class Launcher extends Activity
    implements View.OnClickListener, OnLongClickListener {
    ...
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        mWaitingForResult = false;

        if (resultCode == RESULT_OK && mAddItemCellInfo != null) {
            switch (requestCode) {
                ...
                case REQUEST_PICK_APPWIDGET:
                    addAppWidget(data);
                    break;
                case REQUEST_CREATE_APPWIDGET:
                    completeAddAppWidget(data, mAddItemCellInfo, !mDesktopLocked);
                    break;
            }
        }
        ...
    }
}
```

The method `addAppWidget()` checks to see if the app widget needs to be configured before it's added:

```

void addAppWidget(Intent data) {
    int appWidgetId = data.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, -1);

    String customWidget = data.getStringExtra(EXTRA_CUSTOM_WIDGET);
    AppWidgetProviderInfo appWidget =
        mAppWidgetManager.getAppWidgetInfo(appWidgetId);

    if (appWidget.configure != null) {
        // Launch over to configure widget, if needed.
        Intent intent = new Intent(AppWidgetManager.ACTION_APPWIDGET_CONFIGURE);
        intent.setComponent(appWidget.configure);
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
        startActivityForResult(intent, REQUEST_CREATE_APPWIDGET);
    } else {
        // Otherwise, finish adding the widget.
    }
}

```

For more discussion of configuration, see [Creating an App Widget Configuration Activity](#).

Once the app widget is ready, the next step is to do the actual work of adding it to the workspace. The [original Launcher](#) uses a method called `completeAddAppWidget()` to do this.

Binding app widgets on Android 4.1 and higher

Android 4.1 adds APIs for a more streamlined binding process. These APIs also make it possible for a host to provide a custom UI for binding. To use this improved process, your app must declare the `BIND_APPWIDGET` permission in its manifest:

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

But this is just the first step. At runtime the user must explicitly grant permission to your app to allow it to add app widgets to the host. To test whether your app has permission to add the widget, you use the `bindAppWidgetIdIfAllowed()` method. If `bindAppWidgetIdIfAllowed()` returns `false`, your app must display a dialog prompting the user to grant permission ("allow" or "always allow," to cover all future app widget additions). This snippet gives an example of how to display the dialog:

```

Intent intent = new Intent(AppWidgetManager.ACTION_APPWIDGET_BIND);
intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_PROVIDER, info.componentName);
// This is the options bundle discussed above
intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_OPTIONS, options);
startActivityForResult(intent, REQUEST_BIND_APPWIDGET);

```

The host also has to check whether the user added an app widget that needs configuration. For more discussion of this topic, see [Creating an App Widget Configuration Activity](#).

Host Responsibilities

Widget developers can specify a number of configuration settings for widgets using the [AppWidgetProviderInfo metadata](#). These configuration options, discussed in more detail below, can be retrieved by the host from the `AppWidgetProviderInfo` object associated with a widget provider.

Regardless of the version of Android you are targeting, all hosts have the following responsibilities:

- When adding a widget, you must allocate the widget ID as described above. You must also make sure that when a widget is removed from the host, you call `deleteAppWidgetId()` to deallocate the widget ID.
- When adding a widget, be sure to launch its configuration activity if it exists, as described in [Updating the App Widget from the Configuration Activity](#). This is a necessary step for

What Version are You Targeting?

The approach you use in implementing your host should depend on what Android version you're targeting. Many of the features described in this section were introduced in 3.0 or later. For example:

- Android 3.0 (API Level 11) introduces auto-advance behavior for widgets.
- Android 3.1 (API Level 12) introduces the ability to resize widgets.
- Android 4.0 (API Level 15) introduces a change in padding

many app widgets before they can be properly displayed.

- Every app widget specifies a minimum width and height in dps, as defined in the [AppWidgetProviderInfo](#) metadata (using `android:minWidth` and `android:minHeight`). Make sure that the widget is laid out with at least this many dps. For example, many hosts align icons and widgets in a grid. In this scenario, by default the host should add the app widget using the minimum number of cells that satisfy the `minWidth` and `minHeight` constraints.

In addition to the requirements listed above, specific platform versions introduce features that place new responsibilities on the host. These are described in the following sections.

Android 3.0

Android 3.0 (API Level 11) introduces the ability for a widget to specify `autoAdvanceViewId()`. This view ID should point to an instance of an [Advanceable](#), such as [StackView](#) or [AdapterViewFlipper](#). This indicates that the host should call `advance()` on this view at an interval deemed appropriate by the host (taking into account whether it makes sense to advance the widget—for example, the host probably wouldn't want to advance a widget if it were on another page, or if the screen were turned off).

Android 3.1

Android 3.1 (API Level 12) introduces the ability to resize widgets. A widget can specify that it is resizable using the `android:resizeMode` attribute in the [AppWidgetProviderInfo](#) metadata, and indicate whether it supports horizontal and/or vertical resizing. Introduced in Android 4.0 (API Level 14), the widget can also specify a `android:minResizeWidth` and/or `android:minResizeHeight`.

It is the host's responsibility to make it possible for the widget to be resized horizontally and/or vertically, as specified by the widget. A widget that specifies that it is resizable can be resized arbitrarily large, but should not be resized smaller than the values specified by `android:minResizeWidth` and `android:minResizeHeight`. For a sample implementation, see [AppWidgetResizeFrame](#) in [Launcher2](#).

Android 4.0

Android 4.0 (API Level 15) introduces a change in padding policy that puts the responsibility on the host to manage padding. As of 4.0, app widgets no longer include their own padding. Instead, the system adds padding for each widget, based the characteristics of the current screen. This leads to a more uniform, consistent presentation of widgets in a grid. To assist applications that host app widgets, the platform provides the method `getDefaultPaddingForWidget()`. Applications can call this method to get the system-defined padding and account for it when computing the number of cells to allocate to the widget.

Android 4.1

Android 4.1 (API Level 16) adds an API that allows the widget provider to get more detailed information about the environment in which its widget instances are being hosted. Specifically, the host hints to the widget provider about the size at which the widget is being displayed. It is the host's responsibility to provide this size information.

The host provides this information via `updateAppWidgetSize()`. The size is specified as a minimum and maximum width/height in dps. The reason that a range is specified (as opposed to a fixed size) is because the width and height of a widget may change with orientation. You don't want the host to have to update all of its widgets on rotation, as this could cause serious system slowdown. These values should be updated once upon the widget being placed, any time the widget is resized, and any time the launcher inflates the widget for the first time in a given boot (as the values aren't persisted across boot).

Android 4.2

Android 4.2 (API Level 17) adds the ability for the options bundle to be specified at bind time. This is the ideal way to specify app widget options, including size, as it gives the [AppWidgetProvider](#) immediate access to the options data on the first update. This can be achieved by using the method `bindAppWidgetIdIfAllowed()`. For more discussion of this topic, see [Binding app widgets](#).

Android 4.2 also introduces lockscreen widgets. When hosting widgets on the lockscreen, the host must specify this information within the app widget options bundle (the [AppWidgetProvider](#) can use this information to style the widget appropriately). To designate a widget as a

policy that puts the responsibility on the host to manage padding.

- Android 4.1 (API Level 16) adds an API that allows the widget provider to get more detailed information about the environment in which its widget instances are being hosted.
- Android 4.2 (API Level 17) introduces the options bundle and the `bindAppWidgetIdIfAllowed()` method. It also introduces lockscreen widgets.

If you are targeting earlier devices, refer to the original [Launcher](#) as an example.

lockscreens, use `updateAppWidgetOptions()` and include the field `OPTION_APPWIDGET_HOST_CATEGORY` with the value `WIDGET_CATEGORY_KEYGUARD`. This option defaults to `WIDGET_CATEGORY_HOME_SCREEN`, so it is not explicitly required to set this for a home screen host.

Make sure that your host adds only app widgets that are appropriate for your app—for example, if your host is a home screen, ensure that the `android:widgetCategory` attribute in the `AppWidgetProviderInfo` metadata includes the flag `WIDGET_CATEGORY_HOME_SCREEN`. Similarly, for the lockscreens, ensure that field includes the flag `WIDGET_CATEGORY_KEYGUARD`. For more discussion of this topic, see [Enabling App Widgets on the Lockscreen](#).



App Resources

It takes more than just code to build a great app. Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.

BLOG ARTICLES

New Tools For Managing Screen Sizes

Android 3.2 includes new tools for supporting devices with a wide range of screen sizes. One important result is better support for a new size of screen; what is typically called a "7-inch" tablet. This release also offers several new APIs to simplify developers' work in adjusting to different screen sizes.

Holo Everywhere

Before Android 4.0 the variance in system themes from device to device could make it difficult to design an app with a single predictable look and feel. We set out to improve this situation for the developer community in Ice Cream Sandwich and beyond.

New Mode for Apps on Large Screens

Android tablets are becoming more popular, and we're pleased to note that the vast majority of apps resize to the larger screens just fine. To keep the few apps that don't resize well from frustrating users with awkward-looking apps on their tablets, Android 3.2 introduces a screen compatibility mode that makes these apps more usable on tablets.

TRAINING

Supporting Different Devices

This class teaches you how to use basic platform features that leverage alternative resources and other features so your app can provide an optimized user experience on a variety of Android-compatible devices, using a single application package (APK).

Designing for Multiple Screens

This class shows you how to implement a user interface that's optimized for several screen configurations.

资源概览

主题

- › [提供资源](#)
- › [访问资源](#)
- › [处理运行时变更](#)
- › [本地化](#)
- › [复杂的 XML 资源](#)

参考

- › [资源类型](#)

您应该始终外部化资源，例如图像和应用代码中的字符串，这样有利于您单独维护这些资源。此外，通过外部化资源，您还可以提供支持特定设备配置（例如，不同的语言或屏幕尺寸）的备用资源，随着采用 Android 技术且配置各异的设备越来越多，这些资源的重要性也日益增加。为了提供与不同配置的兼容性，您必须使用各种按类型和配置对资源进行分组的子目录，对项目 `res/` 目录中的资源加以组织。



图 1. 两种不同的设备，均使用默认布局（应用不提供备用布局）。



图 2. 两种不同的设备，分别使用针对不同屏幕尺寸提供的不同布局。

对于任意类型的资源，您均可以为应用指定默认资源和多个备用资源：

- 默认资源系指无论设备配置如何，或者在没有备用资源与当前配置相匹配时，均应使用的资源。
- 备用资源系指设计用于特定配置的资源。要指明某组资源适用于特定配置，请将相应的配置限定符追加到目录名称。

例如，尽管默认 UI 布局保存在 `res/layout/` 目录中，但是您可以指定在屏幕处于横向时要使用不同的布局，方法是将其保存在 `res/layout-land/` 目录中。Android 可以通过将设备的当前配置与资源目录名称进行匹配，自动应用合适的资源。

图 1 说明了在没有备用资源可用时，系统如何为两种不同的设备应用相同布局。图 2 显示的是同一应用针对大屏幕添加了备用布局资源。

以下文档提供了有关如何组织应用资源、如何指定备用资源以及如何在应用中访问这些资源等的完整指南：

提供资源

您可在应用中提供何种资源、可将这些资源保存在何处以及如何为特定设备配置创建备用资源。

访问资源

如何通过从应用代码或其他 XML 资源中引用来使用所提供的资源

处理运行时变更

如何管理在 Activity 运行时发生的配置变更。

本地化

从点到面地指导您使用备用资源本地化应用。尽管这只是备用资源的一种具体运用，但是这对于赢得更多用户非常重要。

复杂的 XML 资源

一种 XML 格式，用于在单个 XML 文件中构建复杂的资源，如动画矢量可绘制对象。

资源类型

有关您可提供的各种资源类型的参考文档，其中描述了这些资源的 XML 元素、属性和语法。例如，此参考文档向您展示了如何为应用菜单、可绘制对象、动画等创建资源。



提供资源

内容快览

- › 不同类型的资源属于 `res/` 的不同子目录
- › 备用资源提供特定于配置的资源文件
- › 始终包含默认资源，使您的应用不依赖于特定的设备配置

本文内容

- › [分组资源类型](#)
- › [提供备用资源](#)
 - › [限定符命名规则](#)
 - › [创建别名资源](#)
- › [利用资源提供最佳设备兼容性](#)
- › [Android 如何查找最佳匹配资源](#)

另请参阅

- › [访问资源](#)
- › [资源类型](#)
- › [支持多种屏幕](#)

您应该始终外部化应用资源，例如图像和代码中的字符串，这样有利于您单独维护这些资源。此外，您还应该为特定设备配置提供备用资源，方法是将它们分组到专门命名的资源目录中。在运行时，Android 会根据当前配置使用适当的资源。例如，您可能需要根据屏幕尺寸提供不同的 UI 布局，或者根据语言设置提供不同的字符串。

外部化应用资源后，即可使用在项目 `R` 类中生成的资源 ID 访问这些资源。有关如何在应用中使用资源，我们将在[访问资源](#)中讨论。本文档介绍如何对 Android 项目中的资源进行分组，以及如何为特定的设备配置提供备用资源。

分组资源类型

您应将各种资源放入项目 `res/` 目录的特定子目录下。例如，以下是一个简单项目的文件层次结构：

```
MyProject/
  src/
    MyActivity.java
  res/
    drawable/
      graphic.png
    layout/
      main.xml
      info.xml
    mipmap/
      icon.png
    values/
      strings.xml
```

正如您在此示例中所看到的那样，`res/` 目录包含所有资源（在子目录下）：一个图像资源、两个布局资源、启动器图标的 `mipmap/` 目录以及一个字符串资源文件。资源目录名称非常重要，将在表 1 中进行介绍。

注：如需了解有关使用 `mipmap` 文件夹的详细信息，请参阅[管理项目概览](#)。

表 1. 项目 `res/` 目录内支持的资源目录。

目录	资源类型
animator/	用于定义 属性动画 的 XML 文件。
anim/	定义 渐变动画 的 XML 文件。（属性动画也可以保存在此目录中，但是为了区分这两种类型，属性动画首选 <code>animator/</code> 目录。）
color/	用于定义颜色状态列表的 XML 文件。请参阅 颜色状态列表资源
drawable/	<p>位图文件（<code>.png</code>、<code>.9.png</code>、<code>.jpg</code>、<code>.gif</code>）或编译为以下可绘制对象资源子类型的 XML 文件：</p> <ul style="list-style-type: none"> 位图文件 九宫格（可调整大小的位图） 状态列表 形状 动画可绘制对象 其他可绘制对象 <p>请参阅可绘制对象资源。</p>
mipmap/	适用于不同启动器图标密度的可绘制对象文件。如需了解有关使用 <code>mipmap/</code> 文件夹管理启动器图标的详细信息，请参阅 管理项目概览 。
layout/	用于定义用户界面布局的 XML 文件。请参阅 布局资源 。
menu/	用于定义应用菜单（如选项菜单、上下文菜单或子菜单）的 XML 文件。请参阅 菜单资源 。
raw/	<p>要以原始形式保存的任意文件。要使用原始 <code>InputStream</code> 打开这些资源，请使用资源 ID（即 <code>R.raw.filename</code>）调用 <code>Resources.openRawResource()</code>。</p> <p>但是，如需访问原始文件名和文件层次结构，则可以考虑将某些资源保存在 <code>assets/</code> 目录下（而不是 <code>res/raw/</code>）。<code>assets/</code> 中的文件没有资源 ID，因此您只能使用 <code>AssetManager</code> 读取这些文件。</p>
values/	<p>包含字符串、整型数和颜色等简单值的 XML 文件。</p> <p>其他 <code>res/</code> 子目录中的 XML 资源文件是根据 XML 文件名定义单个资源，而 <code>values/</code> 目录中的文件可描述多个资源。对于此目录中的文件，<code><resources></code> 元素的每个子元素均定义一个资源。例如，<code><string></code> 元素创建 <code>R.string</code> 资源，<code><color></code> 元素创建 <code>R.color</code> 资源。</p> <p>由于每个资源均用其自己的 XML 元素定义，因此您可以根据自己的需要命名文件，并将不同的资源类型放在一个文件中。但是，为了清晰起见，您可能需要将独特的资源类型放在不同的文件中。例如，对于可在此目录中创建的资源，下面给出了相应的文件名约定：</p> <ul style="list-style-type: none"> <code>arrays.xml</code>，用于资源数组（类型化数组）。 <code>colors.xml</code>：颜色值。 <code>dimens.xml</code>：尺寸值。 <code>strings.xml</code>：字符串值。 <code>styles.xml</code>：样式。 <p>请参阅字符串资源、样式资源和更多资源类型。</p>
xml/	可以在运行时通过调用 <code>Resources.getXML()</code> 读取的任意 XML 文件。各种 XML 配置文件（如 可搜索配置 ）都必须保存在此处。

注意：切勿将资源文件直接保存在 `res/` 目录内，这会导致出现编译错误。

如需了解有关某些资源类型的详细信息，请参阅[资源类型](#)文档。

保存在表 1 中定义的子目录下的资源是“默认”资源。即，这些资源定义应用的默认设计和内容。但是，采用 Android 技术的不同设备类型可能需要不同类型的资源。例如，如果设备的屏幕尺寸大于标准屏幕，则应提供不同的布局资源，以充分利用额外的屏幕空间。或者，如果设备的

语言设置不同，则应提供不同的字符串资源，以转换用户界面中的文本。要为不同的设备配置提供这些不同资源，除了默认资源以外，您还需要提供备用资源。

提供备用资源



图 1. 两种不同的设备，均使用不同的布局资源。

几乎每个应用都应提供备用资源以支持特定的设备配置。例如，对于不同的屏幕密度和语言，您应分别包括备用可绘制对象资源和备用字符串资源。在运行时，Android 会检测当前设备配置并为应用加载合适的资源。

为一组资源指定特定于配置的备用资源：

1. 在 `res/` 中创建一个以 `<resources_name>-<config_qualifier>` 形式命名的新目录。

- `<resources_name>` 是相应默认资源的目录名称（如表 1 中所定义）。
- `<qualifier>` 是指定要使用这些资源的各个配置的名称（如表 2 中所定义）。

您可以追加多个 `<qualifier>`。以短划线将其分隔。

注意：追加多个限定符时，必须按照表 2 中列出的相同顺序放置它们。如果限定符的顺序错误，则该资源将被忽略。

2. 将相应的备用资源保存在此新目录下。这些资源文件的名称必须与默认资源文件完全一样。

例如，以下是一些默认资源和备用资源：

```
res/
  drawable/
    icon.png
    background.png
  drawable-hdpi/
    icon.png
    background.png
```

`hdpi` 限定符表示该目录中的资源适用于屏幕密度较高的设备。其中每个可绘制对象目录中的图像已针对特定的屏幕密度调整大小，但是文件名完全相同。这样一来，用于引用 `icon.png` 或 `background.png` 图像的资源 ID 始终相同，但是 Android 会通过将设备配置信息与资源目录名称中的限定符进行比较，选择最符合当前设备的各个资源版本。

Android 支持若干配置限定符，您可以通过使用短划线分隔每个限定符，向一个目录名称添加多个限定符。表 2 按优先顺序列出了有效的配置限定符；如果对资源目录使用多个限定符，则必须按照表中列出的顺序将它们添加到目录名称。

表 2. 配置限定符名称。

配置	限定符值	说明
MCC 和 MNC	示例： <code>mcc310</code> <code>mcc310-mnc004</code> <code>mcc208-mnc00</code> 等等	移动国家代码 (MCC)，（可选）后跟设备 SIM 卡中的移动网络代码 (MNC)。例如， <code>mcc310</code> 是指美国的任一运营商， <code>mcc310-mnc004</code> 是指美国的 Verizon 公司， <code>mcc208-mnc00</code> 是指法国的 Orange 公司。 如果设备使用无线电连接（GSM 手机），则 MCC 和 MNC 值来自 SIM 卡。 也可以单独使用 MCC（例如，将国家/地区特定的合法资源包括在应用中）。如果只需根据语言指定，则改用“语言和区域”限定符（稍后进行介绍）。如果决定使用 MCC 和 MNC 限定符，请谨慎执行此操作并测试限定符是否按预期工作。 另请参阅配置字段 <code>mcc</code> 和 <code>mnc</code> ，这两个字段分别表示当前的移动国家代码和移动网络代码。

语言和区域	示例： en fr en-rUS fr-rFR fr-rCA 等等	<p>语言通过由两个字母组成的 ISO 639-1 语言代码定义，可以选择后跟两个字母组成的 ISO 3166-1-alpha-2 区域码（前带小写字母“r”）。</p> <p>这些代码不区分大小写；<code>r</code> 前缀用于区分区域码。不能单独指定区域。</p> <p>如果用户更改系统设置中的语言，它有可能在应用生命周期中发生改变。如需了解这会在运行期间给应用带来哪些影响，请参阅处理运行时变更。</p> <p>有关针对其他语言本地化应用的完整指南，请参阅本地化。</p> <p>另请参阅 <code>locale</code> 配置字段，该字段表示当前的语言区域。</p>
布局方向	<code>ldrtl</code> <code>ldltr</code>	<p>应用的布局方向。<code>ldrtl</code> 是指“布局方向从右到左”。<code>ldltr</code> 是指“布局方向从左到右”，这是默认的隐式值。</p> <p>它适用于布局、图片或值等任何资源。</p> <p>例如，若要针对阿拉伯语提供某种特定布局，并针对任何其他“从右到左”语言（如波斯语或希伯来语）提供某种通用布局，则可编码如下：</p> <pre>res/ layout/ main.xml (Default layout) layout-ar/ main.xml (Specific layout for Arabic) layout-ldrtl/ main.xml (Any "right-to-left" language, except for Arabic, because the "ar" language qualifier has a higher precedence.)</pre> <p>注：要为应用启用从右到左的布局功能，必须将 <code>supportsRtl</code> 设置为 "true"，并将 <code>targetSdkVersion</code> 设置为 17 或更高版本。</p> <p>此项为 API 级别 17 中新增配置。</p>
smallestWidth	<code>sw<N>dp</code> 示例： <code>sw320dp</code> <code>sw600dp</code> <code>sw720dp</code> 等等	<p>屏幕的基本尺寸，由可用屏幕区域的最小尺寸指定。具体来说，设备的 <code>smallestWidth</code> 是屏幕可用高度和宽度的最小尺寸（您也可以将其视为屏幕的“最小可能宽度”）。无论屏幕的当前方向如何，您均可使用此限定符确保应用 UI 的可用宽度至少为 <code><N>dp</code>。</p> <p>例如，如果布局要求屏幕区域的最小尺寸始终至少为 600dp，则可使用此限定符创建布局资源 <code>res/layout-sw600dp/</code>。仅当可用屏幕的最小尺寸至少为 600dp 时，系统才会使用这些资源，而不考虑 600dp 所代表的边是用户所认为的高度还是宽度。<code>smallestWidth</code> 是设备的固定屏幕尺寸特性；设备的 <code>smallestWidth</code> 不会随屏幕方向的变化而改变。</p> <p>设备的 <code>smallestWidth</code> 将屏幕装饰元素和系统 UI 考虑在内。例如，如果设备的屏幕上有一些永久性 UI 元素占据沿 <code>smallestWidth</code> 轴的空间，则系统会声明 <code>smallestWidth</code> 小于实际屏幕尺寸，因为这些屏幕像素不适用于您的 UI。因此，使用的值应该是布局所需的实际最小尺寸（通常，无论屏幕的当前方向如何，此值都是布局支持的“最小宽度”）。</p> <p>以下是一些可用于普通屏幕尺寸的值：</p> <ul style="list-style-type: none"> • 320，适用于屏幕配置如下的设备： <ul style="list-style-type: none"> ◦ 240x320 ldpi（QVGA 手机） ◦ 320x480 mdpi（手机） ◦ 480x800 hdpi（高密度手机） • 480，适用于 480x800 mdpi 之类的屏幕（平板电脑/手机）。 • 600，适用于 600x1024 mdpi 之类的屏幕（7 英寸平板电脑）。 • 720，适用于 720x1280 mdpi 之类的屏幕（10 英寸平板电脑）。 <p>应用为多个资源目录提供不同的 <code>smallestWidth</code> 限定符值时，系统会使用最接近（但未超出）设备 <code>smallestWidth</code> 的值。</p>

		<p>此项为 API 级别 13 中新增配置。</p> <p>另请参阅 <code>android:requiresSmallestWidthDp</code> 属性和 <code>smallestScreenWidthDp</code> 配置字段，前者声明与应用兼容的最小 <code>smallestWidth</code>；后者存放设备的 <code>smallestWidth</code> 值。</p> <p>如需了解有关设计不同屏幕和使用此限定符的详细信息，请参阅支持多种屏幕开发者指南。</p>
可用宽度	w<N>dp 示例： <code>w720dp</code> <code>w1024dp</code> 等等	<p>指定资源应该使用的最小可用屏幕宽度，以 <code>dp</code> 为单位，由 <code><N></code> 值定义。在横向和纵向之间切换时，为了匹配当前实际宽度，此配置值也会随之发生变化。</p> <p>应用为多个资源目录提供不同的此配置值时，系统会使用最接近（但未超出）设备当前屏幕宽度的值。此处的值考虑到了屏幕装饰元素，因此如果设备显示屏的左边缘或右边缘上有一些永久性 UI 元素，考虑到这些 UI 元素，它会使用小于实际屏幕尺寸的宽度值，这样会减少应用的可用空间。</p> <p>此项为 API 级别 13 中新增配置。</p> <p>另请参阅 <code>screenWidthDp</code> 配置字段，该字段存放当前屏幕宽度。</p> <p>如需了解有关设计不同屏幕和使用此限定符的详细信息，请参阅支持多种屏幕开发者指南。</p>
可用高度	h<N>dp 示例： <code>h720dp</code> <code>h1024dp</code> 等等	<p>指定资源应该使用的最小可用屏幕高度，以“<code>dp</code>”为单位，由 <code><N></code> 值定义。在横向和纵向之间切换时，为了匹配当前实际高度，此配置值也会随之发生变化。</p> <p>应用为多个资源目录提供不同的此配置值时，系统会使用最接近（但未超出）设备当前屏幕高度的值。此处的值考虑到了屏幕装饰元素，因此如果设备显示屏的上边缘或下边缘有一些永久性 UI 元素，考虑到这些 UI 元素，同时为减少应用的可用空间，它会使用小于实际屏幕尺寸的高度值。非固定的屏幕装饰元素（例如，全屏时可隐藏的手机状态栏）并不在考虑范围内，标题栏或操作栏等窗口装饰也不在考虑范围内，因此应用必须准备好处理稍小于其所指定值的空间。</p> <p>此项为 API 级别 13 中新增配置。</p> <p>另请参阅 <code>screenHeightDp</code> 配置字段，该字段存放当前屏幕宽度。</p> <p>如需了解有关设计不同屏幕和使用此限定符的详细信息，请参阅支持多种屏幕开发者指南。</p>
屏幕尺寸	<code>small</code> <code>normal</code> <code>large</code> <code>xlarge</code>	<p><code>small</code>：尺寸类似于低密度 QVGA 屏幕的屏幕。小屏幕的最小布局尺寸约为 320x426 dp 单位。例如，QVGA 低密度屏幕和 VGA 高密度屏幕。</p> <p><code>normal</code>：尺寸类似于中等密度 HVGA 屏幕的屏幕。标准屏幕的最小布局尺寸约为 320x470 dp 单位。例如，WQVGA 低密度屏幕、HVGA 中等密度屏幕、WVGA 高密度屏幕。</p> <p><code>large</code>：尺寸类似于中等密度 VGA 屏幕的屏幕。大屏幕的最小布局尺寸约为 480x640 dp 单位。例如，VGA 和 WVGA 中等密度屏幕。</p> <p><code>xlarge</code>：明显大于传统中等密度 HVGA 屏幕的屏幕。超大屏幕的最小布局尺寸约为 720x960 dp 单位。在大多数情况下，屏幕超大的设备体积过大，不能放进口袋，最常见的是平板式设备。API 级别 9 中的新增配置。</p> <p>注：使用尺寸限定符并不表示资源仅适用于该尺寸的屏幕。如果没有为备用资源提供最符合当前设备配置的限定符，则系统可能使用其中最匹配的资源。</p> <p>注意：如果所有资源均使用大于当前屏幕的尺寸限定符，则系统不会使用这些资源，并且应用在运行时将会崩溃（例如，如果所有布局资源均用 <code>xlarge</code> 限定符标记，但设备是标准尺寸的屏幕）。</p> <p>此项为 API 级别 4 中新增配置。</p> <p>如需了解详细信息，请参阅支持多种屏幕。</p> <p>另请参阅 <code>screenLayout</code> 配置字段，该字段表示屏幕是小尺寸、标准尺寸还是大尺寸。</p>
屏幕纵横比	<code>long</code> <code>notlong</code>	<p><code>long</code>：宽屏，如 WQVGA、WVGA、FWVGA</p> <p><code>notlong</code>：非宽屏，如 QVGA、HVGA 和 VGA</p> <p>此项为 API 级别 4 中新增配置。</p>

		<p>它完全基于屏幕的纵横比（宽屏较宽），而与屏幕方向无关。</p> <p>另请参阅 screenLayout 配置字段，该字段指示屏幕是否为宽屏。</p>
圆形屏幕	<code>round</code> <code>notround</code>	<p><code>round</code>：圆形屏幕，例如圆形可穿戴式设备</p> <p><code>notround</code>：方形屏幕，例如手机或平板电脑</p> <p>此项为 API 级别 23 中新增配置。</p> <p>另请参阅 isScreenRound() 配置方法，其指示屏幕是否为宽屏。</p>
屏幕方向	<code>port</code> <code>land</code>	<p><code>port</code>：设备处于纵向（垂直）</p> <p><code>land</code>：设备处于横向（水平）</p> <p>如果用户旋转屏幕，它有可能在应用生命周期中发生改变。如需了解这会在运行期间给应用带来哪些影响，请参阅处理运行时变更。</p> <p>另请参阅 orientation 配置字段，该字段指示当前的设备方向。</p>
UI 模式	<code>car</code> <code>desk</code> <code>television</code> <code>appliance</code> <code>watch</code>	<p><code>car</code>：设备正在车载手机座上显示</p> <p><code>desk</code>：设备正在桌面手机座上显示</p> <p><code>television</code>：设备正在电视上显示，为用户提供“十英尺”体验，其 UI 位于远离用户的大屏幕上，主要面向方向键或其他非指针式交互</p> <p><code>appliance</code>：设备用作不带显示屏的装置</p> <p><code>watch</code>：设备配有显示屏，戴在手腕上</p> <p>此项为 API 级别 8 中新增配置，API 13 中新增电视配置，API 20 中新增手表配置。</p> <p>如需了解应用在设备插入手机座或从中移除时的响应方式，请阅读确定并监控插接状态和类型。</p> <p>如果用户将设备放入手机座中，它有可能在应用生命周期中发生改变。可以使用 UiModeManager 启用或禁用其中某些模式。如需了解这会在运行期间给应用带来哪些影响，请参阅处理运行时变更。</p>
夜间模式	<code>night</code> <code>notnight</code>	<p><code>night</code>：夜间</p> <p><code>notnight</code>：白天</p> <p>此项为 API 级别 8 中新增配置。</p> <p>如果夜间模式停留在自动模式（默认），它有可能在应用生命周期中发生改变。在这种情况下，该模式会根据当时的时间进行调整。可以使用 UiModeManager 启用或禁用此模式。如需了解这会在运行期间给应用带来哪些影响，请参阅处理运行时变更。</p>
屏幕像素密度 (dpi)	<code>ldpi</code> <code>mdpi</code> <code>hdpi</code> <code>xhdpi</code> <code>xxhdpi</code> <code>xxxhdpi</code> <code>nodpi</code> <code>tvdpi</code> <code>anydpi</code>	<p><code>ldpi</code>：低密度屏幕；约为 120dpi。</p> <p><code>mdpi</code>：中等密度（传统 HVGA）屏幕；约为 160dpi。</p> <p><code>hdpi</code>：高密度屏幕；约为 240dpi。</p> <p><code>xhdpi</code>：超高密度屏幕；约为 320dpi。此项为 API 级别 8 中新增配置</p> <p><code>xxhdpi</code>：超超高密度屏幕；约为 480dpi。此项为 API 级别 16 中新增配置</p> <p><code>xxxhdpi</code>：超超超高密度屏幕使用（仅限启动器图标，请参阅“支持多种屏幕”中的注释）；约为 640dpi。此项为 API 级别 18 中新增配置</p> <p><code>nodpi</code>：它可用于您不希望缩放以匹配设备密度的位图资源。</p> <p><code>tvdpi</code>：密度介于 mdpi 和 hdpi 之间的屏幕；约为 213dpi。它并不是“主要”密度组，主要用于电视，而大多数应用都不需要它。对于大多数应用而言，提供 mdpi 和 hdpi 资源便已足够，系统将根据需要对其进行缩放。此项为 API 级别 13 中新增配置</p>

		<p><code>anydpi</code>：此限定符适合所有屏幕密度，其优先级高于其他限定符。这对于矢量可绘制对象很有用。此项为 API 级别 21 中新增配置</p> <p>六个主要密度之间的缩放比为 3:4:6:8:12:16（忽略 tvdpi 密度）。因此，9x9 (ldpi) 位图相当于 12x12 (mdpi)、18x18 (hdpi)、24x24 (xhdpi) 位图，依此类推。</p> <p>如果您认为图像资源在电视或其他某些设备上呈现的效果不够好，而想尝试使用 tvdpi 资源，则缩放比例为 1.33*mdpi。例如，mdpi 屏幕的 100px x 100px 图像应该相当于 tvdpi 的 133px x 133px。</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>注：使用密度限定符并不表示资源仅适用于该密度的屏幕。如果没有为备用资源提供最符合当前设备配置的限定符，则系统可能使用其中最匹配的资源。</p> </div> <p>如需了解有关如何处理不同屏幕密度以及 Android 如何缩放位图以适应当前密度的详细信息，请参阅支持多种屏幕。</p>
触摸屏类型	<code>notouch</code> <code>finger</code>	<p><code>notouch</code>：设备没有触摸屏。</p> <p><code>finger</code>：设备有一个专供用户通过手指直接与其交互的触摸屏。</p> <p>另请参阅 touchscreen 配置字段，该字段指示设备上的触摸屏类型。</p>
键盘可用性	<code>keysexposed</code> <code>keyshidden</code> <code>keyssoft</code>	<p><code>keysexposed</code>：设备具有可用的键盘。如果设备启用了软键盘（不无可能），那么即使硬键盘没有展示给用户，哪怕设备没有硬键盘，也可以使用此限定符。如果没有提供或已经禁用软键盘，则只有在显示硬键盘时才会使用此限定符。</p> <p><code>keyshidden</code>：设备具有可用的硬键盘，但它处于隐藏状态，且设备没有启用软键盘。</p> <p><code>keyssoft</code>：设备已经启用软键盘（无论是否可见）。</p> <p>如果提供了 <code>keysexposed</code> 资源，但未提供 <code>keyssoft</code> 资源，那么只要系统已经启用软键盘，就会使用 <code>keysexposed</code> 资源，而不考虑键盘是否可见。</p> <p>如果用户打开硬键盘，它有可能在应用生命周期中发生改变。如需了解这会在运行期间给应用带来哪些影响，请参阅处理运行时变更。</p> <p>另请参阅配置字段 <code>hardKeyboardHidden</code> 和 <code>keyboardHidden</code>，这两个字段分别指示硬键盘的可见性和任何一种键盘（包括软键盘）的可见性。</p>
主要文本输入法	<code>nokeys</code> <code>qwerty</code> <code>12key</code>	<p><code>nokeys</code>：设备没有用于文本输入的硬按键。</p> <p><code>qwerty</code>：设备具有标准硬键盘（无论是否对用户可见）。</p> <p><code>12key</code>：设备具有 12 键硬键盘（无论是否对用户可见）。</p> <p>另请参阅 keyboard 配置字段，该字段指示可用的主要文本输入法。</p>
导航键可用性	<code>navexposed</code> <code>navhidden</code>	<p><code>navexposed</code>：导航键可供用户使用。</p> <p><code>navhidden</code>：导航键不可用（例如，位于密封盖子后面）。</p> <p>如果用户显示导航键，它有可能在应用生命周期中发生改变。如需了解这会在运行期间给应用带来哪些影响，请参阅处理运行时变更。</p> <p>另请参阅 navigationHidden 配置字段，该字段指示导航键是否处于隐藏状态。</p>
主要非触摸导航方法	<code>nonav</code> <code>dpad</code> <code>trackball</code> <code>wheel</code>	<p><code>nonav</code>：除了使用触摸屏以外，设备没有其他导航设施。</p> <p><code>dpad</code>：设备具有用于导航的方向键。</p> <p><code>trackball</code>：设备具有用于导航的轨迹球。</p> <p><code>wheel</code>：设备具有用于导航的方向盘（不常见）。</p> <p>另请参阅 navigation 配置字段，该字段指示可用的导航方法类型。</p>
平台版本 (API 级别)	示例： <code>v3</code>	设备支持的 API 级别。例如， <code>v1</code> 对应于 API 级别 1（带有 Android 1.0 或更高版本系统的设备）， <code>v4</code> 对应于 API 级别 4（带有 Android 1.6 或更高版本系统的设备）。如需了解有关这些值的

v4
v7
等等

详细信息，请参阅 [Android API 级别](#) 文档。

注：有些配置限定符是从 Android 1.0 才开始添加，因此并非所有版本的 Android 系统都支持所有限定符。使用新限定符会隐式添加平台版本限定符，因此较旧版本系统的设备必然会忽略它。例如，使用 `w600dp` 限定符会自动包括 `v13` 限定符，因为可用宽度限定符是 API 级别 13 中的新增配置。为了避免出现任何问题，请始终包含一组默认资源（一组“不带限定符”的资源）。如需了解详细信息，请参阅[利用资源提供最佳设备兼容性部分](#)。

限定符命名规则

以下是一些关于使用配置限定符名称的规则：

- 您可以为单组资源指定多个限定符，并使用短划线分隔。例如，`drawable-en-rUS-land` 适用于横排美国英语设备。
- 这些限定符必须遵循表 2 中列出的顺序。例如：
 - 错误：`drawable-hdpi-port/`
 - 正确：`drawable-port-hdpi/`
- 不能嵌套备用资源目录。例如，您不能拥有 `res/drawable/drawable-en/`。
- 值不区分大小写。在处理之前，资源编译器会将目录名称转换为小写，以避免不区分大小写的文件系统出现问题。名称中使用的任何大写字母只是为了便于认读。
- 对于每种限定符类型，仅支持一个值。例如，若要对西班牙语和法语使用相同的可绘制对象文件，则您肯定不能拥有名为 `drawable-rES-rFR/` 的目录，而是需要两个包含相应文件的资源目录，如 `drawable-rES/` 和 `drawable-rFR/`。然而，实际上您无需将相同的文件都复制到这两个位置。相反，您可以创建指向资源的别名。请参阅下面的[创建别名资源](#)。

将备用资源保存到以这些限定符命名的目录中之后，Android 会根据当前设备配置在应用中自动应用这些资源。每次请求资源时，Android 都会检查备用资源目录是否包含所请求的资源文件，然后[查找最佳匹配资源](#)（下文进行介绍）。如果没有与特定设备配置匹配的备用资源，则 Android 会使用相应的默认资源（一组用于不含配置限定符的特定资源类型的资源）。

创建别名资源

如果您想将某一资源用于多种设备配置（但是不想作为默认资源提供），则无需将同一资源放入多个备用资源目录中。相反，您可以（在某些情况下）创建备用资源，充当保存在默认资源目录下的资源的别名。

注：并非所有资源都会提供相应机制让您创建指向其他资源的别名。特别是，`xml/` 目录中的动画资源、菜单资源、原始资源以及其他未指定资源均不提供此功能。

例如，假设您有一个应用图标 `icon.png`，并且需要不同语言区域的独特版本。但是，加拿大英语和加拿大法语这两种语言区域需要使用同一版本。您可能会认为需要将相同的图像复制到加拿大英语和加拿大法语对应的资源目录中，但事实并非如此。相反，您可以将用于二者的图像另存为 `icon_ca.png`（除 `icon.png` 以外的任何名称），并将其放入默认 `res/drawable/` 目录中。然后，在 `res/drawable-en-rCA/` 和 `res/drawable-fr-rCA/` 中创建 `icon.xml` 文件，使用 `<bitmap>` 元素引用 `icon_ca.png` 资源。这样，您只需存储 PNG 文件的一个版本和两个指向该版本的小型 XML 文件。（XML 文件示例如下。）

可绘制对象

要创建指向现有可绘制对象的别名，请使用 `<bitmap>` 元素。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/icon_ca" />
```

如果将此文件另存为 `icon.xml`（例如，在备用资源目录中，另存为 `res/drawable-en-rCA/`），则会编译到可作为 `R.drawable.icon` 引用的资源中，但实际上它是 `R.drawable.icon_ca` 资源（保存在 `res/drawable/` 中）的别名。

布局

要创建指向现有布局的别名，请使用包装在 `<merge>` 中的 `<include>` 元素。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<merge>
    <include layout="@layout/main_ltr"/>
</merge>
```

如果将此文件另存为 `main.xml`，则会编译到可作为 `R.layout.main` 引用的资源中，但实际上它是 `R.layout.main_ltr` 资源的别名。

字符串和其他简单值

要创建指向现有字符串的别名，只需将所需字符串的资源 ID 用作新字符串的值即可。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello</string>
    <string name="hi">@string/hello</string>
</resources>
```

`R.string.hi` 资源现在是 `R.string.hello` 的别名。

[其他简单值](#) 的原理相同。例如，颜色：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#f00</color>
    <color name="highlight">@color/red</color>
</resources>
```

利用资源提供最佳设备兼容性

要使应用支持多种设备配置，则务必为应用使用的每种资源类型提供默认资源，这一点非常重要。

例如，如果应用支持多种语言，请始终包含不带[语言和区域限定符](#)的 `values/` 目录（用于保存字符串）。相反，如果您将所有字符串放入带有语言和区域限定符的目录中，则在语言设置不支持您的字符串的设备上运行应用时，应用将会崩溃。但是，只要提供默认 `values/` 资源，应用就会正常运行（即使用户不理解该语言，这也总比崩溃要好）。

同样，如果您根据屏幕方向提供不同的布局资源，则应选择一个方向作为默认方向。例如，不要在 `layout-land/` 和 `layout-port/` 中分别提供横向和纵向的布局资源，而是保留其中之一作为默认设置，例如：`layout/` 用于横向，`layout-port/` 用于纵向。

提供默认资源至关重要，这不仅仅因为应用可能在超出预期的配置上运行，也因为新版 Android 有时会添加旧版本不支持的配置限定符。若要使用新的资源限定符，又希望维持对旧版 Android 的代码兼容性，则当旧版 Android 运行应用时，如果不提供默认资源，应用将会崩溃，这是因为它无法使用以新限定符命名的资源。例如，如果将 `minSdkVersion` 设置为 4，并使用[夜间模式](#)（`night` 或 `notnight`，API 级别 8 中新增配置）限定所有可绘制对象资源，则 API 级别 4 设备无法访问可绘制对象资源，而且会崩溃。在这种情况下，您可能希望 `notnight` 成为默认资源，为此，您应排除该限定符，使可绘制对象资源位于 `drawable/` 或 `drawable-night/` 中。

因此，为了提供最佳设备兼容性，请始终为应用正确运行所必需的资源提供默认资源。然后，使用配置限定符为特定的设备配置创建备用资源。

这条规则有一个例外：如果应用的 `minSdkVersion` 为 4 或更高版本，则在提供带[屏幕密度](#)限定符的备用可绘制对象资源时，不需要默认可绘制对象资源。即使没有默认可绘制对象资源，Android 也可以从备用屏幕密度中找到最佳匹配项并根据需要缩放位图。但是，为了在所有类型的设备上提供最佳体验，您应该为所有三种类型的密度提供备用可绘制对象。

Android 如何查找最佳匹配资源

当您请求要为其提供备用资源的资源时，Android 会根据当前的设备配置选择要在运行时使用的备用资源。为演示 Android 如何选择备用资源，假设以下可绘制对象目录分别包含相同图像的不同版本：

```
drawable/  
drawable-en/  
drawable-fr-rCA/  
drawable-en-port/  
drawable-en-notouch-12key/  
drawable-port-ldpi/  
drawable-port-notouch-12key/
```

同时，假设设备配置如下：

语言区域 = en-GB
屏幕方向 = port
屏幕像素密度 = hdpi
触摸屏类型 = notouch
主要文本输入法 = 12key

通过将设备配置与可用的备用资源进行比较，Android 从 `drawable-en-port` 中选择可绘制对象。

系统使用以下逻辑决定要使用的资源：

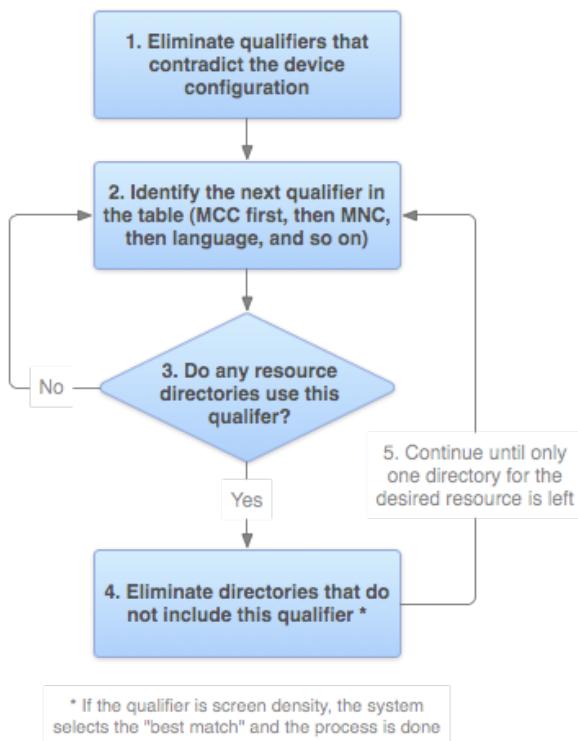


图 2. Android 如何查找最佳匹配资源的流程图。

1. 淘汰与设备配置冲突的资源文件。

`drawable-fr-rCA/` 目录与 `en-GB` 语言区域冲突，因而被淘汰。

```
drawable/  
drawable-en/  
del drawable-fr-rCA/  
drawable-en-port/  
drawable-en-notouch-12key/  
drawable-port-ldpi/  
drawable-port-notouch-12key/
```

例外：屏幕像素密度是唯一一个未因冲突而被淘汰的限定符。尽管设备的屏幕密度为 `hdpi`，但是 `drawable-port-ldpi/` 未被淘汰，因为此时每个屏幕密度均视为匹配。如需了解详细信息，请参阅[支持多种屏幕](#)文档。

2. 选择列表（表 2）中（下一个）优先级最高的限定符。（先从 MCC 开始，然后下移。）

3. 是否有资源目录包括此限定符？

- 若无，请返回到第 2 步，看看下一个限定符。（在该示例中，除非达到语言限定符，否则答案始终为“否”。）

- 若有，请继续执行第 4 步。

4. 淘汰不含此限定符的资源目录。在该示例中，系统会淘汰所有不含语言限定符的目录。

```
drawable/
drawable-en/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

例外：如果涉及的限定符是屏幕像素密度，则 Android 会选择最接近设备屏幕密度的选项。通常，Android 倾向于缩小大型原始图像，而不是放大小型原始图像。请参阅[支持多种屏幕](#)。

5. 返回并重复第 2 步、第 3 步和第 4 步，直到只剩下一个目录为止。在此示例中，屏幕方向是下一个判断是否匹配的限定符。因此，未指定屏幕方向的资源被淘汰：

```
drawable-en/
drawable-en-port/
drawable-en-notouch-12key/
```

剩下的目录是 `drawable-en-port`。

尽管对所请求的每个资源均执行此程序，但是系统仍会对某些方面做进一步优化。例如，系统一旦知道设备配置，即会淘汰可能永远无法匹配的备用资源。比如说，如果配置语言是英语（“en”），则系统绝不会将语言限定符设置为非英语的任何资源目录包含在选中的资源池中（不过，仍会将不带语言限定符的资源目录包含在该池中）。

根据屏幕尺寸限定符选择资源时，如果没有更好的匹配资源，则系统将使用专为小于当前屏幕的屏幕而设计的资源（例如，如有必要，大尺寸屏幕将使用标准尺寸的屏幕资源）。但是，如果唯一可用的资源大于当前屏幕，则系统**不会**使用这些资源，并且如果没有其他资源与设备配置匹配，应用将会崩溃（例如，如果所有布局资源均用 `xlarge` 限定符标记，但设备是标准尺寸的屏幕）。

注：限定符的优先顺序（[表 2](#) 中）比与设备完全匹配的限定符数量更加重要。例如，在上面的第 4 步中，列表剩下的最后选项包括三个与设备完全匹配的限定符（方向、触摸屏类型和输入法），而 `drawable-en` 只有一个匹配参数（语言）。但是，语言的优先顺序高于其他两个限定符，因此 `drawable-port-notouch-12key` 被淘汰。

如需了解有关如何在应用中使用资源的更多信息，请转至[访问资源](#)。



访问资源

内容快览

- › 可以使用 `R.drawable.myimage` 等来自 `R.java` 的整型数在代码中引用资源
- › 可以使用 `@drawable/myimage` 等特殊 XML 语法在资源中引用资源
- › 您还可以通过 `Resources` 中的方法访问您的应用资源

关键类

- › `Resources`

本文内容

- › [在代码中访问资源](#)
- › [在 XML 中访问资源](#)
 - › [引用样式属性](#)
 - › [访问平台资源](#)

另请参阅

- › [提供资源](#)
- › [资源类型](#)

您在应用中提供资源后（[提供资源](#)中对此做了阐述），可通过引用其资源 ID 来应用该资源。所有资源 ID 都在您项目的 `R` 类中定义，后者由 `aapt` 工具自动生成。

编译应用时，`aapt` 会生成 `R` 类，其中包含您的 `res/` 目录中所有资源的资源 ID。每个资源类型都有对应的 `R` 子类（例如，`R.drawable` 对应于所有可绘制对象资源），而该类型的每个资源都有对应的静态整型数（例如，`R.drawable.icon`）。这个整型数就是可用来检索资源的资源 ID。

尽管资源 ID 是在 `R` 类中指定的，但您应该永远都不需要在其中查找资源 ID。资源 ID 始终由以下部分组成：

- **资源类型**：每个资源都被分到一个“类型”组中，例如 `string`、`drawable` 和 `layout`。如需了解有关不同类型的详细信息，请参阅[资源类型](#)。
- **资源名称**，它是不包括扩展名的文件名；或是 XML `android:name` 属性中的值，如果资源是简单值的话（例如字符串）。

访问资源的方法有两种：

- **在代码中**：使用来自 `R` 类的某个子类的静态整型数，例如：

`R.string.hello`

`string` 是资源类型，`hello` 是资源名称。当您提供此格式的资源 ID 时，有许多 Android API 可以访问您的资源。请参阅[在代码中访问资源](#)。

- **在 XML 中**：使用同样与您 `R` 类中定义的资源 ID 对应的特殊 XML 语法，例如：

`@string/hello`

`string` 是资源类型，`hello` 是资源名称。您可以在 XML 资源中任何应该存在您在资源中所提供值的地方使用此语法。请参阅[在 XML 中访问资源](#)。

在代码中访问资源

您可以通过以方法参数的形式传递资源 ID，在代码中使用资源。例如，您可以设置一个 `ImageView`，以利用 `setImageResource()` 使用 `res/drawable/myimage.png` 资源：

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

您还可以利用 `Resources` 中的方法检索个别资源，您可以通过 `getResources()` 获得资源实例。

语法

以下是在代码中引用资源的语法：

```
[<package_name>.]R.<resource_type>.<resource_name>
```

- `<package_name>` 是资源所在包的名称（如果引用的资源来自您自己的资源包，则不需要）。
- `<resource_type>` 是资源类型的 `R` 子类。
- `<resource_name>` 是不带扩展名的资源文件名，或 XML 元素中的 `android:name` 属性值（如果资源是简单值）。

如需了解有关各资源类型及其引用方法的详细信息，请参阅[资源类型](#)。

用例

有许多方法接受资源 ID 参数，您可以利用 `Resources` 中的方法检索资源。您可以通过 `Context.getResources()` 获得 `Resources` 的实例。

以下是一些在代码中访问资源的示例：

```
// Load a background for the current screen from a drawable resource
getWindow().setBackgroundDrawableResource(R.drawable.my_background_image);

// Set the Activity title by getting a string from the Resources object, because
// this method requires a CharSequence rather than a resource ID
getWindow().setTitle(getResources().getText(R.string.main_title));

// Load a custom layout for the current screen
setContentView(R.layout.main_screen);

// Set a slide in animation by getting an Animation from the Resources object
mFlipper.setInAnimation(AnimationUtils.loadAnimation(this,
    R.anim.hyperspace_in));

// Set the text on a TextView object using a resource ID
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

访问原始文件

尽管并不常见，但您的确有可能需要访问原始文件和目录。如果确有需要，则将您的文件保存在 `res/` 中不起作用，因为从 `res/` 读取资源的唯一方法是使用资源 ID。您可以改为将资源保存在 `assets/` 目录中。

保存在 `assets/` 目录中的文件没有资源 ID，因此您无法通过 `R` 类或在 XML 资源中引用它们。您可以改为采用类似普通文件系统的方式查询 `assets/` 目录中的文件，并利用 `AssetManager` 读取原始数据。

不过，如果只需要读取原始数据（例如视频文件或音频文件）的能力，则可将文件保存在 `res/raw/` 目录中，并利用 `openRawResource()` 读取字节流。

注意：切勿手动修改 `R.java` 文件 — 它是在编译您的项目时由 `aapt` 工具生成的。您下次编译时所有更改都会被替代。

您可以利用对现有资源的引用为某些 XML 属性和元素定义值。创建布局文件时，为给您的小部件提供字符串和图像，您经常要这样做。

例如，如果您为布局添加一个 `Button`，应该为按钮文本使用[字符串资源](#)：

```
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/submit" />
```

语法

以下是在 XML 资源中引用资源的语法：

```
@[<package_name>:]<resource_type>/<resource_name>
```

- <package_name> 是资源所在包的名称（如果引用的资源来自相同的包，则不需要）
- <resource_type> 是资源类型的 R 子类
- <resource_name> 是不带扩展名的资源文件名，或 XML 元素中的 android:name 属性值（如果资源是简单值）。

如需了解有关各资源类型及其引用方法的详细信息，请参阅[资源类型](#)。

用例

在某些情况下，您必须使用资源作为 XML 中的值（例如，对小部件应用可绘制图像），但您也可以在 XML 中任何接受简单值的地方使用资源。例如，如果您具有以下资源文件，其中包括一个[颜色资源](#)和一个[字符串资源](#)：

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <color name="opaque_red">#f00</color>  
    <string name="hello">Hello!</string>  
</resources>
```

您可以在以下布局文件中使用这些资源来设置文本颜色和文本字符串：

```
<?xml version="1.0" encoding="utf-8"?>  
<EditText xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:textColor="@color/opaque_red"  
    android:text="@string/hello" />
```

在此情况下，您无需在资源引用中指定包名称，因为资源来自您自己的资源包。要引用系统资源，您需要加入包名称。例如：

```
<?xml version="1.0" encoding="utf-8"?>  
<EditText xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:textColor="@android:color/secondary_text_dark"  
    android:text="@string/hello" />
```

注：您应该始终使用字符串资源，以便将您的应用本地化为其他语言。如需了解有关创建备用资源（例如本地化字符串）的信息，请参阅[提供备用资源](#)。如需查看将您的应用本地化为其他语言的完整指南，请参阅[本地化](#)。

您甚至可以在 XML 中使用资源创建别名。例如，您可以创建一个可绘制对象资源，将其作为另一个可绘制对象资源的别名：

```
<?xml version="1.0" encoding="utf-8"?>  
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"  
    android:src="@drawable/other_drawable" />
```

这听起来多余，但对使用备用资源可能很有帮助。阅读更多关于[创建别名资源](#)的内容。

引用样式属性

您可以通过样式属性资源在当前应用的风格主题中引用某个属性的值。通过引用样式属性，您可以不采用为 UI 元素提供硬编码值这种方式，而是通过为 UI 元素设置样式，使其匹配当前风格主题提供的标准变型来定制这些元素的外观。引用样式属性的实质作用是，“在当前风格主题中使用此属性定义的样式”。

要引用样式属性，名称语法几乎与普通资源格式完全相同，只不过将 at 符号 (@) 改为问号 (?)，资源类型部分为可选项。例如：

```
?[<package_name>:] [<resource_type>/]<resource_name>
```

例如，您可以通过以下代码引用一个属性，将文本颜色设置为与系统风格主题的“主要”文本颜色匹配：

```
<EditText id="text"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textColor="?android:textColorSecondary"  
    android:text="@string/hello_world" />
```

在以上代码中，`android:textColor` 属性指定当前风格主题中某个样式属性的名称。Android 现在会使用应用于 `android:textColorSecondary` 样式属性的值作为 `android:textColor` 在这个小部件中的值。由于系统资源工具知道此环境中肯定存在某个属性资源，因此您无需显式声明类型（类型应为 `?android:attr/textColorSecondary`）— 您可以将 `attr` 类型排除在外。

访问平台资源

Android 包含许多标准资源，例如样式、风格主题和布局。要访问这些资源，请通过 `android` 包名称限定您的资源引用。例如，您可以将 Android 提供的布局资源用于 `ListAdapter` 中的列表项：

```
setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myarray));
```

在上例中，`simple_list_item_1` 是平台为 `ListView` 中的项目定义的布局资源。您可以使用它，而不必自行创建列表项布局。如需了解详细信息，请参阅[列表视图开发者指南](#)。



处理运行时变更

本文内容

- › [在配置变更期间保留对象](#)
- › [自行处理配置变更](#)

另请参阅

- › [提供资源](#)
- › [访问资源](#)
- › [加快屏幕方向变更](#)
- › [多窗口生命周期](#)

有些设备配置可能会在运行时发生变化（例如屏幕方向、键盘可用性及语言）。发生这种变化时，Android 会重启正在运行的 `Activity`（先后调用 `onDestroy()` 和 `onCreate()`）。重启行为旨在通过利用与新设备配置匹配的备用资源自动重新加载您的应用，来帮助它适应新配置。

要妥善处理重启行为，`Activity` 必须通过常规的 [Activity 生命周期](#) 恢复其以前的状态，在 `Activity` 生命周期中，Android 会在销毁 `Activity` 之前调用 `onSaveInstanceState()`，以便您保存有关应用状态的数据。然后，您可以在 `onCreate()` 或 `onRestoreInstanceState()` 期间恢复 `Activity` 状态。

要测试应用能否在保持应用状态完好的情况下自行重启，您应该在应用中执行各种任务时调用配置变更（例如，更改屏幕方向）。您的应用应该能够在不丢失用户数据或状态的情况下随时重启，以便处理如下事件：配置发生变化，或者用户收到来电并在应用进程被销毁很久之后返回到应用。要了解如何恢复 `Activity` 状态，请阅读 [Activity 生命周期](#)。

但是，您可能会遇到这种情况：重启应用并恢复大量数据不仅成本高昂，而且给用户留下糟糕的使用体验。在这种情况下，您有两个其他选择：

a. 在配置变更期间保留对象

允许 `Activity` 在配置变更时重启，但是要将有状态对象传递给 `Activity` 的新实例。

b. 自行处理配置变更

阻止系统在某些配置变更期间重启 `Activity`，但要在配置确实发生变化时接回调，这样，您就能够根据需要手动更新 `Activity`。

在配置变更期间保留对象

如果重启 `Activity` 需要恢复大量数据、重新建立网络连接或执行其他密集操作，那么因配置变更而引起的完全重启可能会给用户留下应用运行缓慢的体验。此外，依靠系统通过 `onSaveInstanceState()` 回调为您保存的 `Bundle`，可能无法完全恢复 `Activity` 状态，因为它并非设计用于携带大型对象（例如位图），而且其中的数据必须先序列化，再进行反序列化，这可能会消耗大量内存并使得配置变更速度缓慢。在这种情况下，如果 `Activity` 因配置变更而重启，则可通过保留 `Fragment` 来减轻重新初始化 `Activity` 的负担。此片段可能包含对您要保留的有状态对象的引用。

当 Android 系统因配置变更而关闭 `Activity` 时，不会销毁您已标记为要保留的 `Activity` 的片段。您可以将此类片段添加到 `Activity` 以保留有状态的对象。

要在运行时配置变更期间将有状态的对象保留在片段中，请执行以下操作：

1. 扩展 `Fragment` 类并声明对有状态对象的引用。
2. 在创建片段后调用 `setRetainInstance(boolean)`。
3. 将片段添加到 `Activity`。

4. 重启 Activity 后，使用 [FragmentManager](#) 检索片段。

例如，按如下方式定义片段：

```
public class RetainedFragment extends Fragment {

    // data object we want to retain
    private MyDataObject data;

    // this method is only called once for this fragment
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // retain this fragment
        setRetainInstance(true);
    }

    public void setData(MyDataObject data) {
        this.data = data;
    }

    public MyDataObject getData() {
        return data;
    }
}
```

注意：尽管您可以存储任何对象，但是切勿传递与 [Activity](#) 绑定的对象，例如，[Drawable](#)、[Adapter](#)、[View](#) 或其他任何与 [Context](#) 关联的对象。否则，它将泄漏原始 Activity 实例的所有视图和资源。（泄漏资源意味着应用将继续持有这些资源，但是无法对其进行垃圾回收，因此可能会丢失大量内存。）

然后，使用 [FragmentManager](#) 将片段添加到 Activity。在运行时配置变更期间再次启动 Activity 时，您可以获得片段中的数据对象。例如，按如下方式定义 Activity：

```
public class MyActivity extends Activity {

    private RetainedFragment dataFragment;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // find the retained fragment on activity restarts
        FragmentManager fm = getFragmentManager();
        dataFragment = (DataFragment) fm.findFragmentByTag("data");

        // create the fragment and data the first time
        if (dataFragment == null) {
            // add the fragment
            dataFragment = new DataFragment();
            fm.beginTransaction().add(dataFragment, "data").commit();
            // load the data from the web
            dataFragment.setData(loadMyData());
        }

        // the data is available in dataFragment.getData()
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // store the data in the fragment
        dataFragment.setData(collectMyLoadedData());
    }
}
```

在此示例中，[onCreate\(\)](#) 添加了一个片段或恢复了对它的引用。此外，[onCreate\(\)](#) 还将有状态的对象存储在片段实例内部。[onDestroy\(\)](#)

对所保留的片段实例内的有状态对象进行更新。

自行处理配置变更

如果应用在特定配置变更期间无需更新资源，并且因性能限制您需要尽量避免重启，则可声明 Activity 将自行处理配置变更，这样可以阻止系统重启 Activity。

注：自行处理配置变更可能导致备用资源的使用更为困难，因为系统不会为您自动应用这些资源。只能在您必须避免 Activity 因配置变更而重启这一万般无奈的情况下，才考虑采用自行处理配置变更这种方法，而且对于大多数应用并不建议使用此方法。

要声明由 Activity 处理配置变更，请在清单文件中编辑相应的 `<activity>` 元素，以包含 `android:configChanges` 属性以及代表要处理的配置的值。`android:configChanges` 属性的文档中列出了该属性的可能值（最常用的值包括 "orientation" 和 "keyboardHidden"，分别用于避免因屏幕方向和可用键盘改变而导致重启）。您可以在该属性中声明多个配置值，方法是用管道 | 字符分隔这些配置值。

例如，以下清单文件代码声明的 Activity 可同时处理屏幕方向变更和键盘可用性变更：

```
<activity android:name=".MyActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name">
```

现在，当其中一个配置发生变化时，`MyActivity` 不会重启。相反，`MyActivity` 会收到对 `onConfigurationChanged()` 的调用。向此方法传递 `Configuration` 对象指定新设备配置。您可以通过读取 `Configuration` 中的字段，确定新配置，然后通过更新界面中使用的资源进行适当的更改。调用此方法时，Activity 的 `Resources` 对象会相应地进行更新，以根据新配置返回资源，这样，您就能够在系统不重启 Activity 的情况下轻松重置 UI 的元素。

注意：从 Android 3.2 (API 级别 13) 开始，当设备在纵向和横向之间切换时，“**屏幕尺寸**”**也会发生变化**。因此，在开发针对 API 级别 13 或更高版本（正如 `minSdkVersion` 和 `targetSdkVersion` 属性中所声明）的应用时，若要避免由于设备方向改变而导致运行时重启，则除了 "orientation" 值以外，您还必须添加 "screenSize" 值。也就是说，您必须声明 `android:configChanges="orientation|screenSize"`。但是，如果您的应用面向 API 级别 12 或更低版本，则 Activity 始终会自行处理此配置变更（即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重启 Activity）。

例如，以下 `onConfigurationChanged()` 实现检查当前设备方向：

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Checks the orientation of the screen
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
}
```

`Configuration` 对象代表所有当前配置，而不仅仅是已经变更的配置。大多数时候，您并不在意配置具体发生了哪些变更，而且您可以轻松地重新分配所有资源，为您提供备用资源。例如，由于 `Resources` 对象现已更新，因此您可以通过 `setImageResource()` 重置任何 `ImageView`，并且使用适合于新配置的资源（如[提供资源](#)中所述）。

请注意，`Configuration` 字段中的值是与 `Configuration` 类中的特定常量匹配的整型数。有关要对每个字段使用哪些常量的文档，请参阅 `Configuration` 参考文档中的相应字段。

请谨记：在声明由 Activity 处理配置变更时，您有责任重置要为其提供备用资源的所有元素。如果您声明由 Activity 处理方向变更，而且有些图像应该在横向和纵向之间切换，则必须在 `onConfigurationChanged()` 期间将每个资源重新分配给每个元素。

如果无需基于这些配置变更更新应用，则可 [不用实现](#) `onConfigurationChanged()`。在这种情况下，仍将使用在配置变更之前用到的所有资源，只是您无需重启 Activity。但是，应用应该始终能够在保持之前状态完好的情况下关闭和重启，因此您不得试图通过此方法来逃避在正常 Activity 生命周期期间保持您的应用状态。这不仅仅是因为还存在其他一些无法禁止重启应用的配置变更，还因为有些事件必须由您处理，例如用户离开应用，而在用户返回应用之前该应用已被销毁。

如需了解有关您可以在 Activity 中处理哪些配置变更的详细信息，请参阅 `android:configChanges` 文档和 `Configuration` 类。



Localizing with Resources

Quickview

- Use resource sets to create a localized app.
- Android loads the correct resource set for the user's language and locale.
- If localized resources are not available, Android loads your default resources.

In this document

- [Overview: Resource-Switching in Android](#)
- [Using Resources for Localization](#)
- [Managing strings for localization](#)
- [Localization Tips](#)
- [Testing Localized Applications](#)

See also

- [Localization Checklist](#)
- [Providing Resources](#)
- [Layouts](#)
- [Activity Lifecycle](#)
- [App Translation Service](#)

Android will run on many devices in many regions. To reach the most users, your application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your application will be used.

This document describes best practices for localizing Android applications.

You should already have a working knowledge of Java and be familiar with Android resource loading, the declaration of user interface elements in XML, development considerations such as Activity lifecycle, and general principles of internationalization and localization.

It is good practice to use the Android resource framework to separate the localized aspects of your application as much as possible from the core Java functionality:

- You can put most or all of the *contents* of your application's user interface into resource files, as described in this document and in [Providing Resources](#).
- The *behavior* of the user interface, on the other hand, is driven by your Java code. For example, if users input data that needs to be formatted or sorted differently depending on locale, then you would use Java to handle the data programmatically. This document does not cover how to localize your Java code.

For a short guide to localizing strings in your app, see the training lesson, [Supporting Different Languages](#).

Overview: Resource-Switching in Android

Resources are text strings, layouts, sounds, graphics, and any other static data that your Android application needs. An application can include multiple sets of resources, each customized for a different device configuration. When a user runs the application, Android automatically selects and loads the resources that best match the device.

(This document focuses on localization and locale. For a complete description of resource-switching and all the types of configurations that you can specify — screen orientation, touchscreen type, and so on — see [Providing Alternative Resources](#).)

When you write your app, you create default and alternative resources for your app to use. When users run your app, the Android system selects which resources to load, based upon the device's locale. To create resources, you place files within specially named subdirectories of the project's `res/` directory.

Why Default Resources Are Important

Whenever the application runs in a locale for which you have not provided locale-specific text, Android will load the default strings from `res/values/strings.xml`. If this default file is absent, or if it is missing a string that your application needs, then your application will not run and will show an error. The example below illustrates what can happen when the default text file is incomplete.

Example:

An application's Java code refers to just two strings, `text_a` and `text_b`. This application includes a localized resource file (`res/values-en/strings.xml`) that defines `text_a` and `text_b` in English. This application also includes a default resource file (`res/values/strings.xml`) that includes a definition for `text_a`, but not for `text_b`:

- When this application is launched on a device with locale set to English, the application might run without a problem, because `res/values-en/strings.xml` contains both of the needed text strings.
- However, **the user will see an error message and a Force Close button** when this application is launched on a device set to a language other than English. The application will not load.

To prevent this situation, make sure that a `res/values/strings.xml` file exists and that it defines every needed string. The situation applies to all types of resources, not just strings: You need to create a set of default resource files containing all the resources that your application calls upon — layouts, drawables, animations, etc. For information about testing, see [Testing for Default Resources](#).

Using Resources for Localization

How to Create Default Resources

Put the application's default text in a file with the following location and name:

`res/values/strings.xml` (required directory)

The text strings in `res/values/strings.xml` should use the default language, which is the language that you expect most of your application's users to speak.

The default resource set must also include any default drawables and layouts, and can include other types of resources such as animations.

`res/drawable/`(required directory holding at least one graphic file, for the application's icon on Google Play)

`res/layout/` (required directory holding an XML file that defines the default layout)

`res/anim/` (required if you have any `res/anim-<qualifiers>` folders)

`res/xml/` (required if you have any `res/xml-<qualifiers>` folders)

`res/raw/` (required if you have any `res/raw-<qualifiers>` folders)

Tip: In your code, examine each reference to an Android resource. Make sure that a default resource is defined for each one. Also make sure that the default string file is complete: A *localized* string file can contain a subset of the strings, but the *default* string file must contain them all.

How to Create Alternative Resources

A large part of localizing an application is providing alternative text for different languages. In some cases you will also provide alternative graphics, sounds, layouts, and other locale-specific resources.

An application can specify many `res/<qualifiers>/` directories, each with different qualifiers. To create an alternative resource for a different locale, you use a qualifier that specifies a language or a language-region combination. (The name of a resource directory must conform to the naming scheme described in [Providing Alternative Resources](#), or else it will not compile.)

Example:

Suppose that your application's default language is English. Suppose also that you want to localize all the text in your application to French,

and most of the text in your application (everything except the application's title) to Japanese. In this case, you could create three alternative `strings.xml` files, each stored in a locale-specific resource directory:

1. `res/values/strings.xml`

Contains English text for all the strings that the application uses, including text for a string named `title`.

2. `res/values-fr/strings.xml`

Contain French text for all the strings, including `title`.

3. `res/values-ja/strings.xml`

Contain Japanese text for all the strings *except* `title`.

If your Java code refers to `R.string.title`, here is what will happen at runtime:

- If the device is set to any language other than French, Android will load `title` from the `res/values/strings.xml` file.
- If the device is set to French, Android will load `title` from the `res/values-fr/strings.xml` file.

Notice that if the device is set to Japanese, Android will look for `title` in the `res/values-ja/strings.xml` file. But because no such string is included in that file, Android will fall back to the default, and will load `title` in English from the `res/values/strings.xml` file.

Which Resources Take Precedence?

If multiple resource files match a device's configuration, Android follows a set of rules in deciding which file to use. Among the qualifiers that can be specified in a resource directory name, **locale almost always takes precedence**.

Example:

Assume that an application includes a default set of graphics and two other sets of graphics, each optimized for a different device setup:

- `res/drawable/`

Contains default graphics.

- `res/drawable-small-land-stylus/`

Contains graphics optimized for use with a device that expects input from a stylus and has a QVGA low-density screen in landscape orientation.

- `res/drawable-ja/`

Contains graphics optimized for use with Japanese.

If the application runs on a device that is configured to use Japanese, Android will load graphics from `res/drawable-ja/`, even if the device happens to be one that expects input from a stylus and has a QVGA low-density screen in landscape orientation.

Exception: The only qualifiers that take precedence over locale in the selection process are MCC and MNC (mobile country code and mobile network code).

Example:

Assume that you have the following situation:

- The application code calls for `R.string.text_a`

- Two relevant resource files are available:

- `res/values-mcc404/strings.xml`, which includes `text_a` in the application's default language, in this case English.
- `res/values-hi/strings.xml`, which includes `text_a` in Hindi.

- The application is running on a device that has the following configuration:

- The SIM card is connected to a mobile network in India (MCC 404).
- The language is set to Hindi (`hi`).

Android will load `text_a` from `res/values-mcc404/strings.xml` (in English), even if the device is configured for Hindi. That is because in the resource-selection process, Android will prefer an MCC match over a language match.

The selection process is not always as straightforward as these examples suggest. Please read [How Android Finds the Best-matching Resource](#) for a more nuanced description of the process. All the qualifiers are described and listed in order of precedence in [Table 2 of Providing Alternative Resources](#).

Referring to Resources in Java

In your application's Java code, you refer to resources using the syntax `R.resource_type.resource_name` or `android.R.resource_type.resource_name`. For more about this, see [Accessing Resources](#).

Managing strings for localization

Move all strings into strings.xml

As you build your apps, do not hard code any string. Instead, declare *all* of your strings as resources in a default `strings.xml` file, which makes it easy to update and localize. Strings in the `strings.xml` file can then be easily extracted, translated, and integrated back into your app (with appropriate qualifiers) without any changes to the compiled code.

If you generate images with text, put those strings in `strings.xml` as well, and regenerate the images after translation.

Follow Android guidelines for UI strings

As you design and develop your UIs, make sure that you pay close attention to *how* you talk to your user. In general, use a [succinct and compressed style](#) that is friendly but brief, and use a consistent style throughout your UIs.

Make sure that you read and follow the Material Design recommendations for [writing style and word choice](#). Doing so will make your apps appear more polished to the user and will help users understand your UI more quickly.

Also, always use Android standard terminology wherever possible—such as for UI elements such as Action Bar, Options Menu, System Bar, Notifications, and so on. Using Android terms correctly and consistently makes translation easier and results in a better end-product for users.

Provide sufficient context for declared strings

As you declare strings in your `strings.xml` file, make sure to describe the context in which the string is used. This information will be invaluable to the translator and result in better quality translation. It will also help you manage your strings more effectively.

Here is an example:

```
<!-- The action for submitting a form. This text is on a button that can fit 30 chars -->
<string name="login_submit_button">Sign in</string>
```

Consider providing context information that may include:

- What is this string for? When and where is it presented to the user?
- Where is this in the layout? For example, if it is a button, translations are less flexible than if it were a text box.

Mark message parts that should not be translated

Often strings contain text that should not be translated into other languages. Common examples might be a piece of code, a placeholder for a value, a special symbol, or a name. As you prepare your strings for translation, look for and mark text that should remain as-is, without translation, so that the translator doesn't change it.

To mark text that should not be translated, use an `<xliif:g>` placeholder tag. Here is an example tag that ensures the text "%1\$s" will not be changed during translation (otherwise it could break the message):

```
<string name="countdown">
  <xliif:g id="time" example="5 days>%1$s</xliif:g>until holiday
</string>
```

When you declare a placeholder tag, always add an id attribute that explains what the placeholder is for. If your apps will later replace the placeholder value, be sure to provide an example attribute to clarify the expected use.

Here are some more examples of placeholder tags:

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <!-- Example placeholder for a special unicode symbol -->
    <string name="star_rating">Check out our 5
        <xlf:g id="star">\u2605</xlf:g>
    </string>
    <!-- Example placeholder for a for a URL -->
    <string name="app_homeurl">
        Visit us at <xlf:g id="application_homepage">http://my/app/home.html</xlf:g>
    </string>
    <!-- Example placeholder for a name -->
    <string name="prod_name">
        Learn more at <xlf:g id="prod_gamegroup">Game Group</xlf:g>
    </string>
    <!-- Example placeholder for a literal -->
    <string name="promo_message">
        Please use the "<xlf:g id="promotion_code">ABCDEFG</xlf:g>" to get a discount.
    </string>
    ...
</resources>
```

Localization Checklist

For a complete overview of the process of localizing and distributing an Android application, see the [Localization Checklist](#) document.

Localization Tips

Design your application to work in any locale

You cannot assume anything about the device on which a user will run your application. The device might have hardware that you were not anticipating, or it might be set to a locale that you did not plan for or that you cannot test. Design your application so that it will function normally or fail gracefully no matter what device it runs on.

Important: Make sure that your application includes a full set of default resources.

Make sure to include `res/drawable/` and a `res/values/` folders (without any additional modifiers in the folder names) that contain all the images and text that your application will need.

If an application is missing even one default resource, it will not run on a device that is set to an unsupported locale. For example, the `res/values/strings.xml` default file might lack one string that the application needs: When the application runs in an unsupported locale and attempts to load `res/values/strings.xml`, the user will see an error message and a Force Close button.

For more information, see [Testing for Default Resources](#).

Design a flexible layout

If you need to rearrange your layout to fit a certain language (for example German with its long words), you can create an alternative layout for that language (for example `res/layout-de/main.xml`). However, doing this can make your application harder to maintain. It is better to create a single layout that is more flexible.

Another typical situation is a language that requires something different in its layout. For example, you might have a contact form that should include two name fields when the application runs in Japanese, but three name fields when the application runs in some other language. You could handle this in either of two ways:

- Create one layout with a field that you can programmatically enable or disable, based on the language, or
- Have the main layout include another layout that includes the changeable field. The second layout can have different configurations for different languages.

Avoid creating more resource files and text strings than you need

You probably do not need to create a locale-specific alternative for every resource in your application. For example, the layout defined in the `res/layout/main.xml` file might work in any locale, in which case there would be no need to create any alternative layout files.

Also, you might not need to create alternative text for every string. For example, assume the following:

- Your application's default language is American English. Every string that the application uses is defined, using American English spellings, in `res/values/strings.xml`.
- For a few important phrases, you want to provide British English spelling. You want these alternative strings to be used when your application runs on a device in the United Kingdom.

To do this, you could create a small file called `res/values-en-GB/strings.xml` that includes only the strings that should be different when the application runs in the U.K. For all the rest of the strings, the application will fall back to the defaults and use what is defined in `res/values/strings.xml`.

Use the Android Context object for manual locale lookup

You can look up the locale using the `Context` object that Android makes available:

```
String locale = context.getResources().getConfiguration().locale.getDisplayName();
```

Use the App Translation Service

The [App Translation Service](#) is integrated into the [Play Console](#), and it is also accessible from [Android Studio](#). It is a quick and simple way to get an instant quote and place an order with a translation company. You can order translations into one or more languages for app UI strings, Play Store Listing text, IAP names, and ad campaign text.

Testing Localized Applications

Testing on a Device

Keep in mind that the device you are testing may be significantly different from the devices available to consumers in other geographies. The locales available on your device may differ from those available on other devices. Also, the resolution and density of the device screen may differ, which could affect the display of strings and drawables in your UI.

To change the locale or language on a device, use the Settings application.

Testing on an Emulator

For details about using the emulator, see See [Android Emulator](#).

Creating and using a custom locale

A "custom" locale is a language/region combination that the Android system image does not explicitly support. You can test how your application will run in a custom locale by creating a custom locale in the emulator. There are two ways to do this:

- Use the Custom Locale application, which is accessible from the Application tab. (After you create a custom locale, switch to it by pressing and holding the locale name.)
- Change to a custom locale from the adb shell, as described below.

When you set the emulator to a locale that is not available in the Android system image, the system itself will display in its default language. Your application, however, should localize properly.

Changing the emulator locale from the adb shell

To change the locale in the emulator by using the adb shell.

1. Pick the locale you want to test and determine its BCP-47 language tag, for example, Canadian French would be `fr-CA`.
2. Launch an emulator.
3. From a command-line shell on the host computer, run the following command:
`adb shell`
or if you have a device attached, specify that you want the emulator by adding the `-e` option:
`adb -e shell`

4. At the adb shell prompt (#), run this command:

```
setprop persist.sys.locale [BCP-47 language tag];stop;sleep 5;start
```

Replace bracketed sections with the appropriate codes from Step 1.

For instance, to test in Canadian French:

```
setprop persist.sys.locale fr-CA;stop;sleep 5;start
```

This will cause the emulator to restart. (It will look like a full reboot, but it is not.) Once the Home screen appears again, re-launch your application, and the application launches with the new locale.

Testing for Default Resources

Here's how to test whether an application includes every string resource that it needs:

1. Set the emulator or device to a language that your application does not support. For example, if the application has French strings in `res/values-fr/` but does not have any Spanish strings in `res/values-es/`, then set the emulator's locale to Spanish. (You can use the Custom Locale application to set the emulator to an unsupported locale.)
2. Run the application.
3. If the application shows an error message and a Force Close button, it might be looking for a string that is not available. Make sure that your `res/values(strings.xml)` file includes a definition for every string that the application uses.

If the test is successful, repeat it for other types of configurations. For example, if the application has a layout file called `res/layout-land/main.xml` but does not contain a file called `res/layout-port/main.xml`, then set the emulator or device to portrait orientation and see if the application will run.



ICU4J Android 框架 API

本文内容：

- [与 ICU4J 的关系](#)
- [从 ICU4J 迁移至 android.icu API](#)
- [授权](#)

另请参阅

- [ICU4J 的文档](#)
- [ICU4J 支持的最新标准](#)

ICU4J 是一个广泛使用的开源 Java 库集合，为软件应用提供 Unicode 和全球化支持。从 Android 7.0 (API 级别 24) 开始，Android 在 `android.icu` 软件包下显示 ICU4J API 子集，供应用开发者使用。这些 API 使用设备上具有的本地化数据。因此，您可以通过不将 ICU4J 库编译到 APK 来减少 APK 占用空间；相反，您可以只在框架中调用它们。（在此情况下，您可能想要提供[多个版本的 APK](#)，这样，运行比 Android 7.0 (API 级别 24) 低的 Android 版本的用户可以下载包含 ICU4J 库的应用版本。）

本文档开头提供了有关支持这些库所需的基本信息。然后，介绍关于 Android 特定的 ICU4J 实现您需要了解的内容。最后，介绍如何在 Android 框架中使用 ICU4J API。

与 ICU4J 的关系

Android 通过`android.icu` 软件包（而非 `com.ibm.icu`）显示 ICU4J API 的子集。由于种种原因，Android 框架可能选择不显示 ICU4J API；例如，Android 不显示一些已弃用的 API 或 ICU 团队尚未将其声明为“稳定”的 API。由于 ICU 团队将来会弃用这些 API，因此，Android 也会将其标记为已弃用，但将继续包含它们。

表 1. Android 中使用的 ICU 和 CLDR 版本]。

Android API 级别	ICU 版本	CLDR 版本
Android 7.0 (API 级别 24)	56	28

以下是几点注意事项：

- ICU4J Android 框架 API 不包含所有的 ICU4J API。
- NDK 开发者应了解 Android ICU4C 不受支持。
- Android 框架中的 API 不会取代 Android 对[使用资源进行本地化](#)的支持。

从 com.ibm.icu 迁移至 android.icu 软件包

如果您已在应用中使用 ICU4J API，且`android.icu` API 符合您的要求，那么要迁移至框架 API，需要将 Java 导入从 `com.ibm.icu` 更改为 `android.icu`。然后，您可以从 APK 移除您自己的 ICU4J 文件的副本。

注：ICU4J 框架 API 使用 `android.icu` 命名空间，而不是 `com.ibm.icu`。这是为了避免在包含自己的 `com.ibm.icu` 库的 APK 中出现命名空间冲突。

从其他 Android SDK API 迁移至 android.icu API

`java` 和 `android` 软件包中的某些类与在 ICU4J 中找到的一些类等效。不过，ICU4J 通常为标准和语言提供更广泛的支持。

下面是一些入门示例：

类	替代项
java.lang.Character	android.icu.lang.UCharacter
java.text.BreakIterator	android.icu.text.BreakIterator
java.text.DecimalFormat	android.icu.text.DecimalFormat
java.util.Calendar	android.icu.util.Calendar
android.text.BidiFormatter	android.icu.text.Bidi
android.text.format.DateFormat	android.icu.text.DateFormat
android.text.format.DateUtils	android.icu.text.DateFormat android.icu.text.RelativeDateFormat

授权

ICU4J 按照 ICU 许可发布。如需了解详情，请参阅 [ICU 用户指南](#)。



语言和语言区域

本文内容：

- [解析语言资源所面临的挑战](#)
- [对资源解析策略的改进](#)
- [设计您的应用以支持附加语言区域](#)

从 Android 7.0 (API 级别 24) 开始，Android 为多语言用户提供增强的支持，让他们可以在设置中选择多个语言区域。Android 通过大幅扩展受支持的语言区域数量并更改系统解析资源的方式来提供此功能。

本文档首先说明低于 7.0 (API 级别 24) 的 Android 版本中的资源解析策略，接着介绍 Android 7.0 中改进的资源解析策略，最后说明如何充分利用扩展的语言区域数量来支持更多的多语言用户。

解析语言资源所面临的挑战

在 Android 7.0 之前，Android 并非始终能够成功匹配应用和系统语言区域。

例如，假设您遇到了以下情况：

- 您的应用的默认语言为 `en_US` (US English)，但它也在 `es_ES` 资源文件中对西班牙字符串进行了本地化。
- 将设备设置为 `es_MX`

当您的 Java 代码引用字符串时，系统会从默认 (`en_US`) 资源文件加载字符串，即使应用在 `es_ES` 下有本地化的西班牙语资源。这是因为当系统无法找到精确匹配时，它会继续通过将国家/地区代码从语言区域中剥离来查找资源。最后，如果未找到匹配，系统会恢复为默认模式，即 `en_US`。

如果用户选择应用根本不支持的语言（如法语），则系统也会默认显示 `en_US`。例如：

表 1. 没有精确语言区域匹配项的资源解析。

用户设置	应用资源	资源解析
<code>fr_CH</code>	默认值 (<code>en</code>) <code>de_DE</code> <code>es_ES</code> <code>fr_FR</code> <code>it_IT</code>	尝试 <code>fr_CH => 失败</code> 尝试 <code>fr => 失败</code> 使用默认值 (<code>en</code>)

在此示例中，系统在不知道用户是否理解英语的情况下显示英语字符串。目前，此行为很常见。

对资源解析策略的改进

Android 7.0 (API 级别 24) 可提供更稳健的资源解析，并自动查找更好的备用方法。不过，为了加速解析和提升可维护性，您应以最常用的母语存储资源。例如，如果您之前将西班牙语资源存储在 `es-US` 目录中，请将它们移动到 `es-419` 目录，该目录包含拉丁美洲西班牙语。同样，如果您在名为 `en-GB` 的文件夹中存储有资源字符串，则将此文件夹重命名为 `en-001` (国际英语)，因为 `en-GB` 字符串的最常用母语为 `en-001`。以下示例介绍为什么这些做法可提升性能和资源解析的可靠性。

资源解析示例

使用 7.0 以上的 Android 版本时，以不同的方式解析表 1 中所描述的案例：

表 2. 针对没有精确语言区域匹配项时改进的资源解析策略。

用户设置	应用资源	资源解析
1. fr_CH	默认值 (en) de_DE es_ES fr_FR it_IT	尝试 fr_CH => 失败 尝试 fr => 失败 尝试 fr 的子项 => fr_FR 使用 fr_FR

现在，用户获得的是法语资源而不是英语。此示例还表明对于 Android 7.0 或更高版本，您为什么应将法语字符串存储在 `fr` (而非 `fr_FR`) 中。此处的操作是匹配最接近的母语，从而使解析更快速且更具预见性。

除了这个改进的解析逻辑外，Android 现在还提供更多的用户语言以供选择。下面，我们将意大利语指定为附加用户语言，但假设应用不支持法语，再次尝试上面的示例。

表 3. 应用仅与用户的次优语言区域设置匹配时的资源解析。

用户设置	应用资源	资源解析
1. fr_CH	默认值 (en)	尝试 fr_CH => 失败
2. it_CH	de_DE es_ES it_IT	尝试 fr => 失败 尝试 fr 的子项 => 失败 尝试 it_CH => 失败 尝试 it => 失败 尝试 it 的子项 => it_IT 使用 it_IT

用户仍会获取他们理解的语言，即使应用不支持法语。

设计您的应用以支持附加语言区域

LocaleList API

从 Android 7.0 (API 级别 24) 开始，Android 显示的 `LocaleList.getDefault()` API 可让应用直接查询用户已指定的语言列表。您可以使用此 API 创建更成熟的应用行为和更优化的内容显示。例如，搜索可以基于用户的设置以多种语言显示结果。浏览器应用可避免翻译以用户理解的语言显示的页面，键盘应用可自动启用所有适用的布局。

格式化程序

直到 Android 6.0 (API 级别 23)，Android 仅支持许多常用语言 (en、es、ar、fr、ru) 的一个或两个语言区域。由于每种语言只有几种变体，因此，应用可以通过在资源文件中将一些数字和日期存储为硬编码字符串解决此问题。不过，随着 Android 扩展了支持的语言区域集，即使在一个语言区域中，日期、时间、货币及类似信息也会存在显著差异。对您的格式进行硬编码会让最终用户困惑不已。因此，在针对 Android 7.0 或更高版本开发应用时请务必使用格式化程序代替硬编码数字和日期字符串。

例如，Android 7.0 和更高版本支持 27 个阿拉伯语言区域。这些语言区域可以共享大多数资源，但其中一些资源首选 ASCII 数字，另一些则首选本地数字。例如，如果您想要创建一个具有数字变量的句子，如“Choose a 4 digit pin”，则按如下所示使用格式化程序：

```
format(locale, "Choose a %d-digit PIN", 4)
```

Resource Types

See also

- [Providing Resources](#)
- [Accessing Resources](#)

Each of the documents in this section describe the usage, format and syntax for a certain type of application resource that you can provide in your resources directory (`res/`).

Here's a brief summary of each resource type:

[Animation Resources](#)

Define pre-determined animations.

Tween animations are saved in `res/anim/` and accessed from the `R.anim` class.

Frame animations are saved in `res/drawable/` and accessed from the `R.drawable` class.

[Color State List Resource](#)

Define a color resources that changes based on the View state.

Saved in `res/color/` and accessed from the `R.color` class.

[Drawable Resources](#)

Define various graphics with bitmaps or XML.

Saved in `res/drawable/` and accessed from the `R.drawable` class.

[Layout Resource](#)

Define the layout for your application UI.

Saved in `res/layout/` and accessed from the `R.layout` class.

[Menu Resource](#)

Define the contents of your application menus.

Saved in `res/menu/` and accessed from the `R.menu` class.

[String Resources](#)

Define strings, string arrays, and plurals (and include string formatting and styling).

Saved in `res/values/` and accessed from the `R.string`, `R.array`, and `R.plurals` classes.

[Style Resource](#)

Define the look and format for UI elements.

Saved in `res/values/` and accessed from the `R.style` class.

[More Resource Types](#)

Define values such as booleans, integers, dimensions, colors, and other arrays.

Saved in `res/values/` but each accessed from unique `R` sub-classes (such as `R.bool`, `R.integer`, `R.dimens`, etc.).

Animation Resources

In this document

- › [Property Animation](#)
- › [View Animation](#)
 - › [Tween animation](#)
 - › [Frame animation](#)

See also

- › [View Animation](#)
- › [Property Animation](#)

An animation resource can define one of two types of animations:

[Property Animation](#)

Creates an animation by modifying an object's property values over a set period of time with an [Animator](#).

[View Animation](#)

There are two types of animations that you can do with the view animation framework:

- [Tween animation](#): Creates an animation by performing a series of transformations on a single image with an [Animation](#)
- [Frame animation](#): or creates an animation by showing a sequence of images in order with an [AnimationDrawable](#).

Property Animation

An animation defined in XML that modifies properties of the target object, such as background color or alpha value, over a set amount of time.

FILE LOCATION:

`res/animator/filename.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a [ValueAnimator](#), [ObjectAnimator](#), or [AnimatorSet](#).

RESOURCE REFERENCE:

In Java: `R.animator.filename`

In XML: `@[package:]animator/filename`

SYNTAX:

```

<set
    android:ordering=["together" | "sequentially">

    <objectAnimator
        android:propertyName="string"
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>

    <animator
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>

    <set>
        ...
    </set>
</set>

```

The file must have a single root element: either `<set>`, `<objectAnimator>`, or `<valueAnimator>`. You can group animation elements together inside the `<set>` element, including other `<set>` elements.

ELEMENTS:

`<set>`

A container that holds other animation elements (`<objectAnimator>`, `<valueAnimator>`, or other `<set>` elements).

Represents an [AnimatorSet](#).

You can specify nested `<set>` tags to further group animations together. Each `<set>` can define its own `ordering` attribute.

attributes:

`android:ordering`

Keyword. Specifies the play ordering of animations in this set.

Value	Description
<code>sequentially</code>	Play animations in this set sequentially
<code>together</code> (default)	Play animations in this set at the same time.

`<objectAnimator>`

Animates a specific property of an object over a specific amount of time. Represents an [ObjectAnimator](#).

attributes:

`android:propertyName`

String. Required. The object's property to animate, referenced by its name. For example you can specify `"alpha"` or `"backgroundColor"` for a View object. The `objectAnimator` element does not expose a `target` attribute, however, so you cannot set the object to animate in the XML declaration. You have to inflate your animation XML resource by calling `loadAnimator()` and call `setTarget()` to set the target object that contains this property.

`android:valueTo`

float, int, or color. **Required.** The value where the animated property ends. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:valueFrom`

float, int, or color. The value where the animated property starts. If not specified, the animation starts at the value obtained by the property's get method. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:duration`

int. The time in milliseconds of the animation. 300 milliseconds is the default.

`android:startOffset`

int. The amount of milliseconds the animation delays after `start()` is called.

`android:repeatCount`

int. How many times to repeat an animation. Set to "`-1`" to infinitely repeat or to a positive integer. For example, a value of "`1`" means that the animation is repeated once after the initial run of the animation, so the animation plays a total of two times. The default value is "`0`", which means no repetition.

`android:repeatMode`

int. How an animation behaves when it reaches the end of the animation. `android:repeatCount` must be set to a positive integer or "`-1`" for this attribute to have an effect. Set to "`reverse`" to have the animation reverse direction with each iteration or "`repeat`" to have the animation loop from the beginning each time.

`android:valueType`

Keyword. Do not specify this attribute if the value is a color. The animation framework automatically handles color values

Value	Description
<code>intType</code>	Specifies that the animated values are integers
<code>floatType</code> (default)	Specifies that the animated values are floats

`<animator>`

Performs an animation over a specified amount of time. Represents a [ValueAnimator](#).

attributes:

`android:valueTo`

float, int, or color. **Required.** The value where the animation ends. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:valueFrom`

float, int, or color. **Required.** The value where the animation starts. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:duration`

int. The time in milliseconds of the animation. 300ms is the default.

`android:startOffset`

int. The amount of milliseconds the animation delays after `start()` is called.

`android:repeatCount`

int. How many times to repeat an animation. Set to `"-1"` to infinitely repeat or to a positive integer. For example, a value of `"1"` means that the animation is repeated once after the initial run of the animation, so the animation plays a total of two times. The default value is `"0"`, which means no repetition.

`android:repeatMode`

int. How an animation behaves when it reaches the end of the animation. `android:repeatCount` must be set to a positive integer or `"-1"` for this attribute to have an effect. Set to `"reverse"` to have the animation reverse direction with each iteration or `"repeat"` to have the animation loop from the beginning each time.

`android:valueType`

Keyword. Do not specify this attribute if the value is a color. The animation framework automatically handles color values.

Value	Description
<code>intType</code>	Specifies that the animated values are integers
<code>floatType</code> (default)	Specifies that the animated values are floats

EXAMPLE:

XML file saved at `res/animator/property_animator.xml`:

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

In order to run this animation, you must inflate the XML resources in your code to an `AnimatorSet` object, and then set the target objects for all of the animations before starting the animation set. Calling `setTarget()` sets a single target object for all children of the `AnimatorSet` as a convenience. The following code shows how to do this:

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,
    R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

SEE ALSO:

- [Property Animation](#)
- [API Demos](#) for examples on how to use the property animation system.

View Animation

The view animation framework supports both tween and frame by frame animations, which can both be declared in XML. The following sections describe how to use both methods.

Tween animation

An animation defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic.

FILE LOCATION:

`res/anim/filename.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to an [Animation](#).

RESOURCE REFERENCE:

In Java: `R.anim.filename`

In XML: `@[package:]anim/filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"] >
    <alpha>
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale>
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate>
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate>
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
    <set>
        ...
    </set>
</set>
```

The file must have a single root element: either an `<alpha>`, `<scale>`, `<translate>`, `<rotate>`, or `<set>` element that holds a group (or groups) of other animation elements (even nested `<set>` elements).

ELEMENTS:

`<set>`

A container that holds other animation elements (`<alpha>`, `<scale>`, `<translate>`, `<rotate>`) or other `<set>` elements. Represents an [AnimationSet](#).

attributes:

`android:interpolator`

Interpolator resource. An [Interpolator](#) to apply on the animation. The value must be a reference to a resource that specifies an interpolator (not an interpolator class name). There are default interpolator resources available from the platform or you can create your own interpolator resource. See the discussion below for more about [Interpolators](#).

`android:shareInterpolator`

Boolean. "true" if you want to share the same interpolator among all child elements.

`<alpha>`

A fade-in or fade-out animation. Represents an [AlphaAnimation](#).

attributes:

`android:fromAlpha`

Float. Starting opacity offset, where 0.0 is transparent and 1.0 is opaque.

`android:toAlpha`

Float. Ending opacity offset, where 0.0 is transparent and 1.0 is opaque.

For more attributes supported by `<alpha>`, see the [Animation](#) class reference (of which, all XML attributes are inherited by this element).

`<scale>`

A resizing animation. You can specify the center point of the image from which it grows outward (or inward) by specifying `pivotX` and `pivotY`. For example, if these values are 0, 0 (top-left corner), all growth will be down and to the right. Represents a [ScaleAnimation](#).

attributes:

`android:fromXScale`

Float. Starting X size offset, where 1.0 is no change.

`android:toXScale`

Float. Ending X size offset, where 1.0 is no change.

`android:fromYScale`

Float. Starting Y size offset, where 1.0 is no change.

`android:toYScale`

Float. Ending Y size offset, where 1.0 is no change.

`android:pivotX`

Float. The X coordinate to remain fixed when the object is scaled.

`android:pivotY`

Float. The Y coordinate to remain fixed when the object is scaled.

For more attributes supported by `<scale>`, see the [Animation](#) class reference (of which, all XML attributes are inherited by this

element).

<translate>

A vertical and/or horizontal motion. Supports the following attributes in any of the following three formats: values from -100 to 100 ending with "%", indicating a percentage relative to itself; values from -100 to 100 ending in "%p", indicating a percentage relative to its parent; a float value with no suffix, indicating an absolute value. Represents a [TranslateAnimation](#).

attributes:

android:fromXDelta

Float or percentage. Starting X offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element width (such as "5%"), or in percentage relative to the parent width (such as "5%p").

android:toXDelta

Float or percentage. Ending X offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element width (such as "5%"), or in percentage relative to the parent width (such as "5%p").

android:fromYDelta

Float or percentage. Starting Y offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element height (such as "5%"), or in percentage relative to the parent height (such as "5%p").

android:toYDelta

Float or percentage. Ending Y offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element height (such as "5%"), or in percentage relative to the parent height (such as "5%p").

For more attributes supported by <`translate`>, see the [Animation](#) class reference (of which, all XML attributes are inherited by this element).

<rotate>

A rotation animation. Represents a [RotateAnimation](#).

attributes:

android:fromDegrees

Float. Starting angular position, in degrees.

android:toDegrees

Float. Ending angular position, in degrees.

android:pivotX

Float or percentage. The X coordinate of the center of rotation. Expressed either: in pixels relative to the object's left edge (such as "5"), in percentage relative to the object's left edge (such as "5%"), or in percentage relative to the parent container's left edge (such as "5%p").

android:pivotY

Float or percentage. The Y coordinate of the center of rotation. Expressed either: in pixels relative to the object's top edge (such as "5"), in percentage relative to the object's top edge (such as "5%"), or in percentage relative to the parent container's top edge (such as "5%p").

For more attributes supported by <`rotate`>, see the [Animation](#) class reference (of which, all XML attributes are inherited by this element).

EXAMPLE:

XML file saved at `res/anim/hyperspace_jump.xml`:

```
<set xmlns:android="http://schemas.android.com/apk/res/android"  
      android:shareInterpolator="false">  
    <scale  
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"  
        android:fromXScale="1.0"  
        android:toXScale="1.4"  
        android:fromYScale="1.0"  
        android:toYScale="0.6"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:fillAfter="false"  
        android:duration="700" />  
    <set  
        android:interpolator="@android:anim/accelerate_interpolator"  
        android:startOffset="700">  
        <scale  
            android:fromXScale="1.4"  
            android:toXScale="0.0"  
            android:fromYScale="0.6"  
            android:toYScale="0.0"  
            android:pivotX="50%"  
            android:pivotY="50%"  
            android:duration="400" />  
        <rotate  
            android:fromDegrees="0"  
            android:toDegrees="-45"  
            android:toYScale="0.0"  
            android:pivotX="50%"  
            android:pivotY="50%"  
            android:duration="400" />  
    </set>  
</set>
```

This application code will apply the animation to an `ImageView` and start the animation:

```
ImageView image = (ImageView) findViewById(R.id.image);  
Animation hyperspaceJump = AnimationUtils.loadAnimation(this, R.anim.hyperspace_jump);  
image.startAnimation(hyperspaceJump);
```

SEE ALSO:

- [2D Graphics: Tween Animation](#)

Interpolators

An interpolator is an animation modifier defined in XML that affects the rate of change in an animation. This allows your existing animation effects to be accelerated, decelerated, repeated, bounced, etc.

An interpolator is applied to an animation element with the `android:interpolator` attribute, the value of which is a reference to an interpolator resource.

All interpolators available in Android are subclasses of the `Interpolator` class. For each interpolator class, Android includes a public resource you can reference in order to apply the interpolator to an animation using the `android:interpolator` attribute. The following table specifies the resource to use for each interpolator:

Interpolator class	Resource ID
<code>AccelerateDecelerateInterpolator</code>	<code>@android:anim/accelerate_decelerate_interpolator</code>
<code>AccelerateInterpolator</code>	<code>@android:anim/accelerate_interpolator</code>
<code>AnticipateInterpolator</code>	<code>@android:anim/anticipate_interpolator</code>
<code>AnticipateOvershootInterpolator</code>	<code>@android:anim/anticipate_overshoot_interpolator</code>

BounceInterpolator	@android:anim/bounce_interpolator
CycleInterpolator	@android:anim/cycle_interpolator
DecelerateInterpolator	@android:anim/decelerate_interpolator
LinearInterpolator	@android:anim/linear_interpolator
OvershootInterpolator	@android:anim/overshoot_interpolator

Here's how you can apply one of these with the `android:interpolator` attribute:

```
<set android:interpolator="@android:anim/accelerate_interpolator">
    ...
</set>
```

Custom interpolators

If you're not satisfied with the interpolators provided by the platform (listed in the table above), you can create a custom interpolator resource with modified attributes. For example, you can adjust the rate of acceleration for the [AnticipateInterpolator](#), or adjust the number of cycles for the [CycleInterpolator](#). In order to do so, you need to create your own interpolator resource in an XML file.

FILE LOCATION:

`res/anim/filename.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to the corresponding interpolator object.

RESOURCE REFERENCE:

In XML: `@[package:]anim/filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<InterpolatorName xmlns:android="http://schemas.android.com/apk/res/android"
    android:attribute_name="value"
/>
```

If you don't apply any attributes, then your interpolator will function exactly the same as those provided by the platform (listed in the table above).

ELEMENTS:

Notice that each [Interpolator](#) implementation, when defined in XML, begins its name in lowercase.

`<accelerateDecelerateInterpolator>`

The rate of change starts and ends slowly but accelerates through the middle.

No attributes.

`<accelerateInterpolator>`

The rate of change starts out slowly, then accelerates.

attributes:

`android:factor`

Float. The acceleration rate (default is 1).

<anticipateInterpolator>

The change starts backward then flings forward.

attributes:

`android:tension`

Float. The amount of tension to apply (default is 2).

<anticipateOvershootInterpolator>

The change starts backward, flings forward and overshoots the target value, then settles at the final value.

attributes:

`android:tension`

Float. The amount of tension to apply (default is 2).

`android:extraTension`

Float. The amount by which to multiply the tension (default is 1.5).

<bounceInterpolator>

The change bounces at the end.

No attributes

<cycleInterpolator>

Repeats the animation for a specified number of cycles. The rate of change follows a sinusoidal pattern.

attributes:

`android:cycles`

Integer. The number of cycles (default is 1).

<decelerateInterpolator>

The rate of change starts out quickly, then decelerates.

attributes:

`android:factor`

Float. The deceleration rate (default is 1).

<linearInterpolator>

The rate of change is constant.

No attributes.

<overshootInterpolator>

The change flings forward and overshoots the last value, then comes back.

attributes:

```
    android:tension
```

Float. The amount of tension to apply (default is 2).

EXAMPLE:

XML file saved at `res/anim/my_overshoot_interpolator.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<overshootInterpolator xmlns:android="http://schemas.android.com/apk/res/android"
    android:tension="7.0"
/>
```

This animation XML will apply the interpolator:

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@anim/my_overshoot_interpolator"
    android:fromXScale="1.0"
    android:toXScale="3.0"
    android:fromYScale="1.0"
    android:toYScale="3.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="700" />
```

Frame animation

An animation defined in XML that shows a sequence of images in order (like a film).

FILE LOCATION:

```
res/drawable/filename.xml
```

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to an [AnimationDrawable](#).

RESOURCE REFERENCE:

In Java: `R.drawable.filename`

In XML: `@[package:]drawable.filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot=["true" | "false"] >
    <item
        android:drawable="@[package:]drawable/drawable_resource_name"
        android:duration="integer" />
</animation-list>
```

ELEMENTS:

```
<animation-list>
```

Required. This must be the root element. Contains one or more `<item>` elements.

attributes:

android:oneshot

Boolean. "true" if you want to perform the animation once; "false" to loop the animation.

<item>

A single frame of animation. Must be a child of a **<animation-list>** element.

attributes:

android:drawable

Drawable resource. The drawable to use for this frame.

android:duration

Integer. The duration to show this frame, in milliseconds.

EXAMPLE:

XML file saved at **res/anim/rocket.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

This application code will set the animation as the background for a View, then play the animation:

```
ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
rocketImage.setBackgroundResource(R.drawable.rocket_thrust);

rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
rocketAnimation.start();
```

SEE ALSO:

- [2D Graphics: Frame Animation](#)



Color State List Resource

See also

➤ [Color \(simple value\)](#)

A `ColorStateList` is an object you can define in XML that you can apply as a color, but will actually change colors, depending on the state of the `View` object to which it is applied. For example, a `Button` widget can exist in one of several different states (pressed, focused, or neither) and, using a color state list, you can provide a different color during each state.

You can describe the state list in an XML file. Each color is defined in an `<item>` element inside a single `<selector>` element. Each `<item>` uses various attributes to describe the state in which it should be used.

During each state change, the state list is traversed top to bottom and the first item that matches the current state will be used—the selection is *not* based on the "best match," but simply the first item that meets the minimum criteria of the state.

Note: If you want to provide a static color resource, use a simple `Color` value.

FILE LOCATION:

`res/color/filename.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a `ColorStateList`.

RESOURCE REFERENCE:

In Java: `R.color.filename`

In XML: `@[package:]color/filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:color="hex_color"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

ELEMENTS:

`<selector>`

Required. This must be the root element. Contains one or more `<item>` elements.

attributes:

`xmlns:android`

String. Required. Defines the XML namespace, which must be "<http://schemas.android.com/apk/res/android>".

`<item>`

Defines a color to use during certain states, as described by its attributes. Must be a child of a `<selector>` element.

attributes:

`android:color`

Hexadesimal color. Required. The color is specified with an RGB value and optional alpha channel.

The value always begins with a pound (#) character and then followed by the Alpha-Red-Green-Blue information in one of the following formats:

- `#RGB`
- `#ARGB`
- `#RRGGBB`
- `#AARRGGBB`

`android:state_pressed`

Boolean. "true" if this item should be used when the object is pressed (such as when a button is touched/clicked); "false" if this item should be used in the default, non-pressed state.

`android:state_focused`

Boolean. "true" if this item should be used when the object is focused (such as when a button is highlighted using the trackball/d-pad); "false" if this item should be used in the default, non-focused state.

`android:state_selected`

Boolean. "true" if this item should be used when the object is selected (such as when a tab is opened); "false" if this item should be used when the object is not selected.

`android:state_checkable`

Boolean. "true" if this item should be used when the object is checkable; "false" if this item should be used when the object is not checkable. (Only useful if the object can transition between a checkable and non-checkable widget.)

`android:state_checked`

Boolean. "true" if this item should be used when the object is checked; "false" if it should be used when the object is unchecked.

`android:state_enabled`

Boolean. "true" if this item should be used when the object is enabled (capable of receiving touch/click events); "false" if it should be used when the object is disabled.

`android:state_window_focused`

Boolean. "true" if this item should be used when the application window has focus (the application is in the foreground); "false" if this item should be used when the application window does not have focus (for example, if the notification shade is pulled down or a dialog appears).

Note: Remember that the first item in the state list that matches the current state of the object will be applied. So if the first item in the list contains none of the state attributes above, then it will be applied every time, which is why your default value should always be last, as demonstrated in the following example.

EXAMPLE:

XML file saved at `res/color/button_text.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:color="#ffff0000"/> <!-- pressed -->
    <item android:state_focused="true"
          android:color="#ff0000ff"/> <!-- focused -->
    <item android:color="#ff000000"/> <!-- default -->
</selector>
```

This layout XML will apply the color list to a View:

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:textColor="@color/button_text" />
```

SEE ALSO:

- [Color \(simple value\)](#)
- [ColorStateList](#)
- [State List Drawable](#)



可绘制对象资源

另请参阅

- [2D 图形](#)
- [Vector Asset Studio](#)

可绘制对象资源是一般概念，是指可在屏幕上绘制的图形，以及可以使用 `getDrawable(int)` 等 API 检索或者应用到具有 `android:drawable` 和 `android:icon` 等属性的其他 XML 资源的图形。共有多种不同类型的可绘制对象：

位图文件

位图图形文件（.png、.jpg 或 .gif）。创建 [BitmapDrawable](#)。

九宫格文件

具有可拉伸区域的 PNG 文件，允许根据内容调整图像大小（.9.png）。创建 [NinePatchDrawable](#)。

图层列表

管理其他可绘制对象阵列的可绘制对象。它们按阵列顺序绘制，因此索引最大的元素绘制在顶部。创建 [LayerDrawable](#)。

状态列表

此 XML 文件为不同状态引用不同位图图形（例如，按下按钮时使用不同的图像）。创建 [StateListDrawable](#)。

级别列表

此 XML 文件用于定义管理大量备选可绘制对象的可绘制对象，每个可绘制对象都分配有最大的备选数量。创建 [LevelListDrawable](#)。

转换可绘制对象

此 XML 文件用于定义可在两种可绘制对象资源之间交错淡出的可绘制对象。创建 [TransitionDrawable](#)。

插入可绘制对象

此 XML 文件用于定义以指定距离插入其他可绘制对象的可绘制对象。当视图需要小于视图实际边界的背景可绘制对象时，此类可绘制对象很有用。

裁剪可绘制对象

此 XML 文件用于定义对其他可绘制对象进行裁剪（根据其当前级别值）的可绘制对象。创建 [ClipDrawable](#)。

缩放可绘制对象

此 XML 文件用于定义更改其他可绘制对象大小（根据其当前级别值）的可绘制对象。创建 [ScaleDrawable](#)。

形状可绘制对象

此 XML 文件用于定义几何形状（包括颜色和渐变）。创建 [ShapeDrawable](#)。

另请参阅[动画资源](#)文档，了解如何创建 [AnimationDrawable](#)。

注：颜色资源也可用作 XML 中的可绘制对象。例如，在创建状态列表可绘制对象时，可以引用 `android:drawable` 属性的颜色资源 (`android:drawable="@color/green"`)。

位图

位图图像。Android 支持以下三种格式的位图文件：`.png`（首选）、`.jpg`（可接受）和 `.gif`（不建议）。

您可以使用文件名作为资源 ID 直接引用位图文件，也可以在 XML 中创建别名资源 ID。

注：在构建过程中，可通过 `aapt` 工具自动优化位图文件，对图像进行无损压缩。例如，不需要超过 256 色的真彩色 PNG 可通过调色板转换为 8 位 PNG。这样产生的图像质量相同，但所需内存更少。因此请注意，此目录中的图像二进制文件在构建时可能会发生变化。如果您计划将图像解读为比特流以将其转换为位图，请改为将图像放在 `res/raw/` 文件夹中，在那里它们不会进行优化。

位图文件

位图文件是 `.png`、`.jpg` 或 `.gif` 文件。当您将这些文件保存到 `res/drawable/` 目录中时，Android 将为它们创建 `Drawable` 资源。

文件位置：

`res/drawable/filename.png` (`.png`、`.jpg` 或 `.gif`)

文件名用作资源 ID。

编译的资源数据类型：

指向 `BitmapDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

示例：

当图像保存为 `res/drawable/myimage.png` 后，此布局 XML 会将图像应用到视图：

```
<ImageView  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:src="@drawable/myimage" />
```

以下应用代码将图像作为 `Drawable` 检索：

```
Resources res = getResources();  
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

另请参阅：

- [2D 图形](#)
- [BitmapDrawable](#)

XML 位图

XML 位图是在 XML 中定义的资源，指向位图文件。实际上是原始位图文件的别名。XML 可以指定位图的其他属性，例如抖动和层叠。

注：您可以将 `<bitmap>` 元素用作 `<item>` 元素的子项。例如，在创建状态列表或图层列表时，可以将 `android:drawable` 属性从 `<item>` 元素中排除，并在其中嵌套用于定义可绘制项的 `<bitmap>`。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 `BitmapDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:antialias=["true" | "false"]
    android:dither=["true" | "false"]
    android:filter=["true" | "false"]
    android:gravity=["top" | "bottom" | "left" | "right" | "center_vertical" |
                    "fill_vertical" | "center_horizontal" | "fill_horizontal" |
                    "center" | "fill" | "clip_vertical" | "clip_horizontal"]
    android:mipMap=["true" | "false"]
    android:tileMode=["disabled" | "clamp" | "repeat" | "mirror"] />
```

元素：

`<bitmap>`

定义位图来源及其属性。

属性：

`xmlns:android`

字符串。定义 XML 命名空间，其必须是 `"http://schemas.android.com/apk/res/android"`。这仅当 `<bitmap>` 是根元素时才需要，当 `<bitmap>` 嵌套在 `<item>` 内时不需要。

`android:src`

可绘制对象资源。**必备**。引用可绘制对象资源。

`android:antialias`

布尔值。启用或停用抗锯齿。

`android:dither`

布尔值。当位图的像素配置与屏幕不同时（例如：ARGB 8888 位图和 RGB 565 屏幕），启用或停用位图抖动。

`android:filter`

布尔值。启用或停用位图过滤。当位图收缩或拉伸以使其外观平滑时使用过滤。

`android:gravity`

关键字。定义位图的重力。重力指示当位图小于容器时，可绘制对象在其容器中放置的位置。

必须是以下一个或多个（用 ‘|’ 分隔）常量值：

值	说明
<code>top</code>	将对象放在其容器顶部，不改变其大小。
<code>bottom</code>	将对象放在其容器底部，不改变其大小。
<code>left</code>	将对象放在其容器左边缘，不改变其大小。
<code>right</code>	将对象放在其容器右边缘，不改变其大小。
<code>center_vertical</code>	将对象放在其容器的垂直中心，不改变其大小。
<code>fill_vertical</code>	按需要扩展对象的垂直大小，使其完全适应其容器。
<code>center_horizontal</code>	将对象放在其容器的水平中心，不改变其大小。
<code>fill_horizontal</code>	按需要扩展对象的水平大小，使其完全适应其容器。
<code>center</code>	将对象放在其容器的水平和垂直轴中心，不改变其大小。
<code>fill</code>	按需要扩展对象的垂直大小，使其完全适应其容器。这是默认值。
<code>clip_vertical</code>	可设置为让子元素的上边缘和/或下边缘裁剪至其容器边界的附加选项。裁剪基于垂直重力：顶部重力裁剪上边缘，底部重力裁剪下边缘，任一重力不会同时裁剪两边。
<code>clip_horizontal</code>	可设置为让子元素的左边和/或右边裁剪至其容器边界的附加选项。裁剪基于水平重力：左边重力裁剪右边缘，右边重力裁剪左边缘，任一重力不会同时裁剪两边。

android:mipMap

布尔值。启用或停用 mipmap 提示。如需了解详细信息，请参阅 [setHasMipMap\(\)](#)。默认值为 false。

android:tileMode

关键字。定义平铺模式。当平铺模式启用时，位图会重复。重力在平铺模式启用时将被忽略。

必须是以下常量值之一：

值	说明
<code>disabled</code>	不平铺位图。这是默认值。
<code>clamp</code>	当着色器绘制范围超出其原边界时复制边缘颜色
<code>repeat</code>	水平和垂直重复着色器的图像。
<code>mirror</code>	水平和垂直重复着色器的图像，交替镜像图像以使相邻图像始终相接。

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/icon"
    android:tileMode="repeat" />
```

另请参阅：

- [BitmapDrawable](#)
- [创建别名资源](#)

九宫格

[NinePatch](#) 是一种 PNG 图像，在其中可定义当视图中的内容超出正常图像边界时 Android 缩放的可拉伸区域。此类图像通常指定为至少有一个尺寸设置为 "wrap_content" 的视图的背景，而且当视图扩展以适应内容时，九宫格图像也会扩展以匹配视图的大小。Android 的标准 [Button](#) 小部件使用的背景就是典型的九宫格图像，其必须拉伸以适应按钮内的文本（或图像）。

与普通位图一样，您可以直接引用九宫格文件，也可以从 XML 定义的资源引用。

如需有关如何创建包含可拉伸区域的九宫格文件的完整论述，请参阅 [2D 图形](#) 文件。

九宫格文件

文件位置：

`res/drawable/filename.9.png`

文件名用作资源 ID。

编译的资源数据类型：

指向 [NinePatchDrawable](#) 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

示例：

当图像保存为 `res/drawable/myninepatch.9.png` 后，此布局 XML 会将九宫格应用到视图：

```
<Button  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:background="@drawable/myninepatch" />
```

另请参阅：

- [2D 图形](#)
- [NinePatchDrawable](#)

XML 九宫格

XML 九宫格是在 XML 中定义的资源，指向九宫格文件。XML 可以为图像指定抖动。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 [NinePatchDrawable](#) 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>  
<nine-patch  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:src="@[package:]drawable/drawable_resource"  
    android:dither=["true" | "false"] />
```

元素：

```
<nine-patch>
```

定义九宫格来源及其属性。

属性：

```
xmlns:android
```

字符串。必备。定义 XML 命名空间，其必须是 "http://schemas.android.com/apk/res/android"。

```
android:src
```

可绘制对象资源。必备。引用九宫格文件。

```
android:dither
```

布尔值。当位图的像素配置与屏幕不同时（例如：ARGB 8888 位图和 RGB 565 屏幕），启用或停用位图抖动。

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<nine-patch xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/myninepatch"
    android:dither="false" />
```

图层列表

[LayerDrawable](#) 是管理其他可绘制对象阵列的可绘制对象。列表中的每个可绘制对象按照列表的顺序绘制，列表中的最后一个可绘制对象绘于顶部。

每个可绘制对象由单一 `<layer-list>` 元素内的 `<item>` 元素表示。

文件位置：

```
res/drawable/filename.xml
```

文件名用作资源 ID。

编译的资源数据类型：

指向 [LayerDrawable](#) 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@[+][package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</layer-list>
```

元素：

```
<layer-list>
```

必备。这必须是根元素。包含一个或多个 `<item>` 元素。

属性：

```
xmlns:android
```

字符串。**必备。**定义 XML 命名空间，其必须是 "<http://schemas.android.com/apk/res/android>"。

```
<item>
```

定义要放在图层可绘制对象中由其属性定义的位置的可绘制对象。必须是 `<selector>` 元素的子项。接受子 `<bitmap>` 元素。

属性：

```
android:drawable
```

可绘制对象资源。**必备。**引用可绘制对象资源。

```
android:id
```

资源 ID。此可绘制对象的唯一资源 ID。要为此项新建资源 ID，请使用以下形式：`@+id/name`。加号表示应创建为新 ID。可以使用此 ID 检索和修改具有 `View.findViewById()` 或 `Activity.findViewById()` 的可绘制对象。

```
android:top
```

整型。顶部偏移（像素）。

```
android:right
```

整型。右边偏移（像素）。

```
android:bottom
```

整型。底部偏移（像素）。

```
android:left
```

整型。左边偏移（像素）。

默认情况下，所有可绘制项都会缩放以适应包含视图的大小。因此，将图像放在图层列表中的不同位置可能会增大视图的大小，并且有些图像会相应地缩放。为避免缩放列表中的项目，请在 `<item>` 元素内使用 `<bitmap>` 元素指定可绘制对象，并且对某些不缩放的项目（例如 "center"）定义重力。例如，以下 `<item>` 定义缩放以适应其容器视图的项目：

```
<item android:drawable="@drawable/image" />
```

为避免缩放，以下示例使用重力居中的 `<bitmap>` 元素：

```
<item>
    <bitmap android:src="@drawable/image"
            android:gravity="center" />
</item>
```

示例：

XML 文件保存在 `res/drawable/layers.xml` 中：

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <bitmap android:src="@drawable/android_red"
            android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
        <bitmap android:src="@drawable/android_green"
            android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
        <bitmap android:src="@drawable/android_blue"
            android:gravity="center" />
    </item>
</layer-list>
```

请注意，此示例使用嵌套的 `<bitmap>` 元素为每个具有“中心”重力的项目定义可绘制对象资源。这可确保没有图像会为了适应容器的大小而缩放，因为偏移图像会造成大小调整。

此布局 XML 会将可绘制对象应用到视图：

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/layers" />
```

结果导致一堆不断偏移的图像：



另请参阅：

- [LayerDrawable](#)

状态列表

`StateListDrawable` 是在 XML 中定义的可绘制对象，它根据对象的状态，使用多个不同的图像来表示同一个图形。例如，`Button` 小部件可以是多种不同状态（按下、聚焦或这两种状态都不是）中的其中一种，而且可以利用状态列表可绘制对象为每种状态提供不同的背景图片。

您可以在 XML 文件中描述状态列表。每个图形由单一 `<selector>` 元素内的 `<item>` 元素表示。每个 `<item>` 均使用各种属性来描述应用作可绘制对象的图形的状态。

在每个状态变更期间，将从上到下遍历状态列表，并使用第一个与当前状态匹配的项目 —— 此选择并非基于“最佳匹配”，而是选择符合状态最低条件的第一个项目。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 `StateListDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android"
    android:constantSize=["true" | "false"]
    android:dither=["true" | "false"]
    android:variablePadding=["true" | "false"] >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_hovered=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_activated=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

元素：

```
<selector>
```

必备。这必须是根元素。包含一个或多个 `<item>` 元素。

属性：

```
xmlns:android
```

字符串。必备。定义 XML 命名空间，其必须是 "<http://schemas.android.com/apk/res/android>"。

```
    android:constantSize
```

布尔值。如果可绘制对象报告的内部大小在状态变更时保持不变，则值为“true”（大小是所有状态的最大值）；如果大小根据当前状态而变化，则值为“false”。默认值为 false。

```
    android:dither
```

布尔值。值为“true”时，将在位图的像素配置与屏幕不同时（例如：ARGB 8888 位图和 RGB 565 屏幕）启用位图的抖动；值为“false”时则停用抖动。默认值为 true。

```
    android:variablePadding
```

布尔值。如果可绘制对象的内边距应根据选择的当前状态而变化，则值为“true”；如果内边距应保持不变（基于所有状态的最大内边距），则值为“false”。启用此功能要求您在状态变更时处理执行布局，这通常不受支持。默认值为 false。

```
<item>
```

定义要在某些状态期间使用的可绘制对象，如其属性所述。必须是 `<selector>` 元素的子项。

属性：

```
    android:drawable
```

可绘制对象资源。必备。引用可绘制对象资源。

```
    android:state_pressed
```

布尔值。如果在按下对象（例如触摸/点按某按钮）时应使用此项目，则值为“true”；如果在默认的未按下状态时应使用此项目，则值为“false”。

```
    android:state_focused
```

布尔值。如果在对象具有输入焦点（例如当用户选择文本输入时）时应使用此项目，则值为“true”；如果在默认的非焦点状态时应使用此项目，则值为“false”。

android:state_hovered

布尔值。如果当光标悬停在对象上时应使用此项目，则值为“true”；如果在默认的非悬停状态时应使用此项目，则值为“false”。通常，这个可绘制对象可能与用于“聚焦”状态的可绘制对象相同。

此项为 API 级别 14 新引入的配置。

android:state_selected

布尔值。如果在使用定向控件浏览（例如使用方向键浏览列表）的情况下对象为当前用户选择时应使用此项目，则值为“true”；如果在未选择对象时应使用此项目，则值为“false”。

当焦点 (android:state_focused) 不充分（例如，列表视图有焦点但使用方向键选择其中的项目）时，使用所选状态。

android:state_checkable

布尔值。如果当对象可选中时应使用此项目，则值为“true”；如果当对象不可选中时应使用此项目，则值为“false”。（仅当对象可在可选中与不可选中小部件之间转换时才有用。）

android:state_checked

布尔值。如果在对象已选中时应使用此项目，则值为“true”；如果在对象未选中时应使用此项目，则值为“false”。

android:state_enabled

布尔值。如果在对象启用（能够接收触摸/点击事件）时应使用此项目，则值为“true”；如果在对象停用时应使用此项目，则值为“false”。

android:state_activated

布尔值。如果在对象激活作为持续选择（例如，在持续导航视图中“突出显示”之前选中的列表项）时应使用此项目，则值为“true”；如果在对象未激活时应使用此项目，则值为“false”。

此项为 API 级别 11 新引入的配置。

android:state_window_focused

布尔值。如果当应用窗口有焦点（应用在前台）时应使用此项目，则值为“true”；如果当应用窗口没有焦点（例如，通知栏下拉或对话框出现）时应使用此项目，则值为“false”。

注：请记住，Android 将应用状态列表中第一个与对象当前状态匹配的项目。因此，如果列表中的第一个项目不含上述任何状态属性，则每次都会应用它，这就是默认值应始终放在最后的原因（如以下示例所示）。

示例：

XML 文件保存在 `res/drawable/button.xml` 中：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

此布局 XML 将状态列表可绘制对象应用到按钮：

```
<Button  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:background="@drawable/button" />
```

另请参阅：

- [StateListDrawable](#)

级别列表

管理大量备选可绘制对象的可绘制对象，每个可绘制对象都分配有最大的备选数量。使用 [setLevel\(\)](#) 设置可绘制对象的级别值会加载级别列表中 `android:maxLevel` 值大于或等于传递到方法的值的可绘制对象资源。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 [LevelListDrawable](#) 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>  
<level-list  
    xmlns:android="http://schemas.android.com/apk/res/android" >  
    <item  
        android:drawable="@drawable/drawable_resource"  
        android:maxLevel="integer"  
        android:minLevel="integer" />  
</level-list>
```

元素：

`<level-list>`

这必须是根元素。包含一个或多个 `<item>` 元素。

属性：

`xmlns:android`

字符串。**必备**。定义 XML 命名空间，其必须是 `"http://schemas.android.com/apk/res/android"`。

`<item>`

定义要在某特定级别使用的可绘制对象。

属性：

`android:drawable`

可绘制对象资源。**必备**。引用要插入的可绘制对象资源。

`android:maxLevel`

整型。此项目允许的最高级别。

`android:minLevel`

整型。此项目允许的最低级别。

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<level-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/status_off"
        android:maxLevel="0" />
    <item
        android:drawable="@drawable/status_on"
        android:maxLevel="1" />
</level-list>
```

在此项目应用到 `View` 后，可通过 `setLevel()` 或 `setImageLevel()` 更改级别。

另请参阅：

- [LevelListDrawable](#)

转换可绘制对象

`TransitionDrawable` 是可在两种可绘制对象资源之间交错淡出的可绘制对象。

每个可绘制对象由单一 `<transition>` 元素内的 `<item>` 元素表示。不支持超过两个项目。要向前转换，请调用 `startTransition()`。要向后转换，则调用 `reverseTransition()`。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 `TransitionDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<transition
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@[+][package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</transition>
```

元素：

```
<transition>
```

必备。这必须是根元素。包含一个或多个 `<item>` 元素。

属性：

```
xmlns:android
```

字符串。**必备。**定义 XML 命名空间，其必须是 "<http://schemas.android.com/apk/res/android>"。

```
<item>
```

定义要用作可绘制对象转换一部分的可绘制对象。必须是 `<transition>` 元素的子项。接受子 `<bitmap>` 元素。

属性：

```
android:drawable
```

可绘制对象资源。**必备。**引用可绘制对象资源。

```
android:id
```

资源 ID。此可绘制对象的唯一资源 ID。要为此项新建资源 ID，请使用以下形式：`@+id/name`。加号表示应创建为新 ID。可以使用此 ID 检索和修改具有 `View.findViewById()` 或 `Activity.findViewById()` 的可绘制对象。

```
android:top
```

整型。顶部偏移（像素）。

```
android:right
```

整型。右边偏移（像素）。

```
android:bottom
```

整型。底部偏移（像素）。

```
android:left
```

整型。左边偏移（像素）。

示例：

XML 文件保存在 `res/drawable/transition.xml` 中：

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

此布局 XML 会将可绘制对象应用到视图：

```
<ImageButton
    android:id="@+id/button"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/transition" />
```

以下代码从第一个项目到第二个项目执行 500ms 的转换：

```
ImageButton button = (ImageButton) findViewById(R.id.button);
TransitionDrawable drawable = (TransitionDrawable) button.getDrawable();
drawable.startTransition(500);
```

另请参阅：

- [TransitionDrawable](#)

插入可绘制对象

在 XML 文件中定义的以指定距离插入其他可绘制对象的可绘制对象。当视图需要小于视图实际边界的背景时，此类可绘制对象很有用。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 `InsetDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<inset
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:insetTop="dimension"
    android:insetRight="dimension"
    android:insetBottom="dimension"
    android:insetLeft="dimension" />
```

元素：

`<inset>`

定义插入可绘制对象。这必须是根元素。

属性：

`xmlns:android`

字符串。**必备**。定义 XML 命名空间，其必须是 `"http://schemas.android.com/apk/res/android"`。

`android:drawable`

可绘制对象资源。**必备**。引用要插入的可绘制对象资源。

`android:insetTop`

尺寸。顶部插入，表示为尺寸值或尺寸资源

`android:insetRight`

尺寸。右边插入，表示为尺寸值或尺寸资源

```
    android:insetBottom
```

尺寸。底部插入，表示为尺寸值或[尺寸资源](#)

```
    android:insetLeft
```

尺寸。左边插入，表示为尺寸值或[尺寸资源](#)

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<inset xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/background"
    android:insetTop="10dp"
    android:insetLeft="10dp" />
```

另请参阅：

- [InsetDrawable](#)

裁剪可绘制对象

在 XML 文件中定义的对其他可绘制对象进行裁剪（根据其当前级别）的可绘制对象。您可以根据级别以及用于控制其在整个容器中位置的重力，来控制子可绘制对象的裁剪宽度和高度。通常用于实现进度栏之类的项目。

文件位置：

```
res/drawable/filename.xml
```

文件名用作资源 ID。

编译的资源数据类型：

指向 [ClipDrawable](#) 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<clip
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:clipOrientation=["horizontal" | "vertical"]
    android:gravity=[["top" | "bottom" | "left" | "right" | "center_vertical" |
                    "fill_vertical" | "center_horizontal" | "fill_horizontal" |
                    "center" | "fill" | "clip_vertical" | "clip_horizontal"] />
```

元素：

```
<clip>
```

定义裁剪可绘制对象。这必须是根元素。

属性：

```
    xmlns:android
```

字符串。**必备**。定义 XML 命名空间，其必须是 "http://schemas.android.com/apk/res/android"。

android:drawable

可绘制对象资源。**必备**。引用要裁剪的可绘制对象资源。

android:clipOrientation

关键字。裁剪方向。

必须是以下常量值之一：

值	说明
horizontal	水平裁剪可绘制对象。
vertical	垂直裁剪可绘制对象。

android:gravity

关键字。指定可绘制对象中要裁剪的位置。

必须是以下一个或多个（用 " " 分隔）常量值：

值	说明
top	将对象放在其容器顶部，不改变其大小。当 <code>clipOrientation</code> 是 "vertical" 时，在可绘制对象的底部裁剪。
bottom	将对象放在其容器底部，不改变其大小。当 <code>clipOrientation</code> 是 "vertical" 时，在可绘制对象的顶部裁剪。
left	将对象放在其容器左边缘，不改变其大小。这是默认值。当 <code>clipOrientation</code> 是 "horizontal" 时，在可绘制对象的右边裁剪。这是默认值。
right	将对象放在其容器右边缘，不改变其大小。当 <code>clipOrientation</code> 是 "horizontal" 时，在可绘制对象的左边裁剪。
center_vertical	将对象放在其容器的垂直中心，不改变其大小。裁剪行为与重力为 "center" 时相同。
fill_vertical	按需要扩展对象的垂直大小，使其完全适应其容器。当 <code>clipOrientation</code> 是 "vertical" 时，不会进行裁剪，因为可绘制对象会填充垂直空间（除非可绘制对象级别为 0，此时它不可见）。
center_horizontal	将对象放在其容器的水平中心，不改变其大小。裁剪行为与重力为 "center" 时相同。
fill_horizontal	按需要扩展对象的水平大小，使其完全适应其容器。当 <code>clipOrientation</code> 是 "horizontal" 时，不会进行裁剪，因为可绘制对象会填充水平空间（除非可绘制对象级别为 0，此时它不可见）。
center	将对象放在其容器的水平和垂直轴中心，不改变其大小。当 <code>clipOrientation</code> 是 "horizontal" 时，在左边和右边裁剪。当 <code>clipOrientation</code> 是 "vertical" 时，在顶部和底部裁剪。
fill	按需要扩展对象的垂直大小，使其完全适应其容器。不会进行裁剪，因为可绘制对象会填充水平和垂直空间（除非可绘制对象级别为 0，此时它不可见）。
clip_vertical	可设置为让子元素的上边缘和/或下边缘裁剪至其容器边界的附加选项。裁剪基于垂直重力：顶部重力裁剪上边缘，底部重力裁剪下边缘，任一重力不会同时裁剪两边。
clip_horizontal	可设置为让子元素的左边和/或右边裁剪至其容器边界的附加选项。裁剪基于水平重力：左边重力裁剪右边缘，右边重力裁剪左边缘，任一重力不会同时裁剪两边。

示例：

XML 文件保存在 `res/drawable/clip.xml` 中：

```
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/android"
    android:clipOrientation="horizontal"
    android:gravity="left" />
```

以下布局 XML 会将裁剪可绘制对象应用到视图：

```
<ImageView
    android:id="@+id/image"
    android:background="@drawable/clip"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

以下代码用于获取可绘制对象，并增加裁剪量以便逐渐显示图像：

```
ImageView imageview = (ImageView) findViewById(R.id.image);
ClipDrawable drawable = (ClipDrawable) imageview.getDrawable();
drawable.setLevel(drawable.getLevel() + 1000);
```

增大级别可减少裁剪量并慢慢显示图像。此处的级别为 7000：



注：默认级别为 0，即完全裁剪，使图像不可见。当级别为 10,000 时，图像不会裁剪，而是完全可见。

另请参阅：

- [ClipDrawable](#)

缩放可绘制对象

在 XML 文件中定义的更改其他可绘制对象大小（根据其当前级别）的可绘制对象。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 `ScaleDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```

<?xml version="1.0" encoding="utf-8"?>
<scale
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:scaleGravity=["top" | "bottom" | "left" | "right" | "center_vertical" |
        "fill_vertical" | "center_horizontal" | "fill_horizontal" |
        "center" | "fill" | "clip_vertical" | "clip_horizontal"]
    android:scaleHeight="percentage"
    android:scaleWidth="percentage" />

```

元素：

<scale>

定义缩放可绘制对象。这必须是根元素。

属性：

`xmlns:android`

字符串。**必备**。定义 XML 命名空间，其必须是 "<http://schemas.android.com/apk/res/android>"。

`android:drawable`

可绘制对象资源。**必备**。引用可绘制对象资源。

`android:scaleGravity`

关键字。指定缩放后的重力位置。

必须是以下一个或多个（用 ' | ' 分隔）常量值：

值	说明
<code>top</code>	将对象放在其容器顶部，不改变其大小。
<code>bottom</code>	将对象放在其容器底部，不改变其大小。
<code>left</code>	将对象放在其容器左边缘，不改变其大小。这是默认值。
<code>right</code>	将对象放在其容器右边缘，不改变其大小。
<code>center_vertical</code>	将对象放在其容器的垂直中心，不改变其大小。
<code>fill_vertical</code>	按需要扩展对象的垂直大小，使其完全适应其容器。
<code>center_horizontal</code>	将对象放在其容器的水平中心，不改变其大小。
<code>fill_horizontal</code>	按需要扩展对象的水平大小，使其完全适应其容器。
<code>center</code>	将对象放在其容器的水平和垂直轴中心，不改变其大小。
<code>fill</code>	按需要扩展对象的垂直大小，使其完全适应其容器。
<code>clip_vertical</code>	可设置为让子元素的上边缘和/或下边缘裁剪至其容器边界的附加选项。裁剪基于垂直重力：顶部重力裁剪上边缘，底部重力裁剪下边缘，任一重力不会同时裁剪两边。
<code>clip_horizontal</code>	可设置为让子元素的左边和/或右边裁剪至其容器边界的附加选项。裁剪基于水平重力：左边重力裁剪右边缘，右边重力裁剪左边缘，任一重力不会同时裁剪两边。

`android:scaleHeight`

百分比。缩放高度，表示为可绘制对象边界的百分比。值的格式为 XX%。例如：100%、12.5% 等。

`android:scaleWidth`

百分比。缩放宽度，表示为可绘制对象边界的百分比。值的格式为 XX%。例如：100%、12.5% 等。

示例：

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/logo"
    android:scaleGravity="center_vertical|center_horizontal"
    android:scaleHeight="80%"
    android:scaleWidth="80%" />
```

另请参阅：

- [ScaleDrawable](#)

形状可绘制对象

这是在 XML 中定义的一般形状。

文件位置：

`res/drawable/filename.xml`

文件名用作资源 ID。

编译的资源数据类型：

指向 `GradientDrawable` 的资源指针。

资源引用：

在 Java 中：`R.drawable.filename`

在 XML 中：`@[package:]drawable/filename`

语法：

```

<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <gradient
        android:angle="integer"
        android:centerX="float"
        android:centerY="float"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=["linear" | "radial" | "sweep"]
        android:useLevel=[true" | "false"] />
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <size
        android:width="integer"
        android:height="integer" />
    <solid
        android:color="color" />
    <stroke
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
</shape>

```

元素：

<shape>

形状可绘制对象。这必须是根元素。

属性：

xmlns:android

字符串。**必备**。定义 XML 命名空间，其必须是 "<http://schemas.android.com/apk/res/android>"。

android:shape

关键字。定义形状的类型。有效值为：

值	描述
"rectangle"	填充包含视图的矩形。这是默认形状。
"oval"	适应包含视图尺寸的椭圆形状。
"line"	跨越包含视图宽度的水平线。此形状需要 < stroke > 元素定义线宽。
"ring"	环形。

仅当 `android:shape="ring"` 如下时才使用以下属性：

`android:innerRadius`

尺寸。环内部（中间的孔）的半径，以尺寸值或尺寸资源表示。

`android:innerRadiusRatio`

浮点型。环内部的半径，以环宽度的比率表示。例如，如果 `android:innerRadiusRatio="5"`，则内半径等于环宽度除以 5。此值被 `android:innerRadius` 覆盖。默认值为 9。

`android:thickness`

尺寸。环的厚度，以尺寸值或尺寸资源表示。

`android:thicknessRatio`

浮点型。环的厚度，表示为环宽度的比率。例如，如果 `android:thicknessRatio="2"`，则厚度等于环宽度除以 2。此值被 `android:innerRadius` 覆盖。默认值为 3。

`android:useLevel`

布尔值。如果这用作 `LevelListDrawable`，则此值为“true”。这通常应为“false”，否则形状不会显示。

`<corners>`

为形状产生圆角。仅当形状为矩形时适用。

属性：

`android:radius`

尺寸。所有角的半径，以尺寸值或尺寸资源表示。对于每个角，这会被以下属性覆盖。

`android:topLeftRadius`

尺寸。左上角的半径，以尺寸值或尺寸资源表示。

`android:topRightRadius`

尺寸。右上角的半径，以尺寸值或尺寸资源表示。

`android:bottomLeftRadius`

尺寸。左下角的半径，以尺寸值或尺寸资源表示。

`android:bottomRightRadius`

尺寸。右下角的半径，以尺寸值或尺寸资源表示。

注：（最初）必须为每个角提供大于 1 的角半径，否则无法产生圆角。如果希望特定角不要倒圆角，解决方法是使用 `android:radius` 设置大于 1 的默认角半径，然后使用实际所需的值替换每个角，为不希望倒圆角的角提供零（“0dp”）。

`<gradient>`

指定形状的渐变颜色。

属性：

`android:angle`

整型。渐变的角度（度）。0 为从左到右，90 为从上到下。必须是 45 的倍数。默认值为 0。

`android:centerX`

浮点型。渐变中心的相对 X 轴位置 (0 - 1.0)。

`android:centerY`

浮点型。渐变中心的相对 Y 轴位置 (0 - 1.0)。

android:centerColor

颜色。起始颜色与结束颜色之间的可选颜色，以十六进制值或[颜色资源](#)表示。

android:endColor

颜色。结束颜色，表示为十六进制值或[颜色资源](#)。

android:gradientRadius

浮点型。渐变的半径。仅在 `android:type="radial"` 时适用。

android:startColor

颜色。起始颜色，表示为十六进制值或[颜色资源](#)。

android:type

关键字。要应用的渐变图案的类型。有效值为：

值	说明
"linear"	线性渐变。这是默认值。
"radial"	径向渐变。起始颜色为中心颜色。
"sweep"	流线型渐变。

android:useLevel

布尔值。如果这用作 [LevelListDrawable](#)，则此值为“true”。

<padding>

要应用到包含视图元素的内边距（这会填充视图内容的位置，而非形状）。

属性：

android:left

尺寸。左内边距，表示为尺寸值或[尺寸资源](#)

android:top

尺寸。上内边距，表示为尺寸值或[尺寸资源](#)

android:right

尺寸。右内边距，表示为尺寸值或[尺寸资源](#)

android:bottom

尺寸。下内边距，表示为尺寸值或[尺寸资源](#)

<size>

形状的大小。

属性：

android:height

尺寸。形状的高度，表示为尺寸值或**尺寸资源**

android:width

尺寸。形状的宽度，表示为尺寸值或**尺寸资源**

注：默认情况下，形状按照此处定义的尺寸按比例缩放至容器视图的大小。在 [ImageView](#) 中使用形状时，可通过将 **android:scaleType** 设置为 "center" 来限制缩放。

<solid>

用于填充形状的纯色。

属性：

android:color

颜色。应用于形状的颜色，以十六进制值或**颜色资源**表示。

<stroke>

形状的笔划中线。

属性：

android:width

尺寸。线宽，以尺寸值或**尺寸资源**表示。

android:color

颜色。线的颜色，表示为十六进制值或**颜色资源**。

android:dashGap

尺寸。短划线的间距，以尺寸值或**尺寸资源**表示。仅在设置了 **android:dashWidth** 时有效。

android:dashWidth

尺寸。每个短划线的大小，以尺寸值或**尺寸资源**表示。仅在设置了 **android:dashGap** 时有效。

示例：

XML 文件保存在 `res/drawable/gradient_box.xml` 中：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <corners android:radius="8dp" />
</shape>
```

此布局 XML 会将形状可绘制对象应用到视图：

```
<TextView  
    android:background="@drawable/gradient_box"  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content" />
```

此应用代码将获取形状可绘制对象，并将其应用到视图：

```
Resources res = getResources();  
Drawable shape = res.getDrawable(R.drawable.gradient_box);  
  
TextView tv = (TextView) findViewById(R.id.textview);  
tv.setBackground(shape);
```

另请参阅：

- [ShapeDrawable](#)

Layout Resource

See also

› [Layouts](#)

A layout resource defines the architecture for the UI in an Activity or a component of a UI.

FILE LOCATION:

`res/layout/filename.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a [View](#) (or subclass) resource.

RESOURCE REFERENCE:

In Java: `R.layout.filename`

In XML: `@[package:]layout/filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "match_parent" | "wrap_content"]
    android:layout_width=["dimension" | "match_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "match_parent" | "wrap_content"]
        android:layout_width=["dimension" | "match_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

Note: The root element can be either a [ViewGroup](#), a [View](#), or a [<merge>](#) element, but there must be only one root element and it must contain the `xmlns:android` attribute with the `android` namespace as shown.

ELEMENTS:

`<ViewGroup>`

A container for other [View](#) elements. There are many different kinds of [ViewGroup](#) objects and each one lets you specify the layout of the child elements in different ways. Different kinds of [ViewGroup](#) objects include [LinearLayout](#), [RelativeLayout](#), and [FrameLayout](#).

You should not assume that any derivation of [ViewGroup](#) will accept nested [Views](#). Some [ViewGroups](#) are implementations of the [AdapterView](#) class, which determines its children only from an [Adapter](#).

attributes:

`android:id`

Resource ID. A unique resource name for the element, which you can use to obtain a reference to the [ViewGroup](#) from your application. See more about the [value for android:id](#) below.

`android:layout_height`

Dimension or keyword. **Required**. The height for the group, as a dimension value (or [dimension resource](#)) or a keyword ("match_parent" or "wrap_content"). See the [valid values](#) below.

`android:layout_width`

Dimension or keyword. **Required**. The width for the group, as a dimension value (or [dimension resource](#)) or a keyword ("match_parent" or "wrap_content"). See the [valid values](#) below.

More attributes are supported by the [ViewGroup](#) base class, and many more are supported by each implementation of [ViewGroup](#). For a reference of all available attributes, see the corresponding reference documentation for the [ViewGroup](#) class (for example, the [LinearLayout XML attributes](#)).

`<View>`

An individual UI component, generally referred to as a "widget". Different kinds of [View](#) objects include [TextView](#), [Button](#), and [CheckBox](#).

attributes:

`android:id`

Resource ID. A unique resource name for the element, which you can use to obtain a reference to the [View](#) from your application. See more about the [value for android:id](#) below.

`android:layout_height`

Dimension or keyword. **Required**. The height for the element, as a dimension value (or [dimension resource](#)) or a keyword ("match_parent" or "wrap_content"). See the [valid values](#) below.

`android:layout_width`

Dimension or keyword. **Required**. The width for the element, as a dimension value (or [dimension resource](#)) or a keyword ("match_parent" or "wrap_content"). See the [valid values](#) below.

More attributes are supported by the [View](#) base class, and many more are supported by each implementation of [View](#). Read [Layouts](#) for more information. For a reference of all available attributes, see the corresponding reference documentation (for example, the [TextView XML attributes](#)).

`<requestFocus>`

Any element representing a [View](#) object can include this empty element, which gives its parent initial focus on the screen. You can have only one of these elements per file.

`<include>`

Includes a layout file into this layout.

attributes:

layout

Layout resource. **Required.** Reference to a layout resource.

android:id

Resource ID. Overrides the ID given to the root view in the included layout.

android:layout_height

Dimension or keyword. Overrides the height given to the root view in the included layout. Only effective if `android:layout_width` is also declared.

android:layout_width

Dimension or keyword. Overrides the width given to the root view in the included layout. Only effective if `android:layout_height` is also declared.

You can include any other layout attributes in the `<include>` that are supported by the root element in the included layout and they will override those defined in the root element.

Caution: If you want to override layout attributes using the `<include>` tag, you must override both `android:layout_height` and `android:layout_width` in order for other layout attributes to take effect.

Another way to include a layout is to use `ViewStub`. It is a lightweight View that consumes no layout space until you explicitly inflate it, at which point, it includes a layout file defined by its `android:layout` attribute. For more information about using `ViewStub`, read [Loading Views On Demand](#).

<merge>

An alternative root element that is not drawn in the layout hierarchy. Using this as the root element is useful when you know that this layout will be placed into a layout that already contains the appropriate parent View to contain the children of the `<merge>` element. This is particularly useful when you plan to include this layout in another layout file using `<include>` and this layout doesn't require a different `ViewGroup` container. For more information about merging layouts, read [Re-using Layouts with <include/>](#).

Value for android:id

For the ID value, you should usually use this syntax form: "`@+id/name`". The plus symbol, `+`, indicates that this is a new resource ID and the `aapt` tool will create a new resource integer in the `R.java` class, if it doesn't already exist. For example:

```
<TextView android:id="@+id/nameTextbox"/>
```

The `nameTextbox` name is now a resource ID attached to this element. You can then refer to the `TextView` to which the ID is associated in Java:

```
findViewById(R.id.nameTextbox);
```

This code returns the `TextView` object.

However, if you have already defined an `ID resource` (and it is not already used), then you can apply that ID to a `View` element by excluding the plus symbol in the `android:id` value.

Value for android:layout_height and android:layout_width:

The height and width value can be expressed using any of the `dimension units` supported by Android (px, dp, sp, pt, in, mm) or with the following keywords:

Value	Description

<code>match_parent</code>	Sets the dimension to match that of the parent element. Added in API Level 8 to deprecate <code>fill_parent</code> .
<code>wrap_content</code>	Sets the dimension only to the size required to fit the content of this element.

Custom View elements

You can create your own custom [View](#) and [ViewGroup](#) elements and apply them to your layout the same as a standard layout element. You can also specify the attributes supported in the XML element. To learn more, see the [Custom Components](#) developer guide.

EXAMPLE:

XML file saved at `res/layout/main_activity.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

This application code will load the layout for an [Activity](#), in the `onCreate()` method:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
}
```

SEE ALSO:

- [Layouts](#)
- [View](#)
- [ViewGroup](#)



Menu Resource

See also

➤ [Menus](#)

A menu resource defines an application menu (Options Menu, Context Menu, or submenu) that can be inflated with [MenuInflater](#).

For a guide to using menus, see the [Menus](#) developer guide.

FILE LOCATION:

`res/menu/filename.xml`

The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a [Menu](#) (or subclass) resource.

RESOURCE REFERENCE:

In Java: `R.menu.filename`

In XML: `@[package :]menu.filename`

SYNTAX:

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@[+][package:]id/resource_name"
          android:title="string"
          android:titleCondensed="string"
          android:icon="@[package:]drawable/drawable_resource_name"
          android:onClick="method name"
          android:showAsAction=["ifRoom" | "never" | "withText" | "always" | "collapseActionView"]
          android:actionLayout="@[package:]layout/layout_resource_name"
          android:actionViewClass="class name"
          android:actionProviderClass="class name"
          android:alphabeticShortcut="string"
          android:alphabeticModifiers=["META" | "CTRL" | "ALT" | "SHIFT" | "SYM" | "FUNCTION"]
          android:numericShortcut="string"
          android:numericModifiers=["META" | "CTRL" | "ALT" | "SHIFT" | "SYM" | "FUNCTION"]
          android:checkable=["true" | "false"]
          android:visible=["true" | "false"]
          android:enabled=["true" | "false"]
          android:menuCategory=["container" | "system" | "secondary" | "alternative"]
          android:orderInCategory="integer" />
    <group android:id="@[+][package:]id/resource_name"
           android:checkableBehavior=["none" | "all" | "single"]
           android:visible=["true" | "false"]
           android:enabled=["true" | "false"]
           android:menuCategory=["container" | "system" | "secondary" | "alternative"]
           android:orderInCategory="integer" >
        <item />
    </group>
    <item >
        <menu>
            <item />
        </menu>
    </item>
</menu>

```

ELEMENTS:

<menu>

Required. This must be the root node. Contains <item> and/or <group> elements.

attributes:

xmlns:android

XML namespace. **Required.** Defines the XML namespace, which must be "http://schemas.android.com/apk/res/android".

<item>

A menu item. May contain a <menu> element (for a Sub Menu). Must be a child of a <menu> or <group> element.

attributes:

android:id

Resource ID. A unique resource ID. To create a new resource ID for this item, use the form: "@+id/name". The plus symbol indicates that this should be created as a new ID.

android:title

String resource. The menu title as a string resource or raw string.

android:titleCondensed

String resource. A condensed title as a string resource or a raw string. This title is used for situations in which the normal

title is too long.

`android:icon`

Drawable resource. An image to be used as the menu item icon.

`android:onClick`

Method name. The method to call when this menu item is clicked. The method must be declared in the activity as public and accept a `MenuItem` as its only parameter, which indicates the item clicked. This method takes precedence over the standard callback to `onOptionsItemSelected()`. See the example at the bottom.

Warning: If you obfuscate your code using `ProGuard` (or a similar tool), be sure to exclude the method you specify in this attribute from renaming, because it can break the functionality.

Introduced in API Level 11.

`android:showAsAction`

Keyword. When and how this item should appear as an action item in the app bar. A menu item can appear as an action item only when the activity includes an app bar. Valid values:

Value	Description
<code>ifRoom</code>	Only place this item in the app bar if there is room for it. If there is not room for all the items marked " <code>ifRoom</code> ", the items with the lowest <code>orderInCategory</code> values are displayed as actions, and the remaining items are displayed in the overflow menu.
<code>withText</code>	Also include the title text (defined by <code>android:title</code>) with the action item. You can include this value along with one of the others as a flag set, by separating them with a pipe .
<code>never</code>	Never place this item in the app bar. Instead, list the item in the app bar's overflow menu.
<code>always</code>	Always place this item in the app bar. Avoid using this unless it's critical that the item always appear in the action bar. Setting multiple items to always appear as action items can result in them overlapping with other UI in the app bar.
<code>collapseActionView</code>	The action view associated with this action item (as declared by <code>android:actionLayout</code> or <code>android:actionViewClass</code>) is collapsible. Introduced in API Level 14.

See the [Adding the App Bar](#) training class for more information.

Introduced in API Level 11.

`android:actionLayout`

Layout resource. A layout to use as the action view.

See [Action Views and Action Providers](#) for more information.

Introduced in API Level 11.

`android:actionViewClass`

Class name. A fully-qualified class name for the `View` to use as the action view. For example, `"android.widget.SearchView"` to use `SearchView` as an action view.

See [Action Views and Action Providers](#) for more information.

Warning: If you obfuscate your code using `ProGuard` (or a similar tool), be sure to exclude the class you specify in this attribute from renaming, because it can break the functionality.

Introduced in API Level 11.

`android:actionProviderClass`

Class name. A fully-qualified class name for the [ActionProvider](#) to use in place of the action item. For example, `"android.widget.ShareActionProvider"` to use [ShareActionProvider](#).

See [Action Views and Action Providers](#) for more information.

Warning: If you obfuscate your code using [ProGuard](#) (or a similar tool), be sure to exclude the class you specify in this attribute from renaming, because it can break the functionality.

Introduced in API Level 14.

`android:alphabeticShortcut`

Char. A character for the alphabetic shortcut key.

`android:numericShortcut`

Integer. A number for the numeric shortcut key.

`android:alphabeticModifiers`

Keyword. A modifier for the menu item's alphabetic shortcut. The default value corresponds to the Control key. Valid values:

Value	Description
META	Corresponds to the Meta meta key
CTRL	Corresponds to the Control meta key
ALT	Corresponds to the Alt meta key
SHIFT	Corresponds to the Shift meta key
SYM	Corresponds to the Sym meta key
FUNCTION	Corresponds to the Function meta key

Note: You can specify multiple keywords in an attribute. For example,

`android:alphabeticModifiers="CTRL|SHIFT"` indicates that to trigger the corresponding menu item, the user needs to press both Control and Shift meta keys along with the shortcut.

You can use the `setAlphabeticShortcut()` method to set the attribute values programmatically. For more information about the [alphabeticModifier](#) attribute, go to [alphabeticModifiers](#).

`android:numericModifiers`

Keyword. A modifier for the menu item's numeric shortcut. The default value corresponds to the Control key. Valid values:

Value	Description
META	Corresponds to the Meta meta key
CTRL	Corresponds to the Control meta key
ALT	Corresponds to the Alt meta key
SHIFT	Corresponds to the Shift meta key
SYM	Corresponds to the Sym meta key
FUNCTION	Corresponds to the Function meta key

Note: You can specify multiple keywords in an attribute. For example, `android:numericModifiers="CTRL|SHIFT"` indicates that to trigger the corresponding menu item, the user needs to press both Control and Shift meta keys along with the shortcut.

You can use the `setNumericShortcut()` method to set the attribute values programmatically. For more information about the `numericModifier` attribute, go to [numericModifiers](#).

`android:checkable`

Boolean. "true" if the item is checkable.

`android:checked`

Boolean. "true" if the item is checked by default.

`android:visible`

Boolean. "true" if the item is visible by default.

`android:enabled`

Boolean. "true" if the item is enabled by default.

`android:menuCategory`

Keyword. Value corresponding to [Menu CATEGORY_*](#) constants, which define the item's priority. Valid values:

Value	Description
<code>container</code>	For items that are part of a container.
<code>system</code>	For items that are provided by the system.
<code>secondary</code>	For items that are user-supplied secondary (infrequently used) options.
<code>alternative</code>	For items that are alternative actions on the data that is currently displayed.

`android:orderInCategory`

Integer. The order of "importance" of the item, within a group.

`<group>`

A menu group (to create a collection of items that share traits, such as whether they are visible, enabled, or checkable).

Contains one or more `<item>` elements. Must be a child of a `<menu>` element.

attributes:

`android:id`

Resource ID. A unique resource ID. To create a new resource ID for this item, use the form: "`@+id/name`". The plus symbol indicates that this should be created as a new ID.

`android:checkableBehavior`

Keyword. The type of checkable behavior for the group. Valid values:

Value	Description
<code>none</code>	Not checkable
<code>all</code>	All items can be checked (use checkboxes)
<code>single</code>	Only one item can be checked (use radio buttons)

`android:visible`

Boolean. "true" if the group is visible.

`android:enabled`

Boolean. "true" if the group is enabled.

`android:menuCategory`

Keyword. Value corresponding to `Menu CATEGORY_*` constants, which define the group's priority. Valid values:

Value	Description
<code>container</code>	For groups that are part of a container.
<code>system</code>	For groups that are provided by the system.
<code>secondary</code>	For groups that are user-supplied secondary (infrequently used) options.
<code>alternative</code>	For groups that are alternative actions on the data that is currently displayed.

`android:orderInCategory`

Integer. The default order of the items within the category.

EXAMPLE:

XML file saved at `res/menu/example_menu.xml`:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:title="@string/item1"
          android:icon="@drawable/group_item1_icon"
          android:showAsAction="ifRoom|withText" />
    <group android:id="@+id/group">
        <item android:id="@+id/group_item1"
              android:onClick="onGroupItemClick"
              android:title="@string/group_item1"
              android:icon="@drawable/group_item1_icon" />
        <item android:id="@+id/group_item2"
              android:onClick="onGroupItemClick"
              android:title="@string/group_item2"
              android:icon="@drawable/group_item2_icon" />
    </group>
    <item android:id="@+id_submenu"
          android:title="@string/submenu_title"
          android:showAsAction="ifRoom|withText" >
        <menu>
            <item android:id="@+id_submenu_item1"
                  android:title="@string_submenu_item1" />
        </menu>
    </item>
</menu>
```

The following application code inflates the menu from the `onCreateOptionsMenu(Menu)` callback and also declares the on-click callback for two of the items:

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.example_menu, menu);
    return true;
}

public void onGroupItemClick(MenuItem item) {
    // One of the group items (using the onClick attribute) was clicked
    // The item parameter passed here indicates which item it is
    // All other menu item clicks are handled by onOptionsItemSelected()
}
```



字符串资源

字符串资源为您的应用提供具有可选文本样式和格式设置的文本字符串。共有三种类型的资源可为您的应用提供字符串：

String

提供单个字符串的 XML 资源。

String Array

提供字符串数组的 XML 资源。

Quantity Strings (Plurals)

带有用于多元化的不同字符串的 XML 资源。

所有字符串都能应用某些样式设置标记和格式设置参数。如需了解有关样式和格式设置字符串的信息，请参阅有关[格式和样式设置](#)的部分。

String

可从应用或从其他资源文件（如 XML 布局）引用的单个字符串。

注：字符串是一种使用 `name` 属性（并非 XML 文件的名称）中提供的值进行引用的简单资源。因此，您可以在一个 XML 文件中将字符串资源与其他简单资源合并在一起，放在 `<resources>` 元素之下。

文件位置：

`res/values/filename.xml`

`filename` 是任意值。`<string>` 元素的 `name` 将用作资源 ID。

编译的资源数据类型：

指向 `String` 的资源指针。

资源引用：

在 Java 中：`R.string.string_name`

在 XML 中：`@string/string_name`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string
        name="string_name"
        >text_string</string>
</resources>
```

元素：

`<resources>`

必备。此元素必须是根节点。

无属性。

```
<string>
```

一个字符串，可包括样式设置标记。请注意，您必须将撇号和引号转义。如需了解有关如何正确设置字符串样式和格式的详细信息，请参阅下文的[格式和样式设置](#)。

属性：

`name`

String。字符串的名称。该名称将用作资源 ID。

示例：

保存在 `res/values/strings.xml` 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

该布局 XML 会对视图应用一个字符串：

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

以下应用代码用于检索字符串：

```
String string = getString(R.string.hello);
```

您可以使用 `getString(int)` 或 `getText(int)` 来检索字符串。`getText(int)` 将保留应用于字符串的任何富文本样式设置。

String Array

可从应用引用的字符串数组。

注：字符串数组是一种使用 `name` 属性（并非 XML 文件的名称）中提供的值进行引用的简单资源。因此，您可以在一个 XML 文件中将字符串数组资源与其他简单资源合并在一起，放在 `<resources>` 元素之下。

文件位置：

```
res/values/filename.xml
```

`filename` 是任意值。`<string-array>` 元素的 `name` 将用作资源 ID。

编译的资源数据类型：

指向 `String` 数组的资源指针。

资源引用：

在 Java 中：`R.array.string_array_name`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="string_array_name">
        <item
            >text_string</item>
    </string-array>
</resources>
```

元素：

```
<resources>
```

必备。此元素必须是根节点。

无属性。

```
<string-array>
```

定义一个字符串数组。包含一个或多个 `<item>` 元素。

属性：

```
name
```

String。数组的名称。该名称将用作资源 ID 来引用数组。

```
<item>
```

一个字符串，可包括样式设置标记。其值可以是对另一字符串资源的引用。必须是 `<string-array>` 元素的子项。请注意，您必须将撇号和引号转义。如需了解有关如何正确设置字符串样式和格式的信息，请参阅下文的[格式和样式设置](#)。

无属性。

示例：

保存在 `res/values/strings.xml` 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

以下应用代码用于检索字符串数组：

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Quantity Strings (Plurals)

不同语言在语法数量一致上具有不同的规则。例如，在英语中，数量 1 是一种特殊情况。我们会写成“1 book”，但如果是任何其他数量，我们则会写成“*n* books”。这种对单复数的区分很常见，但其他语言进行了更加细致的区分。Android 支持的完整集合包括 `zero`、`one`、`two`、`few`、`many` 和 `other`。

决定为给定语言和数量使用哪一种情况的规则可能非常复杂，因此 Android 为您提供了 `getQuantityString()` 等方法来选择适合您的资源。

尽管历史上被称作“数量字符串”（并且在 API 中也仍然这样叫），但数量字符串 只应用于表示复数。例如，使用数量字符串来实现 Gmail

的“Inbox”之类的情况是错误的，正确的做法是使用它们来实现“Inbox (12)”这种存在未读邮件的情况。 使用数量字符串来替代 `if` 语句似乎更为方便，但必须注意的是，某些语言（如中文）根本不做这些语法区分，因此您获取的始终是 `other` 字符串。

选择使用哪一个字符串完全取决于语法上的必要性。在英语中，即使数量是 0，一个表示 `zero` 的字符串也会被忽略，因为在语法上 0 与 2 或 1 以外的任何其他数字没有区别（“zero books”、“one book”、“two books”、等等）。相反，在韩语中，得到使用的就只有 `other` 字符串。

不要被某些事实误导，比如 `two` 听起来只能应用于数量 2：某种语言可能规定，2、12、102（等等）均相同对待，但与其他数量则区分对待。可以依靠翻译人员来了解他们的语言实际的区分要求。

通常可以利用“Books: 1”之类的数量中性表示来避免使用数量字符串。如果这是一种符合您的应用需要的样式，就能减轻您和翻译人员的工作负荷。

注：Plurals 集合是一种使用 `name` 属性（并非 XML 文件的名称）中提供的值进行引用的简单资源。因此，您可以在一个 XML 文件中将 plurals 资源与其他简单资源合并在一起，放在 `<resources>` 元素之下。

文件位置：

`res/values/filename.xml`

`filename` 是任意值。`<plurals>` 元素的 `name` 将用作资源 ID。

资源引用：

在 Java 中：`R.plurals_plural_name`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals
        name="plural_name">
        <item
            quantity=["zero" | "one" | "two" | "few" | "many" | "other"]
            >text_string</item>
    </plurals>
</resources>
```

元素：

`<resources>`

必备。此元素必须是根节点。

无属性。

`<plurals>`

一个字符串集合，根据事物数量提供其中的一个字符串。包含一个或多个 `<item>` 元素。

属性：

`name`

`String`。字符串对的名称。该名称将用作资源 ID。

`<item>`

一个复数或单数字符串。其值可以是对另一字符串资源的引用。必须是 `<plurals>` 元素的子项。请注意，您必须将撇号和引号转义。如需了解有关如何正确设置字符串样式和格式的信息，请参阅下文的[格式和样式设置](#)。

属性：

`quantity`

关键字。表示应在何时使用该字符串的值。以下是其有效值，括号内的示例并不详尽：

值	说明
zero	当语言要求对数字 0 做特殊对待时（如阿拉伯语的要求）。
one	当语言要求对 1 这类数字做特殊对待时（如英语和大多数其他语言中对数字 1 的对待要求；在俄语中，任何末尾是 1 但不是 11 的数字均属此类）。
two	当语言要求对 2 这类数字做特殊对待时（如威尔士语中对 2 的要求，或斯洛文尼亚语中对 102 的要求）。
few	当语言要求对“小”数字做特殊对待时（如捷克语中的 2、3 和 4；或波兰语中末尾是 2、3 或 4 但不是 12、13 或 14 的数字）。
many	当语言要求对“大”数字做特殊对待时（如马耳他语中末尾是 11-99 的数字）。
other	当语言不要求对给定数量做特殊对待时（如中文中的所有数字，或英语中的 42）。

示例：

保存在 res/values/strings.xml 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <!--
            As a developer, you should always supply "one" and "other"
            strings. Your translators will know which strings are actually
            needed for their language. Always include %d in "one" because
            translators will need to use %d for languages where "one"
            doesn't mean 1 (as explained above).
        -->
        <item quantity="one">%d song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>
```

保存在 res/values-pl/strings.xml 的 XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">Znaleziono %d piosenkę.</item>
        <item quantity="few">Znaleziono %d piosenki.</item>
        <item quantity="other">Znaleziono %d piosenek.</item>
    </plurals>
</resources>
```

Java 代码：

```
int count = getNumberOfSongsAvailable();
Resources res = getResources();
String songsFound = res.getQuantityString(R.plurals.numberOfSongsAvailable, count, count);
```

使用 `getQuantityString()` 方法时，如果您的字符串包括的字符串格式设置带有数字，则需要传递 `count` 两次。例如，对于字符串 `%d songs found`，第一个 `count` 参数选择相应的复数字符串，第二个 `count` 参数将插入 `%d` 占位符内。如果您的复数字符串不包括字符串格式设置，则无需向 `getQuantityString` 传递第三个参数。

格式和样式设置

关于如何正确设置字符串资源的格式和样式，您应该了解下面这几个要点。

转义撇号和引号

如果字符串中包含撇号 (')，您必须用反斜杠 (\') 将其转义，或为字符串加上双引号 ("")。例如，以下是一些有效和无效的字符串：

```
<string name="good_example">This\'ll work</string>
<string name="good_example_2">"This'll also work"</string>
<string name="bad_example">This doesn't work</string>
    <!-- Causes a compile error -->
```

如果字符串中包含双引号，您必须将其转义（使用 \"）。为字符串加上单引号不起作用。

```
<string name="good_example">This is a \"good string\".</string>
<string name="bad_example">This is a "bad string".</string>
    <!-- Quotes are stripped; displays as: This is a bad string. -->
<string name="bad_example_2">'This is another "bad string".'</string>
    <!-- Causes a compile error -->
```

设置字符串格式

如果您需要使用 `String.format(String, Object...)` 设置字符串格式，可以通过在字符串资源中加入格式参数来实现。例如，对于以下资源：

```
<string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

在本例中，格式字符串有两个参数：`%1$s` 是一个字符串，而 `%2$d` 是一个十进制数字。您可以像下面这样使用应用中的参数设置字符串格式：

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
```

使用 HTML 标记设置样式

您可以使用 HTML 标记为字符串添加样式设置。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

支持的 HTML 元素包括：

- `` 表示**粗体文本**。
- `<i>` 表示**斜体文本**。
- `<u>` 表示**下划线文本**。

有时，您可能想让自己创建的带样式文本资源同时也用作格式字符串。正常情况下，这是行不通的，因为 `String.format(String, Object...)` 方法会去除字符串中的所有样式信息。这个问题的解决方法是编写带转义实体的 HTML 标记，在完成格式设置后，这些实体可通过 `fromHtml(String)` 恢复。例如：

1. 将您带样式的文本资源存储为 HTML 转义字符串：

```
<resources>
    <string name="welcome_messages">Hello, %1$s! You have &lt;b>%2$d new messages&lt;/b>.</string>
</resources>
```

在这个带格式的字符串中，添加了 `` 元素。请注意，开括号使用 `<` 表示法进行了 HTML 转义。

2. 然后照常设置字符串格式，但还要调用 `fromHtml(String)` 以将 HTML 文本转换成带样式文本：

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

由于 `fromHtml(String)` 方法将设置所有 HTML 实体的格式，因此务必要使用 `htmlEncode(String)` 对您用于带格式文本的字符串中任何可能的 HTML 字符进行转义。例如，如果您向 `String.format()` 传递的字符串参数可能包含“<”或“&”之类的字符，则必须在设置格式前进行转义，这样在通过 `fromHtml(String)` 传递带格式字符串时，字符就能以原始形式显示出来。例如：

```
String escapedUsername = TextUtil.htmlEncode(username);

Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), escapedUsername, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

使用 Spannable 设置样式

`Spannable` 是一种文本对象，让您可以使用颜色和字体粗细等字体属性进行样式设置。您可以使用 `SpannableStringBuilder` 生成文本，然后对文本应用 `android.text.style` 包中定义的样式。

您可以利用下列辅助工具方法来设置许多 spannable 文本创建工作：

```
/***
 * Returns a CharSequence that concatenates the specified array of CharSequence
 * objects and then applies a list of zero or more tags to the entire range.
 *
 * @param content an array of character sequences to apply a style to
 * @param tags the styled span objects to apply to the content
 *             such as android.text.style.StyleSpan
 */
private static CharSequence apply(CharSequence[] content, Object... tags) {
    SpannableStringBuilder text = new SpannableStringBuilder();
    openTags(text, tags);
    for (CharSequence item : content) {
        text.append(item);
    }
    closeTags(text, tags);
    return text;
}

/**
 * Iterates over an array of tags and applies them to the beginning of the specified
 * Spannable object so that future text appended to the text will have the styling
 * applied to it. Do not call this method directly.
 */
private static void openTags(Spannable text, Object[] tags) {
    for (Object tag : tags) {
        text.setSpan(tag, 0, 0, Spannable.SPAN_MARK_MARK);
    }
}

/**
 * "Closes" the specified tags on a Spannable by updating the spans to be
 * endpoint-exclusive so that future text appended to the end will not take
 * on the same styling. Do not call this method directly.
 */
private static void closeTags(Spannable text, Object[] tags) {
    int len = text.length();
    for (Object tag : tags) {
        if (len > 0) {
            text.setSpan(tag, 0, len, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
        } else {
            text.removeSpan(tag);
        }
    }
}
```

以下 `bold`、`italic` 和 `color` 方法向您展示了如何调用这些帮助程序方法来应用 `android.text.style` 包中定义的样式。您可以创建类似方法来进行其他类型的文本样式设置。

```
/**  
 * Returns a CharSequence that applies boldface to the concatenation  
 * of the specified CharSequence objects.  
 */  
public static CharSequence bold(CharSequence... content) {  
    return apply(content, new StyleSpan(Typeface.BOLD));  
}  
  
/**  
 * Returns a CharSequence that applies italics to the concatenation  
 * of the specified CharSequence objects.  
 */  
public static CharSequence italic(CharSequence... content) {  
    return apply(content, new StyleSpan(Typeface.ITALIC));  
}  
  
/**  
 * Returns a CharSequence that applies a foreground color to the  
 * concatenation of the specified CharSequence objects.  
 */  
public static CharSequence color(int color, CharSequence... content) {  
    return apply(content, new ForegroundColorSpan(color));  
}
```

下面这个示例展示了如何将这些方法链接起来，创建出对不同词语应用不同类型样式的字符序列：

```
// Create an italic "hello, " a red "world",  
// and bold the entire sequence.  
CharSequence text = bold(italic(res.getString(R.string.hello)),  
    color(Color.RED, res.getString(R.string.world)));
```

Style Resource

See also

➤ [Styles and Themes](#)

A style resource defines the format and look for a UI. A style can be applied to an individual [View](#) (from within a layout file) or to an entire [Activity](#) or application (from within the manifest file).

For more information about creating and applying styles, please read [Styles and Themes](#).

Note: A style is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine style resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

`res/values/filename.xml`

The filename is arbitrary. The element's `name` will be used as the resource ID.

RESOURCE REFERENCE:

In XML: `@[package:]style/style_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style
        name="style_name"
        parent="@[package:]style/style_to_inherit">
        <item
            name="@[package:]style_property_name"
            >style_value</item>
    </style>
</resources>
```

ELEMENTS:

`<resources>`

Required. This must be the root node.

No attributes.

`<style>`

Defines a single style. Contains `<item>` elements.

attributes:

`name`

String. **Required.** A name for the style, which is used as the resource ID to apply the style to a View, Activity, or application.

parent

Style resource. Reference to a style from which this style should inherit style properties.

<item>

Defines a single property for the style. Must be a child of a <**style**> element.

attributes:

name

Attribute resource. **Required.** The name of the style property to be defined, with a package prefix if necessary (for example `android:textColor`).

EXAMPLE:

XML file for the style (saved in `res/values/`):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText" parent="@style/Text">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

XML file that applies the style to a `TextView` (saved in `res/layout/`):

```
<?xml version="1.0" encoding="utf-8"?>
<EditText
    style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

More Resource Types

This page defines more types of resources you can externalize, including:

Bool

XML resource that carries a boolean value.

Color

XML resource that carries a color value (a hexadecimal color).

Dimension

XML resource that carries a dimension value (with a unit of measure).

ID

XML resource that provides a unique identifier for application resources and components.

Integer

XML resource that carries an integer value.

Integer Array

XML resource that provides an array of integers.

Typed Array

XML resource that provides a `TypedArray` (which you can use for an array of drawables).

Bool

A boolean value defined in XML.

Note: A bool is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine bool resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

`res/values/filename.xml`

The filename is arbitrary. The `<bool>` element's `name` will be used as the resource ID.

RESOURCE REFERENCE:

In Java: `R.bool.bool_name`

In XML: `@[package:]bool/bool_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool
        name="bool_name"
        >[true | false]</bool>
</resources>
```

ELEMENTS:

<resources>

Required. This must be the root node.

No attributes.

<bool>

A boolean value: **true** or **false**.

attributes:

name

String. A name for the bool value. This will be used as the resource ID.

EXAMPLE:

XML file saved at `res/values-small/bools.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="screen_small">true</bool>
    <bool name="adjust_view_bounds">true</bool>
</resources>
```

This application code retrieves the boolean:

```
Resources res = getResources();
boolean screenIsSmall = res.getBoolean(R.bool.screen_small);
```

This layout XML uses the boolean for an attribute:

```
<ImageView
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:src="@drawable/logo"
    android:adjustViewBounds="@bool/adjust_view_bounds" />
```

Color

A color value defined in XML. The color is specified with an RGB value and alpha channel. You can use a color resource any place that accepts a hexadecimal color value. You can also use a color resource when a drawable resource is expected in XML (for example, `android:drawable="@color/green"`).

The value always begins with a pound (#) character and then followed by the Alpha-Red-Green-Blue information in one of the following formats:

- #RGB
- #ARGB

- #RRGGBB
- #AARRGGBB

Note: A color is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine color resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

`res/values/colors.xml`

The filename is arbitrary. The `<color>` element's `name` will be used as the resource ID.

RESOURCE REFERENCE:

In Java: `R.color.color_name`

In XML: `@[package:]color/color_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color
        name="color_name"
        >hex_color</color>
</resources>
```

ELEMENTS:

`<resources>`

Required. This must be the root node.

No attributes.

`<color>`

A color expressed in hexadecimal, as described above.

attributes:

`name`

String. A name for the color. This will be used as the resource ID.

EXAMPLE:

XML file saved at `res/values/colors.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <color name="translucent_red">#80ff0000</color>
</resources>
```

This application code retrieves the color resource:

```
Resources res = getResources();
int color = res.getColor(R.color.opaque_red);
```

This layout XML applies the color to an attribute:

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textColor="@color/translucent_red"  
    android:text="Hello"/>
```

Dimension

A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp. The following units of measure are supported by Android:

dp

Density-independent Pixels - An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

sp

Scale-independent Pixels - This is like the dp unit, but it is also scaled by the user's font size preference. It is recommended you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.

pt

Points - 1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.

px

Pixels - Corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; each device may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

mm

Millimeters - Based on the physical size of the screen.

in

Inches - Based on the physical size of the screen.

Note: A dimension is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine dimension resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

`res/values/filename.xml`

The filename is arbitrary. The `<dimen>` element's `name` will be used as the resource ID.

RESOURCE REFERENCE:

In Java: `R.dimen.dimension_name`

In XML: `@[package:]dimen/dimension_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen
        name="dimension_name"
        >dimension</dimen>
</resources>
```

ELEMENTS:

<resources>

Required. This must be the root node.

No attributes.

<dimen>

A dimension, represented by a float, followed by a unit of measurement (dp, sp, pt, px, mm, in), as described above.

attributes:

name

String. A name for the dimension. This will be used as the resource ID.

EXAMPLE:

XML file saved at `res/values/dimens.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="ball_radius">30dp</dimen>
    <dimen name="font_size">16sp</dimen>
</resources>
```

This application code retrieves a dimension:

```
Resources res = getResources();
float fontSize = res.getDimension(R.dimen.font_size);
```

This layout XML applies dimensions to attributes:

```
<TextView
    android:layout_height="@dimen/textview_height"
    android:layout_width="@dimen/textview_width"
    android:textSize="@dimen/font_size"/>
```

ID

A unique resource ID defined in XML. Using the name you provide in the `<item>` element, the Android developer tools create a unique integer in your project's `R.java` class, which you can use as an identifier for an application resources (for example, a `View` in your UI layout) or a unique integer for use in your application code (for example, as an ID for a dialog or a result code).

Note: An ID is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine ID resources with other simple resources in the one XML file, under one `<resources>` element. Also, remember that an ID resources does not reference an actual resource item; it is simply a unique ID that you can attach to other resources or use as a unique integer in your application.

FILE LOCATION:

`res/values/filename.xml`

The filename is arbitrary.

RESOURCE REFERENCE:

In Java: `R.id.name`

In XML: `@[package:]id/name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item
        type="id"
        name="id_name" />
</resources>
```

ELEMENTS:

`<resources>`

Required. This must be the root node.

No attributes.

`<item>`

Defines a unique ID. Takes no value, only attributes.

attributes:

`type`

Must be "id".

`name`

String. A unique name for the ID.

EXAMPLE:

XML file saved at `res/values/ids.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item type="id" name="button_ok" />
    <item type="id" name="dialog_exit" />
</resources>
```

Then, this layout snippet uses the "button_ok" ID for a Button widget:

```
<Button android:id="@+id/button_ok"
        style="@style/button_style" />
```

Notice that the `android:id` value does not include the plus sign in the ID reference, because the ID already exists, as defined in the `ids.xml` example above. (When you specify an ID to an XML resource using the plus sign—in the format `android:id="@+id/name"`—it means that the "name" ID does not exist and should be created.)

As another example, the following code snippet uses the "dialog_exit" ID as a unique identifier for a dialog:

```
showDialog(R.id.dialog_exit);
```

In the same application, the "dialog_exit" ID is compared when creating a dialog:

```
protected Dialog onCreateDialog(int) (int id) {  
    Dialog dialog;  
    switch(id) {  
        case R.id.dialog_exit:  
            ...  
            break;  
        default:  
            dialog = null;  
    }  
    return dialog;  
}
```

Integer

An integer defined in XML.

Note: An integer is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine integer resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

`res/values/filename.xml`

The filename is arbitrary. The `<integer>` element's `name` will be used as the resource ID.

RESOURCE REFERENCE:

In Java: `R.integer.integer_name`

In XML: `@[package:]integer/integer_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <integer  
        name="integer_name"  
        >integer</integer>  
    </resources>
```

ELEMENTS:

`<resources>`

Required. This must be the root node.

No attributes.

`<integer>`

An integer.

attributes:

`name`

String. A name for the integer. This will be used as the resource ID.

EXAMPLE:

XML file saved at `res/values/integers.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="max_speed">75</integer>
    <integer name="min_speed">5</integer>
</resources>
```

This application code retrieves an integer:

```
Resources res = getResources();
int maxSpeed = res.getInteger(R.integer.max_speed);
```

Integer Array

An array of integers defined in XML.

Note: An integer array is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file). As such, you can combine integer array resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

`res/values/filename.xml`

The filename is arbitrary. The `<integer-array>` element's `name` will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to an array of integers.

RESOURCE REFERENCE:

In Java: `R.array.integer_array_name`

In XML: `@[package:]array.integer_array_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array
        name="integer_array_name">
        <item
            >integer</item>
    </integer-array>
</resources>
```

ELEMENTS:

`<resources>`

Required. This must be the root node.

No attributes.

`<integer-array>`

Defines an array of integers. Contains one or more child `<item>` elements.

attributes:

```
    android:name
```

String. A name for the array. This name will be used as the resource ID to reference the array.

```
<item>
```

An integer. The value can be a reference to another integer resource. Must be a child of a `<integer-array>` element.

No attributes.

EXAMPLE:

XML file saved at `res/values/integers.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="bits">
        <item>4</item>
        <item>8</item>
        <item>16</item>
        <item>32</item>
    </integer-array>
</resources>
```

This application code retrieves the integer array:

```
Resources res = getResources();
int[] bits = res.getIntArray(R.array.bits);
```

Typed Array

A `TypedArray` defined in XML. You can use this to create an array of other resources, such as drawables. Note that the array is not required to be homogeneous, so you can create an array of mixed resource types, but you must be aware of what and where the data types are in the array so that you can properly obtain each item with the `TypedArray's get...()` methods.

Note: A typed array is a simple resource that is referenced using the value provided in the `name` attribute (not the name of the XML file).

As such, you can combine typed array resources with other simple resources in the one XML file, under one `<resources>` element.

FILE LOCATION:

```
res/values/filename.xml
```

The filename is arbitrary. The `<array>` element's `name` will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a `TypedArray`.

RESOURCE REFERENCE:

In Java: `R.array.array_name`

In XML: `@[package:]array.array_name`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array
        name="integer_array_name">
        <item>resource</item>
    </array>
</resources>
```

ELEMENTS:

<resources>

Required. This must be the root node.

No attributes.

<array>

Defines an array. Contains one or more child <item> elements.

attributes:

 android:name

String. A name for the array. This name will be used as the resource ID to reference the array.

<item>

A generic resource. The value can be a reference to a resource or a simple data type. Must be a child of an <array> element.

No attributes.

EXAMPLE:

XML file saved at res/values/arrays.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="icons">
        <item>@drawable/home</item>
        <item>@drawable/settings</item>
        <item>@drawable/logout</item>
    </array>
    <array name="colors">
        <item>#FFFF0000</item>
        <item>#FF00FF00</item>
        <item>#FF0000FF</item>
    </array>
</resources>
```

This application code retrieves each array and then obtains the first entry in each array:

```
Resources res = getResources();
TypedArray icons = res.obtainTypedArray(R.array.icons);
Drawable drawable = icons.getDrawable(0);

TypedArray colors = res.obtainTypedArray(R.array.colors);
int color = colors.getColor(0, 0);
```



应用清单

本文内容

- › [清单文件结构](#)
- › [文件约定](#)
- › [文件功能](#)
 - › [Intent 过滤器](#)
 - › [图标和标签](#)
 - › [权限](#)
 - › [库](#)

每个应用的根目录中都必须包含一个 `AndroidManifest.xml` 文件（且文件名精确无误）。清单文件向 Android 系统提供应用的必要信息，系统必须具有这些信息方可运行应用的任何代码。

此外，清单文件还可执行以下操作：

- 为应用的 Java 软件包命名。软件包名称充当应用的唯一标识符。
- 描述应用的各个组件，包括构成应用的 Activity、服务、广播接收器和内容提供程序。它还为实现每个组件的类命名并发布其功能，例如它们可以处理的 Intent 消息。这些声明向 Android 系统告知有关组件以及可以启动这些组件的条件的信息。
- 确定托管应用组件的进程。
- 声明应用必须具备哪些权限才能访问 API 中受保护的部分并与其它应用交互。还声明其它应用与该应用组件交互所需具备的权限
- 列出 `Instrumentation` 类，这些类可在应用运行时提供分析和其他信息。这些声明只会在应用处于开发阶段时出现在清单中，在应用发布之前将移除。
- 声明应用所需的最低 Android API 级别
- 列出应用必须链接到的库

注：准备要在 Chromebook 上运行的 Android 应用时，要考虑一些重要的硬件和软件功能限制。如需了解详细信息，请参阅 [Chromebook 的应用清单兼容性](#) 文档。

清单文件结构

下面的代码段显示了清单文件的通用结构及其可包含的每个元素。每个元素及其所有属性全部记录在一个单独的文件中。

提示：要查看本文档提及的任何元素的详细信息，只需点按元素名称。

下面是清单文件的示例：

```

<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

        <uses-library />

    </application>

</manifest>

```

以下列表包含可出现在清单文件中的所有元素，按字母顺序列出：

- <[action](#)>
- <[activity](#)>
- <[activity-alias](#)>
- <[application](#)>
- <[category](#)>
- <[data](#)>
- <[grant-uri-permission](#)>
- <[instrumentation](#)>

- <intent-filter>
- <manifest>
- <meta-data>
- <permission>
- <permission-group>
- <permission-tree>
- <provider>
- <receiver>
- <service>
- <supports-screens>
- <uses-configuration>
- <uses-feature>
- <uses-library>
- <uses-permission>
- <uses-sdk>

注：这些是仅有的合法元素 – 您无法添加自己的元素或属性。

文件约定

本节描述普遍适用于清单文件中所有元素和属性的约定和规则。

元素

只有 <manifest> 和 <application> 元素是必需的，它们都必须存在并且只能出现一次。其他大部分元素可以出现多次或者根本不出现。但清单文件中必须至少存在其中某些元素才有用。

如果一个元素包含某些内容，也就包含其他元素。所有值均通过属性进行设置，而不是通过元素内的字符数据设置。

同级别的元素通常不分先后顺序。例如，<activity>、<provider> 和 <service> 元素可以按任何顺序混合在一起。这条规则有两个主要例外：

- <activity-alias> 元素必须跟在别名所指的 <activity> 之后。
- <application> 元素必须是 <manifest> 元素内最后一个元素。换言之，</manifest> 结束标记必须紧接在 </application> 结束标记后。

属性

从某种意义上说，所有属性都是可选的。但是，必须指定某些属性，元素才可实现其目的。请使用本文档作为参考。对于真正可选的属性，它将指定默认值或声明缺乏规范时将执行何种操作。

除了根 <manifest> 元素的一些属性外，所有属性名称均以 android: 前缀开头。例如，android:alwaysRetainTaskState。由于该前缀是通用的，因此在按名称引用属性时，本文档通常会将其忽略。

声明类名

许多元素对应于 Java 对象，包括应用本身的元素（<application> 元素）及其主要组件：Activity（<activity>）、服务（<service>）、广播接收器（<receiver>）以及内容提供程序（<provider>）。

如果按照您针对组件类（Activity、Service 和 BroadcastReceiverContentProvider）几乎一直采用的方式来定义子类，则该子类需通过 name 属性来声明。该名称必须包含完整的软件包名称。例如，Service 子类可能会声明如下：

```
<manifest . . . >
    <application . . . >
        <service android:name="com.example.project.SecretService" . . . >
            .
            .
            </service>
        .
        .
    </application>
</manifest>
```

但是，如果字符串的第一个字符是句点，则应用的软件包名称（如 `<manifest>` 元素的 `package` 属性所指定）将附加到该字符串。以下赋值与上述方法相同：

```
<manifest package="com.example.project" . . . >
    <application . . . >
        <service android:name=".SecretService" . . . >
            .
            .
            </service>
        .
        .
    </application>
</manifest>
```

当启动组件时，Android 系统会创建已命名子类的实例。如果未指定子类，则会创建基类的实例。

多个值

如果可以指定多个值，则几乎总是在重复此元素，而不是列出单个元素内的多个值。例如，`intent` 过滤器可以列出多个操作：

```
<intent-filter . . . >
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    .
    .
</intent-filter>
```

资源值

某些属性的值可以显示给用户，例如，Activity 的标签和图标。这些属性的值应该本地化，并通过资源或主题进行设置。资源值用以下格式表示：

```
@[<i>package</i>:</i>]<i>type</i>/<i>name</i>
```

如果资源与应用在同一个软件包中，可以省略软件包名称。类型是资源类型，例如字符串或可绘制对象，名称是标识特定资源的名称。下面是示例：

```
<activity android:icon="@drawable/smallPic" . . . >
```

主题中的值用类似的方法表示，但是以 `?` 开头，而不是以 `@` 开头：

```
?[<i>package</i>:</i>]<i>type</i>/<i>name</i>
```

字符串值

如果属性值为字符串，则必须使用双反斜杠 (`\\"`) 转义字符，例如，使用 `\n` 表示换行符或使用 `\u0000` 表示 Unicode 字符。

文件功能

下文介绍在清单文件中体现某些 Android 特性的方式。

Intent 过滤器

应用的核心组件（例如其 Activity、服务和广播接收器）由 *intent* 激活。Intent 是一系列用于描述所需操作的信息（Intent 对象），其中包括要执行操作的数据、应执行操作的组件类别以及其他相关说明。Android 系统会查找合适的组件来响应 intent，根据需要启动组件的新实例，并将其传递到 Intent 对象。

组件将通过 intent 过滤器公布它们可响应的 intent 类型。由于 Android 系统在启动某组件之前必须了解该组件可以处理的 intent，因此 intent 过滤器在清单中被指定为 `<intent-filter>` 元素。一个组件可有任意数量的过滤器，其中每个过滤器描述一种不同的功能。

显式命名目标组件的 intent 将激活该组件，因此过滤器不起作用。不按名称指定目标的 intent 只有在能够通过组件的一个过滤器时才可激活该组件。

如需了解有关如何根据 intent 过滤器测试 Intent 对象的信息，请参阅 Intent 和 Intent 过滤器 文档。

图标和标签

对于可以显示给用户的小图标和文本标签，大量元素具有 `icon` 和 `label` 属性。此外，对于同样可以显示在屏幕上的较长说明文本，某些元素还具有 `description` 属性。例如，`<permission>` 元素具有所有这三个属性。因此，当系统询问用户是否授权给请求获得权限的应用时，权限图标、权限名称以及所需信息的说明均会呈现给用户。

无论何种情况下，在包含元素中设置的图标和标签都将成为所有容器子元素的默认 `icon` 和 `label` 设置。因此，在 `<application>` 元素中设置的图标和标签是每个应用组件的默认图标和标签。同样，为组件（例如 `<activity>` 元素）设置的图标和标签是组件每个 `<intent-filter>` 元素的默认设置。如果 `<application>` 元素设置标签，但是 Activity 及其 intent 过滤器不执行此操作，则应用标签将被视为 Activity 和 intent 过滤器的标签。

在实现过滤器公布的功能时，只要向用户呈现组件，系统便会使用为 intent 过滤器设置的图标和标签表示该组件。例如，具有 `android.intent.action.MAIN` 和 `android.intent.category.LAUNCHER` 设置的过滤器将 Activity 公布为可启动应用的功能，即，公布为应显示在应用启动器中的功能。在过滤器中设置的图标和标签显示在启动器中。

权限

权限是一种限制，用于限制对部分代码或设备上数据的访问。施加限制是为了保护可能被误用以致破坏或损害用户体验的关键数据和代码。

每种权限均由一个唯一的标签标识。标签通常指示受限制的操作。以下是 Android 定义的一些权限：

- `android.permission.CALL_EMERGENCY_NUMBERS`
- `android.permission.READ_OWNER_DATA`
- `android.permission.SET_WALLPAPER`
- `android.permission.DEVICE_POWER`

一个功能只能由一种权限保护。

如果应用需要访问受权限保护的功能，则必须在清单中使用 `<uses-permission>` 元素声明应用需要该权限。将应用安装到设备上之后，安装程序会通过检查签署应用证书的颁发机构并（在某些情况下）询问用户，确定是否授予请求的权限。如果授予权限，则应用能够使用受保护的功能。否则，其访问这些功能的尝试将会失败，并且不会向用户发送任何通知。

应用也可以使用权限保护自己的组件。它可以采用由 Android 定义（如 `android.Manifest.permission` 中所列）或由其他应用声明的任何权限。它也可以定义自己的权限。新权限用 `<permission>` 元素来声明。例如，Activity 可受到如下保护：

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . . />
    <uses-permission android:name="com.example.project.DEBIT_ACCT" />

    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
            android:permission="com.example.project.DEBIT_ACCT"
            . . . >

        </activity>
    </application>
</manifest>
```

请注意，在此示例中，`DEBIT_ACCT` 权限不仅是通过 `<permission>` 元素来声明，而且其使用也是通过 `<uses-permission>` 元素来请求。要让应用的其他组件也能够启动受保护的 Activity，您必须请求其使用权限，即便保护是由应用本身施加的亦如此。

同样还是在此示例中，如果将 `permission` 属性设置为在其他位置（例如，`android.permission.CALL_EMERGENCY_NUMBERS`）声明的权限，则无需使用 `<permission>` 元素再次声明。但是，仍有必要通过 `<uses-permission>` 请求其使用权限。

`<permission-tree>` 元素声明为代码中定义的一组权限声明命名空间，`<permission-group>` 为一组权限定义标签，包括在清单中使用 `<permission>` 元素声明的权限以及在其他位置声明的权限。这将影响如何对提供给用户的权限进行分组。`<permission-group>` 元素并不指定属于该组的权限，而只是为组提供名称。可通过向 `<permission>` 元素的 `permissionGroup` 属性分配组名，将权限放入组中。

库

每个应用均链接到默认的 Android 库，该库中包括用于开发应用（以及通用类，如 Activity、服务、intent、视图、按钮、应用、ContentProvider）的基本软件包。

但是，某些软件包驻留在自己的库中。如果应用使用来自其中任一软件包的代码，则必须明确要求其链接到这些软件包。清单必须包含单独的 `<uses-library>` 元素来命名其中每个库。库名称可在软件包的文档中找到。



<action>

SYNTAX:

```
<action android:name="string" />
```

CONTAINED IN:

```
<intent-filter>
```

DESCRIPTION:

Adds an action to an intent filter. An `<intent-filter>` element must contain one or more `<action>` elements. If there are no `<action>` elements in an intent filter, the filter doesn't accept any `Intent` objects. See [Intents and Intent Filters](#) for details on intent filters and the role of action specifications within a filter.

ATTRIBUTES:

`android:name`

The name of the action. Some standard actions are defined in the `Intent` class as `ACTION_string` constants. To assign one of these actions to this attribute, prepend "`android.intent.action.`" to the `string` that follows `ACTION_`. For example, for `ACTION_MAIN`, use "`android.intent.action.MAIN`" and for `ACTION_WEB_SEARCH`, use "`android.intent.action.WEB_SEARCH`".

For actions you define, it's best to use your app's package name as a prefix to ensure uniqueness. For example, a `TRANSMOGRIFY` action might be specified as follows:

```
<action android:name="com.example.project.TRANSMOGRIFY" />
```

INTRODUCED IN:

API Level 1

SEE ALSO:

```
<intent-filter>
```



<activity>

语法：

```
<activity android:allowEmbedded=["true" | "false"]
          android:allowTaskReparenting=["true" | "false"]
          android:alwaysRetainTaskState=["true" | "false"]
          android:autoRemoveFromRecents=["true" | "false"]
          android:banner="drawable resource"
          android:clearTaskOnLaunch=["true" | "false"]
          android:configChanges=["mcc", "mnc", "locale",
                                 "touchscreen", "keyboard", "keyboardHidden",
                                 "navigation", "screenLayout", "fontScale",
                                 "uiMode", "orientation", "screenSize",
                                 "smallestScreenSize"]
          android:documentLaunchMode=["intoExisting" | "always" |
                                       "none" | "never"]
          android:enabled=["true" | "false"]
          android:excludeFromRecents=["true" | "false"]
          android:exported=["true" | "false"]
          android:finishOnTaskLaunch=["true" | "false"]
          android:hardwareAccelerated=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:launchMode=["standard" | "singleTop" |
                             "singleTask" | "singleInstance"]
          android:maxRecents="integer"
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:noHistory=["true" | "false"]
          android:parentActivityName="string"
          android:permission="string"
          android:process="string"
          android:relinquishTaskIdentity=["true" | "false"]
          android:resizeableActivity=["true" | "false"]
          android:screenOrientation=["unspecified" | "behind" |
                                      "landscape" | "portrait" |
                                      "reverseLandscape" | "reversePortrait" |
                                      "sensorLandscape" | "sensorPortrait" |
                                      "userLandscape" | "userPortrait" |
                                      "sensor" | "fullSensor" | "nosensor" |
                                      "user" | "fullUser" | "locked"]
          android:stateNotNeeded=["true" | "false"]
          android:supportsPictureInPicture=["true" | "false"]
          android:taskAffinity="string"
          android:theme="resource or theme"
          android:uiOptions=["none" | "splitActionBarWhenNarrow"]
          android:windowSoftInputMode=["stateUnspecified",
                                      "stateUnchanged", "stateHidden",
                                      "stateAlwaysHidden", "stateVisible",
                                      "stateAlwaysVisible", "adjustUnspecified",
                                      "adjustResize", "adjustPan"] >
</activity>
```

包含它的文件：

```
<application>
```

可包含：

```
<intent-filter>
<meta-data>
```

说明：

声明一个实现应用的部分可视化用户界面的 Activity（一个 `Activity` 子类）。所有 Activity 都必须由清单文件中的 `<activity>` 元素表示。任何未在该处声明的 Activity 都将对系统不可见，并且也永远不会被运行。

属性：

`android:allowEmbedded`

表示该 Activity 可作为另一 Activity 的嵌入式子项启动。它尤其适用于子项所在的容器（如 `Display`）为另一 Activity 所拥有的情况。例如，用于 Wear 自定义通知的 Activity 必须声明此项，以便 Wear 在其上下文流中显示 Activity，后者位于另一进程中。

该属性的默认值为 `false`。

`android:allowTaskReparenting`

当启动 Activity 的任务接下来转至前台时，Activity 是否能从该任务转移至与其有亲和关系的任务——“`true`”表示它可以转移，“`false`”表示它仍须留在启动它的任务处。

如果未设置该属性，则对 Activity 应用由 `<application>` 元素的相应 `allowTaskReparenting` 属性设置的值。默认值为“`false`”。

正常情况下，当 Activity 启动时，会与启动它的任务关联，并在其整个生命周期中一直留在该任务处。您可以利用该属性强制 Activity 在其当前任务不再显示时将其父项更改为与其有亲和关系的任务。该属性通常用于使应用的 Activity 转移至与该应用关联的主任务。

例如，如果电子邮件包含网页链接，则点击链接会调出可显示网页的 Activity。该 Activity 由浏览器应用定义，但作为电子邮件任务的一部分启动。如果将其父项更改为浏览器任务，它会在浏览器下一次转至前台时显示，当电子邮件任务再次转至前台时则会消失。

Activity 的亲和关系由 `taskAffinity` 属性定义。任务的亲和关系通过读取其根 Activity 的亲和关系来确定。因此，按照定义，根 Activity 始终位于具有相同亲和关系的任务之中。由于具有“`singleTask`”或“`singleInstance`”启动模式的 Activity 只能位于任务的根，因此更改父项仅限于“`standard`”和“`singleTop`”模式。（另请参阅 `launchMode` 属性。）

`android:alwaysRetainTaskState`

系统是否始终保持 Activity 所在任务的状态——“`true`”表示保持，“`false`”表示允许系统在特定情况下将任务重置到其初始状态。默认值为“`false`”。该属性只对任务的根 Activity 有意义；对于所有其他 Activity，均忽略该属性。

正常情况下，当用户从主屏幕重新选择某个任务时，系统会在特定情况下清除该任务（从根 Activity 之上的堆栈中移除所有 Activity）。系统通常会在用户一段时间（如 30 分钟）内未访问任务时执行此操作。

不过，如果该属性的值是“`true`”，则无论用户如何到达任务，将始终返回到最后状态的任务。例如，在网络浏览器这类存在大量用户不愿失去的状态（如多个打开的标签）的应用中，该属性会很有用。

`android:autoRemoveFromRecents`

由具有该属性的 Activity 启动的任务是否一直保留在 `概览屏幕` 中，直至任务中的最后一个 Activity 完成为止。若为 `true`，则自动从概览屏幕中移除任务。它会替换调用方使用的 `FLAG_ACTIVITY_RETAIN_IN_RECENTS`。它必须是布尔值“`true`”或“`false`”。

`android:banner`

一种为其关联项提供扩展图形化横幅的 `可绘制资源`。将其与 `<activity>` 标记联用可为特定 Activity 提供默认横幅，也可与 `<application>` 标记联用，为所有应用 Activity 提供横幅。

系统使用横幅在 Android TV 主屏幕上表示应用。由于横幅只显示在主屏幕上，因此只应由包含的 Activity 能够处理 `CATEGORY_LEANBACK_LAUNCHER` Intent 的应用指定。

必须将该属性设置为对包含图像的可绘制资源的引用（例如 “`@drawable/banner`”）。没有默认横幅。

如需了解详细信息，请参阅“面向电视的 UI 模式”设计指南中的[横幅](#)，以及“电视应用入门指南”中的[提供主屏幕横幅](#)。

android:clearTaskOnLaunch

是否每当从主屏幕重新启动任务时都从中移除根 Activity 之外的所有 Activity —“`true`”表示始终将任务清除到只剩其根 Activity；“`false`”表示不做清除。默认值为“`false`”。该属性只对启动新任务的 Activity（根 Activity）有意义；对于任务中的所有其他 Activity，均忽略该属性。

当值为“`true`”时，每次用户再次启动任务时，无论用户最后在任务中正在执行哪个 Activity，也无论用户是使用[返回](#)还是[主屏幕](#)按钮离开，都会将用户转至任务的根 Activity。当值为“`false`”时，可在某些情况下清除任务中的 Activity（请参阅[alwaysRetainTaskState 属性](#)），但并非一律可以。

例如，假定有人从主屏幕启动了 Activity P，然后从那里转到 Activity Q。该用户接着按了[主屏幕](#)按钮，然后返回到 Activity P。正常情况下，用户将看到 Activity Q，因为那是其最后在 P 的任务中执行的 Activity。不过，如果 P 将此标志设置为“`true`”，则当用户按下[主屏幕](#)将任务转入后台时，其上的所有 Activity（在本例中为 Q）都会被移除。因此用户返回任务时只会看到 P。

如果该属性和 [allowTaskReparenting](#) 的值均为“`true`”，则如上所述，任何可以更改父项的 Activity 都将转移到与其有亲和关系的任务；其余 Activity 随即被移除。

android:configChanges

列出 Activity 将自行处理的配置更改。在运行时发生配置更改时，默认情况下会关闭 Activity 然后将其重新启动，但使用该属性声明配置将阻止 Activity 重新启动。Activity 反而会保持运行状态，并且系统会调用其 [onConfigurationChanged\(\)](#) 方法。

注：应避免使用该属性，并且只应在万不得已的情况下使用。如需了解有关如何正确处理配置更改所致重新启动的详细信息，请阅读[处理运行时变更](#)。

任何或所有下列字符串均是该属性的有效值。多个值使用“|”分隔 — 例如，“`locale|navigation|orientation`”。

值	说明
“ <code>mcc</code> ”	IMSI 移动国家/地区代码 (MCC) 发生了变化 - 检测到了 SIM 并更新了 MCC。
“ <code>mnc</code> ”	IMSI 移动网络代码 (MNC) 发生了变化 - 检测到了 SIM 并更新了 MNC。
“ <code>locale</code> ”	语言区域发生了变化 — 用户为文本选择了新的显示语言。
“ <code>touchscreen</code> ”	触摸屏发生了变化。 (这种情况通常永远不会发生。)
“ <code>keyboard</code> ”	键盘类型发生了变化 — 例如，用户插入了一个外置键盘。
“ <code>keyboardHidden</code> ”	键盘无障碍功能发生了变化 — 例如，用户显示了硬件键盘。
“ <code>navigation</code> ”	导航类型（轨迹球/方向键）发生了变化。 (这种情况通常永远不会发生。)
“ <code>screenLayout</code> ”	屏幕布局发生了变化 — 这可能是由激活了其他显示方式所致。
“ <code>fontScale</code> ”	字体缩放系数发生了变化 — 用户选择了新的全局字号。
“ <code>uiMode</code> ”	用户界面模式发生了变化 — 这可能是因用户将设备放入桌面/车载基座或夜间模式发生变化所致。请参阅 UiModeManager 。此项为 API 级别 8 中新增配置。
“ <code>orientation</code> ”	屏幕方向发生了变化 — 用户旋转了设备。 <p>注：如果您的应用面向 API 级别 13 或更高级别（按照 <code>minSdkVersion</code> 和 <code>targetSdkVersion</code> 属性所声明的级别），则还应声明 “<code>screenSize</code>” 配置，因为当设备在横向与纵向之间切换时，该配置也会发生变化。</p>
“ <code>screenSize</code> ”	当前可用屏幕尺寸发生了变化。它表示当前可用尺寸相对于当前纵横比的变化，因此会在用户在横向与纵向之间切换时发生变化。不过，如果您的应用面向 API 级别 12 或更低级别，则 Activity 始终会自行处理此配置变更（即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动 Activity）。 <p>此项为 API 级别 13 中新增配置。</p>
“ <code>smallestScreenSize</code> ”	物理屏幕尺寸发生了变化。它表示与方向无关的尺寸变化，因此只有在实际物理屏幕尺寸发生变

	化（如切换到外部显示器）时才会变化。对此配置的变更对应于 smallestWidth 配置 的变化。不过，如果您的应用面向 API 级别 12 或更低级别，则 Activity 始终会自行处理此配置变更（即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动 Activity）。 此项为 API 级别 13 中新增配置。
“layoutDirection”	布局方向发生了变化。例如，从左至右 (LTR) 更改为从右至左 (RTL)。 此项为 API 级别 17 中新增配置。

所有这些配置变更都可能影响应用看到的资源值。因此，调用 [onConfigurationChanged\(\)](#) 时，通常有必要再次获取所有资源（包括视图布局、可绘制对象等），以正确处理变化。

android:documentLaunchMode

指定每次启动任务时应如何向其中添加新的 Activity 实例。该属性允许用户让多个来自同一应用的文档出现在 [概览屏幕](#) 中。

该属性有四个值，会在用户使用该应用打开文档时产生以下效果：

值	说明
“intoExisting”	Activity 会为文档重复使用现有任务。使用该值与不设置 FLAG_ACTIVITY_MULTIPLE_TASK 标志、但设置 FLAG_ACTIVITY_NEW_DOCUMENT 标志所产生的效果相同，如 使用 Intent 标志添加任务 中所述。
“always”	Activity 为文档创建新任务，即便文档已打开也是如此。这与同时设置 FLAG_ACTIVITY_NEW_DOCUMENT 和 FLAG_ACTIVITY_MULTIPLE_TASK 标志的效果相同。
“none”	该 Activity 不会为 Activity 创建新任务。这是默认值，它只会在设置了 FLAG_ACTIVITY_NEW_TASK 时创建新任务。概览屏幕将按其默认方式对待此 Activity：为应用显示单个任务，该任务将从用户上次调用的任意 Activity 开始继续执行。
“never”	即使 Intent 包含 FLAG_ACTIVITY_NEW_DOCUMENT ，该 Activity 也不会启动到新文档之中。设置此值会替代 FLAG_ACTIVITY_NEW_DOCUMENT 和 FLAG_ACTIVITY_MULTIPLE_TASK 标志的行为（如果在 Activity 中设置了其中一个标志），并且概览屏幕将为应用显示单个任务，该任务将从用户上次调用的任意 Activity 开始继续执行。

注：对于除“none”和“never”以外的值，必须使用 [launchMode="standard"](#) 定义 Activity。如果未指定此属性，则使用 [documentLaunchMode="none"](#)。

android:enabled

系统是否可将 Activity 实例化 — “true” 表示可以，“false”表示不可以。默认值为“true”。

[application](#) 元素具有自己的 [enabled](#) 属性，该属性适用于所有应用组件，包括 Activity。 [application](#) 和 [activity](#) 属性必须都是“true”（因为它们都默认使用该值），系统才能将 Activity 实例化。如果任何一个属性是“false”，则无法进行实例化。

android:excludeFromRecents

是否应将该 Activity 启动的任务排除在最近使用的应用列表（即 [概览屏幕](#)）之外。也就是说，当该 Activity 是新任务的根 Activity 时，此属性确定任务是否应出现在最近使用的应用列表中。如果应将任务排除在列表之外，请设置“true”；如果应将其包括在内，则设置“false”。默认值为“false”。

android:exported

Activity 是否可由其他应用的组件启动 —“true”表示可以，“false”表示不可以。若为“false”，则 Activity 只能由同一应用的组件或使用同一用户 ID 的不同应用启动。

默认值取决于 Activity 是否包含 Intent 过滤器。没有任何过滤器意味着 Activity 只能通过指定其确切的类名称进行调用。这意味着 Activity 专供应用内部使用（因为其他应用不知晓其类名称）。因此，在这种情况下，默认值为“false”。另一方面，至少存在一个过滤器意味着 Activity 专供外部使用，因此默认值为“true”。

该属性并非限制 Activity 对其他应用开放度的唯一手段。您还可以利用权限来限制哪些外部实体可以调用 Activity（请参阅 [permission](#) 属性）。

android:finishOnTaskLaunch

每当用户再次启动其任务（在主屏幕上选择任务）时，是否应关闭（完成）现有 Activity 实例 —“`true`”表示应关闭，“`false`”表示不应关闭。默认值为“`false`”。

如果该属性和 `allowTaskReparenting` 均为“`true`”，则优先使用该属性。Activity 的亲和关系会被忽略。系统不是更改 Activity 的父项，而是将其销毁。

android:hardwareAccelerated

是否应为此 Activity 启用硬件加速渲染 —“`true`”表示应启用，“`false`”表示不应启用。默认值为“`false`”。

从 Android 3.0 开始，为应用提供了硬件加速 OpenGL 渲染器，以改善许多常见 2D 图形运算的性能。启用硬件加速渲染器时，Canvas、Paint、Xfermode、ColorFilter、Shader 和 Camera 中的大多数运算都会得到加速。这可以提高动画、滚动的流畅度和总体响应速度，即便是并不明确使用框架 OpenGL 库的应用也会受益。由于启用硬件加速会增加资源消耗，因此您的应用将占用更多内存。

请注意，并非所有 OpenGL 2D 运算都会得到加速。如果您启用硬件加速渲染器，请对应用进行测试，以确保其在利用渲染器时不会出错。

android:icon

一个表示 Activity 的图标。该图标会在需要在屏幕上表示 Activity 时显示给用户。例如，代表启动任务的 Activity 的图标显示在启动器窗口中。该图标通常附带标签（请参阅 `android:label` 属性）。

必须将该属性设置为对包含图像定义的可绘制资源的引用。如果未设置该属性，则改为使用为应用整体指定的图标（请参阅 `<application>` 元素的 `icon` 属性）。

这个 Activity 的图标 — 无论设置于此处还是由 `<application>` 元素设置 — 同时也是 Activity 所有 Intent 过滤器的默认图标（请参阅 `<intent-filter>` 元素的 `icon` 属性）。

android:label

一种可由用户读取的 Activity 标签。该标签会在必须将 Activity 呈现给用户时显示在屏幕上。它通常与 Activity 图标一并显示。

如果未设置该属性，则改为使用为应用整体设置的标签（请参阅 `<application>` 元素的 `label` 属性）。

这个 Activity 的标签 — 无论设置于此处还是由 `<application>` 元素设置 — 同时也是 Activity 所有 Intent 过滤器的默认标签（请参阅 `<intent-filter>` 元素的 `label` 属性）。

应将该标签设置为对字符串资源的引用，以便可以像用户界面中的其他字符串那样进行本地化。不过，为便于您开发应用，也可将其设置为原始字符串。

android:launchMode

有关应如何启动 Activity 的指令。共有四种模式与 `Intent` 对象中的 Activity 标志（`FLAG_ACTIVITY_*` 常量）协同工作，以确定在调用 Activity 处理 Intent 时应执行的操作。这些模式是：

```
"standard"  
"singleTop"  
"singleTask"  
"singleInstance"
```

默认模式是“`standard`”。

如下表所示，这些模式分为两大类，“`standard`”和“`singleTop`”Activity 为一类，“`singleTask`”和“`singleInstance`”为另一类。使用“`standard`”或“`singleTop`”启动模式的 Activity 可多次实例化。实例可归属任何任务，并且可以位于 Activity 堆栈中的任何位置。它们通常启动到名为 `startActivity()` 的任务之中（除非 Intent 对象包含 `FLAG_ACTIVITY_NEW_TASK` 指令，在此情况下会选择其他任务 — 请参阅 `taskAffinity` 属性）。

相比之下，“`singleTask`”和“`singleInstance`”Activity 只能启动任务。它们始终位于 Activity 堆栈的根位置。此外，设备一次只能保留一个 Activity 实例 — 只允许一个此类任务。

“standard”和“singleTop”模式只在一个方面有差异：每次“standard”Activity 有新的 Intent 时，系统都会创建新的类实例来响应该 Intent。每个实例处理单个 Intent。同理，也可创建新的“singleTop”Activity 实例来处理新的 Intent。不过，如果目标任务在其堆栈顶部已有一个 Activity 实例，那么该实例将接收新 Intent（通过调用 `onNewIntent()`）；此时不会创建新实例。在其他情况下 — 例如，如果“singleTop”的一个现有实例虽在目标任务内，但未处于堆栈顶部，或者虽然位于堆栈顶部，但不在目标任务中 — 则系统会创建一个新实例并将其推送到堆栈上。

同理，如果您向上导航到当前堆栈上的某个 Activity，该行为由父 Activity 的启动模式决定。如果父 Activity 有启动模式 `singleTop`（或 up Intent 包含 `FLAG_ACTIVITY_CLEAR_TOP`），则系统会将该父项置于堆栈顶部，并保留其状态。导航 Intent 由父 Activity 的 `onNewIntent()` 方法接收。如果父 Activity 有启动模式 `standard`（并且 up Intent 不包含 `FLAG_ACTIVITY_CLEAR_TOP`），则系统会将当前 Activity 及其父项同时弹出堆栈，并创建一个新的父 Activity 实例来接收导航 Intent。

“singleTask”和“singleInstance”模式同样只在一个方面有差异：“singleTask”Activity 允许其他 Activity 成为其任务的组成部分。它始终位于其任务的根位置，但其他 Activity（必然是“standard”和“singleTop”Activity）可以启动到该任务中。相反，“singleInstance”Activity 则不允许其他 Activity 成为其任务的组成部分。它是任务中唯一的 Activity。如果它启动另一个 Activity，系统会将该 Activity 分配给其他任务 — 就好像 Intent 中包含 `FLAG_ACTIVITY_NEW_TASK` 一样。

用例	启动模式	多个实例？	注释
大多数 Activity 的正常启动	“standard”	是	默认值。系统始终会在目标任务中创建新的 Activity 实例并向其传送 Intent。
	“singleTop”	有条件的	如果目标任务的顶部已存在一个 Activity 实例，则系统会通过调用该实例的 <code>onNewIntent()</code> 方法向其传送 Intent，而不是创建新的 Activity 实例。
专用启动 (不建议用作常规用途)	“singleTask”	否	系统在新任务的根位置创建 Activity 并向其传送 Intent。不过，如果已存在一个 Activity 实例，则系统会通过调用该实例的 <code>onNewIntent()</code> 方法向其传送 Intent，而不是创建新的 Activity 实例。
	“singleInstance”	否	与“singleTask”相同，只是系统不会将任何其他 Activity 启动到包含实例的任务中。该 Activity 始终是其任务唯一仅有的成员。

如上表所示，`standard` 是默认模式，并且适用于大多数的 Activity 类型。对许多类型的 Activity 而言，`SingleTop` 也是一个常见并且有用的启动模式。其他模式 — `singleTask` 和 `singleInstance` - 不适合 **大多数应用** 因为它们所形成的交互模式可能让用户感到陌生，并且与大多数其他应用迥异。

无论您选择哪一种启动模式，请务必在启动期间以及使用返回按钮从其他 Activity 和任务返回该 Activity 时对其进行易用性测试。

如需了解有关启动模式及其与 Intent 标志交互的详细信息，请参阅[任务和返回栈](#)文档。

`android:maxRecents`

`概览屏幕`中位于此 Activity 根位置的任务数上限。达到该条目数时，系统会从概览屏幕中移除最近最少使用的实例。有效值为 1-50（低内存设备使用 25）；0 为无效值。该值必须是整数，例如 50。默认值为 16。

`android:multiprocess`

是否可以将 Activity 实例启动到启动该实例的组件进程内 — `true` 表示可以，`false` 表示不可以。默认值为 `false`。

正常情况下，新的 Activity 实例会启动到定义它的应用进程中，因此所有 Activity 实例都在同一进程中运行。不过，如果该标志设置为 `true`，Activity 实例便可在多个进程中运行，这样系统就能在任何使用实例的地方创建实例（前提是权限允许这样做），但这几乎毫无必要性或可取之处。

`android:name`

实现 Activity 的类的名称，是 `Activity` 的子类。该属性值应为完全限定类名称（例如，“`com.example.project.ExtracurricularActivity`”）。不过，为了简便起见，如果名称的第一个字符是句点（例

如，“`.ExtracurricularActivity`”），则名称将追加到 `<manifest>` 元素中指定的软件包名称。

应用一旦发布，即**不应更改该名称**（除非您设置了 `android:exported="false"`）。

没有默认值。必须指定该名称。

android:noHistory

当用户离开 Activity 并且其在屏幕上不再可见时，是否应从 Activity 堆栈中将其移除并完成（调用其 `finish()` 方法）—“`true`”表示应将其完成，“`false`”表示不应将其完成。默认值为“`false`”。

“`true`”一值表示 Activity 不会留下历史轨迹。它不会留在任务的 Activity 堆栈内，因此用户将无法返回 Activity。在此情况下，如果您启动另一个 Activity 来获取该 Activity 的结果，系统永远不会调用 `onActivityResult()`。

该属性是在 API 级别 3 引入的。

android:parentActivityName

Activity 逻辑父项的类名称。此处的名称必须与为相应 `<activity>` 元素的 `android:name` 属性指定的类名称一致。

系统会读取该属性，以确定当用户按下操作栏中的“向上”按钮时应该启动哪一个 Activity。系统还可以利用这些信息通过 `TaskStackBuilder` 合成 Activity 的返回栈。

要支持 API 级别 4 - 16，您还可以使用为 “`android.support.PARENT_ACTIVITY`” 指定值的 `<meta-data>` 元素来声明父 Activity。例如：

```
<activity
    android:name="com.example.app.ChildActivity"
    android:label="@string/title_child_activity"
    android:parentActivityName="com.example.app.MainActivity" >
    <!-- Parent activity meta-data to support API level 4+ -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.app.MainActivity" />
</activity>
```

如需了解有关声明父 Activity 以支持向上导航的详细信息，请阅读[提供向上导航](#)。

该属性是在 API 级别 16 引入的。

android:permission

客户端启动 Activity 或以其他方式令其响应 Intent 而必须具备的权限的名称。如果系统尚未向 `startActivity()` 或 `startActivityForResult()` 的调用方授予指定权限，其 Intent 将不会传递给 Activity。

如果未设置该属性，则对 Activity 应用 `<application>` 元素的 `permission` 属性设置的权限。如果这两个属性均未设置，则 Activity 不受权限保护。

如需了解有关权限的详细信息，请参阅简介的[权限](#)一节和另一份文档[安全与权限](#)。

android:process

应在其中运行 Activity 的进程的名称。正常情况下，应用的所有组件都在为应用创建的默认进程名称内运行，您无需使用该属性。但在必要时，您可以使用该属性替换默认进程名称，以便让应用组件散布到多个进程中。

如果为该属性分配的名称以冒号（“`:`”）开头，则会在需要时创建应用专用的新进程，并且 Activity 会在该进程中运行。如果进程名称以小写字符开头，Activity 将在该名称的全局进程中运行，前提是它拥有相应的权限。这可以让不同应用中的组件共享一个进程，从而减少资源占用。

`<application>` 元素的 `process` 属性可为所有组件设置一个不同的默认进程名称。

android:relinquishTaskIdentity

Activity 是否将其任务标识符交给任务栈中在其之上的 Activity。如果任务根 Activity 的该属性设置为“`true`”，则任务会用其内的下一个 Activity 的 Intent 替换基本 Intent。如果下一个 Activity 的该属性也设置为“`true`”，则该 Activity 会将基本 Intent 给予其在同

一任务中启动的任何 Activity。系统继续为每个 Activity 执行此过程，直至遇到的某个 Activity 将该属性设置为“`false`”为止。默认值为“`false`”。

如果该属性设置为“`true`”，则 Activity 还可利用 `ActivityManager.TaskDescription` 来更改概览屏幕中的标签、颜色和图标。

`resizeableActivity`

指定应用是否支持多窗口显示。您可以在 `<activity>` 或 `<application>` 元素中设置该属性。

如果您将该属性设置为 `true`，则用户可以分屏和自由形状模式启动 Activity。如果您将该属性设置为 `false`，Activity 将不支持多窗口模式。如果该值为 `false`，且用户尝试在多窗口模式下启动 Activity，该 Activity 将全屏显示。

如果您的应用面向 API 级别 24 或更高级别，但未对该属性指定值，则该属性的值默认设为 `true`。

该属性是在 API 级别 24 添加的。

`android:screenOrientation`

Activity 在设备上的显示方向。如果 Activity 是在多窗口模式下运行，系统会忽略该属性。

其值可以是下列任一字符串：

<code>"unspecified"</code>	默认值。由系统选择方向。在不同设备上，系统使用的政策以及基于政策在特定上下文所做的选择可能有所差异。
<code>"behind"</code>	与 Activity 栈中紧接着它的 Activity 的方向相同。
<code>"landscape"</code>	横向方向（显示的宽度大于高度）。
<code>"portrait"</code>	纵向方向（显示的高度大于宽度）。
<code>"reverseLandscape"</code>	与正常横向方向相反的横向方向。API 级别 9 中的新增配置。
<code>"reversePortrait"</code>	与正常纵向方向相反的纵向方向。API 级别 9 中的新增配置。
<code>"sensorLandscape"</code>	横向方向，但根据设备传感器，可以是正常或反向的横向方向。API 级别 9 中的新增配置。
<code>"sensorPortrait"</code>	纵向方向，但根据设备传感器，可以是正常或反向的纵向方向。API 级别 9 中的新增配置。
<code>"userLandscape"</code>	横向方向，但根据设备传感器和用户的传感器首选项，可以是正常或反向的横向方向。如果用户锁定了基于传感器的旋转，其行为与 <code>landscape</code> 相同，否则，其行为与 <code>sensorLandscape</code> 相同。API 级别 18 中的新增配置。
<code>"userPortrait"</code>	纵向方向，但根据设备传感器和用户的传感器首选项，可以是正常或反向的纵向方向。如果用户锁定了基于传感器的旋转，其行为与 <code>portrait</code> 相同，否则，其行为与 <code>sensorPortrait</code> 相同。API 级别 18 中的新增配置。
<code>"sensor"</code>	方向由设备方向传感器决定。显示方向取决于用户如何手持设备，它会在用户旋转设备时发生变化。但一些设备默认情况下不会旋转到所有四种可能的方向。要允许全部四种方向，请使用 <code>"fullSensor"</code> 。
<code>"fullSensor"</code>	方向由 4 种方向中任一方向的设备方向传感器决定。这与 <code>"sensor"</code> 类似，不同的是它允许所有 4 种可能的屏幕方向，无论设备正常情况下采用什么方向（例如，一些设备正常情况下不使用反向纵向或反向横向，但它支持这些方向）。API 级别 9 中的新增配置。
<code>"nosensor"</code>	决定方向时不考虑物理方向传感器。传感器会被忽略，因此显示不会随用户对设备的移动而旋转。除了这个区别，系统在选择方向时使用的政策与“ <code>unspecified</code> ”设置相同。
<code>"user"</code>	用户当前的首选方向。
<code>"fullUser"</code>	如果用户锁定了基于传感器的旋转，其行为与 <code>user</code> 相同，否则，其行为与 <code>fullSensor</code> 相同，允许所有 4 种可能的屏幕方向。API 级别 18 中的新增配置。
<code>"locked"</code>	将方向锁定在其当前的任意旋转方向。API 级别 18 中的新增配置。

注：如果您声明其中一个横向或纵向值，系统将其视为对 Activity 运行方向的硬性要求。因此，您声明的值支持通过 Google Play 之类的服务进行过滤，这样就能将您的应用只提供给支持 Activity 所要求方向的设备。例如，如果您声明了 `"landscape"`、`"reverseLandscape"` 或 `"sensorLandscape"`，则您的应用将只提供给支持横向方向的设备。不过，您还

应通过 `<uses-feature>` 元素明确声明，您的应用要求采用纵向或横向方向。例如，`<uses-feature android:name="android.hardware.screen.portrait"/>`。这纯粹是 Google Play（以及其他支持它的服务）提供的一种过滤行为，平台本身并不能控制当设备仅支持特定方向时您的应用能否安装。

android:stateNotNeeded

能否在不保存 Activity 状态的情况下将其终止并成功重新启动 —“`true`”表示可在不考虑其之前状态的情况下重新启动，“`false`”表示需要之前状态。默认值为“`false`”。

正常情况下，为保存资源而暂时关闭 Activity 前，系统会调用其 `onSaveInstanceState()` 方法。该方法将 Activity 的当前状态存储在一个 `Bundle` 对象中，然后在 Activity 重新启动时将其传递给 `onCreate()`。如果该属性设置为“`true`”，系统可能不会调用 `onSaveInstanceState()`，并且会向 `onCreate()` 传递 `null` 而不是 `Bundle` - 这与它在 Activity 首次启动时完全一样。

“`true`”设置可确保 Activity 能够在未保留状态时重新启动。例如，显示主屏幕的 Activity 可以使用该设置来确保其由于某种原因崩溃时不会被移除。

supportsPictureInPicture

指定 Activity 是否支持 [画中画](#) 显示。如果 `android:resizeableActivity` 是 `false`，系统会忽略该属性。

该属性是在 API 级别 24 添加的。

android:taskAffinity

与 Activity 有着亲和关系的任务。从概念上讲，具有相同亲和关系的 Activity 归属同一任务（从用户的角度来看，则是归属同一“应用”）。任务的亲和关系由其根 Activity 的亲和关系确定。

亲和关系确定两件事 - Activity 更改到的父项任务（请参阅 `allowTaskReparenting` 属性）和通过 `FLAG_ACTIVITY_NEW_TASK` 标志启动 Activity 时将用来容纳它的任务。

默认情况下，应用中的所有 Activity 都具有相同的亲和关系。您可以设置该属性来以不同方式组合它们，甚至可以将在不同应用中定义的 Activity 置于同一任务内。要指定 Activity 与任何任务均无亲和关系，请将其设置为空字符串。

如果未设置该属性，则 Activity 继承为应用设置的亲和关系（请参阅 `<application>` 元素的 `taskAffinity` 属性）。应用默认亲和关系的名称是 `<manifest>` 元素设置的软件包名称。

android:theme

对定义 Activity 总体主题的样式资源的引用。它会自动将 Activity 的上下文设置为使用该主题（请参阅 `setTheme()`），它还可以引发 Activity 启动前的“启动”动画（以更加符合 Activity 的实际外观）。

如果未设置该属性，则 Activity 继承通过 `<application>` 元素的 `theme` 属性为应用整体设置的主题。如果该属性也未设置，则使用默认系统主题。如需了解详细信息，请参阅[样式和主题](#)开发者指南。

android:uiOptions

针对 Activity UI 的附加选项。

必须是下列值之一。

值	说明
<code>"none"</code>	无附加 UI 选项。这是默认值。
<code>"splitActionBarWhenNarrow"</code>	当水平空间受限时（例如在手持设备上的纵向模式下时）在屏幕底部添加一个栏以显示应用栏（也称为操作栏）中的操作项）。应用栏不是以少量操作项形式出现在屏幕顶部的应用栏中，而是分成了顶部导航区和底部操作项栏。这可以确保操作项以及顶部的导航和标题元素都能获得合理的空间。菜单项不会拆分到两个栏中，它们始终一起出现。

如需了解有关应用栏的详细信息，请参阅[添加应用栏](#)培训课。

该属性是在 API 级别 14 添加的。

android:windowSoftInputMode

Activity 的主窗口与包含屏幕软键盘的窗口的交互方式。该属性的设置影响两个方面：

- 当 Activity 成为用户注意的焦点时软键盘的状态 — 隐藏还是可见。
- 对 Activity 主窗口所做的调整 — 是否将其尺寸调小以为软键盘腾出空间，或者当窗口部分被软键盘遮挡时是否平移其内容以使当前焦点可见。

该设置必须是下表所列的值之一，或者是一个“state...”值加上一个“adjust...”值的组合。在任一组中设置多个值（例如，多个“state...”值）都会产生未定义结果。各值之间使用垂直条（|）分隔。例如：

```
<activity android:windowSoftInputMode="stateVisible|adjustResize" . . . >
```

此处设置的值（“stateUnspecified”和“adjustUnspecified”除外）替换主题中设置的值。

值	说明
“stateUnspecified”	不指定软键盘的状态（隐藏还是可见）。将由系统选择合适的状态，或依赖主题中的设置。 这是对软键盘行为的默认设置。
“stateUnchanged”	当 Activity 转至前台时保留软键盘最后所处的任何状态，无论是可见还是隐藏。
“stateHidden”	当用户选择 Activity 时 — 也就是说，当用户确实是向前导航到 Activity，而不是因离开另一 Activity 而返回时 — 隐藏软键盘。
“stateAlwaysHidden”	当 Activity 的主窗口有输入焦点时始终隐藏软键盘。
“stateVisible”	在正常的适宜情况下（当用户向前导航到 Activity 的主窗口时）显示软键盘。
“stateAlwaysVisible”	当用户选择 Activity 时 — 也就是说，当用户确实是向前导航到 Activity，而不是因离开另一 Activity 而返回时 — 显示软键盘。
“adjustUnspecified”	不指定 Activity 的主窗口是否调整尺寸以为软键盘腾出空间，或者窗口内容是否进行平移以在屏幕上显露当前焦点。系统会根据窗口的内容是否存在任何可滚动其内容的布局视图来自动选择其中一种模式。如果存在这样的视图，窗口将进行尺寸调整，前提是可通过滚动在较小区域内看到窗口的所有内容。 这是对主窗口行为的默认设置。
“adjustResize”	始终调整 Activity 主窗口的尺寸来为屏幕上的软键盘腾出空间。
“adjustPan”	不调整 Activity 主窗口的尺寸来为软键盘腾出空间，而是自动平移窗口的内容，使当前焦点永远不被键盘遮盖，让用户始终都能看到其输入的内容。这通常不如尺寸调正可取，因为用户可能需要关闭软键盘以到达被遮盖的窗口部分或与这些部分进行交互。

该属性是在 API 级别 3 引入的。

引入的版本：

API 级别 1，为 noHistory 和 windowSoftInputMode 之外的所有属性引入，这两个属性则是在 API 级别 3 中增加。

另请参阅：

```
<application>
<activity-alias>
```



<activity-alias>

SYNTAX:

```
<activity-alias android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:icon="drawable resource"
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:targetActivity="string" >
    .
</activity-alias>
```

CONTAINED IN:

```
<application>
```

CAN CONTAIN:

```
<intent-filter>
<meta-data>
```

DESCRIPTION:

An alias for an activity, named by the `targetActivity` attribute. The target must be in the same application as the alias and it must be declared before the alias in the manifest.

The alias presents the target activity as an independent entity. It can have its own set of intent filters, and they, rather than the intent filters on the target activity itself, determine which intents can activate the target through the alias and how the system treats the alias. For example, the intent filters on the alias may specify the "`android.intent.action.MAIN`" and "`android.intent.category.LAUNCHER`" flags, causing it to be represented in the application launcher, even though none of the filters on the target activity itself set these flags.

With the exception of `targetActivity`, `<activity-alias>` attributes are a subset of `<activity>` attributes. For attributes in the subset, none of the values set for the target carry over to the alias. However, for attributes not in the subset, the values set for the target activity also apply to the alias.

ATTRIBUTES:

`android:enabled`

Whether or not the target activity can be instantiated by the system through this alias — "`true`" if it can be, and "`false`" if not. The default value is "`true`".

The `<application>` element has its own `enabled` attribute that applies to all application components, including activity aliases. The `<application>` and `<activity-alias>` attributes must both be "`true`" for the system to be able to instantiate the target activity through the alias. If either is "`false`", the alias does not work.

`android:exported`

Whether or not components of other applications can launch the target activity through this alias — "`true`" if they can, and "`false`" if not. If "`false`", the target activity can be launched through the alias only by components of the same application as the alias or applications with the same user ID.

The default value depends on whether the alias contains intent filters. The absence of any filters means that the activity can be invoked through the alias only by specifying the exact name of the alias. This implies that the alias is intended only for application-internal use (since others would not know its name) — so the default value is "`false`". On the other hand, the presence of at least one filter implies that the alias is intended for external use — so the default value is "`true`".

`android:icon`

An icon for the target activity when presented to users through the alias. See the [`<activity>`](#) element's `icon` attribute for more information.

`android:label`

A user-readable label for the alias when presented to users through the alias. See the [`<activity>`](#) element's `label` attribute for more information.

`android:name`

A unique name for the alias. The name should resemble a fully qualified class name. But, unlike the name of the target activity, the alias name is arbitrary; it does not refer to an actual class.

`android:permission`

The name of a permission that clients must have to launch the target activity or get it to do something via the alias. If a caller of `startActivity()` or `startActivityForResult()` has not been granted the specified permission, the target activity will not be activated.

This attribute supplants any permission set for the target activity itself. If it is not set, a permission is not needed to activate the target through the alias.

For more information on permissions, see the [Permissions](#) section in the introduction.

`android:targetActivity`

The name of the activity that can be activated through the alias. This name must match the `name` attribute of an [`<activity>`](#) element that precedes the alias in the manifest.

INTRODUCED IN:

API Level 1

SEE ALSO:

[`<activity>`](#)



<application>

SYNTAX:

```
<application android:allowTaskReparenting=["true" | "false"]
    android:allowBackup=["true" | "false"]
    android:backupAgent="string"
    android:backupInForeground=["true" | "false"]
    android:banner="drawable resource"
    android:debuggable=["true" | "false"]
    android:description="string resource"
    android:directBootAware=["true" | "false"]
    android:enabled=["true" | "false"]
    android:extractNativeLibs=["true" | "false"]
    android:fullBackupContent="string"
    android:fullBackupOnly=["true" | "false"]
    android:hasCode=["true" | "false"]
    android:hardwareAccelerated=["true" | "false"]
    android:icon="drawable resource"
    android:isGame=["true" | "false"]
    android:killAfterRestore=["true" | "false"]
    android:largeHeap=["true" | "false"]
    android:label="string resource"
    android:logo="drawable resource"
    android:manageSpaceActivity="string"
    android:name="string"
    android:networkSecurityConfig="xml resource"
    android:permission="string"
    android:persistent=["true" | "false"]
    android:process="string"
    android:restoreAnyVersion=["true" | "false"]
    android:requiredAccountType="string"
    android:resizeableActivity=["true" | "false"]
    android:restrictedAccountType="string"
    android:supportsRtl=["true" | "false"]
    android:taskAffinity="string"
    android:testOnly=["true" | "false"]
    android:theme="resource or theme"
    android:uiOptions=["none" | "splitActionBarWhenNarrow"]
    android:usesCleartextTraffic=["true" | "false"]
    android:vmSafeMode=["true" | "false"] >
    .
    .
</application>
```

CONTAINED IN:

```
<manifest>
```

CAN CONTAIN:

```
<activity>
<activity-alias>
<meta-data>
<service>
<receiver>
<provider>
<uses-library>
```

DESCRIPTION:

The declaration of the application. This element contains subelements that declare each of the application's components and has attributes that can affect all the components. Many of these attributes (such as `icon`, `label`, `permission`, `process`, `taskAffinity`, and `allowTaskReparenting`) set default values for corresponding attributes of the component elements. Others (such as `debuggable`, `enabled`, `description`, and `allowClearUserData`) set values for the application as a whole and cannot be overridden by the components.

ATTRIBUTES

`android:allowTaskReparenting`

Whether or not activities that the application defines can move from the task that started them to the task they have an affinity for when that task is next brought to the front — `"true"` if they can move, and `"false"` if they must remain with the task where they started. The default value is `"false"`.

The `<activity>` element has its own `allowTaskReparenting` attribute that can override the value set here. See that attribute for more information.

`android:allowBackup`

Whether to allow the application to participate in the backup and restore infrastructure. If this attribute is set to false, no backup or restore of the application will ever be performed, even by a full-system backup that would otherwise cause all application data to be saved via adb. The default value of this attribute is true.

`android:backupAgent`

The name of the class that implements the application's backup agent, a subclass of `BackupAgent`. The attribute value should be a fully qualified class name (such as, `"com.example.project.MyBackupAgent"`). However, as a shorthand, if the first character of the name is a period (for example, `".MyBackupAgent"`), it is appended to the package name specified in the `<manifest>` element.

There is no default. The name must be specified.

`android:backupInForeground`

Indicates that `Auto Backup` operations may be performed on this app even if the app is in a foreground-equivalent state. The system shuts down an app during auto backup operation, so use this attribute with caution. Setting this flag to true can impact app behavior while the app is active.

The default value is `false`, which means that the OS will avoid backing up the app while it is running in the foreground (such as a music app that is actively playing music via a service in the `startForeground()` state).

`android:banner`

A `drawable resource` providing an extended graphical banner for its associated item. Use with the `<application>` tag to supply a default banner for all application activities, or with the `<activity>` tag to supply a banner for a specific activity.

The system uses the banner to represent an app in the Android TV home screen. Since the banner is displayed only in the home screen, it should only be specified by applications with an activity that handles the `CATEGORY_LEANBACK_LAUNCHER` intent.

This attribute must be set as a reference to a drawable resource containing the image (for example `"@drawable/banner"`). There is no default banner.

See [Banners](#) in the UI Patterns for TV design guide, and [Provide a home screen banner](#) in Get Started with TV Apps for more information.

`android:debuggable`

Whether or not the application can be debugged, even when running on a device in user mode — `"true"` if it can be, and `"false"` if not. The default value is `"false"`.

android:description

User-readable text about the application, longer and more descriptive than the application label. The value must be set as a reference to a string resource. Unlike the label, it cannot be a raw string. There is no default value.

android:directBootAware

Whether or not the application is *direct-boot aware*; that is, whether or not it can run before the user unlocks the device. If you're using a custom subclass of [Application](#), and if any component inside your application is direct-boot aware, then your entire custom application is considered to be direct-boot aware.

Note: During [Direct Boot](#), your application can only access the data that is stored in *device protected* storage.

The default value is "false".

android:enabled

Whether or not the Android system can instantiate components of the application — "true" if it can, and "false" if not. If the value is "true", each component's `enabled` attribute determines whether that component is enabled or not. If the value is "false", it overrides the component-specific values; all components are disabled.

The default value is "true".

android:extractNativeLibs

Whether or not the package installer extracts native libraries from the APK to the filesystem. If set to `false`, then your native libraries must be page aligned and stored uncompressed in the APK. No code changes are required as the linker loads the libraries directly from the APK at runtime.

The default value is "true".

android:fullBackupContent

This attribute points to an XML file that contains full backup rules for [Auto Backup](#). These rules determine what files get backed up. For more information, see [XML Config Syntax](#) for Auto Backup.

This attribute is optional. If it is not specified, by default, Auto Backup includes most of your app's files. For more information, see [Files that are backed up](#).

android:fullBackupOnly

This attribute indicates whether or not to use [Auto Backup](#) on devices where it is available. If set to `true`, then your app performs Auto Backup when installed on a device running Android 6.0 (API level 23) or higher. On older devices, your app ignores this attribute and performs [Key/Value Backups](#).

The default value is "false".

android:hasCode

Whether or not the application contains any code — "true" if it does, and "false" if not. When the value is "false", the system does not try to load any application code when launching components. The default value is "true".

An application would not have any code of its own only if it's using nothing but built-in component classes, such as an activity that uses the [AliasActivity](#) class, a rare occurrence.

android:hardwareAccelerated

Whether or not hardware-accelerated rendering should be enabled for all activities and views in this application — "true" if it should be enabled, and "false" if not. The default value is "true" if you've set either `minSdkVersion` or `targetSdkVersion` to "14" or higher; otherwise, it's "false".

Starting from Android 3.0 (API level 11), a hardware-accelerated OpenGL renderer is available to applications, to improve

performance for many common 2D graphics operations. When the hardware-accelerated renderer is enabled, most operations in Canvas, Paint, Xfermode, ColorFilter, Shader, and Camera are accelerated. This results in smoother animations, smoother scrolling, and improved responsiveness overall, even for applications that do not explicitly make use of the framework's OpenGL libraries.

Note that not all of the OpenGL 2D operations are accelerated. If you enable the hardware-accelerated renderer, test your application to ensure that it can make use of the renderer without errors.

For more information, read the [Hardware Acceleration](#) guide.

`android:icon`

An icon for the application as whole, and the default icon for each of the application's components. See the individual `icon` attributes for `<activity>`, `<activity-alias>`, `<service>`, `<receiver>`, and `<provider>` elements.

This attribute must be set as a reference to a drawable resource containing the image (for example "`@drawable/icon`"). There is no default icon.

`android:isGame`

Whether or not the application is a game. The system may group together applications classified as games or display them separately from other applications.

The default is `false`.

`android:killAfterRestore`

Whether the application in question should be terminated after its settings have been restored during a full-system restore operation. Single-package restore operations will never cause the application to be shut down. Full-system restore operations typically only occur once, when the phone is first set up. Third-party applications will not normally need to use this attribute.

The default is `true`, which means that after the application has finished processing its data during a full-system restore, it will be terminated.

`android:largeHeap`

Whether your application's processes should be created with a large Dalvik heap. This applies to all processes created for the application. It only applies to the first application loaded into a process; if you're using a shared user ID to allow multiple applications to use a process, they all must use this option consistently or they will have unpredictable results.

Most apps should not need this and should instead focus on reducing their overall memory usage for improved performance. Enabling this also does not guarantee a fixed increase in available memory, because some devices are constrained by their total available memory.

To query the available memory size at runtime, use the methods `getMemoryClass()` or `getLargeMemoryClass()`.

`android:label`

A user-readable label for the application as a whole, and a default label for each of the application's components. See the individual `label` attributes for `<activity>`, `<activity-alias>`, `<service>`, `<receiver>`, and `<provider>` elements.

The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

`android:logo`

A logo for the application as whole, and the default logo for activities.

This attribute must be set as a reference to a drawable resource containing the image (for example "`@drawable/logo`"). There is no default logo.

`android:manageSpaceActivity`

The fully qualified name of an Activity subclass that the system can launch to let users manage the memory occupied by the application on the device. The activity should also be declared with an [activity](#) element.

android:name

The fully qualified name of an [Application](#) subclass implemented for the application. When the application process is started, this class is instantiated before any of the application's components.

The subclass is optional; most applications won't need one. In the absence of a subclass, Android uses an instance of the base Application class.

android:networkSecurityConfig

Specifies the name of the XML file that contains your application's [Network Security Configuration](#). The value must be a reference to the XML resource file containing the configuration.

This attribute was added in API level 24.

android:permission

The name of a permission that clients must have in order to interact with the application. This attribute is a convenient way to set a permission that applies to all of the application's components. It can be overwritten by setting the [permission](#) attributes of individual components.

For more information on permissions, see the [Permissions](#) section in the introduction and another document, [Security and Permissions](#).

android:persistent

Whether or not the application should remain running at all times — "[true](#)" if it should, and "[false](#)" if not. The default value is "[false](#)". Applications should not normally set this flag; persistence mode is intended only for certain system applications.

android:process

The name of a process where all components of the application should run. Each component can override this default by setting its own [process](#) attribute.

By default, Android creates a process for an application when the first of its components needs to run. All components then run in that process. The name of the default process matches the package name set by the [manifest](#) element.

By setting this attribute to a process name that's shared with another application, you can arrange for components of both applications to run in the same process — but only if the two applications also share a user ID and be signed with the same certificate.

If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, is created when it's needed. If the process name begins with a lowercase character, a global process of that name is created. A global process can be shared with other applications, reducing resource usage.

android:restoreAnyVersion

Indicates that the application is prepared to attempt a restore of any backed-up data set, even if the backup was stored by a newer version of the application than is currently installed on the device. Setting this attribute to [true](#) will permit the Backup Manager to attempt restore even when a version mismatch suggests that the data are incompatible. *Use with caution!*

The default value of this attribute is [false](#).

android:requiredAccountType

Specifies the account type required by the application in order to function. If your app requires an [Account](#), the value for this attribute must correspond to the account authenticator type used by your app (as defined by [AuthenticatorDescription](#)), such as "com.google".

The default value is null and indicates that the application can work *without* any accounts.

Because restricted profiles currently cannot add accounts, specifying this attribute **makes your app unavailable from a restricted profile** unless you also declare `android:restrictedAccountType` with the same value.

Caution: If the account data may reveal personally identifiable information, it's important that you declare this attribute and leave `android:restrictedAccountType` null, so that restricted profiles cannot use your app to access personal information that belongs to the owner user.

This attribute was added in API level 18.

`resizeableActivity`

Specifies whether the app supports [multi-window display](#). You can set this attribute in either the `<activity>` or `<application>` element.

If you set this attribute to true, the user can launch the activity in split-screen and freeform modes. If you set the attribute to false, the activity does not support multi-window mode. If this value is false, and the user attempts to launch the activity in multi-window mode, the activity takes over the full screen.

If your app targets API level 24 or higher, but you do not specify a value for this attribute, the attribute's value defaults to true.

This attribute was added in API level 24.

`android:restrictedAccountType`

Specifies the account type required by this application and indicates that restricted profiles are allowed to access such accounts that belong to the owner user. If your app requires an [Account](#) and restricted profiles **are allowed to access** the primary user's accounts, the value for this attribute must correspond to the account authenticator type used by your app (as defined by [AuthenticatorDescription](#)), such as "com.google".

The default value is null and indicates that the application can work *without* any accounts.

Caution: Specifying this attribute allows restricted profiles to use your app with accounts that belong to the owner user, which may reveal personally identifiable information. If the account may reveal personal details, you **should not** use this attribute and you should instead declare the `android:requiredAccountType` attribute to make your app unavailable to restricted profiles.

This attribute was added in API level 18.

`android:supportsRtl`

Declares whether your application is willing to support right-to-left (RTL) layouts.

If set to `true` and `targetSdkVersion` is set to 17 or higher, various RTL APIs will be activated and used by the system so your app can display RTL layouts. If set to `false` or if `targetSdkVersion` is set to 16 or lower, the RTL APIs will be ignored or will have no effect and your app will behave the same regardless of the layout direction associated to the user's Locale choice (your layouts will always be left-to-right).

The default value of this attribute is `false`.

This attribute was added in API level 17.

`android:taskAffinity`

An affinity name that applies to all activities within the application, except for those that set a different affinity with their own `taskAffinity` attributes. See that attribute for more information.

By default, all activities within an application share the same affinity. The name of that affinity is the same as the package name set by the `<manifest>` element.

`android:testOnly`

Indicates whether this application is only for testing purposes. For example, it may expose functionality or data outside of itself that would cause a security hole, but is useful for testing. This kind of APK can be installed only through `adb`—you cannot publish it to Google Play.

Android Studio automatically adds this attribute when you click **Run** .

`android:theme`

A reference to a style resource defining a default theme for all activities in the application. Individual activities can override the default by setting their own `theme` attributes. For more information, see the [Styles and Themes](#) developer guide.

`android:uiOptions`

Extra options for an activity's UI.

Must be one of the following values.

Value	Description
<code>"none"</code>	No extra UI options. This is the default.
<code>"splitActionBarWhenNarrow"</code>	Add a bar at the bottom of the screen to display action items in the <i>app bar</i> (also known as the <i>action bar</i>), when constrained for horizontal space (such as when in portrait mode on a handset). Instead of a small number of action items appearing in the app bar at the top of the screen, the app bar is split into the top navigation section and the bottom bar for action items. This ensures a reasonable amount of space is made available not only for the action items, but also for navigation and title elements at the top. Menu items are not split across the two bars; they always appear together.

For more information about the app bar, see the [Adding the App Bar](#) training class.

This attribute was added in API level 14.

`android:usesCleartextTraffic`

Indicates whether the app intends to use cleartext network traffic, such as cleartext HTTP. The default value is `"true"`.

When the attribute is set to `"false"`, platform components (for example, HTTP and FTP stacks, `DownloadManager`, `MediaPlayer`) will refuse the app's requests to use cleartext traffic. Third-party libraries are strongly encouraged to honor this setting as well. The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering: a network attacker can eavesdrop on transmitted data and also modify it without being detected.

This flag is honored on a best effort basis because it's impossible to prevent all cleartext traffic from Android applications given the level of access provided to them. For example, there's no expectation that the `Socket` API will honor this flag because it cannot determine whether its traffic is in cleartext. However, most network traffic from applications is handled by higher-level network stacks/components which can honor this flag by either reading it from `ApplicationInfo.flags` or `NetworkSecurityPolicy.isCleartextTrafficPermitted()`.

NOTE: `WebView` does not honor this flag.

During app development, StrictMode can be used to identify any cleartext traffic from the app: see `StrictMode.VmPolicy.Builder.detectCleartextNetwork()`.

This attribute was added in API level 23.

This flag is ignored on Android 7.0 (API level 24) and above if an Android Network Security Config is present.

`android:vmSafeMode`

Indicates whether the app would like the virtual machine (VM) to operate in safe mode. The default value is `"false"`.

This attribute was added in API level 8 where a value of "true" disabled the Dalvik just-in-time (JIT) compiler.

This attribute was adapted in API level 22 where a value of "true" disabled the ART ahead-of-time (AOT) compiler.

INTRODUCED IN:

API Level 1

SEE ALSO:

[<activity>](#)
[<service>](#)
[<receiver>](#)
[<provider>](#)

<category>

SYNTAX:

```
<category android:name="string" />
```

CONTAINED IN:

```
<intent-filter>
```

DESCRIPTION:

Adds a category name to an intent filter. See [Intents and Intent Filters](#) for details on intent filters and the role of category specifications within a filter.

ATTRIBUTES:

`android:name`

The name of the category. Standard categories are defined in the [Intent](#) class as `CATEGORY_name` constants. The name assigned here can be derived from those constants by prefixing "`android.intent.category.`" to the `name` that follows `CATEGORY_`. For example, the string value for `CATEGORY_LAUNCHER` is "`android.intent.category.LAUNCHER`".

Note: In order to receive implicit intents, you must include the `CATEGORY_DEFAULT` category in the intent filter. The methods `startActivity()` and `startActivityForResult()` treat all intents as if they declared the `CATEGORY_DEFAULT` category. If you do not declare it in your intent filter, no implicit intents will resolve to your activity.

Custom categories should use the package name as a prefix, to ensure that they are unique.

INTRODUCED IN:

API Level 1

SEE ALSO:

```
<action>
<data>
```



<compatible-screens>

SYNTAX:

```
<compatible-screens>
    <screen android:screenSize=["small" | "normal" | "large" | "xlarge"]
            android:screenDensity=["ldpi" | "mdpi" | "hdpi" | "xhdpi"
            | "280" | "360" | "420" | "480" | "560" ] />
    ...
</compatible-screens>
```

CONTAINED IN:

<manifest>

DESCRIPTION:

Specifies each screen configuration with which the application is compatible. Only one instance of the `<compatible-screens>` element is allowed in the manifest, but it can contain multiple `<screen>` elements. Each `<screen>` element specifies a specific screen size-density combination with which the application is compatible.

The Android system *does not* read the `<compatible-screens>` manifest element (neither at install-time nor at runtime). This element is informational only and may be used by external services (such as Google Play) to better understand the application's compatibility with specific screen configurations and enable filtering for users. Any screen configuration that is *not* declared in this element is a screen with which the application is *not* compatible. Thus, external services (such as Google Play) should not provide the application to devices with such screens.

Caution: Normally, **you should not use this manifest element**. Using this element can dramatically reduce the potential user base for your application, by not allowing users to install your application if they have a device with a screen configuration that you have not listed. You should use it only as a last resort, when the application absolutely does not work with specific screen configurations. Instead of using this element, you should follow the guide to [Supporting Multiple Screens](#) to provide scalable support for multiple screens using alternative layouts and bitmaps for different screen sizes and densities.

If you want to set only a minimum screen size for your application, then you should use the `<supports-screens>` element. For example, if you want your application to be available only for *large* and *xlarge* screen devices, the `<supports-screens>` element allows you to declare that your application does not support *small* and *normal* screen sizes. External services (such as Google Play) will filter your application accordingly. You can also use the `<supports-screens>` element to declare whether the system should resize your application for different screen sizes.

Also see the [Filters on Google Play](#) document for more information about how Google Play filters applications using this and other manifest elements.

CHILD ELEMENTS:

<screen>

Specifies a single screen configuration with which the application is compatible.

At least one instance of this element must be placed inside the `<compatible-screens>` element. This element *must include both* the `android:screenSize` and `android:screenDensity` attributes (if you do not declare both attributes, then the element is ignored).

attributes:

`android:screenSize`

Required. Specifies the screen size for this screen configuration.

Accepted values:

- `small`
- `normal`
- `large`
- `xlarge`

For information about the different screen sizes, see [Supporting Multiple Screens](#).

`android:screenDensity`

Required. Specifies the screen density for this screen configuration.

Accepted values:

- `"ldpi"` (approximately 120 dpi)
- `"mdpi"` (approximately 160 dpi)
- `"hdpi"` (approximately 240 dpi)
- `"xhdpi"` (approximately 320 dpi)
- `"280"`
- `"360"`
- `"420"`
- `"480"`
- `"560"`

For information about the different screen densities, see [Supporting Multiple Screens](#).

EXAMPLE

If your application is compatible with only small and normal screens, regardless of screen density, then you must specify twelve different `<screen>` elements, because each screen size has six different density configurations. You must declare each one of these; any combination of size and density that you do *not* specify is considered a screen configuration with which your application is *not* compatible. Here's what the manifest entry looks like if your application is compatible with only small and normal screens:

```
<manifest ... >
  ...
  <compatible-screens>
    <!-- all small size screens -->
    <screen android:screenSize="small" android:screenDensity="ldpi" />
    <screen android:screenSize="small" android:screenDensity="mdpi" />
    <screen android:screenSize="small" android:screenDensity="hdpi" />
    <screen android:screenSize="small" android:screenDensity="xhdpi" />
    <screen android:screenSize="small" android:screenDensity="xxhdpi" />
    <screen android:screenSize="small" android:screenDensity="xxxhdpi" />
    <!-- all normal size screens -->
    <screen android:screenSize="normal" android:screenDensity="ldpi" />
    <screen android:screenSize="normal" android:screenDensity="mdpi" />
    <screen android:screenSize="normal" android:screenDensity="hdpi" />
    <screen android:screenSize="normal" android:screenDensity="xhdpi" />
    <screen android:screenSize="normal" android:screenDensity="xxhdpi" />
    <screen android:screenSize="normal" android:screenDensity="xxxhdpi" />
  </compatible-screens>
  <application ... >
    ...
  </application>
</manifest>
```

INTRODUCED IN:

API Level 9

SEE ALSO:

[Supporting Multiple Screens](#)

[Filters on Google Play](#)



<data>

SYNTAX:

```
<data android:scheme="string"
      android:host="string"
      android:port="string"
      android:path="string"
      android:pathPattern="string"
      android:pathPrefix="string"
      android:mimeType="string" />
```

CONTAINED IN:

```
<intent-filter>
```

DESCRIPTION:

Adds a data specification to an intent filter. The specification can be just a data type (the `mimeType` attribute), just a URI, or both a data type and a URI. A URI is specified by separate attributes for each of its parts:

```
<scheme>://<host>:<port>[<path>|<pathPrefix>|<pathPattern>]
```

These attributes that specify the URL format are optional, but also mutually dependent:

- If a `scheme` is not specified for the intent filter, all the other URI attributes are ignored.
- If a `host` is not specified for the filter, the `port` attribute and all the path attributes are ignored.

All the `<data>` elements contained within the same `<intent-filter>` element contribute to the same filter. So, for example, the following filter specification,

```
<intent-filter . . . >
  <data android:scheme="something" android:host="project.example.com" />
  .
</intent-filter>
```

is equivalent to this one:

```
<intent-filter . . . >
  <data android:scheme="something" />
  <data android:host="project.example.com" />
  .
</intent-filter>
```

You can place any number of `<data>` elements inside an `<intent-filter>` to give it multiple data options. None of its attributes have default values.

Information on how intent filters work, including the rules for how Intent objects are matched against filters, can be found in another document, [Intents and Intent Filters](#). See also the [Intent Filters](#) section in the manifest file overview.

ATTRIBUTES:

`android:scheme`

The scheme part of a URI. This is the minimal essential attribute for specifying a URI; at least one `scheme` attribute must be set for the filter, or none of the other URI attributes are meaningful.

A scheme is specified without the trailing colon (for example, `http`, rather than `http:`).

If the filter has a data type set (the `mimeType` attribute) but no scheme, the `content:` and `file:` schemes are assumed.

Note: Scheme matching in the Android framework is case-sensitive, unlike the RFC. As a result, you should always specify schemes using lowercase letters.

`android:host`

The host part of a URI authority. This attribute is meaningless unless a `scheme` attribute is also specified for the filter. To match multiple subdomains, use an asterisk (*) to match zero or more characters in the host. For example, the host `*.google.com` matches `www.google.com`, `.google.com`, and `developer.google.com`.

The asterisk must be the first character of the host attribute. For example, the host `google.co.*` is invalid because the asterisk wildcard is not the first character.

Note: host name matching in the Android framework is case-sensitive, unlike the formal RFC. As a result, you should always specify host names using lowercase letters.

`android:port`

The port part of a URI authority. This attribute is meaningful only if the `scheme` and `host` attributes are also specified for the filter.

`android:path`

`android:pathPrefix`

`android:pathPattern`

The path part of a URI which must begin with a /. The `path` attribute specifies a complete path that is matched against the complete path in an Intent object. The `pathPrefix` attribute specifies a partial path that is matched against only the initial part of the path in the Intent object. The `pathPattern` attribute specifies a complete path that is matched against the complete path in the Intent object, but it can contain the following wildcards:

- An asterisk ('*') matches a sequence of 0 to many occurrences of the immediately preceding character.
- A period followed by an asterisk (".*") matches any sequence of 0 to many characters.

Because '\\' is used as an escape character when the string is read from XML (before it is parsed as a pattern), you will need to double-escape: For example, a literal '*' would be written as "\\\\"*\" and a literal '\\' would be written as "\\\\\\". This is basically the same as what you would need to write if constructing the string in Java code.

For more information on these three types of patterns, see the descriptions of `PATTERN_LITERAL`, `PATTERN_PREFIX`, and `PATTERN_SIMPLE_GLOB` in the `PatternMatcher` class.

These attributes are meaningful only if the `scheme` and `host` attributes are also specified for the filter.

`android:mimeType`

A MIME media type, such as `image/jpeg` or `audio/mpeg4-generic`. The subtype can be the asterisk wildcard (*) to indicate that any subtype matches.

It's common for an intent filter to declare a `<data>` that includes only the `android:mimeType` attribute.

Note: MIME type matching in the Android framework is case-sensitive, unlike formal RFC MIME types. As a result, you should always specify MIME types using lowercase letters.

INTRODUCED IN:

API Level 1

SEE ALSO:

<action>

<category>



<grant-uri-permission>

SYNTAX:

```
<grant-uri-permission android:path="string"  
                      android:pathPattern="string"  
                      android:pathPrefix="string" />
```

CONTAINED IN:

```
<provider>
```

DESCRIPTION:

Specifies which data subsets of the parent content provider permission can be granted for. Data subsets are indicated by the path part of a **content:** URI. (The authority part of the URI identifies the content provider.) Granting permission is a way of enabling clients of the provider that don't normally have permission to access its data to overcome that restriction on a one-time basis.

If a content provider's **grantUriPermissions** attribute is "**true**", permission can be granted for any the data under the provider's purview. However, if that attribute is "**false**", permission can be granted only to data subsets that are specified by this element. A provider can contain any number of **<grant-uri-permission>** elements. Each one can specify only one path (only one of the three possible attributes).

For information on how permission is granted, see the **<intent-filter>** element's **grantUriPermissions** attribute.

ATTRIBUTES:

```
android:path  
android:pathPrefix  
android:pathPattern
```

A path identifying the data subset or subsets that permission can be granted for. The **path** attribute specifies a complete path; permission can be granted only to the particular data subset identified by that path. The **pathPrefix** attribute specifies the initial part of a path; permission can be granted to all data subsets with paths that share that initial part. The **pathPattern** attribute specifies a complete path, but one that can contain the following wildcards:

- An asterisk ("*") matches a sequence of 0 to many occurrences of the immediately preceding character.
- A period followed by an asterisk (".*") matches any sequence of 0 to many characters.

Because '\' is used as an escape character when the string is read from XML (before it is parsed as a pattern), you will need to double-escape: For example, a literal '*' would be written as "*" and a literal '\' would be written as "\\\". This is basically the same as what you would need to write if constructing the string in Java code.

For more information on these types of patterns, see the descriptions of **PATTERN_LITERAL**, **PATTERN_PREFIX**, and **PATTERN_SIMPLE_GLOB** in the **PatternMatcher** class.

INTRODUCED IN:

API Level 1

SEE ALSO:

the **grantUriPermissions** attribute of the **<provider>** element



<instrumentation>

SYNTAX:

```
<instrumentation android:functionalTest=["true" | "false"]
                 android:handleProfiling=["true" | "false"]
                 android:icon="drawable resource"
                 android:label="string resource"
                 android:name="string"
                 android:targetPackage="string" />
```

CONTAINED IN:

<[manifest](#)>

DESCRIPTION:

Declares an [Instrumentation](#) class that enables you to monitor an application's interaction with the system. The Instrumentation object is instantiated before any of the application's components.

ATTRIBUTES:

android:functionalTest

Whether or not the Instrumentation class should run as a functional test — "true" if it should, and "false" if not. The default value is "false".

android:handleProfiling

Whether or not the Instrumentation object will turn profiling on and off — "true" if it determines when profiling starts and stops, and "false" if profiling continues the entire time it is running. A value of "true" enables the object to target profiling at a specific set of operations. The default value is "false".

android:icon

An icon that represents the Instrumentation class. This attribute must be set as a reference to a drawable resource.

android:label

A user-readable label for the Instrumentation class. The label can be set as a raw string or a reference to a string resource.

android:name

The name of the [Instrumentation](#) subclass. This should be a fully qualified class name (such as, "com.example.project.StringInstrumentation"). However, as a shorthand, if the first character of the name is a period, it is appended to the package name specified in the <[manifest](#)> element.

There is no default. The name must be specified.

android:targetPackage

The application that the Instrumentation object will run against. An application is identified by the package name assigned in its manifest file by the <[manifest](#)> element.

INTRODUCED IN:

API Level 1

<intent-filter>

SYNTAX:

```
<intent-filter android:icon="drawable resource"
              android:label="string resource"
              android:priority="integer" >
    ...
</intent-filter>
```

CONTAINED IN:

```
<activity>
<activity-alias>
<service>
<receiver>
```

MUST CONTAIN:

```
<action>
```

CAN CONTAIN:

```
<category>
<data>
```

DESCRIPTION:

Specifies the types of intents that an activity, service, or broadcast receiver can respond to. An intent filter declares the capabilities of its parent component — what an activity or service can do and what types of broadcasts a receiver can handle. It opens the component to receiving intents of the advertised type, while filtering out those that are not meaningful for the component.

Most of the contents of the filter are described by its `<action>`, `<category>`, and `<data>` subelements.

For a more detailed discussion of filters, see the separate [Intents and Intent Filters](#) document, as well as the [Intents Filters](#) section in the introduction.

ATTRIBUTES:

android:icon

An icon that represents the parent activity, service, or broadcast receiver when that component is presented to the user as having the capability described by the filter.

This attribute must be set as a reference to a drawable resource containing the image definition. The default value is the icon set by the parent component's `icon` attribute. If the parent does not specify an icon, the default is the icon set by the `<application>` element.

For more on intent filter icons, see [Icons and Labels](#) in the introduction.

android:label

A user-readable label for the parent component. This label, rather than the one set by the parent component, is used when the component is presented to the user as having the capability described by the filter.

The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

The default value is the label set by the parent component. If the parent does not specify a label, the default is the label set by the `<application>` element's `label` attribute.

For more on intent filter labels, see [Icons and Labels](#) in the introduction.

`android:priority`

The priority that should be given to the parent component with regard to handling intents of the type described by the filter. This attribute has meaning for both activities and broadcast receivers:

- It provides information about how able an activity is to respond to an intent that matches the filter, relative to other activities that could also respond to the intent. When an intent could be handled by multiple activities with different priorities, Android will consider only those with higher priority values as potential targets for the intent.
- It controls the order in which broadcast receivers are executed to receive broadcast messages. Those with higher priority values are called before those with lower values. (The order applies only to synchronous messages; it's ignored for asynchronous messages.)

Use this attribute only if you really need to impose a specific order in which the broadcasts are received, or want to force Android to prefer one activity over others.

The value must be an integer, such as "`100`". Higher numbers have a higher priority. The default value is 0. The value must be greater than -1000 and less than 1000.

Also see [setPriority\(\)](#).

INTRODUCED IN:

API Level 1

SEE ALSO:

`<action>`
`<category>`
`<data>`

<manifest>

SYNTAX:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="string"
    android:sharedUserId="string"
    android:sharedUserLabel="string resource"
    android:versionCode="integer"
    android:versionName="string"
    android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
    .
    .
</manifest>
```

CONTAINED IN:

none

MUST CONTAIN:

```
<application>
```

CAN CONTAIN:

```
<compatible-screens>
<instrumentation>
<permission>
<permission-group>
<permission-tree>
<supports-gl-texture>
<supports-screens>
<uses-configuration>
<uses-feature>
<uses-permission>
<uses-permission-sdk-23>
<uses-sdk>
```

DESCRIPTION:

The root element of the AndroidManifest.xml file. It must contain an `<application>` element and specify `xmlns:android` and `package` attributes.

ATTRIBUTES:

`xmlns:android`

Defines the Android namespace. This attribute should always be set to "`http://schemas.android.com/apk/res/android`".

`package`

A full Java-language-style package name for the Android application. The name should be unique. The name may contain uppercase or lowercase letters ('A' through 'Z'), numbers, and underscores ('_'). However, individual package name parts may only start with letters.

To avoid conflicts with other developers, you should use Internet domain ownership as the basis for your package names (in reverse). For example, applications published by Google start with `com.google`. You should also never use the `com.example` namespace when publishing your applications.

The package name serves as a unique identifier for the application. It's also the default name for the application process (see the `<application>` element's `process` process attribute) and the default task affinity of an activity (see the `<activity>` element's `taskAffinity` attribute).

Caution: Once you publish your application, you **cannot change the package name**. The package name defines your application's identity, so if you change it, then it is considered to be a different application and users of the previous version cannot update to the new version.

`android:sharedUserId`

The name of a Linux user ID that will be shared with other applications. By default, Android assigns each application its own unique user ID. However, if this attribute is set to the same value for two or more applications, they will all share the same ID — provided that their certificate sets are identical. Applications with the same user ID can access each other's data and, if desired, run in the same process.

`android:sharedUserLabel`

A user-readable label for the shared user ID. The label must be set as a reference to a string resource; it cannot be a raw string.

This attribute was introduced in API Level 3. It is meaningful only if the `sharedUserId` attribute is also set.

`android:versionCode`

An internal version number. This number is used only to determine whether one version is more recent than another, with higher numbers indicating more recent versions. This is not the version number shown to users; that number is set by the `versionName` attribute.

The value must be set as an integer, such as "100". You can define it however you want, as long as each successive version has a higher number. For example, it could be a build number. Or you could translate a version number in "x.y" format to an integer by encoding the "x" and "y" separately in the lower and upper 16 bits. Or you could simply increase the number by one each time a new version is released.

`android:versionName`

The version number shown to users. This attribute can be set as a raw string or as a reference to a string resource. The string has no other purpose than to be displayed to users. The `versionCode` attribute holds the significant version number used internally.

`android:installLocation`

The default install location for the application.

The following keyword strings are accepted:

Value	Description
<code>"internalOnly"</code>	The application must be installed on the internal device storage only. If this is set, the application will never be installed on the external storage. If the internal storage is full, then the system will not install the application. This is also the default behavior if you do not define <code>android:installLocation</code> .
<code>"auto"</code>	The application may be installed on the external storage, but the system will install the application on the internal storage by default. If the internal storage is full, then the system will install it on the external storage. Once installed, the user can move the application to either internal or external storage through the system settings.
<code>"preferExternal"</code>	The application prefers to be installed on the external storage (SD card). There is no guarantee that the system will honor this request. The application might be installed on internal storage if the

external media is unavailable or full. Once installed, the user can move the application to either internal or external storage through the system settings.

Note: By default, your application will be installed on the internal storage and cannot be installed on the external storage unless you define this attribute to be either "auto" or "preferExternal".

When an application is installed on the external storage:

- The `.apk` file is saved to the external storage, but any application data (such as databases) is still saved on the internal device memory.
- The container in which the `.apk` file is saved is encrypted with a key that allows the application to operate only on the device that installed it. (A user cannot transfer the SD card to another device and use applications installed on the card.) Though, multiple SD cards can be used with the same device.
- At the user's request, the application can be moved to the internal storage.

The user may also request to move an application from the internal storage to the external storage. However, the system will not allow the user to move the application to external storage if this attribute is set to `internalOnly`, which is the default setting.

Read [App Install Location](#) for more information about using this attribute (including how to maintain backward compatibility).

Introduced in: API Level 8.

INTRODUCED IN:

API Level 1 for all attributes, unless noted otherwise in the attribute description.

SEE ALSO:

[`<application>`](#)



<meta-data>

SYNTAX:

```
<meta-data android:name="string"  
          android:resource="resource specification"  
          android:value="string" />
```

CONTAINED IN:

```
<activity>  
<activity-alias>  
<application>  
<provider>  
<receiver>  
<service>
```

DESCRIPTION:

A name-value pair for an item of additional, arbitrary data that can be supplied to the parent component. A component element can contain any number of <meta-data> subelements. The values from all of them are collected in a single [Bundle](#) object and made available to the component as the [PackageItemInfo.metaData](#) field.

Ordinary values are specified through the [value](#) attribute. However, to assign a resource ID as the value, use the [resource](#) attribute instead. For example, the following code assigns whatever value is stored in the [@string/kangaroo](#) resource to the "zoo" name:

```
<meta-data android:name="zoo" android:value="@string/kangaroo" />
```

On the other hand, using the [resource](#) attribute would assign "zoo" the numeric ID of the resource, not the value stored in the resource:

```
<meta-data android:name="zoo" android:resource="@string/kangaroo" />
```

It is highly recommended that you avoid supplying related data as multiple separate <meta-data> entries. Instead, if you have complex data to associate with a component, store it as a resource and use the [resource](#) attribute to inform the component of its ID.

ATTRIBUTES:

android:name

A unique name for the item. To ensure that the name is unique, use a Java-style naming convention — for example, "com.example.project.activity.fred".

android:resource

A reference to a resource. The ID of the resource is the value assigned to the item. The ID can be retrieved from the meta-data Bundle by the [Bundle.getInt\(\)](#) method.

android:value

The value assigned to the item. The data types that can be assigned as values and the [Bundle](#) methods that components use to

retrieve those values are listed in the following table:

Type	Bundle method
String value, using double backslashes (\\\) to escape characters — such as "\\n" and "\\uxxxx" for a Unicode character.	<code>getString()</code>
Integer value, such as "100"	<code>getInt()</code>
Boolean value, either "true" or "false"	<code>getBoolean()</code>
Color value, in the form "#rgb", "#argb", "#rrggbb", or "#aarrggbb"	<code>getInt()</code>
Float value, such as "1.23"	<code>getFloat()</code>

INTRODUCED IN:

API Level 1



<path-permission>

SYNTAX:

```
<path-permission android:path="string"  
                 android:pathPrefix="string"  
                 android:pathPattern="string"  
                 android:permission="string"  
                 android:readPermission="string"  
                 android:writePermission="string" />
```

CONTAINED IN:

[`<provider>`](#)

DESCRIPTION:

Defines the path and required permissions for a specific subset of data within a content provider. This element can be specified multiple times to supply multiple paths.

ATTRIBUTES:

`android:path`

A complete URI path for a subset of content provider data. Permission can be granted only to the particular data identified by this path. When used to provide search suggestion content, it must be appended with "/search_suggest_query".

`android:pathPrefix`

The initial part of a URI path for a subset of content provider data. Permission can be granted to all data subsets with paths that share this initial part.

`android:pathPattern`

A complete URI path for a subset of content provider data, but one that can use the following wildcards:

- An asterisk ('*'). This matches a sequence of 0 to many occurrences of the immediately preceding character.
- A period followed by an asterisk (".*"). This matches any sequence of 0 or more characters.

Because '\' is used as an escape character when the string is read from XML (before it is parsed as a pattern), you will need to double-escape. For example, a literal '*' would be written as "*" and a literal '\' would be written as "\\\". This is basically the same as what you would need to write if constructing the string in Java code.

For more information on these types of patterns, see the descriptions of [PATTERN_LITERAL](#), [PATTERN_PREFIX](#), and [PATTERN_SIMPLE_GLOB](#) in the [PatternMatcher](#) class.

`android:permission`

The name of a permission that clients must have in order to read or write the content provider's data. This attribute is a convenient way of setting a single permission for both reading and writing. However, the `readPermission` and `writePermission` attributes take precedence over this one.

`android:readPermission`

A permission that clients must have in order to query the content provider.

`android:writePermission`

A permission that clients must have in order to make changes to the data controlled by the content provider.

INTRODUCED IN:

API Level 4

SEE ALSO:

[SearchManager](#)

[Manifest.permission](#)

[Security and Permissions](#)

<permission>

SYNTAX:

```
<permission android:description="string resource"  
           android:icon="drawable resource"  
           android:label="string resource"  
           android:name="string"  
           android:permissionGroup="string"  
           android:protectionLevel=[ "normal" | "dangerous" |  
                                "signature" | "signatureOrSystem" ] />
```

CONTAINED IN:

<manifest>

DESCRIPTION:

Declares a security permission that can be used to limit access to specific components or features of this or other applications. See the [Permissions](#) section in the introduction, and the [Security and Permissions](#) document for more information on how permissions work.

ATTRIBUTES:

android:description

A user-readable description of the permission, longer and more informative than the label. It may be displayed to explain the permission to the user — for example, when the user is asked whether to grant the permission to another application.

This attribute must be set as a reference to a string resource; unlike the **label** attribute, it cannot be a raw string.

android:icon

A reference to a drawable resource for an icon that represents the permission.

android:label

A name for the permission, one that can be displayed to users.

As a convenience, the label can be directly set as a raw string while you're developing the application. However, when the application is ready to be published, it should be set as a reference to a string resource, so that it can be localized like other strings in the user interface.

android:name

The name of the permission. This is the name that will be used in code to refer to the permission — for example, in a <[uses-permission](#)> element and the **permission** attributes of application components.

Note: The system does not allow multiple packages to declare a permission with the same name, unless all the packages are signed with the same certificate. If a package declares a permission, the system does not permit the user to install other packages with the same permission name, unless those packages are signed with the same certificate as the first package. To avoid naming collisions, we recommend using reverse-domain-style naming for custom permissions, for example `com.example.myapp.ENGAGE_HYPERSPACE`.

android:permissionGroup

Assigns this permission to a group. The value of this attribute is the name of the group, which must be declared with the [`<permission-group>`](#) element in this or another application. If this attribute is not set, the permission does not belong to a group.

`android:protectionLevel`

Characterizes the potential risk implied in the permission and indicates the procedure the system should follow when determining whether or not to grant the permission to an application requesting it. The value can be set to one of the following strings:

Value	Meaning
<code>"normal"</code>	The default value. A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval (though the user always has the option to review these permissions before installing).
<code>"dangerous"</code>	A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities.
<code>"signature"</code>	A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.
<code>"signatureOrSystem"</code>	A permission that the system grants only to applications that are in the Android system image <i>or</i> that are signed with the same certificate as the application that declared the permission. Please avoid using this option, as the <code>signature</code> protection level should be sufficient for most needs and works regardless of exactly where applications are installed. The " <code>signatureOrSystem</code> " permission is used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.

INTRODUCED IN:

API Level 1

SEE ALSO:

[`<uses-permission>`](#)
[`<permission-tree>`](#)
[`<permission-group>`](#)



<permission-group>

SYNTAX:

```
<permission-group android:description="string resource"
                  android:icon="drawable resource"
                  android:label="string resource"
                  android:name="string" />
```

CONTAINED IN:

<manifest>

DESCRIPTION:

Declares a name for a logical grouping of related permissions. Individual permission join the group through the [permissionGroup](#) attribute of the [<permission>](#) element. Members of a group are presented together in the user interface.

Note that this element does not declare a permission itself, only a category in which permissions can be placed. See the [<permission>](#) element for information on declaring permissions and assigning them to groups.

ATTRIBUTES:

[android:description](#)

User-readable text that describes the group. The text should be longer and more explanatory than the label. This attribute must be set as a reference to a string resource. Unlike the [label](#) attribute, it cannot be a raw string.

[android:icon](#)

An icon representing the permission. This attribute must be set as a reference to a drawable resource containing the image definition.

[android:label](#)

A user-readable name for the group. As a convenience, the label can be directly set as a raw string while you're developing the application. However, when the application is ready to be published, it should be set as a reference to a string resource, so that it can be localized like other strings in the user interface.

[android:name](#)

The name of the group. This is the name that can be assigned to a [<permission>](#) element's [<permissionGroup>](#) attribute.

INTRODUCED IN:

API Level 1

SEE ALSO:

[<permission>](#)
[<permission-tree>](#)
[<uses-permission>](#)

<permission-tree>

SYNTAX:

```
<permission-tree android:icon="drawable resource"
                 android:label="string resource" ]
                 android:name="string" />
```

CONTAINED IN:

```
<manifest>
```

DESCRIPTION:

Declares the base name for a tree of permissions. The application takes ownership of all names within the tree. It can dynamically add new permissions to the tree by calling [PackageManager.addPermission\(\)](#). Names within the tree are separated by periods ('.'). For example, if the base name is `com.example.project.taxes`, permissions like the following might be added:

```
com.example.project.taxes.CALCULATE
com.example.project.taxes.deductions.MAKE_SOME_UP
com.example.project.taxes.deductions.EXAGGERATE
```

Note that this element does not declare a permission itself, only a namespace in which further permissions can be placed. See the [<permission>](#) element for information on declaring permissions.

ATTRIBUTES:

`android:icon`

An icon representing all the permissions in the tree. This attribute must be set as a reference to a drawable resource containing the image definition.

`android:label`

A user-readable name for the group. As a convenience, the label can be directly set as a raw string for quick and dirty programming. However, when the application is ready to be published, it should be set as a reference to a string resource, so that it can be localized like other strings in the user interface.

`android:name`

The name that's at the base of the permission tree. It serves as a prefix to all permission names in the tree. Java-style scoping should be used to ensure that the name is unique. The name must have more than two period-separated segments in its path — for example, `com.example.base` is OK, but `com.example` is not.

INTRODUCED IN:

API Level 1

SEE ALSO:

```
<permission>
<permission-group>
<uses-permission>
```

<provider>

SYNTAX:

```
<provider android:authorities="list"
    android:directBootAware=["true" | "false"]
    android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:grantUriPermissions=["true" | "false"]
    android:icon="drawable resource"
    android:initOrder="integer"
    android:label="string resource"
    android:multiprocess=["true" | "false"]
    android:name="string"
    android:permission="string"
    android:process="string"
    android:readPermission="string"
    android:syncable=["true" | "false"]
    android:writePermission="string" >
    .
    .
</provider>
```

CONTAINED IN:

```
<application>
```

CAN CONTAIN:

```
<meta-data>
<grant-uri-permission>
<path-permission>
```

DESCRIPTION:

Declares a content provider component. A content provider is a subclass of [ContentProvider](#) that supplies structured access to data managed by the application. All content providers in your application must be defined in a [`<provider>`](#) element in the manifest file; otherwise, the system is unaware of them and doesn't run them.

You only declare content providers that are part of your application. Content providers in other applications that you use in your application should not be declared.

The Android system stores references to content providers according to an **authority** string, part of the provider's **content URI**. For example, suppose you want to access a content provider that stores information about health care professionals. To do this, you call the method [ContentResolver.query\(\)](#), which among other arguments takes a URI that identifies the provider:

```
content://com.example.project.healthcareprovider/nurses/rn
```

The **content: scheme** identifies the URI as a content URI pointing to an Android content provider. The authority [com.example.project.healthcareprovider](#) identifies the provider itself; the Android system looks up the authority in its list of known providers and their authorities. The substring [nurses/rn](#) is a **path**, which the content provider can use to identify subsets of the provider data.

Notice that when you define your provider in the [`<provider>`](#) element, you don't include the scheme or the path in the [android:name](#) argument, only the authority.

For information on using and developing content providers, see the API Guide, [Content Providers](#).

ATTRIBUTES:

`android:authorities`

A list of one or more URI authorities that identify data offered by the content provider. Multiple authorities are listed by separating their names with a semicolon. To avoid conflicts, authority names should use a Java-style naming convention (such as `com.example.provider.cartoonprovider`). Typically, it's the name of the [ContentProvider](#) subclass that implements the provider

There is no default. At least one authority must be specified.

`android:enabled`

Whether or not the content provider can be instantiated by the system — "`true`" if it can be, and "`false`" if not. The default value is "`true`".

The `<application>` element has its own `enabled` attribute that applies to all application components, including content providers. The `<application>` and `<provider>` attributes must both be "`true`" (as they both are by default) for the content provider to be enabled. If either is "`false`", the provider is disabled; it cannot be instantiated.

`android:directBootAware`

Whether or not the content provider is *direct-boot aware*; that is, whether or not it can run before the user unlocks the device.

Note: During [Direct Boot](#), a content provider in your application can only access the data that is stored in *device protected* storage.

The default value is "`false`".

`android:exported`

Whether the content provider is available for other applications to use:

- `true`: The provider is available to other applications. Any application can use the provider's content URI to access it, subject to the permissions specified for the provider.
- `false`: The provider is not available to other applications. Set `android:exported="false"` to limit access to the provider to your applications. Only applications that have the same user ID (UID) as the provider will have access to it.

Because this attribute was introduced in API level 17, all devices running API level 16 and lower behave as though this attribute is set "`true`". If you set `android:targetSdkVersion` to 17 or higher, then the default value is "`false`" for devices running API level 17 and higher.

You can set `android:exported="false"` and still limit access to your provider by setting permissions with the `permission` attribute.

`android:grantUriPermissions`

Whether or not those who ordinarily would not have permission to access the content provider's data can be granted permission to do so, temporarily overcoming the restriction imposed by the `readPermission`, `writePermission`, and `permission` attributes — "`true`" if permission can be granted, and "`false`" if not. If "`true`", permission can be granted to any of the content provider's data. If "`false`", permission can be granted only to the data subsets listed in `<grant-uri-permission>` subelements, if any. The default value is "`false`".

Granting permission is a way of giving an application component one-time access to data protected by a permission. For example, when an e-mail message contains an attachment, the mail application may call upon the appropriate viewer to open it, even though the viewer doesn't have general permission to look at all the content provider's data.

In such cases, permission is granted by `FLAG_GRANT_READ_URI_PERMISSION` and `FLAG_GRANT_WRITE_URI_PERMISSION` flags

in the Intent object that activates the component. For example, the mail application might put `FLAG_GRANT_READ_URI_PERMISSION` in the Intent passed to `Context.startActivity()`. The permission is specific to the URI in the Intent.

If you enable this feature, either by setting this attribute to "true" or by defining `<grant-uri-permission>` subelements, you must call `Context.revokeUriPermission()` when a covered URI is deleted from the provider.

See also the `<grant-uri-permission>` element.

`android:icon`

An icon representing the content provider. This attribute must be set as a reference to a drawable resource containing the image definition. If it is not set, the icon specified for the application as a whole is used instead (see the `<application>` element's `icon` attribute).

`android:initOrder`

The order in which the content provider should be instantiated, relative to other content providers hosted by the same process. When there are dependencies among content providers, setting this attribute for each of them ensures that they are created in the order required by those dependencies. The value is a simple integer, with higher numbers being initialized first.

`android:label`

A user-readable label for the content provided. If this attribute is not set, the label set for the application as a whole is used instead (see the `<application>` element's `label` attribute).

The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

`android:multiprocess`

If the app runs in multiple processes, this attribute determines whether multiple instances of the content provider are created. If `true`, each of the app's processes has its own content provider object. If `false`, the app's processes share only one content provider object. The default value is `false`.

Setting this flag to `true` may improve performance by reducing the overhead of interprocess communication, but it also increases the memory footprint of each process.

`android:name`

The name of the class that implements the content provider, a subclass of `ContentProvider`. This should be a fully qualified class name (such as, "`com.example.project.TransportationProvider`"). However, as a shorthand, if the first character of the name is a period, it is appended to the package name specified in the `<manifest>` element.

There is no default. The name must be specified.

`android:permission`

The name of a permission that clients must have to read or write the content provider's data. This attribute is a convenient way of setting a single permission for both reading and writing. However, the `readPermission` and `writePermission` attributes take precedence over this one. If the `readPermission` attribute is also set, it controls access for querying the content provider. And if the `writePermission` attribute is set, it controls access for modifying the provider's data.

For more information on permissions, see the [Permissions](#) section in the introduction and a separate document, [Security and Permissions](#).

`android:process`

The name of the process in which the content provider should run. Normally, all components of an application run in the default process created for the application. It has the same name as the application package. The `<application>` element's `process` attribute can set a different default for all components. But each component can override the default with its own `process`

attribute, allowing you to spread your application across multiple processes.

If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, is created when it's needed and the activity runs in that process. If the process name begins with a lowercase character, the activity will run in a global process of that name, provided that it has permission to do so. This allows components in different applications to share a process, reducing resource usage.

`android:readPermission`

A permission that clients must have to query the content provider. See also the [permission](#) and [writePermission](#) attributes.

`android:syncable`

Whether or not the data under the content provider's control is to be synchronized with data on a server — "`true`" if it is to be synchronized, and "`false`" if not.

`android:writePermission`

A permission that clients must have to make changes to the data controlled by the content provider. See also the [permission](#) and [readPermission](#) attributes.

INTRODUCED IN:

API Level 1

SEE ALSO:

[Content Providers](#)

<receiver>

SYNTAX:

```
<receiver android:directBootAware=["true" | "false"]
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    ...
</receiver>
```

CONTAINED IN:

[<application>](#)

CAN CONTAIN:

[<intent-filter>](#)
[<meta-data>](#)

DESCRIPTION:

Declares a broadcast receiver (a [BroadcastReceiver](#) subclass) as one of the application's components. Broadcast receivers enable applications to receive intents that are broadcast by the system or by other applications, even when other components of the application are not running.

There are two ways to make a broadcast receiver known to the system: One is declare it in the manifest file with this element. The other is to create the receiver dynamically in code and register it with the [Context.registerReceiver\(\)](#) method. For more information about how to dynamically create receivers, see the [BroadcastReceiver](#) class description.

Warning: Limit how many broadcast receivers you set in your app. Having too many broadcast receivers can affect your app's performance and the battery life of users' devices. For more information about APIs you can use instead of the [BroadcastReceiver](#) class for scheduling background work, see [Background Optimizations](#).

ATTRIBUTES:

android:directBootAware

Whether or not the broadcast receiver is *direct-boot aware*; that is, whether or not it can run before the user unlocks the device.

Note: During [Direct Boot](#), a broadcast receiver in your application can only access the data that is stored in *device protected* storage.

The default value is "[false](#)".

android:enabled

Whether or not the broadcast receiver can be instantiated by the system — "[true](#)" if it can be, and "[false](#)" if not. The default value is "[true](#)".

The `<application>` element has its own `enabled` attribute that applies to all application components, including broadcast receivers. The `<application>` and `<receiver>` attributes must both be "`true`" for the broadcast receiver to be enabled. If either is "`false`", it is disabled; it cannot be instantiated.

`android:exported`

Whether or not the broadcast receiver can receive messages from sources outside its application — "`true`" if it can, and "`false`" if not. If "`false`", the only messages the broadcast receiver can receive are those sent by components of the same application or applications with the same user ID.

The default value depends on whether the broadcast receiver contains intent filters. The absence of any filters means that it can be invoked only by Intent objects that specify its exact class name. This implies that the receiver is intended only for application-internal use (since others would not normally know the class name). So in this case, the default value is "`false`". On the other hand, the presence of at least one filter implies that the broadcast receiver is intended to receive intents broadcast by the system or other applications, so the default value is "`true`".

This attribute is not the only way to limit a broadcast receiver's external exposure. You can also use a permission to limit the external entities that can send it messages (see the `permission` attribute).

`android:icon`

An icon representing the broadcast receiver. This attribute must be set as a reference to a drawable resource containing the image definition. If it is not set, the icon specified for the application as a whole is used instead (see the `<application>` element's `icon` attribute).

The broadcast receiver's icon — whether set here or by the `<application>` element — is also the default icon for all the receiver's intent filters (see the `<intent-filter>` element's `icon` attribute).

`android:label`

A user-readable label for the broadcast receiver. If this attribute is not set, the label set for the application as a whole is used instead (see the `<application>` element's `label` attribute).

The broadcast receiver's label — whether set here or by the `<application>` element — is also the default label for all the receiver's intent filters (see the `<intent-filter>` element's `label` attribute).

The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

`android:name`

The name of the class that implements the broadcast receiver, a subclass of `BroadcastReceiver`. This should be a fully qualified class name (such as, "`com.example.project.ReportReceiver`"). However, as a shorthand, if the first character of the name is a period (for example, ". `ReportReceiver`"), it is appended to the package name specified in the `<manifest>` element.

Once you publish your application, you [should not change this name](#) (unless you've set `android:exported="false"`).

There is no default. The name must be specified.

`android:permission`

The name of a permission that broadcasters must have to send a message to the broadcast receiver. If this attribute is not set, the permission set by the `<application>` element's `permission` attribute applies to the broadcast receiver. If neither attribute is set, the receiver is not protected by a permission.

For more information on permissions, see the [Permissions](#) section in the introduction and a separate document, [Security and Permissions](#).

`android:process`

The name of the process in which the broadcast receiver should run. Normally, all components of an application run in the

default process created for the application. It has the same name as the application package. The [`<application>`](#) element's **process** attribute can set a different default for all components. But each component can override the default with its own **process** attribute, allowing you to spread your application across multiple processes.

If the name assigned to this attribute begins with a colon (`:`), a new process, private to the application, is created when it's needed and the broadcast receiver runs in that process. If the process name begins with a lowercase character, the receiver will run in a global process of that name, provided that it has permission to do so. This allows components in different applications to share a process, reducing resource usage.

INTRODUCED IN:

API Level 1



<service>

SYNTAX:

```
<service android:description="string resource"  
        android:directBootAware=["true" | "false"]  
        android:enabled=["true" | "false"]  
        android:exported=["true" | "false"]  
        android:icon="drawable resource"  
        android:isolatedProcess=["true" | "false"]  
        android:label="string resource"  
        android:name="string"  
        android:permission="string"  
        android:process="string" >  
  
    . . .  
</service>
```

CONTAINED IN:

```
<application>
```

CAN CONTAIN:

```
<intent-filter>  
<meta-data>
```

DESCRIPTION:

Declares a service (a [Service](#) subclass) as one of the application's components. Unlike activities, services lack a visual user interface. They're used to implement long-running background operations or a rich communications API that can be called by other applications.

All services must be represented by `<service>` elements in the manifest file. Any that are not declared there will not be seen by the system and will never be run.

ATTRIBUTES:

android:description

A string that describes the service to users. The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface.

android:directBootAware

Whether or not the service is *direct-boot aware*; that is, whether or not it can run before the user unlocks the device.

Note: During [Direct Boot](#), a service in your application can only access the data that is stored in *device protected* storage.

The default value is "**false**".

android:enabled

Whether or not the service can be instantiated by the system — "**true**" if it can be, and "**false**" if not. The default value is "**true**".

The `<application>` element has its own [enabled](#) attribute that applies to all application components, including services. The

`<application>` and `<service>` attributes must both be `"true"` (as they both are by default) for the service to be enabled. If either is `"false"`, the service is disabled; it cannot be instantiated.

android:exported

Whether or not components of other applications can invoke the service or interact with it — `"true"` if they can, and `"false"` if not. When the value is `"false"`, only components of the same application or applications with the same user ID can start the service or bind to it.

The default value depends on whether the service contains intent filters. The absence of any filters means that it can be invoked only by specifying its exact class name. This implies that the service is intended only for application-internal use (since others would not know the class name). So in this case, the default value is `"false"`. On the other hand, the presence of at least one filter implies that the service is intended for external use, so the default value is `"true"`.

This attribute is not the only way to limit the exposure of a service to other applications. You can also use a permission to limit the external entities that can interact with the service (see the `permission` attribute).

android:icon

An icon representing the service. This attribute must be set as a reference to a drawable resource containing the image definition. If it is not set, the icon specified for the application as a whole is used instead (see the `<application>` element's `icon` attribute).

The service's icon — whether set here or by the `<application>` element — is also the default icon for all the service's intent filters (see the `<intent-filter>` element's `icon` attribute).

android:isolatedProcess

If set to true, this service will run under a special process that is isolated from the rest of the system and has no permissions of its own. The only communication with it is through the Service API (binding and starting).

android:label

A name for the service that can be displayed to users. If this attribute is not set, the label set for the application as a whole is used instead (see the `<application>` element's `label` attribute).

The service's label — whether set here or by the `<application>` element — is also the default label for all the service's intent filters (see the `<intent-filter>` element's `label` attribute).

The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

android:name

The name of the `Service` subclass that implements the service. This should be a fully qualified class name (such as, `"com.example.project.RoomService"`). However, as a shorthand, if the first character of the name is a period (for example, `".RoomService"`), it is appended to the package name specified in the `<manifest>` element.

Once you publish your application, you `should not change this name` (unless you've set `android:exported="false"`).

There is no default. The name must be specified.

android:permission

The name of a permission that an entity must have in order to launch the service or bind to it. If a caller of `startService()`, `bindService()`, or `stopService()`, has not been granted this permission, the method will not work and the Intent object will not be delivered to the service.

If this attribute is not set, the permission set by the `<application>` element's `permission` attribute applies to the service. If neither attribute is set, the service is not protected by a permission.

For more information on permissions, see the [Permissions](#) section in the introduction and a separate document, [Security and Permissions](#).

[Permissions](#).

android:process

The name of the process where the service is to run. Normally, all components of an application run in the default process created for the application. It has the same name as the application package. The [`<application>`](#) element's `process` attribute can set a different default for all components. But component can override the default with its own `process` attribute, allowing you to spread your application across multiple processes.

If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, is created when it's needed and the service runs in that process. If the process name begins with a lowercase character, the service will run in a global process of that name, provided that it has permission to do so. This allows components in different applications to share a process, reducing resource usage.

SEE ALSO:

[`<application>`](#)
[`<activity>`](#)

INTRODUCED IN:

API Level 1

<supports-gl-texture>

SYNTAX:

```
<supports-gl-texture
    android:name="string" />
```

CONTAINED IN:

`<manifest>`

DESCRIPTION:

Declares a single GL texture compression format that is supported by the application.

An application "supports" a GL texture compression format if it is capable of providing texture assets that are compressed in that format, once the application is installed on a device. The application can provide the compressed assets locally, from inside the `.apk`, or it can download them from a server at runtime.

Each `<supports-gl-texture>` element declares exactly one supported texture compression format, specified as the value of a `android:name` attribute. If your application supports multiple texture compression formats, you can declare multiple `<supports-gl-texture>` elements. For example:

```
<supports-gl-texture android:name="GL_OES_compressed_ETC1_RGB8_texture" />
<supports-gl-texture android:name="GL_OES_compressed_paletted_texture" />
```

Declared `<supports-gl-texture>` elements are informational, meaning that the Android system itself does not examine the elements at install time to ensure matching support on the device. However, other services (such as Google Play) or applications can check your application's `<supports-gl-texture>` declarations as part of handling or interacting with your application. For this reason, it's very important that you declare all of the texture compression formats (from the list below) that your application is capable of supporting.

Applications and devices typically declare their supported GL texture compression formats using the same set of well-known strings, as listed below. The set of format strings may grow over time, as needed, and since the values are strings, applications are free to declare other formats as needed.

Assuming that the application is built with SDK Platform Tools r3 or higher, filtering based on the `<supports-gl-texture>` element is activated for all API levels.

ATTRIBUTES:

`android:name`

Specifies a single GL texture compression format supported by the application, as a descriptor string. Common descriptor values are listed in the table below.

Texture Compression Format Descriptor	Comments
<code>GL_OES_compressed_ETC1_RGB8_texture</code>	Ericsson texture compression. Specified in OpenGL ES 2.0 and available in all Android-powered devices that support OpenGL ES 2.0.
<code>GL_OES_compressed_paletted_texture</code>	Generic paletted texture compression.
<code>GL_AMD_compressed_3DC_texture</code>	ATI 3Dc texture compression.



Google Play Filtering

Google Play filters applications according to the texture compression formats that they support, to ensure that they can be installed only on devices that can handle their textures properly. You can use texture compression filtering as a way of targeting specific device types, based on GPU platform.

For important information about how Google Play uses `<supports-gl-texture>` elements as the basis for filtering, please read [Google Play and texture compression filtering](#), below.

<code>GL_AMD_compressed_ATC_texture</code>	ATI texture compression. Available on devices running Adreno GPU, including HTC Nexus One, Droid Incredible, EVO, and others. For widest compatibility, devices may also declare a <code><supports-gl-texture></code> element with the descriptor <code>GL_ATI_texture_compression_attc</code> .
<code>GL_EXT_texture_compression_latc</code>	Luminance alpha texture compression.
<code>GL_EXT_texture_compression_dxt1</code>	S3 DXT1 texture compression. Supported on devices running Nvidia Tegra2 platform, including Motorola Xoom, Motorola Atrix, Droid Bionic, and others.
<code>GL_EXT_texture_compression_s3tc</code>	S3 texture compression, nonspecific to DXT variant. Supported on devices running Nvidia Tegra2 platform, including Motorola Xoom, Motorola Atrix, Droid Bionic, and others. If your application requires a specific DXT variant, declare that descriptor instead of this one.
<code>GL_IMG_texture_compression_pvrtc</code>	PowerVR texture compression. Available in devices running PowerVR SGX530/540 GPU, such as Motorola DROID series; Samsung Galaxy S, Nexus S, and Galaxy Tab; and others.

SEE ALSO:

- [Filters on Google Play](#)

Google Play and texture compression filtering

Google Play filters the applications that are visible to users, so that users can see and download only those applications that are compatible with their devices. One of the ways it filters applications is by texture compression compatibility, giving you control over the availability of your application to various devices, based on the capabilities of their GPUs.

To determine an application's texture compression compatibility with a given user's device, Google Play compares:

- Texture compression formats that are supported by the application — an application declares its supported texture compression formats in `<supports-gl-texture>` elements in its manifest with...
- Texture compression formats that are supported by the GPU on the device — a device reports the formats it supports as read-only system properties.

Each time you upload an application to the Google Play Console, Google Play scans the application's manifest file and looks for any `<supports-gl-texture>` elements. It extracts the format descriptors from the elements and stores them internally as metadata associated with the application `.apk` and the application version.

When a user searches or browses for applications on Google Play, the service compares the texture compression formats supported by the application with those supported by the user's device. The comparison is based on the format descriptor strings and a match must be exact.

If *any* of an application's supported texture compression formats is also supported by the device, Google Play allows the user to see the application and potentially download it. Otherwise, if none of the application's formats is supported by the device, Google Play filters the application so that it is not available for download.

If an application does not declare any `<supports-gl-texture>` elements, Google Play does not apply any filtering based on GL texture compression format.



<supports-screens>

SYNTAX:

```
<supports-screens android:resizeable=["true" | "false"]
                  android:smallScreens=["true" | "false"]
                  android:normalScreens=["true" | "false"]
                  android:largeScreens=["true" | "false"]
                  android:xlargeScreens=["true" | "false"]
                  android:anyDensity=["true" | "false"]
                  android:requiresSmallestWidthDp="integer"
                  android:compatibleWidthLimitDp="integer"
                  android:largestWidthLimitDp="integer" />
```

CONTAINED IN:

<manifest>

DESCRIPTION:

Lets you specify the screen sizes your application supports and enable [screen compatibility mode](#) for screens larger than what your application supports. It's important that you always use this element in your application to specify the screen sizes your application supports.

An application "supports" a given screen size if it resizes properly to fill the entire screen. Normal resizing applied by the system works well for most applications and you don't have to do any extra work to make your application work on screens larger than a handset device. However, it's often important that you optimize your application's UI for different screen sizes by providing [alternative layout resources](#). For instance, you might want to modify the layout of an activity when it is on a tablet compared to when running on a handset device.

However, if your application does not work well when resized to fit different screen sizes, you can use the attributes of the [`<supports-screens>`](#) element to control whether your application should be distributed to smaller screens or have its UI scaled up ("zoomed") to fit larger screens using the system's [screen compatibility mode](#). When you have not designed for larger screen sizes and the normal resizing does not achieve the appropriate results, screen compatibility mode will scale your UI by emulating a *normal* size screen and medium density, then zooming in so that it fills the entire screen. Beware that this causes pixelation and blurring of your UI, so it's better if you optimize your UI for large screens.

Note: Android 3.2 introduces new attributes: `android:requiresSmallestWidthDp`, `android:compatibleWidthLimitDp`, and `android:largestWidthLimitDp`. If you're developing your application for Android 3.2 and higher, you should use these attributes to declare your screen size support, instead of the attributes based on generalized screen sizes.

For more information about how to properly support different screen sizes so that you can avoid using screen compatibility mode with your application, read [Supporting Multiple Screens](#).

ATTRIBUTES:

`android:resizeable`

Indicates whether the application is resizeable for different screen sizes. This attribute is true, by default. If set false, the system will run your application in [screen compatibility mode](#) on large screens.

This attribute is deprecated. It was introduced to help applications transition from Android 1.5 to 1.6, when support for multiple screens was first introduced. You should not use it.

`android:smallScreens`

Indicates whether the application supports smaller screen form-factors. A small screen is defined as one with a smaller aspect ratio than the "normal" (traditional HVGA) screen. An application that does not support small screens *will not be available* for small screen devices from external services (such as Google Play), because there is little the platform can do to make such an application work on a smaller screen. This is "`true`" by default.

`android:normalScreens`

Indicates whether an application supports the "normal" screen form-factors. Traditionally this is an HVGA medium density screen, but WQVGA low density and WVGA high density are also considered to be normal. This attribute is "true" by default.

`android:largeScreens`

Indicates whether the application supports larger screen form-factors. A large screen is defined as a screen that is significantly larger than a "normal" handset screen, and thus might require some special care on the application's part to make good use of it, though it may rely on resizing by the system to fill the screen.

The default value for this actually varies between some versions, so it's better if you explicitly declare this attribute at all times. Beware that setting it "false" will generally enable [screen compatibility mode](#).

`android:xlargeScreens`

Indicates whether the application supports extra large screen form-factors. An xlarge screen is defined as a screen that is significantly larger than a "large" screen, such as a tablet (or something larger) and may require special care on the application's part to make good use of it, though it may rely on resizing by the system to fill the screen.

The default value for this actually varies between some versions, so it's better if you explicitly declare this attribute at all times. Beware that setting it "false" will generally enable [screen compatibility mode](#).

This attribute was introduced in API level 9.

`android:anyDensity`

Indicates whether the application includes resources to accommodate any screen density.

For applications that support Android 1.6 (API level 4) and higher, this is "true" by default and **you should not set it "false"** unless you're absolutely certain that it's necessary for your application to work. The only time it might be necessary to disable this is if your app directly manipulates bitmaps (see the [Supporting Multiple Screens](#) document for more information).

`android:requiresSmallestWidthDp`

Specifies the minimum `smallestWidth` required. The `smallestWidth` is the shortest dimension of the screen space (in `dp` units) that must be available to your application UI—that is, the shortest of the available screen's two dimensions. So, in order for a device to be considered compatible with your application, the device's `smallestWidth` must be equal to or greater than this value. (Usually, the value you supply for this is the "smallest width" that your layout supports, regardless of the screen's current orientation.)

For example, a typical handset screen has a `smallestWidth` of 320dp, a 7" tablet has a `smallestWidth` of 600dp, and a 10" tablet has a `smallestWidth` of 720dp. These values are generally the `smallestWidth` because they are the shortest dimension of the screen's available space.

The size against which your value is compared takes into account screen decorations and system UI. For example, if the device has some persistent UI elements on the display, the system declares the device's `smallestWidth` as one that is smaller than the actual screen size, accounting for these UI elements because those are screen pixels not available for your UI. Thus, the value you use should be the minimum width required by your layout, regardless of the screen's current orientation.

If your application properly resizes for smaller screen sizes (down to the `small` size or a minimum width of 320dp), you do not need to use this attribute. Otherwise, you should use a value for this attribute that matches the smallest value used by your application for the `smallest screen width qualifier` (`sw<N>dp`).

Caution: The Android system does not pay attention to this attribute, so it does not affect how your application behaves at runtime. Instead, it is used to enable filtering for your application on services such as Google Play. However, **Google Play currently does not support this attribute for filtering** (on Android 3.2), so you should continue using the other size attributes if your application does not support small screens.

This attribute was introduced in API level 13.

`android:compatibleWidthLimitDp`

This attribute allows you to enable [screen compatibility mode](#) as a user-optional feature by specifying the maximum "smallest screen width" for which your application is designed. If the smallest side of a device's available screen is greater than your value here, users can still install your application, but are offered to run it in screen compatibility mode. By default, screen compatibility mode is disabled and your layout is resized to fit the screen as usual, but a button is available in the system bar that allows the user to toggle screen compatibility mode on and off.

If your application is compatible with all screen sizes and its layout properly resizes, you do not need to use this attribute.

Note: Currently, screen compatibility mode emulates only handset screens with a 320dp width, so screen compatibility mode is not applied if your value for `android:compatibleWidthLimitDp` is larger than 320.

This attribute was introduced in API level 13.

`android:largestWidthLimitDp`

This attribute allows you to force-enable [screen compatibility mode](#) by specifying the maximum "smallest screen width" for which your application is designed. If the smallest side of a device's available screen is greater than your value here, the application runs in screen compatibility mode with no way for the user to disable it.

If your application is compatible with all screen sizes and its layout properly resizes, you do not need to use this attribute. Otherwise, you should first consider using the `android:compatibleWidthLimitDp` attribute. You should use the `android:largestWidthLimitDp` attribute only when your application is functionally broken when resized for larger screens and screen compatibility mode is the only way that users should use your application.

Note: Currently, screen compatibility mode emulates only handset screens with a 320dp width, so screen compatibility mode is not applied if your value for `android:largestWidthLimitDp` is larger than 320.

This attribute was introduced in API level 13.

INTRODUCED IN:

API Level 4

SEE ALSO:

- [Supporting Multiple Screens](#)
- [DisplayMetrics](#)



<uses-configuration>

SYNTAX:

```
<uses-configuration
    android:reqFiveWayNav=["true" | "false"]
    android:reqHardKeyboard=["true" | "false"]
    android:reqKeyboardType=["undefined" | "nokeys" | "qwerty" | "twelvekey"]
    android:reqNavigation=["undefined" | "nonav" | "dpad" | "trackball" | "wheel"]
    android:reqTouchScreen=["undefined" | "notouch" | "stylus" | "finger"] />
```

CONTAINED IN:

<manifest>

DESCRIPTION:

Indicates what hardware and software features the application requires. For example, an application might specify that it requires a physical keyboard or a particular navigation device, like a trackball. The specification is used to avoid installing the application on devices where it will not work.

Note: Most apps should not use this manifest tag. You should *always* support input with a directional pad (d-pad) in order to assist sight-impaired users and support devices that provide d-pad input in addition to or instead of touch. For information about how to support d-pad input in your app, read [Enabling Focus Navigation](#). If your app absolutely cannot function without a touchscreen, then instead use the <[uses-feature](#)> tag to declare the required touchscreen type, ranging from "android.hardware.faketouch" for basic touch-style events to more advanced touch types such as "android.hardware.touchscreen.multitouch.jazzhand" for distinct input from multiple fingers.

ATTRIBUTES:

android:reqFiveWayNav

Whether or not the application requires a five-way navigation control — "true" if it does, and "false" if not. A five-way control is one that can move the selection up, down, right, or left, and also provides a way of invoking the current selection. It could be a D-pad (directional pad), trackball, or other device.

If an application requires a directional control, but not a control of a particular type, it can set this attribute to "true" and ignore the [reqNavigation](#) attribute. However, if it requires a particular type of directional control, it can ignore this attribute and set [reqNavigation](#) instead.

android:reqHardKeyboard

Whether or not the application requires a hardware keyboard — "true" if it does, and "false" if not.

android:reqKeyboardType

The type of keyboard the application requires, if any at all. This attribute does not distinguish between hardware and software keyboards. If a hardware keyboard of a certain type is required, specify the type here and also set the [reqHardKeyboard](#) attribute to "true".

The value must be one of the following strings:

Value	Description
-------	-------------

<code>"undefined"</code>	The application does not require a keyboard. (A keyboard requirement is not defined.) This is the default value.
<code>"nokeys"</code>	The application does not require a keyboard.
<code>"qwerty"</code>	The application requires a standard QWERTY keyboard.
<code>"twelvekey"</code>	The application requires a twelve-key keypad, like those on most phones — with keys for the digits from <code>0</code> through <code>9</code> plus star (<code>*</code>) and pound (<code>#</code>) keys.

`android:reqNavigation`

The navigation device required by the application, if any. The value must be one of the following strings:

Value	Description
<code>"undefined"</code>	The application does not require any type of navigation control. (The navigation requirement is not defined.) This is the default value.
<code>"nonav"</code>	The application does not require a navigation control.
<code>"dpad"</code>	The application requires a D-pad (directional pad) for navigation.
<code>"trackball"</code>	The application requires a trackball for navigation.
<code>"wheel"</code>	The application requires a navigation wheel.

If an application requires a navigational control, but the exact type of control doesn't matter, it can set the `reqFiveWayNav` attribute to `"true"` rather than set this one.

`android:reqTouchScreen`

The type of touch screen the application requires, if any at all. The value must be one of the following strings:

Value	Description
<code>"undefined"</code>	The application doesn't require a touch screen. (The touch screen requirement is undefined.) This is the default value.
<code>"notouch"</code>	The application doesn't require a touch screen.
<code>"stylus"</code>	The application requires a touch screen that's operated with a stylus.
<code>"finger"</code>	The application requires a touch screen that can be operated with a finger. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> Note: If some type of touch input is required for your app, you should instead use the <code><uses-feature></code> tag to declare the required touchscreen type, beginning with <code>"android.hardware.faketouch"</code> for basic touch-style events. </div>

INTRODUCED IN:

API Level 3

SEE ALSO:

- `configChanges` attribute of the `<activity>` element
- [ConfigurationInfo](#)

<uses-feature>

本文内容

- › [Google Play 和根据功能进行过滤](#)
 - › [根据显式声明的功能进行过滤](#)
 - › [根据隐含功能进行过滤](#)
 - › [针对蓝牙功能的特殊处理](#)
 - › [测试应用需要的功能](#)
- › [功能参考资料](#)
 - › [硬件功能](#)
 - › [软件功能](#)
 - › [隐含功能要求的权限](#)

语法：

```
<uses-feature
    android:name="string"
    android:required=["true" | "false"]
    android:glEsVersion="integer" />
```

包含它的文件：

```
<manifest>
```

说明：

声明应用使用的单一硬件或软件功能。

<uses-feature> 声明的用途是将您的应用依赖的硬件和软件功能集通知任何外部实体。该元素提供的 `required` 属性可让您指定您的应用是需要声明的功能并且没有该功能便无法正常工作，还是使用该功能只是一种优先选择，没有它仍然可以正常工作。由于功能支持可能因 Android 设备而异，<uses-feature> 元素发挥的重要作用是，能够让应用描述其使用的随设备而变化的功能。

您的应用声明的可用功能集对应于 Android `PackageManager` 提供的功能常量集，方便起见，本文末尾的[功能参考资料](#)部分列出了这些功能常量。

您必须在单独的 <uses-feature> 元素中指定每个功能，因此如果您的应用需要多个功能，就需要声明多个 <uses-feature> 元素。例如，要求设备同时具有蓝牙和相机功能的应用会声明以下这两个元素：

```
<uses-feature android:name="android.hardware.bluetooth" />
<uses-feature android:name="android.hardware.camera" />
```

一般而言，您始终都应确保为应用需要的所有功能声明 <uses-feature> 元素。

声明的 <uses-feature> 元素仅供参考，这意味着 Android 系统本身在安装应用前不会检查设备是否提供相应的功能支持。不过，其他服务（如 Google Play）或应用可能会在处理您的应用或与其交互的过程中检查它的 <uses-feature> 声明。因此，声明您的应用使用的所有功能（见下文列表）非常重要。

对某些功能而言，您或许可以通过特定属性定义功能的版本，例如所用 Open GL 的版本（使用 `glEsVersion` 声明）。设备具有或不具有的其他功能（如相机）使用 `name` 属性进行声明。

尽管只为运行 API 级别 4 或更高版本系统的设备激活了 <uses-feature> 元素，我们仍然建议为所有应用加入这些元素，即使



Google Play 过滤

Google Play 利用在您的应用清单中声明的 <uses-feature> 元素从不符合应用的硬件和软件功能要求的设备上将应用滤除。

通过指定您的应用要求的功能，可以让 Google Play 只向设备符合应用功能要求的用户而不是所有用户提供您的应用。

如需了解有关 Google Play 如何以功能为依据进行过滤的重要信息，请阅读下面的 [Google Play 和根据功能进行过滤](#)。

`minSdkVersion` 为“3”或更低也要这样做。 运行较低版本平台的设备会直接忽略该元素。

注：声明功能时，切记您还必须视情况请求权限。例如，您仍须请求 `CAMERA` 权限，才能让您的应用访问 Camera API。请求权限可授予应用对相应硬件和软件的访问权，而声明应用使用的功能则可确保设备兼容性。

属性：

`android:name`

以描述符字符串形式指定应用使用的单一硬件或软件功能。[硬件功能](#)和[软件功能](#)部分列出了有效的属性值。这些属性值区分大小写。

`android:required`

表示应用是否需要 `android:name` 中所指定功能的布尔值。

- 当您为某项功能声明 `android:required="true"` 时，即是规定当设备不具有该指定功能时，应用无法正常工作，或设计为无法正常工作。
- 当您为某项功能声明 `android:required="false"` 时，则意味着如果设备具有该功能，应用会在必要时优先使用该功能，但应用设计为不使用该指定功能也可正常工作。

如果未予声明，`android:required` 的默认值为 “`true`”。

`android:glEsVersion`

应用需要的 OpenGL ES 版本。高 16 位表示主版本号，低 16 位表示次版本号。例如，要指定 OpenGL ES 2.0 版，您需要将其值设置为“`0x00020000`”；要指定 OpenGL ES 3.2，则需将其值设置为“`0x00030002`”。

应用应在其清单中至多指定一个 `android:glEsVersion`。如果指定不止一个，将使用数值最高的 `android:glEsVersion`，任何其他值都会被忽略。

如果应用不指定 `android:glEsVersion` 属性，则系统假定应用只需要 OpenGL ES 1.0，即所有 Android 设备都支持的版本。

应用可以假定，如果平台支持给定 OpenGL ES 版本，也同样支持所有数值更低的 OpenGL ES 版本。因此，同时需要 OpenGL ES 1.0 和 OpenGL ES 2.0 的应用必须指定它需要 OpenGL ES 2.0。

能够使用几种 OpenGL ES 版本中任一版本的应用只应指定它需要的数值最低的 OpenGL ES 版本。（它可以在运行时检查是否有更高版本的 OpenGL ES 可用）。

如需了解有关如何使用 OpenGL ES（包括如何在运行时检查支持的 OpenGL ES 版本）的详细信息，请参阅 [OpenGL ES API 指南](#)。

引入的版本：

API 级别 4

另请参阅：

- [PackageManager](#)
- [FeatureInfo](#)
- [ConfigurationInfo](#)
- [`<uses-permission>`](#)
- [Google Play 上的过滤器](#)

Google Play 和根据功能进行过滤

Google Play 会过滤对用户可见的应用，让用户只能看到和下载与其设备兼容的应用。它过滤应用的其中一种方法是按功能兼容性进行过滤。

为确定应用与给定用户设备的功能兼容性，Google Play 会比较：

- 应用需要的功能 — 应用在其清单内的 `<uses-feature>` 元素中声明这些功能
与...
- 设备上提供的硬件或软件功能 — 设备以只读系统属性形式报告其支持的功能。

为确保功能比较的准确性，Android 软件包管理器提供了一个共享的功能常量集，应用和设备均利用该集来声明功能要求和支持。本文末尾的[功能参考资料](#)部分以及 `PackageManager` 的类文档中列出了这些可用的功能常量。

当用户启动 Google Play 时，应用通过调用 `getSystemAvailableFeatures()` 在软件包管理器中查询设备上提供的功能列表。然后，Google 商店应用会在为用户建立会话时将功能列表向上传递给 Google Play。

您每次向 Google Play Developer Console 上传应用时，Google Play 都会扫描应用的清单文件。它会寻找 `<uses-feature>` 元素并在某些情况下结合其他元素（例如 `<uses-sdk>` 和 `<uses-permission>` 元素）对其进行评估。在建立应用所需的功能集之后，它会在内部将该列表存储为与应用 `.apk` 和应用版本关联的元数据。

当用户利用 Google Play 应用搜索或浏览应用时，该服务会将各应用需要的功能与用户设备上提供的功能进行比较。如果设备提供了应用所需的全部功能，则 Google Play 允许用户看到该应用并可能允许用户下载该应用。如果设备不支持任何所需功能，Google Play 会滤除该应用，令其对用户不可见，也无法供用户下载。

由于您在 `<uses-feature>` 元素中声明的功能会直接影响 Google Play 对您的应用的过滤方式，因此必须了解 Google Play 如何评估应用的清单以及如何建立所需功能集。下文提供了详细信息。

根据显式声明的功能进行过滤

显式声明的功能是指您的应用在 `<uses-feature>` 元素中声明的功能。功能声明可包括 `android:required=["true" | "false"]` 属性（如果您编译的应用面向 API 级别 5 或更高版本），您可以通过它指定应用是绝对需要该功能，没有它便无法正常工作（设置为 `"true"` 时），还是应用会在提供了该功能时予以优先使用，但应用本身设计为不使用它也能正常运行（设置为 `"false"` 时）。

Google Play 按以下方式处理显式声明的功能：

- 如果一项功能被显式声明为所需功能，Google Play 会将该功能添加到应用的所需功能列表。然后，它会从不提供该功能的设备上滤除该应用，让用户无法看到。例如：

```
<uses-feature android:name="android.hardware.camera" android:required="true" />
```

- 如果一项功能被显式声明为并非所需功能，Google Play 不会将该功能添加到所需功能列表。因此，在过滤应用时，从不会考虑显式声明的非所需功能。即使设备不提供声明的功能，Google Play 仍会考虑与设备兼容的应用并将其显示给用户，除非有其他过滤规则适用。例如：

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

- 如果显式声明了某个功能，但未加入 `android:required` 属性，则 Google Play 会假定该功能是所需功能，并对其设置过滤。

一般而言，如果您的应用设计为在 Android 1.6 及更低版本上运行，API 中不提供 `android:required` 属性，Google Play 会假定任何以及所有 `<uses-feature>` 声明都是所需功能。

注：通过显式声明某项功能并加入 `android:required="false"` 属性，您可以在 Google Play 上有效停用所有针对指定功能的过滤。

根据隐含功能进行过滤

隐含功能是指应用正常工作所必需，但未在清单文件的 `<uses-feature>` 元素中进行声明的功能。严格地讲，每个应用都应始终声明其使用或需要的所有功能，因此应将缺少应用所使用功能的声明视为错误。不过，作为给用户和开发者提供的一种保障措施，Google Play 会寻找每个应用中的隐含功能并设置针对这些功能的过滤，就像它为显式声明功能所做的那样。

应用可能需要某项功能却不会声明它，这是因为：

- 应用的编译目标是旧版本的 Android 内容库（Android 1.5 或更低版本），并且 `<uses-feature>` 元素不可用。
- 开发者错误地认为所有设备都提供该功能，没必要进行声明。

- 开发者无意中遗漏了功能声明。
- 开发者显式声明了功能，但声明无效。例如，`<uses-feature>` 元素名称拼写错误或无法识别的 `android:name` 属性字符串值都会令功能声明无效。

为将以上情况考虑在内，Google Play 会尝试通过检查清单文件中声明的其他元素（具体地讲，`<uses-permission>` 元素）来发现应用的隐含功能要求。

如果应用请求硬件相关权限，Google Play 假定该应用使用基础的硬件功能，并因此需要这些功能，即使可能没有对应的 `<uses-feature>` 声明。对于此类权限，Google Play 会向它为应用存储的元数据添加这些基础的硬件功能，并设置针对它们的过滤。

例如，如果应用请求 `CAMERA` 权限，但没有针对 `android.hardware.camera` 声明 `<uses-feature>` 元素，Google Play 会认为该应用需要相机，并且不应向没有相机的设备用户显示。

如果您不想让 Google Play 根据特定隐含功能进行过滤，可以停用该行为。要执行此操作，请在一个 `<uses-feature>` 元素中显式声明该功能，并加入一个 `android:required="false"` 属性。例如，要停用源自 `CAMERA` 权限的过滤，您需要声明该功能，如下所示。

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

必须了解的是，您在 `<uses-permission>` 元素中请求的权限可能会直接影响 Google Play 对您的应用的过滤方式。参考资料部分[隐含功能要求的权限](#)完整列出了一组隐含功能要求并因此触发过滤的权限。

针对蓝牙功能的特殊处理

Google Play 在决定针对蓝牙的过滤时应用的规则与上述略有差异。

如果某个应用在 `<uses-permission>` 元素中声明了蓝牙权限，但未在 `<uses-feature>` 元素中显式声明蓝牙功能，则 Google Play 会按照 `<uses-sdk>` 元素中的指定检查作为应用设计运行目标的 Android 平台版本。

如下表所示，只有在应用将其最低或目标平台声明为 Android 2.0 (API 级别 5) 或更高级别时，Google Play 才会启用针对蓝牙功能的过滤。但请注意，当应用在 `<uses-feature>` 元素中显式声明蓝牙功能时，Google Play 会应用正常过滤规则。

表 1. Google Play 如何确定请求蓝牙权限却未在 `<uses-feature>` 元素中声明蓝牙功能的应用的蓝牙功能要求。

如果 <code>minSdkVersion...</code>	或 <code>targetSdkVersion</code>	结果
<code><=4</code> (或未声明 <code>uses-sdk</code>)	<code><=4</code>	Google Play 将不会根据设备所报告的对 <code>android.hardware.bluetooth</code> 功能的支持情况从任何设备中滤除该应用。
<code><=4</code>	<code>>=5</code>	Google Play 会从任何不支持 <code>android.hardware.bluetooth</code> 功能 (包括旧版本) 的设备中滤除该应用。
<code>>=5</code>	<code>>=5</code>	

以下示例根据 Google Play 对蓝牙功能的处理方式说明了不同的过滤效果。

在第一个示例中，一个设计为在旧版 API 级别上运行的应用声明了蓝牙权限，但未在 `<uses-feature>` 元素中声明蓝牙功能。

结果：Google Play 不会从任何设备中滤除该应用。

```
<manifest ...>
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
  <uses-sdk android:minSdkVersion="3" />
  ...
</manifest>
```

在下面的第二个示例中，同一应用还声明了目标 API 级别“5”。

结果：Google Play 现在认为该功能是所需功能，将从所有不报告蓝牙支持的设备 (包括运行旧版本平台的设备) 上滤除该应用。

```
<manifest ...>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="5" />
    ...
</manifest>
```

在本例中，同一应用现在明确声明了蓝牙功能。

结果：与上例完全相同（应用了过滤）。

```
<manifest ...>
    <uses-feature android:name="android.hardware.bluetooth" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="5" />
    ...
</manifest>
```

最后，在以下示例中，同一应用添加了一个 `android:required="false"` 属性。

结果：Google Play 针对所有设备停用根据蓝牙功能支持进行过滤。

```
<manifest ...>
    <uses-feature android:name="android.hardware.bluetooth" android:required="false" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="5" />
    ...
</manifest>
```

测试应用需要的功能

您可以利用 Android SDK 中包括的 `aapt` 工具来确定 Google Play 如何根据应用声明的功能和权限对其进行过滤。要执行该操作，请使用 `dump badging` 命令运行 `aapt`。这会使 `aapt` 解析您的应用的清单，并应用与 Google Play 所用相同的规则来确定您的应用需要的功能。

要使用该工具，请按以下步骤操作：

1. 首先，生成您的应用并将其导出为未签署的 `.apk`。如果您是在 Android Studio 中进行开发，请使用 Gradle 生成您的应用：
 - a. 打开项目并选择 **Run > Edit Configurations**。
 - b. 选择靠近 **Run/Debug Configurations** 窗口左上角的加号。
 - c. 选择 **Gradle**。
 - d. 在 **Name** 中输入 `Unsigned APK`。
 - e. 从 **Gradle project** 部分选择您的模块。
 - f. 在 **Tasks** 中输入 `assemble`。
 - g. 选择 **OK** 完成新配置。
 - h. 确保在工具栏中选择了 `Unsigned APK` 运行配置，并选择 **Run > Run 'Unsigned APK'**。
- 您可以在 `<ProjectName>/app/build/outputs/apk/` 目录中找到您的未签署 `.apk`。
- 接下来，如果 `aapt` 工具尚未出现在您的 PATH 中，定位该工具。如果您使用的是 SDK Tools r8 或更高版本，可以在 `<SDK>/platform-tools/` 目录中找到 `aapt`。

注：您必须使用为可用的最新 Platform-Tools 组件提供的 `aapt` 版本。如果您没有最新的 Platform-Tools 组件，请使用 [Android SDK 管理器](#) 下载它。
3. 使用以下语法运行 `aapt`：

```
$ aapt dump badging <path_to_exported_.apk>
```

以下是针对上面第二个蓝牙示例的命令输出示例：

```
$ ./aapt dump badging BTExample.apk
package: name='com.example.android.btexample' versionCode='1' versionName=''
uses-permission:'android.permission.BLUETOOTH_ADMIN'
uses-feature:'android.hardware.bluetooth'
sdkVersion:'3'
targetSdkVersion:'5'
application: label='BT Example' icon='res/drawable/app_bt_ex.png'
launchable activity name='com.example.android.btexample.MyActivity' label='' icon=''
uses-feature:'android.hardware.touchscreen'
main
supports-screens: 'small' 'normal' 'large'
locales: '--_--'
densities: '160'
```

功能参考资料

下文提供有关硬件功能、软件功能以及隐含具体功能要求的权限集的参考信息。

硬件功能

此部分介绍最新平台版本支持的硬件功能。要指出您的应用使用或需要某项硬件功能，请在 `android:name` 属性中声明相应的值（以 `"android.hardware"` 开头）。请在您每次声明一项硬件功能时使用单独的 `<uses-feature>` 元素。

音频硬件功能

`android.hardware.audio.low_latency`

应用使用设备的低延迟时间音频管道，该管道可以减少处理声音输入或输出时的滞后和延迟。

`android.hardware.audio.output`

应用使用设备的音响设备、音频耳机插孔、蓝牙流式传输能力或类似机制传输声音。

`android.hardware.audio.pro`

应用使用设备的高端音频功能和性能能力。

`android.hardware.microphone`

应用使用设备的麦克风录音音频。

蓝牙硬件功能

`android.hardware.bluetooth`

应用使用设备的蓝牙功能，通常是为了与其他支持蓝牙的设备通信。

`android.hardware.bluetooth_le`

应用使用设备的低功耗蓝牙无线电功能。

相机硬件功能

`android.hardware.camera`

应用使用设备的后置相机。只有前置相机的设备不会列出该功能，因此如果您的应用可与任何朝向的相机通信，请改用

`android.hardware.camera.any` 功能。

`android.hardware.camera.any`

应用使用设备的其中一个相机或用户为设备连接的外置相机。如果您的应用不要求相机必须是后置式，请使用此值来替代 `android.hardware.camera`。

`android.hardware.camera.autofocus`

应用使用设备相机支持的自动对焦功能。

应用通过使用该功能暗示其还使用 `android.hardware.camera` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

`android.hardware.camera.capability.manual_post_processing`

应用使用设备相机支持的 `MANUAL_POST_PROCESSING` 功能。

您的应用可以通过该功能替换相机的自动白平衡功能。使用 `android.colorCorrection.transform`、`android.colorCorrection.gains` 以及 `TRANSFORM_MATRIX` 的 `android.colorCorrection.mode`。

`android.hardware.camera.capability.manual_sensor`

应用使用设备相机支持的 `MANUAL_SENSOR` 功能。

该功能隐含对自动曝光锁定 (`android.control.aeLock`) 的支持，该支持可以让相机的曝光时间和灵敏度一直固定在特定值。

`android.hardware.camera.capability.raw`

应用使用设备相机支持的 `RAW` 功能。

该功能暗示设备可以保存 DNG（原始）文件，并且设备的相机提供您的应用直接处理这些原始图像所需的 DNG 相关元数据。

`android.hardware.camera.external`

应用与用户为设备连接的外置相机通信。但该功能不能保证外置相机可供您的应用使用。

`android.hardware.camera.flash`

应用使用设备相机支持的闪光功能。

应用通过使用该功能暗示其还使用 `android.hardware.camera` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

`android.hardware.camera.front`

应用使用设备的前置相机。

应用通过使用该功能暗示其还使用 `android.hardware.camera` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

`android.hardware.camera.level.full`

应用使用设备的至少一个相机提供的 `FULL` 级图像捕捉支持。提供 `FULL` 支持的相机可提供快速捕捉功能、逐帧控制和手动后期处理控制。

设备 UI 硬件功能

`android.hardware.type.automotive`

应用设计为在车辆内的一组屏幕上显示其 UI。用户使用硬按钮、触摸、旋转控制器以及类鼠标界面与应用进行交互。车辆的屏幕通常

出现在车辆的中控台或仪表板中。这些屏幕的尺寸和分辨率通常有限。

注：切记，由于用户是在驾车时使用这类应用 UI，应用必须尽量不要让驾驶员分心。

`android.hardware.type.television`

(已弃用；请改用 `android.software.leanback`。)

应用设计为在电视上显示其 UI。该功能将“电视”定义为一种典型的起居室电视体验：显示在大屏幕上，用户坐在远处，主流输入形式是类似方向键的东西，一般不使用鼠标、指示器或触摸设备。

`android.hardware.type.watch`

应用设计为在手表上显示其 UI。手表佩戴在身体（例如手腕）上。用户在很近的距离与设备互动。

指纹硬件功能

`android.hardware.fingerprint`

应用使用设备的生物识别硬件读取指纹。

手柄硬件功能

`android.hardware.gamepad`

应用捕获来自设备本身或其连接的手柄的游戏控制器输入。

红外线硬件功能

`android.hardware.consumerir`

应用使用设备的红外线 (IR) 功能，通常是为了与其他消费 IR 设备通信。

定位硬件功能

`android.hardware.location`

应用使用设备上的一项或多项功能来确定位置，例如 GPS 位置、网络位置或基站位置。

`android.hardware.location.gps`

应用使用从设备上的全球定位系统 (GPS) 接收器获得的精确位置坐标。

应用通过使用该功能暗示其还使用 `android.hardware.location` 功能，除非这个父功能在声明时使用了属性 `android:required="false"`。

`android.hardware.location.network`

应用使用从设备上支持的基于网络的地理定位系统获得的粗略位置坐标。

应用通过使用该功能暗示其还使用 `android.hardware.location` 功能，除非这个父功能在声明时使用了属性 `android:required="false"`。

NFC 硬件功能

`android.hardware.nfc`

应用使用设备的近距离无线通信 (NFC) 功能。

`android.hardware.nfc.hce`

(已弃用。)

应用使用设备上托管的 NFC 卡模拟。

OpenGL ES 硬件功能

`android.hardware.opengles.aep`

应用使用设备上安装的 [OpenGL ES Android 扩展包](#)。

传感器硬件功能

`android.hardware.sensor.accelerometer`

应用使用从设备的加速计读取的运动信息来检测设备的当前方向。例如，应用可以使用加速计读数来确定何时在纵向与横向方向之间切换。

`android.hardware.sensor.ambient_temperature`

应用使用设备的外界（环境）温度传感器。例如，天气应用可以报告室内或室外温度。

`android.hardware.sensor.barometer`

应用使用设备的气压计。例如，天气应用可以报告气压。

`android.hardware.sensor.compass`

应用使用设备的磁力计（罗盘）。例如，导航应用可以用户当前面朝的方向。

`android.hardware.sensor.gyroscope`

应用使用设备的陀螺仪来检测旋转和倾斜，从而形成一个六轴方向系统。通过使用该传感器，应用可以更顺利地检测其是否需要在纵向与横向方向之间切换。

`android.hardware.sensor.hifi_sensors`

应用使用设备的高保真 (Hi-Fi) 传感器。例如，游戏应用可以检测用户的高精度移动。

`android.hardware.sensor.heartrate`

应用使用设备的心率监测器。例如，健身应用可以报告用户心率随时间的变化趋势。

`android.hardware.sensor.heartrate.ecg`

应用使用设备的超声波心动图 (ECG) 心率传感器。例如，健身应用可以报告有关用户心率的更详细信息。

`android.hardware.sensor.light`

应用使用设备的光传感器。例如，应用可以根据环境光照条件显示两种不同配色方案中的一种。

`android.hardware.sensor.proximity`

应用使用设备的近程传感器。例如，电话应用可以在其检测到用户握持的设备贴近身体时关闭设备的屏幕。

`android.hardware.sensor.relative_humidity`

应用使用设备的相对湿度传感器。例如，天气应用可以利用湿度来计算和报告当前露点。

`android.hardware.sensor.stepcounter`

应用使用设备的计步器。例如，健身应用可以报告用户需要走多少步才能达到每天的计步目标。

android.hardware.sensor.stepdetector

应用使用设备的步测器。例如，健身应用可以利用每步的间隔时间来推测用户正在进行的锻炼类型。

屏幕硬件功能

android.hardware.screen.landscape

android.hardware.screen.portrait

应用要求设备使用纵向或横向方向。如果您的应用同时支持这两种方向，则无需声明任一功能。

例如，如果您的应用要求纵向方向，则应声明以下功能，使得只有支持纵向方向（始终或由用户选择）的设备才能运行您的应用：

```
<uses-feature android:name="android.hardware.screen.portrait" />
```

默认情况下假定两种方向均非要求的方向，这样您的应用就可以安装在支持一种或同时支持两种方向的设备上。不过，如果应用的任何 Activity 利用 `android:screenOrientation` 属性请求在特定方向下运行，则此声明意味着您的应用要求该方向。例如，如果您使用 "landscape"、"reverseLandscape" 或 "sensorLandscape" 声明 `android:screenOrientation`，则您的应用将只能安装在支持横向方向的设备上。

最佳做法是，您仍应使用 `<uses-feature>` 元素来声明对该方向的要求。如果您使用 `android:screenOrientation` 为 Activity 声明了某个方向，但实际并无此要求，可通过使用 `<uses-feature>` 元素并加入 `android:required="false"` 声明该方向来停用这一要求。

为实现后向兼容性，任何运行 Android 3.1 (API 级别 12) 或更低版本的设备都同时支持横向和纵向方向。

电话硬件功能

android.hardware.telephony

应用使用设备的电话功能，例如提供数据通信服务的无线电话。

android.hardware.telephony.cdma

应用使用码分多址接入 (CDMA) 无线电话系统。

应用通过使用该功能暗示其还使用 `android.hardware.telephony` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

android.hardware.telephony.gsm

应用使用全球移动通信系统 (GSM) 无线电话系统。

应用通过使用该功能暗示其还使用 `android.hardware.telephony` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

触摸屏硬件功能

android.hardware.faketouch

应用使用基本的触摸交互事件，例如点按和拖动。

声明为必需时，此功能表示应用只兼容模拟触摸屏（“假触摸”界面）或实际具有触摸屏的设备。

带有假触摸界面的设备为用户提供模拟部分触摸屏功能的用户输入系统。例如，鼠标或遥控器可以驱动屏幕光标。如果您的应用需要基本的点击式交互（换言之，它在只使用方向键控制器的情况下无法正常工作），则应声明该功能。由于这是最低水平的触摸交互，因此您还可以在提供更复杂触摸界面的设备上使用声明该功能的应用。

注：默认情况下应用需要 `android.hardware.touchscreen`。如果您希望自己的应用可供提供假触摸界面的设备使用，则必须按如下方式显式声明不要求提供触摸屏：

```
<uses-feature android:name="android.hardware.touchscreen" android:required="false" />
```

android.hardware.faketouch.multitouch.distinct

应用在假触摸界面上区分两个或更多个“手指”的触摸轨迹。这是 `android.hardware.faketouch` 功能的一个超集。声明为必需时，此功能表示应用只兼容模拟区分两个或更多个手指的触摸轨迹或实际具有触摸屏的设备。

不同于 `android.hardware.touchscreen.multitouch.distinct` 所定义的分区式多点触摸，通过假触摸界面支持分区式多点触摸的输入设备并不支持所有双指手势，因为输入会转换成屏幕上的光标移动。也就是说，在此类设备上的单指手势移动光标，双指划动触发单指触摸事件，而其他双指手势则触发相应的双指触摸事件。

提供双指触摸触控板进行光标移动的设备可支持该功能。

android.hardware.faketouch.multitouch.jazzhand

应用在假触摸界面上区分五个或更多个“手指”的触摸轨迹。这是 `android.hardware.faketouch` 功能的一个超集。声明为必需时，此功能表示应用只兼容模拟区分五个或更多个手指的触摸轨迹或实际具有触摸屏的设备。

不同于 `android.hardware.touchscreen.multitouch.jazzhand` 所定义的分区式多点触摸，通过假触摸界面支持单手多点触摸的输入设备并不支持所有五指手势，因为输入会转换成屏幕上的光标移动。也就是说，在此类设备上的单指手势移动光标，多指手势触发单指触摸事件，而其他多指手势则触发相应的多指触摸事件。

提供五指触摸触控板进行光标移动的设备可支持该功能。

android.hardware.touchscreen

应用利用设备的触摸屏功能来实现比基本触摸事件交互性更强的手势，例如滑屏。这是 `android.hardware.faketouch` 功能的一个超集。

默认情况下，您的应用需要该功能。因此，您的应用不可供默认情况下只提供模拟触摸界面（“假触摸”）的设备使用。如果您希望自己的应用可供提供假触摸界面的设备（甚至只提供方向键控制器的设备）使用，则必须通过在声明 `android.hardware.touchscreen` 时加入 `android:required="false"` 来显式声明不要求提供触摸屏。如果您的应用使用（但并不需要）真触摸屏界面，则应添加此声明。

如果您的应用确实需要触摸界面（以便执行滑屏之类的更高级触摸手势），则您无需声明任何触摸界面功能，因为它们在默认情况下是必需功能。不过，最好还是显式声明您的应用使用的所有功能。

如果您需要进行更复杂的触摸交互（例如多指手势），则应声明您的应用使用高级触摸屏功能。

android.hardware.touchscreen.multitouch

应用使用设备的基本两点式多点触摸功能（例如实现张合手势的功能），但应用不需要独立追踪触摸轨迹。这是 `android.hardware.touchscreen` 功能的一个超集。

应用通过使用该功能暗示其还使用 `android.hardware.touchscreen` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

android.hardware.touchscreen.multitouch.distinct

应用使用设备的高级多点触摸功能来独立追踪两点或更多点的轨迹。该功能是 `android.hardware.touchscreen.multitouch` 功能的一个超集。

应用通过使用该功能暗示其还使用 `android.hardware.touchscreen.multitouch` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

android.hardware.touchscreen.multitouch.jazzhand

应用使用设备的高级多点触摸功能来独立追踪五点或更多点的轨迹。该功能是 `android.hardware.touchscreen.multitouch` 功能的一个超集。

应用通过使用该功能暗示其还使用 `android.hardware.touchscreen.multitouch` 功能，除非这个父功能在声明时使用了

```
    android:required="false"。
```

USB 硬件功能

android.hardware.usb.accessory

应用表现为 USB 设备，与 USB 主机相连。

android.hardware.usb.host

应用使用与设备相连的 USB 附件。设备充当 USB 主机。

Wi-Fi 硬件功能

android.hardware.wifi

应用使用设备上的 802.11 网络 (Wi-Fi) 功能。

android.hardware.wifi.direct

应用使用设备上的 Wi-Fi Direct 网络功能。

软件功能

此部分介绍最新平台版本支持的软件功能。要指出您的应用使用或需要某项软件功能，请在 `android:name` 属性中声明相应的值（以 `"android.software"` 开头）。请在您每次声明一项软件功能时使用单独的 `<uses-feature>` 元素。

通信软件功能

android.software.sip

应用使用会话发起协议 (SIP) 服务。通过使用 SIP，应用可以支持互联网电话操作，例如视频会议和即时消息传递。

android.software.sip.voip

应用使用基于 SIP 的互联网语音协议 (VoIP) 服务。通过使用 VoIP，应用可以支持实时互联网电话操作，例如双向视频会议。

应用通过使用该功能暗示其还使用 `android.software.sip` 功能，除非这个父功能在声明时使用了 `android:required="false"`。

android.software.webview

应用显示来自互联网的内容。

自定义输入软件功能

android.software.input_methods

应用使用新的输入法，该输入法由开发者在 `InputMethodService` 中定义。

设备管理软件功能

android.software.backup

应用加入处理备份和恢复操作的逻辑。

android.software.device_admin

应用通过设备管理员来强制执行设备规范。

android.software.managed_users

应用支持二级用户和托管配置文件。

`android.software.securely_removes_users`

应用可**永久性**移除用户及其相关数据。

`android.software.verified_boot`

应用加入处理设备验证启动功能结果的逻辑，该逻辑可检测设备的配置在重新启动操作期间是否发生了变化。

媒体软件功能

`android.software.midi`

应用利用乐器数字化接口 (MIDI) 协议连接到乐器或输出声音。

`android.software.print`

应用加入打印设备上所显示文档的命令。

`android.software.leanback`

应用呈现专为在大屏幕（例如电视）上观看而设计的 UI。

`android.software.live_tv`

应用流式传输直播电视节目。

屏幕界面软件功能

`android.software.app_widgets`

应用使用或提供应用小部件，并且只应安装在带有可供用户嵌入应用小部件的主屏幕或类似位置的设备上。

`android.software.home_screen`

应用起到替代设备主屏幕的作用。

`android.software.live_wallpaper`

应用使用或提供包含动画的壁纸。

隐含功能要求的权限

一些硬件和软件功能常量是在相应 API 发布后提供给应用使用；例如，`android.hardware.bluetooth` 功能是在 Android 2.2 (API 级别 8) 中添加的，但它引用的 Bluetooth API 则是在 Android 2.0 (API 级别 5) 添加的。因此，一些应用在能够利用 `<uses-feature>` 系统声明其需要某个 API 之前就已经具备了使用该 API 的能力。

为防止无意间提供这些应用，Google Play 会假定特定硬件相关权限规定，默认情况下需要底层硬件功能。例如，使用蓝牙的应用必须在 `<uses-permission>` 元素中请求 `BLUETOOTH` 权限 — 对于旧版应用，Google Play 假定权限声明意味着，应用需要底层 `android.hardware.bluetooth` 功能，并根据该功能设置过滤。表 2 所列权限隐含的功能要求等同于 `<uses-feature>` 元素中声明的功能。

请注意，`<uses-feature>` 声明（包括任何声明的 `android:required` 属性）始终优先于表 2 中权限所隐含的功能。对于上述任一权限，您都可以通过在 `<uses-feature>` 元素中使用 `android:required="false"` 属性显式声明隐含功能来停用根据隐含功能进行过滤。例如，要想停用根据 `CAMERA` 权限进行任何过滤，您需要向清单文件添加下面这个 `<uses-feature>` 声明：

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

表 2. 隐含设备硬件用途的设备权限。

类别	此权限...	...隐含此功能要求
蓝牙	BLUETOOTH	android.hardware.bluetooth (如需了解详情, 请参阅 针对蓝牙功能的特殊处理 。)
	BLUETOOTH_ADMIN	android.hardware.bluetooth
相机	CAMERA	android.hardware.camera 和 android.hardware.camera.autofocus
定位	ACCESS_MOCK_LOCATION	android.hardware.location
	ACCESS_LOCATION_EXTRA_COMMANDS	android.hardware.location
	INSTALL_LOCATION_PROVIDER	android.hardware.location
	ACCESS_COARSE_LOCATION	android.hardware.location.network 和 android.hardware.location
	ACCESS_FINE_LOCATION	android.hardware.location.gps 和 android.hardware.location
麦克风	RECORD_AUDIO	android.hardware.microphone
电话	CALL_PHONE	android.hardware.telephony
	CALL_PRIVILEGED	android.hardware.telephony
	MODIFY_PHONE_STATE	android.hardware.telephony
	PROCESS_OUTGOING_CALLS	android.hardware.telephony
	READ_SMS	android.hardware.telephony
	RECEIVE_SMS	android.hardware.telephony
	RECEIVE_MMS	android.hardware.telephony
	RECEIVE_WAP_PUSH	android.hardware.telephony
	SEND_SMS	android.hardware.telephony
	WRITE_APN_SETTINGS	android.hardware.telephony
	WRITE_SMS	android.hardware.telephony
Wi-Fi	ACCESS_WIFI_STATE	android.hardware.wifi
	CHANGE_WIFI_STATE	android.hardware.wifi
	CHANGE_WIFI_MULTICAST_STATE	android.hardware.wifi



<uses-library>

SYNTAX:

```
<uses-library  
    android:name="string"  
    android:required=["true" | "false"] />
```

CONTAINED IN:

<application>

DESCRIPTION:

Specifies a shared library that the application must be linked against. This element tells the system to include the library's code in the class loader for the package.

All of the **android** packages (such as `android.app`, `android.content`, `android.view`, and `android.widget`) are in the default library that all applications are automatically linked against. However, some packages (such as `maps`) are in separate libraries that are not automatically linked. Consult the documentation for the packages you're using to determine which library contains the package code.

This element also affects the installation of the application on a particular device and the availability of the application on Google Play:

Installation

If this element is present and its `android:required` attribute is set to `true`, the `PackageManager` framework won't let the user install the application unless the library is present on the user's device.

The `android:required` attribute is described in detail in the following section.

ATTRIBUTES:

`android:name`

The name of the library. The name is provided by the documentation for the package you are using. An example of this is `"android.test.runner"`, a package that contains Android test classes.

`android:required`

Boolean value that indicates whether the application requires the library specified by `android:name`:

- `"true"`: The application does not function without this library. The system will not allow the application on a device that does not have the library.
- `"false"`: The application can use the library if present, but is designed to function without it if necessary. The system will allow the application to be installed, even if the library is not present. If you use `"false"`, you are responsible for checking at runtime that the library is available.

To check for a library, you can use reflection to determine if a particular class is available.

The default is `"true"`.

Introduced in: API Level 7.



Google Play Filtering

Google Play uses the `<uses-library>` elements declared in your app manifest to filter your app from devices that do not meet its library requirements. For more information about filtering, see the topic [Google Play filters](#).

INTRODUCED IN:

API Level 1

SEE ALSO:

- [PackageManager](#)

<uses-permission>

语法：

```
<uses-permission android:name="string"
    android:maxSdkVersion="integer" />
```

包含它的文件：

<manifest>

说明：

请求应用正确运行所必须获取的权限。权限在安装应用时（在运行 Android 5.1 和更低版本的设备上）或者在应用运行时（在运行 Android 6.0 和更高版本的设备上）由用户授予。

如需了解有关权限的详细信息，请参阅简介的[权限部分](#)和单独的[系统权限 API 指南](#)。在[android.Manifest.permission](#) 上可以找到基础平台定义的权限列表。

属性：

[android:name](#)

权限的名称。可以是应用通过[<permission>](#)元素定义的权限、另一个应用定义的权限，或者一个标准系统权限（例如["android.permission.CAMERA"](#) 或 ["android.permission.READ_CONTACTS"](#)）。如这些示例所示，权限名称通常以软件包名称为前缀。

[android:maxSdkVersion](#)

此权限应授予应用的最高 API 级别。如果应用需要的权限从某个 API 级别开始不再需要，则设置此属性很有用。

例如，从 Android 4.4（API 级别 19）开始，应用在外部存储空间写入其特定目录（[getExternalFilesDir\(\)](#) 提供的目录）时不再需要请求[WRITE_EXTERNAL_STORAGE](#) 权限。但 API 级别 18 和更低版本需要此权限。因此，您可以使用如下声明，声明只有 API 级别 18 及以前版本才需要此权限：

```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18" />
```

这样，从 API 级别 19 开始，系统将不再向您的应用授予[WRITE_EXTERNAL_STORAGE](#) 权限。

此属性为 API 级别 19 中新增属性。

引入的版本：

API 级别 1

另请参阅：

- [<permission>](#)
- [<uses-permission-sdk-23>](#)



Google Play 过滤

在某些情况下，您通过[<uses-permission>](#) 请求的权限可影响 Google Play 如何过滤您的应用。

如果您请求硬件相关的权限（例如—[CAMERA](#)），—Google Play 会假设您的应用需要底层硬件功能，因此从设备过滤掉不提供该功能的应用。

要控制过滤，务必在[<uses-feature>](#) 元素中显式声明硬件功能，而不是依赖 Google Play“发现”[<uses-permission>](#) 元素中的要求。然后，如果要对特定功能停用过滤，可将[android:required="false"](#) 属性添加到[<uses-feature>](#) 声明中。

有关暗示硬件功能的权限列表，请参阅[<uses-feature>](#) 元素的文档。

- <uses-feature>



<uses-permission-sdk-23>

语法：

```
<uses-permission-sdk-23 android:name="string"  
    android:maxSdkVersion="integer" />
```

包含它的文件：

```
<manifest>
```

说明：

指明应用需要特定权限，但仅当应用在 API 级别 23 或更高版本的设备上运行时才需要。如果设备运行的是 API 级别 22 或更低版本，则应用没有指定的权限。

当您更新应用以包含需要其他权限的新功能时，此元素很有用。如果用户在运行 API 级别 22 或更低版本的设备更新应用，系统在安装时会提示用户授予在该更新中声明的所有新权限。如果某个新功能无关紧要，您可能想同时在这些设备上停用该功能，以便用户不需要授予额外权限即可更新应用。如果使用 `<uses-permission-sdk-23>` 元素而不使用 `<uses-permission>`，则仅当应用在支持**运行时权限模式**（用户在应用运行时向其授予权限）的平台上运行时才可请求权限。

如需了解有关权限的详细信息，请参阅简介的[权限部分](#)和单独的[系统权限 API 指南](#)。在 `android.Manifest.permission` 上可以找到基础平台定义的权限列表。

属性：

`android:name`

权限的名称。此权限可以是应用通过 `<permission>` 元素定义的权限、另一个应用定义的权限，或者一个标准系统权限（例如 `"android.permission.CAMERA"` 或 `"android.permission.READ_CONTACTS"`）。

`android:maxSdkVersion`

此权限应授予应用的最高 API 级别。如果应用安装在 API 级别较高的设备上，则应用不会被授予权限，无法使用任何相关功能。

引入的版本：

API 级别 23

另请参阅：

- [`<permission>`](#)
- [`<uses-permission>`](#)
- [`<uses-feature>`](#)

<uses-sdk>

本文内容

- › [什么是 API 级别？](#)
- › [API 级别在 Android 中的使用](#)
- › [开发注意事项](#)
 - › [应用向前兼容性](#)
 - › [应用向后兼容性](#)
 - › [选择平台版本和 API 级别](#)
 - › [声明最低 API 级别](#)
 - › [针对更高 API 级别进行测试](#)
- › [按 API 级别过滤参考文档](#)

语法：

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```



Google Play 过滤

Google Play 利用在您的应用清单中声明的 `<uses-sdk>` 属性从不符合其平台版本要求的设备上滤除您的应用。在设置这些属性前，确保您了解 [Google Play 过滤器](#)。

包含它的文件：

```
<manifest>
```

说明：

您可以通过整数形式的 API 级别表示应用与一个或多个版本的 Android 平台的兼容性。应用表示的 API 级别将与给定 Android 系统的 API 级别进行比较，其结果在不同 Android 设备上可能有所差异。

尽管名称是这样，但该元素实际上是用于指定 API 级别，而不是 SDK（软件开发工具包）或 Android 平台的版本号。API 级别始终是单个整数。您无法通过 API 级别关联的 Android 版本号来推知 API 级别（例如，它不同于主版本号，也不同于主版本号与次版本号之和）。

另请阅读有关 [对您的应用进行版本控制](#) 的文档。

属性：

`android:minSdkVersion`

一个用于指定应用运行所需最低 API 级别的整数。如果系统的 API 级别低于该属性中指定的值，Android 系统将阻止用户安装应用。您应该始终声明该属性。

注意：如果您不声明该属性，系统将假定默认值为“1”，这表示您的应用兼容所有 Android 版本。如果您的应用并不兼容所有版本（例如，它使用 API 级别 3 中引入的 API），并且您尚未声明正确的 `minSdkVersion`，则当应用安装在 API 级别小于 3 的系统上时，应用将在运行时尝试访问不可用的 API 时发生崩溃。因此，请务必在 `minSdkVersion` 属性中声明合适的 API 级别。

`android:targetSdkVersion`

一个用于指定应用的目标 API 级别的整数。如果未设置，其默认值与为 `minSdkVersion` 指定的值相等。

该属性用于通知系统，您已针对目标版本进行测试，并且系统不应启用任何兼容性行为来保持您的应用与目标版本的向前兼容性。应用仍可在较低版本上运行（最低版本为 `minSdkVersion`）。

在 Android 随着每个新版本的推出而进化的过程中，一些行为甚至是外观可能会发生变化。不过，如果平台的 API 级别高于您的应用的 `targetSdkVersion` 所声明的版本，系统就可以通过启用兼容性行为来确保您的应用继续以您所期望的方式工作。您可以通过将 `targetSdkVersion` 指定为与应用所运行平台的 API 级别一致来停用此类兼容性行为。例如，如果将该值设置为“11”或更高，系统便可在您的应用运行在 Android 3.0 或更高版本的平台上时对其应用新的默认主题 (Holo)，还可在您的应用运行在更大屏幕上时停用屏幕兼容性模式（因为对 API 级别 11 的支持隐含了对更大屏幕的支持）。

系统可根据您为该属性设置的值启用许多兼容性行为。[Build.VERSION_CODES](#) 参考资料中的相应平台版本对其中的几种行为做了说明。

要让您的应用与各 Android 版本保持同步，您应该增加该属性的值，使其与最新 API 级别一致，然后在相应平台版本上对您的应用进行全面测试。

引入的版本：API 级别 4

`android:maxSdkVersion`

一个指定作为应用设计运行目标的最高 API 级别的整数。

在 Android 1.5、1.6、2.0 及 2.0.1 中，系统会在安装应用以及系统更新后重新验证应用时检查该属性的值。在任一情况下，如果应用的 `maxSdkVersion` 属性低于系统本身使用的 API 级别，系统均不允许安装应用。在系统更新后重新验证这种情况下，这实际上相当于将您的应用从设备中移除。

为说明该属性在系统更新后对您的应用的影响，请看看下面这个示例：

Google Play 上发布了一个在其清单中声明了 `maxSdkVersion="5"` 的应用。一位设备运行 Android 1.6 (API 级别 4) 的用户下载并安装了该应用。几周后，该用户收到了 Android 2.0 (API 级别 5) OTA 系统更新。更新安装后，系统检查该应用的 `maxSdkVersion` 并顺利完成了对其的重新验证。应用仍可照常工作。不过，一段时间后，设备又收到了一个系统更新，这次是更新到 Android 2.0.1 (API 级别 6)。更新完成后，系统无法再重新验证应用，因为此时系统本身的 API 级别 (6) 已超过该应用支持的最高级别 (5)。系统会使该应用对用户不可见，这实际上相当于将它从设备上删除。

警告：不建议声明该属性。首先，没有必要设置该属性，将其作为阻止您的应用部署到 Android 平台新发布版本上的一种手段。从设计上讲，新版本平台完全向后兼容。只要您的应用只使用标准 API 并遵循部署最佳实践，应该能够在新版本平台上正常工作。其次，请注意在某些情况下，声明该属性可能导致您的应用在系统更新至更高 API 级别后被从用户设备中移除。大多数可能安装您的应用的设备都会定期收到 OTA 系统更新，因此您应该在设置该属性前考虑这些更新对您的应用的影响。

引入的版本：API 级别 4

未来的 Android 版本（Android 2.0.1 之后的版本）在安装或重新验证时将不再检查或强制执行 `maxSdkVersion` 属性。不过，Google Play 在向用户呈现可供下载的应用时将继续使用该属性作为过滤器。

引入的版本：

API 级别 1

什么是 API 级别？

API 级别是一个对 Android 平台版本提供的框架 API 修订版进行唯一标识的整数值。

Android 平台提供了一种框架 API，应用可利用它与底层 Android 系统进行交互。该框架 API 由以下部分组成：

- 一组核心软件包和类
- 一组用于声明清单文件的 XML 元素和属性
- 一组用于声明和访问资源的 XML 元素和属性
- 一组 Intent
- 一组应用可请求的权限，以及系统中包括的权限强制执行。

每个后续版本的 Android 平台均可包括对其提供的 Android 应用框架 API 的更新。

框架 API 更新的设计用途是使新 API 与早期版本的 API 保持兼容。也就是说，大多数 API 更改都是新增更改，会引入新功能或替代功能。在 API 的某些部分得到升级时，旧版的被替换部分将被弃用，但不会被移除，这样现有应用仍可使用它们。在极少数情况下，可能会修改或移除 API 的某些部分，但通常只有在为了确保 API 稳健性以及应用或系统安全性时，才需要进行此类更改。所有其他来自早期修订版的 API 部分都将结转，不做任何修改。

Android 平台提供的框架 API 使用叫做“API 级别”的整数标识符指定。每个 Android 平台版本恰好支持一个 API 级别，但隐含了对所有早期 API 级别（低至 API 级别 1）的支持。Android 平台初始版本提供的是 API 级别 1，后续版本的 API 级别递增。

下表列出了各 Android 平台版本支持的 API 级别。如需了解有关运行各版本的设备的相对数量的信息，请参阅[“平台版本”信息中心页面](#)。

平台版本	API 级别	VERSION_CODE	备注
Android 7.0	24	N	平台亮点
Android 6.0	23	M	平台亮点
Android 5.1	22	LOLLIPOP_MR1	平台亮点
Android 5.0	21	LOLLIPOP	
Android 4.4W	20	KITKAT_WATCH	仅限 KitKat for Wearables
Android 4.4	19	KITKAT	平台亮点
Android 4.3	18	JELLY_BEAN_MR2	平台亮点
Android 4.2、4.2.2	17	JELLY_BEAN_MR1	平台亮点
Android 4.1、4.1.1	16	JELLY_BEAN	平台亮点
Android 4.0.3、4.0.4	15	ICE_CREAM SANDWICH_MR1	平台亮点
Android 4.0、4.0.1、4.0.2	14	ICE_CREAM SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	平台亮点
Android 3.0.x	11	HONEYCOMB	平台亮点
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	平台亮点
Android 2.3.2	9	GINGERBREAD	
Android 2.3.1 Android 2.3			
Android 2.2.x	8	FROYO	平台亮点
Android 2.1.x	7	ECLAIR_MR1	平台亮点
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	
Android 1.6	4	DONUT	平台亮点
Android 1.5	3	CUPCAKE	平台亮点
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	

API 级别在 Android 中的使用

API 级别标识符在确保尽可能为用户和应用开发者提供最佳体验方面发挥着重要作用：

- 它允许 Android 平台描述其支持的最高框架 API 修订版

- 它允许应用描述其需要的框架 API 修订版
- 它允许系统协商应用在用户设备上的安装，从而不安装不兼容版本的应用。

每个 Android 平台版本都将其 API 级别标识符存储在 Android 系统自身内部。

应用可以利用框架 API 提供的清单元素 (`<uses-sdk>`) 来说明其可以运行的最低和最高 API 级别，以及其在设计上支持的首选 API 级别。该元素具有以下三个重要属性：

- `android:minSdkVersion` - 指定能够运行应用的最低 API 级别。默认值为“1”。
- `android:targetSdkVersion` - 指定运行应用的目标 API 级别。在某些情况下，这允许应用使用在目标 API 级别中定义的清单文件元素或行为，而不是仅限于使用那些针对最低 API 级别定义的元素。
- `android:maxSdkVersion` - 指定能够运行应用的最高 API 级别。 **重要说明：**在使用该属性之前，请先阅读 `<uses-sdk>` 文档。

例如，要指定应用运行所需的最低系统 API 级别，应用需要在其清单中加入一个带 `android:minSdkVersion` 属性的 `<uses-sdk>` 元素。`android:minSdkVersion` 是一个整数值，对应于能够运行应用的最低版本 Android 平台的 API 级别。

当用户试图安装应用，或在系统更新后重新验证应用时，Android 系统会先检查应用清单中的 `<uses-sdk>` 属性，然后将这些属性的值与其自己的内部 API 级别进行对比。只有在符合以下条件时，系统才允许安装开始：

- 如果声明了 `android:minSdkVersion` 属性，其值必须小于或等于系统的 API 级别整数。如果未声明，则系统假定应用需要 API 级别 1。
- 如果声明了 `android:maxSdkVersion` 属性，其值必须大于或等于系统的 API 级别整数。如果未声明，则系统假定应用没有最高 API 级别。如需了解有关系统如何处理该属性的详细信息，请阅读 `<uses-sdk>` 文档。

如果在应用清单中进行了声明，`<uses-sdk>` 元素的内容可能如下所示：

```
<manifest>
  <uses-sdk android:minSdkVersion="5" />
  ...
</manifest>
```

应用在 `android:minSdkVersion` 中声明 API 级别的主要原因是，告知 Android 系统，它使用了在指定 API 级别引入的 API。如果由于某种原因将应用安装在 API 级别更低的平台上，则它会在运行时试图访问不存在的 API 时发生崩溃。系统通过在应用需要的最低 API 级别高于目标设备上的平台版本时不允许安装应用来防止出现这种结果。

例如，`android.appwidget` 软件包是随 API 级别 3 引入的。如果应用使用该 API，则必须使用“3”一值声明 `android:minSdkVersion` 属性。随后，应用便可安装在 Android 1.5 (API 级别 3) 和 Android 1.6 (API 级别 4) 等平台上，但不能安装在 Android 1.1 (API 级别 2) 和 Android 1.0 (API 级别 1) 平台上。

如需了解有关如何指定应用 API 级别要求的详细信息，请参阅清单文件文档的 `<uses-sdk>` 部分。

开发注意事项

下文提供有关您在开发应用时应该考虑的 API 级别的信息。

应用向前兼容性

Android 应用一般向前兼容新版本的 Android 平台。

因为几乎所有对框架 API 的更改都是新增更改，所以使用 API 任何给定版本（其 API 级别所指定版本）开发的 Android 应用均向前兼容更新版本的 Android 平台以及更高 API 级别。应用应该能够在所有后期版本的 Android 平台上运行，除非在个别情况下应用使用的某个 API 部分后来由于某种原因被移除。

向前兼容性很重要，因为许多 Android 设备都接收 OTA 系统更新。用户可以安装您的应用并顺利使用，然后通过接收 OTA 更新升级到新版本的 Android 平台。安装更新后，您的应用将运行在新运行时版本的环境中，但这个版本具有您的应用所依赖的 API 和系统功能。

在某些情况下，在该 API 之下所做的更改（例如对底层系统本身的更改）可能会影响运行在新环境中的应用。因此，作为应用开发者，您必须了解应用在各系统环境中的外观和行为。为帮助您在各种版本的 Android 平台上测试您的应用，Android SDK 提供了多个可供您下载的平台。每个平台都包括一个兼容的系统映像，您可以在 AVD 中运行该映像来测试您的应用。

应用向后兼容性

Android 应用不一定向后兼容比其编译时所针对的目标版本更久远的 Android 平台版本。

每个新版本的 Android 平台都可能包括新的框架 API，例如那些能够让应用使用新的平台功能或者替换现有 API 部分的 API。应用可以在运行于新平台时使用这些新 API，如上所述，也可以在运行于更新版本的平台（API 级别所指定的平台）上时使用这些新 API。反之，由于早期版本的平台不包括新 API，因此使用新 API 的应用无法运行在这些平台上。

尽管 Android 设备降级至之前版本平台的情况不太可能发生，但必须意识到，在现实情况下可能有许多设备运行的是早期版本的平台。即便是在能够获得 OTA 更新的设备中，也可能发生一些设备获得的更新滞后，可能在相当长时间内无法获得更新的情况。

选择平台版本和 API 级别

当您开发应用时，需要选择您所编译的应用所针对的目标平台版本。一般而言，您编译的应用所针对的目标版本应该是您的应用所能支持的最低平台版本。

您可以通过在编译应用时依次降低其所针对的目标编译版本来确定可能的最低平台版本。确定最低版本后，您应该使用相应平台版本（和 API 级别）创建一个 AVD，然后对您的应用进行全面测试。请务必在应用的清单中声明 `android:minSdkVersion` 属性，并将其值设置为平台版本的 API 级别。

声明最低 API 级别

如果您构建的应用使用的 API 或系统功能是在最新平台版本中引入的，则应将 `android:minSdkVersion` 属性的值设置为最新平台版本的 API 级别。这样做可确保用户只能将您的应用安装在运行兼容版本 Android 平台的设备上，并进而确保您的应用可以在用户设备上正常工作。

如果您的应用使用在最新平台版本中引入的 API，但未声明 `android:minSdkVersion` 属性，则应用可以在运行最新版本平台的设备上正常工作，但无法在运行早期版本平台的设备上正常工作。在后一种情况下，应用将在运行时试图使用早期版本上不存在的 API 时发生崩溃。

针对更高 API 级别进行测试

编译应用之后，您应该确保在应用的 `android:minSdkVersion` 属性所指定的平台上对其进行测试。为此，请使用您的应用所需的平台版本创建一个 AVD。此外，为确保向前兼容性，您还应在所有使用的 API 级别高于您的应用所用 API 级别的平台上运行并测试您的应用。

Android SDK 提供了多个可供您使用的平台版本（包括最新版本），并提供了可供您在必要时下载其他平台版本的更新器工具。

要访问该更新器，请使用位于 `<sdk>/tools` 目录的 `android` 命令行工具。您可以通过执行 `android sdk` 来启动 SDK 更新器。您还可以直接双击 `android.bat` (Windows) 或 `Android` (OS X/Linux) 文件。

要在模拟器中于不同平台版本上运行您的应用，请为您想测试的每个平台版本创建一个 AVD。如需了解有关 AVD 的详细信息，请参阅 [创建和管理虚拟设备](#)。如果您要使用物理设备进行测试，请确保您知晓它所运行的 Android 平台的 API 级别。请参阅本文顶部表格中所列的平台版本及其 API 级别。

按 API 级别过滤参考文档

Android Developers 网站在每个参考文档页面的右上角提供了一个“Filter by API Level”控件。您可以利用该控件，根据应用在其清单文件的 `android:minSdkVersion` 属性中指定的 API 级别，只显示您的应用实际可以访问的 API 部分的对应文档。

要使用过滤，请选择用于启用过滤的复选框，该复选框就在页面搜索框下方。然后将“Filter by API Level”控件设置为您的应用所指定的同一 API 级别。请注意，后期 API 级别中引入的 API 随即灰显，并且其内容会被屏蔽，因为您的应用无法访问它们。

在文档中按 API 级别进行过滤并不能让您查看每个 API 级别新增或引入的 API - 它仅仅是提供了一种途径，让您能够查看与给定 API 级别关联的整个 API，同时将后期 API 级别中引入的 API 元素排除在外。

如果您决定不想过滤 API 文档，只需使用复选框停用该功能。默认情况下系统会停用 API 级别过滤，这样无论 API 级别如何，您都能查看完整的框架 API。

还请注意，各 API 元素的参考文档指定的是引入各元素的 API 级别。软件包和类的 API 级别在各文档页面内容区域的右上角以“Since <api 级别>”形式指定。类成员的 API 级别在其位于右边距的详细说明标头中指定。



用户界面

应用的用户界面包含用户可查看并与之交互的所有内容。Android 提供丰富多样的预置 UI 组件，例如结构化布局对象和 UI 控件，您可以利用这些组件为您的应用构建图形界面。Android 还提供其他 UI 模块用于构建特殊界面，例如对话框、通知和菜单。

博客文章

从此告别菜单按钮

随着 Ice Cream Sandwich 向更多设备的推广，您必须开始将您的设计迁移至操作栏，以促进一致的 Android 用户体验。

新布局小部件：Space 和 GridLayout

Ice Cream Sandwich (ICS) 支持两款新的小部件，这两款小部件经过专门的设计，支持大显示屏可能提供的更丰富的用户界面：Space 和 GridLayout。

自定义操作栏

通过在针对 Honeycomb 的应用中使用操作栏，可以让您的用户以其熟悉的方式与您的应用交互。

利用 ViewPager 滑动水平视图

无论您是刚着手开发 Android 应用，还是 Android 应用资深开发者，都可能要不了多久，您就需要实现可水平滚动的视图集。

培训

实现有效导航

本课程将向您介绍如何为应用规划高级屏幕层次结构，然后选择合适的导航形式，让用户能够有效且直观地浏览您的内容。

面向多种屏幕的设计

Android 支持数以百计具有若干不同屏幕尺寸的设备类型，涵盖从小到手机、大到电视的各类设备。本课程向您介绍如何实现已针对多种屏幕配置进行了优化的用户界面。

改善布局性能

布局是 Android 应用中直接影响用户体验的关键部分。如果实施不当，您的布局可能会导致应用消耗大量内存，并且 UI 速度缓慢。本课程将向您介绍如何避免此类问题。

UI 概览

Android 应用中的所有用户界面元素都是使用 `View` 和 `ViewGroup` 对象构建而成。`View` 对象用于在屏幕上绘制可供用户交互的内容。`ViewGroup` 对象用于储存其他 `View` (和 `ViewGroup`) 对象，以便定义界面的布局。

Android 提供了一系列 `View` 和 `ViewGroup` 子类，可为您提供常用输入控件（如按钮和文本字段）和各种布局模式（如线性布局或相对布局）。

用户界面布局

如图 1 所示，每个应用组件的用户界面都是使用 `View` 和 `ViewGroup` 对象的层次结构定义的。每个视图组都是一个用于组织子视图的不可见容器，而子视图可以是输入控件或其他可绘制某一 UI 部分的小部件。此层次结构树可繁可简，随需而定（但是简单的结构可提供最佳性能）。

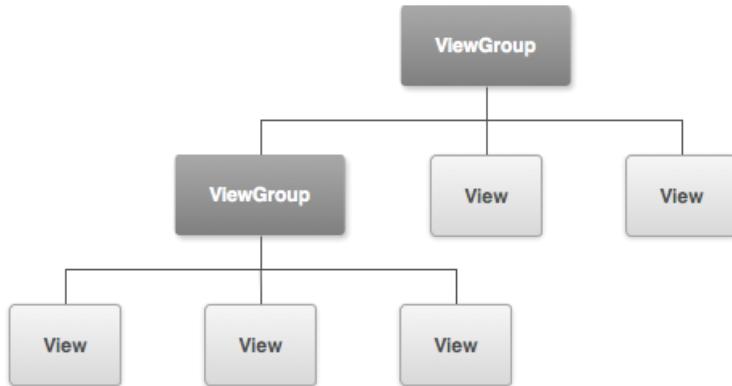


图 1. 视图层次结构的图示，它定义了一个 UI 布局。

要声明布局，您可以实例化代码中的 `View` 对象并开始构建树，但是定义布局最简单且最有效的方法是使用 XML 文件。如同 HTML 一样，XML 也为布局提供了一种用户可读结构。

视图的 XML 元素名称与其代表的 Android 类相对应。因此，`<TextView>` 元素用于在 UI 中创建一个 `TextView` 小部件，而 `<LinearLayout>` 元素用于创建一个 `LinearLayout` 视图组。

例如，包含文本视图和按钮的简单垂直布局如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button" />
</LinearLayout>
```

在应用中加载布局资源时，Android 会将布局的每个节点初始化为运行时对象，供您定义其他行为、查询对象状态或修改布局。

有关创建 UI 布局的完整指南，请参阅 [XML 布局](#)。

用户界面组件

您无需使用 [View](#) 和 [ViewGroup](#) 对象构建所有 UI。Android 提供了几个带有标准 UI 布局的应用组件，您只需定义内容。这些 UI 组件均拥有一组唯一的 API，具体描述可参阅相应的文档，如[添加应用栏](#)、[对话框](#)和[状态通知](#)。



输入事件

本文内容

- › [事件监听器](#)
- › [事件处理程序](#)
- › [触摸模式](#)
- › [处理焦点](#)

在 Android 系统中，从用户与应用的交互中截获事件的方法不止一种。如考虑截获用户界面内的事件，则可从用户与之交互的特定视图对象中捕获事件。为此，View 类提供了多种方法。

在您将用于构建布局的各种 View 类中，您可能会注意到几种看起来适用于 UI 事件的公共回调方法。当该对象上发生相应的操作时，Android 框架会调用这些方法。例如，在触摸一个视图对象（例如“按钮”）时，对该对象调用 `onTouchEvent()` 方法。不过，为了截获此事件，您必须扩展 View 类并重写该方法。然而，为了处理此类事件而扩展每个视图对象并不现实。正因如此，View 类还包含一系列嵌套接口以及您可以更加轻松定义的回调。这些接口称为 [事件监听器](#)，是您捕获用户与 UI 之间交互的票证。

尽管您通常会使用事件监听器来监听用户交互，但有时您确实需要扩展 View 类以构建自定义组件。也许，您想扩展 [Button](#) 类来丰富某些内容的样式。在这种情况下，您将能够使用该类的[事件处理程序](#)为类定义默认事件行为。

事件监听器

事件监听器是 [View](#) 类中包含一个回调方法的接口。当用户与 UI 项目之间的交互触发已注册此视图的监听器时，Android 框架将调用这些方法。

各事件监听器接口包含的回调方法如下：

`onClick()`

在 [View.OnClickListener](#) 中。当用户触摸项目（处于触摸模式下）时，或者使用导航键或轨迹球聚焦于项目，然后按适用的“Enter”键或按下轨迹球时，将调用此方法。

`onLongClick()`

在 [View.OnLongClickListener](#) 中。当用户触摸并按住项目（处于触摸模式下）时，或者使用导航键或轨迹球聚焦于项目，然后按住适用的“Enter”键或按住轨迹球（持续一秒钟）时，将调用此方法。

`onFocusChange()`

在 [View.OnFocusChangeListener](#) 中。当用户使用导航键或轨迹球导航到或远离项目时，将调用此方法。

`onKey()`

在 [View.OnKeyListener](#) 中。当用户聚焦于项目并按下或释放设备上的硬按键时，将调用此方法。

`onTouch()`

在 [View.OnTouchListener](#) 中。当用户执行可视为触摸事件的操作时，其中包括按下、释放或屏幕上的任何移动手势（在项目边界内），将调用此方法。

`onCreateContextMenu()`

在 `View.OnCreateContextMenuListener` 中。当（因持续“长按”而）生成上下文菜单时，将调用此方法。请参见[菜单开发者指南](#)中有
关上下文菜单的阐述。

这些方法是其相应接口的唯一成员。要定义其中一个方法并处理事件，请在 Activity 中实现嵌套接口或将其定义为匿名类。然后，将实现的实
例传递给相应的 `View.set...Listener()` 方法。（例如，调用 `setOnClickListener()` 并向其传递 `OnClickListener` 实现。）

以下示例显示了如何为按钮注册点击侦听器。

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

您可能还会发现，将 `OnClickListener` 作为 Activity 的一部分来实现更为方便。这样可以避免加载额外的类和分配对象。例如：

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
        // do something when the button is clicked
    }
    ...
}
```

请注意，上述示例中的 `onClick()` 回调没有返回值，但是其他某些事件侦听器方法必须返回布尔值。具体原因取决于事件。对于这几个事件
侦听器，必须返回布尔值的原因如下：

- `onLongClick()`：此方法返回一个布尔值，表示您是否已处理完事件，以及是否应该将它继续传下去。也就是说，返回 `true` 表示您已经处
理事件且事件应就此停止；如果您尚未处理事件和/或事件应该继续传递给其他任何点击侦听器，则返回 `false`。
- `onKey()`：此方法返回一个布尔值，表示您是否已处理完事件，以及是否应该将它继续传下去。也就是说，返回 `true` 表示您已经处理事件
且事件应就此停止；如果您尚未处理事件和/或事件应该继续传递给其他任何按键侦听器，则返回 `false`。
- `onTouch()`：此方法返回一个布尔值，表示侦听器是否处理完此事件。重要的是，此事件可以拥有多个分先后顺序的操作。因此，如果在
收到关闭操作事件时返回 `false`，则表示您并未处理完此事件，而且对其后续操作也不感兴趣。因此，您无需执行事件内的任何其他操作，
如手势或最终操作事件。

请记住，硬按键事件总是传递给目前处于焦点的视图对象。它们从视图层次结构的顶部开始分派，然后向下，直至到达合适的目的地。如果您的
视图对象（或视图对象的子项）目前具有焦点，那么您可以看到事件经由 `dispatchKeyEvent()` 方法的分派过程。除了通过视图捕获按键事件，
您还可以使用 `onKeyDown()` 和 `onKeyUp()` 接收 Activity 内部的所有事件。

此外，考虑应用的文本输入时，请记住：许多设备只有软件输入法。此类方法无需基于按键；某些可能使用语音输入、手写等。尽管输入法提供
了类似键盘的界面，但它通常**不会**触发 `onKeyDown()` 系列的事件。除非您要将应用限制为带有硬键盘的设备，否则，在设计 UI 时切勿要求
必须通过特定按键进行控制。特别是，当用户按下返回键时，不要依赖这些方法验证输入；请改用 `IME_ACTION_DONE` 等操作让输入法知晓您的
应用预计会作何反应，这样，可以通过一种有意义的方式更改其 UI。不要推断软件输入法应如何工作，只要相信它能够为应用提供已设置格式
的文本即可。

注：Android 会先调用事件处理器，然后从类定义调用合适的默认处理器。因此，从这些事件侦听器返回 `true` 会停止将事件传播到其

他事件侦听器，还会阻止回调视图对象中的默认事件处理程序。因此，在返回 `true` 时请确保您要终止事件。

事件处理程序

如果您从视图构建自定义组件，则将能够定义几种用作默认事件处理程序的回调方法。在有关[自定义组件](#)的文档中，您将了解某些用于事件处理的常见回调，其中包括：

- `onKeyDown(int, KeyEvent)`：在发生新的按键事件时调用
- `onKeyUp(int, KeyEvent)`：在发生按键弹起事件时调用
- `onTrackballEvent(MotionEvent)`：在发生轨迹球运动事件时调用
- `onTouchEvent(MotionEvent)`：在发生触摸屏运动事件时调用
- `onFocusChanged(boolean, int, Rect)`：在视图获得或失去焦点时调用

还有一些其他方法值得您注意，尽管它们并非 `View` 类的一部分，但可能会直接影响所能采取的事件处理方式。因此，在管理布局内更复杂的事件时，请考虑使用以下其他方法：

- `Activity.dispatchTouchEvent(MotionEvent)`：此方法允许 `Activity` 在分派给窗口之前截获所有触摸事件。
- `ViewGroup.onInterceptTouchEvent(MotionEvent)`：此方法允许 `ViewGroup` 监视分派给子视图的事件。
- `ViewParent.requestDisallowInterceptTouchEvent(boolean)`：对父视图调用此方法表明不应使用 `onInterceptTouchEvent(MotionEvent)` 截获触摸事件。

触摸模式

当用户使用方向键或轨迹球导航用户界面时，必须聚焦到可操作项目上（如按钮），以便用户看到将接受输入的对象。但是，如果设备具有触摸功能且用户开始通过触摸界面与之交互，则不再需要突出显示项目或聚焦到特定视图对象上。因此，有一种交互模式称为“触摸模式”。

对于支持触摸功能的设备，当用户触摸屏幕时，设备会立即进入触摸模式。自此以后，只有 `isFocusableInTouchMode()` 为 `true` 的视图才可聚焦，如文本编辑小部件。其他可触摸的视图（如按钮）在用户触摸时不会获得焦点；按下时它们只是触发点击侦听器。

无论何时，只要用户点击方向键或滚动轨迹球，设备就会退出触摸模式并找到一个视图使其获得焦点。现在，用户可在不触摸屏幕的情况下继续与用户界面交互。

整个系统（所有窗口和 `Activity`）都将保持触摸模式状态。要查询当前状态，您可以调用 `isInTouchMode()` 来检查设备目前是否处于触摸模式。

处理焦点

该框架将处理例行焦点移动来响应用户输入。其中包括在视图被移除或隐藏时或在新视图变得可用时更改焦点。视图对象表示愿意通过 `isFocusable()` 方法获得焦点。要设置视图能否获得焦点，请调用 `setFocusable()`。在触摸模式中，您可以使用 `isFocusableInTouchMode()` 查询视图是否允许聚焦，而且可以使用 `setFocusableInTouchMode()` 对此进行更改。

焦点移动所使用的算法会查找指定方向上距离最近的元素。在极少数情况下，默认算法可能与开发者的期望行为不一致。在这些情况下，您可以在布局文件中显式重写以下 XML 属性：`nextFocusDown`、`nextFocusLeft`、`nextFocusRight` 和 `nextFocusUp`。将其中一个属性添加到失去焦点的视图。将属性的值定义为应该聚焦的视图的 ID。例如：

```
<LinearLayout
    android:orientation="vertical"
    ...
    >
    <Button android:id="@+id/top"
        android:nextFocusUp="@+id/bottom"
        ...
    />
    <Button android:id="@+id/bottom"
        android:nextFocusDown="@+id/top"
        ...
    />
</LinearLayout>
```

一般来说，在此垂直布局中，从第一个按钮向上导航或从第二个按钮向下导航，焦点都不会移到任何其他位置。现在，顶部按钮已将底部按钮定义为 `nextFocusUp`（反之亦然），因而导航焦点将自上而下和自下而上循环往复。

若要将一个视图声明为在 UI 中可聚焦（传统上并非如此），请在布局声明中将 `android:focusable` XML 属性添加到该视图。将值设置为 `true`。此外，您还可以使用 `android:focusableInTouchMode` 将视图声明为在触摸模式下可聚焦。

要请求要获得焦点的特定视图，请调用 `requestFocus()`。

要侦听焦点事件（视图获得或失去焦点时会收到通知），请使用 `onFocusChange()`，如上文的[事件侦听器](#)部分中所述。

菜单

本文内容

- › [使用 XML 定义菜单](#)
- › [创建选项菜单](#)
 - › [处理点击事件](#)
 - › [在运行时更改菜单项](#)
- › [创建上下文菜单](#)
 - › [创建浮动上下文菜单](#)
 - › [使用上下文操作模式](#)
- › [创建弹出菜单](#)
 - › [处理点击事件](#)
- › [创建菜单组](#)
 - › [使用可选中的菜单项](#)
- › [添加基于 Intent 的菜单项](#)
 - › [允许将 Activity 添加到其他菜单中](#)

关键类

- › [Menu](#)
- › [MenuItem](#)
- › [ContextMenu](#)
- › [ActionMode](#)

另请参阅

- › [添加应用栏](#)
- › [菜单资源](#)
- › [从此告别菜单按钮](#)

菜单是许多应用类型中常见的用户界面组件。要提供熟悉而一致的用户体验，您应使用 [Menu API](#) 呈现 Activity 中的用户操作和其他选项。

从 Android 3.0 (API 级别 11) 开始，采用 Android 技术的设备不必再提供一个专用“菜单”按钮。随着这种改变，Android 应用需摆脱对包含 6 个项目的传统菜单面板的依赖，取而代之的是要提供一个应用栏来呈现常见的用户操作。

尽管某些菜单项的设计和用户体验已发生改变，但定义一系列操作和选项所使用的语义仍是以 [Menu API](#) 为基础。本指南将介绍所有 Android 版本系统中三种基本菜单或操作呈现效果的创建方法：

选项菜单和应用栏

[选项菜单](#)是某个 Activity 的主菜单项，供您放置对应用产生全局影响的操作，如“搜索”、“撰写电子邮件”和“设置”。

请参阅[创建选项菜单](#)部分。

上下文菜单和上下文操作模式

上下文菜单是用户长按某一元素时出现的[浮动菜单](#)。它提供的操作将影响所选内容或上下文框架。

[上下文操作模式](#)在屏幕顶部栏显示影响所选内容的操作项目，并允许用户选择多项。

请参阅[创建上下文菜单](#)部分。

弹出菜单

弹出菜单将以垂直列表形式显示一系列项目，这些项目将锚定到调用该菜单的视图中。它特别适用于提供与特定内容相关的大量操作，或者为命令的另一部分提供选项。弹出菜单中的操作**不会**直接影响对应的内容，而上下文操作则会影响。相反，弹出菜单适用于与您 Activity 中的内容区域相关的扩展操作。

请参阅[创建弹出菜单](#)部分。

使用 XML 定义菜单

对于所有菜单类型，Android 提供了标准的 XML 格式来定义菜单项。您应在 XML [菜单资源](#)中定义菜单及其所有项，而不是在 Activity 的代码中构建菜单。定义后，您可以在 Activity 或片段中扩充菜单资源（将其作为 `Menu` 对象加载）。

使用菜单资源是一种很好的做法，原因如下：

- 更易于使用 XML 可视化菜单结构
- 将菜单内容与应用的行为代码分离
- 允许您利用[应用资源](#)框架，为不同的平台版本、屏幕尺寸和其他配置创建备用菜单配置

要定义菜单，请在项目 `res/menu/` 目录内创建一个 XML 文件，并使用以下元素构建菜单：

`<menu>`

定义 `Menu`，即菜单项的容器。`<menu>` 元素必须是该文件的根节点，并且能够包含一个或多个 `<item>` 和 `<group>` 元素。

`<item>`

创建 `MenuItem`，此元素表示菜单中的一项，可能包含嵌套的 `<menu>` 元素，以便创建子菜单。

`<group>`

`<item>` 元素的不可见容器（可选）。它支持您对菜单项进行分类，使其共享活动状态和可见性等属性。如需了解详细信息，请参阅[创建菜单组](#)部分。

以下是名为 `game_menu.xml` 的菜单示例：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

`<item>` 元素支持多个属性，您可使用这些属性定义项目的外观和行为。上述菜单中的项目包括以下属性：

`android:id`

项目特有的资源 ID，让应用能够在用户选择项目时识别该项目。

`android:icon`

引用一个要用作项目图标的可绘制对象。

`android:title`

引用一个要用作项目标题的字符串。

`android:showAsAction`

指定此项应作为操作项目显示在应用栏中的时间和方式。

这些是您应使用的最重要属性，但还有许多其他属性。有关所有受支持属性的信息，请参阅[菜单资源](#)文档。

您可以通过以 `<item>` 子元素的形式添加 `<menu>` 元素，向任何菜单（子菜单除外）中的某个菜单项添加子菜单。当应用具有大量可按主题进行组织的功能时，类似于 PC 应用程序菜单栏中的菜单项（“文件”、“编辑”、“视图”等），子菜单非常有用。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

要在 Activity 中使用菜单，您需要使用 `MenuInflater.inflate()` 扩充菜单资源（将 XML 资源转换为可编程对象）。在下文中，您将了解如何扩充每种类型的菜单。

创建选项菜单

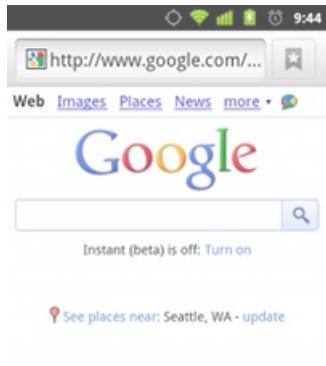


图 1. Android 2.3 系统上浏览器中的选项菜单。

在选项菜单中，您应当包括与当前 Activity 上下文相关的操作和其他选项，如“搜索”、“撰写电子邮件”和“设置”。

选项菜单中的项目在屏幕上的显示位置取决于您开发的应用所适用的 Android 版本：

- 如果您开发的应用适用于 **Android 2.3.x (API 级别 10) 或更低版本**，则当用户按“菜单”按钮时，选项菜单的内容会出现在屏幕底部，如图 1 所示。打开时，第一个可见部分是图标菜单，其中包含多达 6 个菜单项。如果菜单包括 6 个以上项目，则 Android 会将第六项和其余项目放入溢出菜单。用户可以通过选择“更多”打开该菜单。
- 如果您开发的应用适用于 **Android 3.0 (API 级别 11) 及更高版本**，则选项菜单中的项目将出现在应用栏中。默认情况下，系统会将所有项目均放入操作溢出菜单中。用户可以使用应用栏右侧的操作溢出菜单图标（或者，通过按设备“菜单”按钮（如有））显示操作溢出菜单。要支持快速访问重要操作，您可以将 `android:showAsAction="ifRoom"` 添加到对应的 `<item>` 元素，从而将几个项目提升到应用栏中（请参阅图 2）。

如需了解有关操作项目和其他应用栏行为的详细信息，请参阅[添加应用栏](#)培训课程。



图 2. Honeycomb Gallery 应用中的应用栏，其中显示了导航标签和相机操作项目（以及操作溢出菜单按钮）。

您可以通过 `Activity` 子类或 `Fragment` 子类为选项菜单声明项目。如果您的 `Activity` 和片段均为选项菜单声明项目，则这些项目将合并到 UI 中。系统将首先显示 `Activity` 的项目，随后按每个片段添加到 `Activity` 中的顺序显示各片段的项目。如有必要，您可以使用 `android:orderInCategory` 属性，对需要移动的每个 `<item>` 中的菜单项重新排序。

要为 `Activity` 指定选项菜单，请重写 `onCreateOptionsMenu()`（片段会提供自己的 `onCreateOptionsMenu()` 回调）。通过此方法，您可以将菜单资源（使用 XML 定义）扩充到回调中提供的 `Menu` 中。例如：

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.game_menu, menu);  
    return true;  
}
```

此外，您还可以使用 `add()` 添加菜单项，并使用 `findItem()` 检索项目，以便使用 `MenuItem` API 修改其属性。

如果您开发的应用适用于 Android 2.3.x 及更低版本，则当用户首次打开选项菜单时，系统会调用 `onCreateOptionsMenu()` 来创建该菜单。如果您开发的应用适用于 Android 3.0 及更高版本，则系统将在启动 `Activity` 时调用 `onCreateOptionsMenu()`，以便向应用栏显示项目。

处理点击事件

用户从选项菜单中选择项目（包括应用栏中的操作项目）时，系统将调用 `Activity` 的 `onOptionsItemSelected()` 方法。此方法将传递所选的 `MenuItem`。您可以通过调用 `getItemId()` 方法来识别项目，该方法将返回菜单项的唯一 ID（由菜单资源中的 `android:id` 属性定义，或通过提供给 `add()` 方法的整型数定义）。您可以将此 ID 与已知的菜单项匹配，以执行适当的操作。例如：

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle item selection  
    switch (item.getItemId()) {  
        case R.id.new_game:  
            newGame();  
            return true;  
        case R.id.help:  
            showHelp();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

成功处理菜单项后，系统将返回 `true`。如果未处理菜单项，则应调用 `onOptionsItemSelected()` 的超类实现（默认实现将返回 `false`）。

如果 `Activity` 包括片段，则系统将依次为 `Activity` 和每个片段（按照每个片段的添加顺序）调用 `onOptionsItemSelected()`，直到有一个返回结果为 `true` 或所有片段均调用完毕为止。

提示：Android 3.0 新增了一项功能，支持您在 XML 中使用 `android:onClick` 属性为菜单项定义点击行为。该属性的值必须是 `Activity` 使用菜单定义的方法的名称。该方法必须是公用的，且接受单个 `MenuItem` 参数；当系统调用此方法时，它会传递所选的菜单项。如需了解详细信息和示例，请参阅[菜单资源](#)文档。

提示：如果应用包含多个 `Activity`，且其中某些 `Activity` 提供相同的选项菜单，则可考虑创建一个仅实现 `onCreateOptionsMenu()` 和

`onOptionsItemSelected()` 方法的 Activity。然后，为每个应共享相同选项菜单的 Activity 扩展此类。通过这种方式，您可以管理一个用于处理菜单操作的代码集，且每个子级类均会继承菜单行为。若要将菜单项添加到一个子级 Activity，请重写该 Activity 中的 `onCreateOptionsMenu()`。调用 `super.onCreateOptionsMenu(menu)`，以便创建原始菜单项，然后使用 `menu.add()` 添加新菜单项。此外，您还可以替代各个菜单项的超类行为。

在运行时更改菜单项

系统调用 `onCreateOptionsMenu()` 后，将保留您填充的 `Menu` 实例。除非菜单由于某些原因而失效，否则不会再次调用 `onCreateOptionsMenu()`。但是，您仅应使用 `onCreateOptionsMenu()` 来创建初始菜单状态，而不应使用它在 Activity 生命周期中执行任何更改。

如需根据在 Activity 生命周期中发生的事件修改选项菜单，则可通过 `onPrepareOptionsMenu()` 方法执行此操作。此方法向您传递 `Menu` 对象（因为该对象目前存在），以便您能够对其进行修改，如添加、移除或禁用项目。（此外，片段还提供 `onPrepareOptionsMenu()` 回调。）

在 Android 2.3.x 及更低版本中，每当用户打开选项菜单时（按“菜单”按钮），系统均会调用 `onPrepareOptionsMenu()`。

在 Android 3.0 及更高版本中，当菜单项显示在应用栏中时，选项菜单被视为始终处于打开状态。发生事件时，如果您要执行菜单更新，则必须调用 `invalidateOptionsMenu()` 来请求系统调用 `onPrepareOptionsMenu()`。

注：切勿根据目前处于焦点的 `View` 更改选项菜单中的项目。处于触摸模式（用户未使用轨迹球或方向键）时，视图无法形成焦点，因此切勿根据焦点修改选项菜单中的项目。若要为 `View` 提供上下文相关的菜单项，请使用 [上下文菜单](#)。

创建上下文菜单

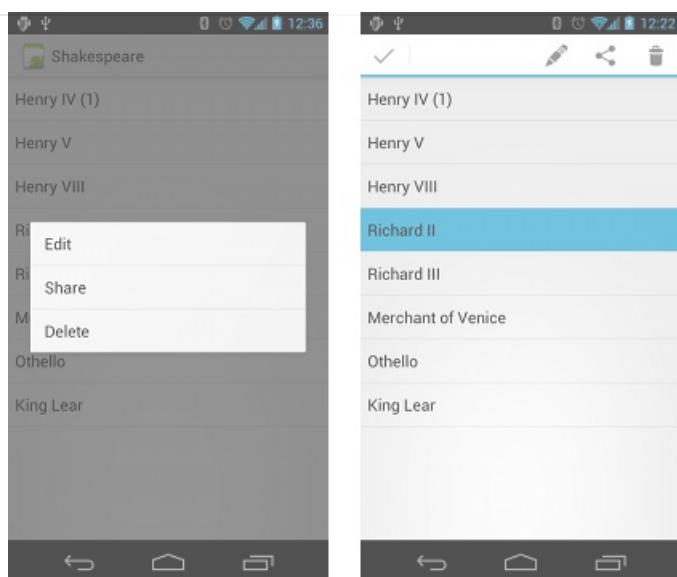


图 3. 浮动上下文菜单（左）和上下文操作栏（右）的屏幕截图。

上下文菜单提供了许多操作，这些操作影响 UI 中的特定项目或上下文框架。您可以为任何视图提供上下文菜单，但这些菜单通常用于 `ListView`、`GridView` 或用户可直接操作每个项目的其他视图集合中的项目。

提供上下文操作的方法有两种：

- 使用[浮动上下文菜单](#)。用户长按（按住）一个声明支持上下文菜单的视图时，菜单显示为菜单项的浮动列表（类似于对话框）。用户一次可对一个项目执行上下文操作。
- 使用[上下文操作模式](#)。此模式是 `ActionMode` 的系统实现，它将在屏幕顶部显示上下文操作栏，其中包括影响所选项的操作项目。当此模式处于活动状态时，用户可以同时对多项执行操作（如果应用允许）。

注：上下文操作模式可用于 Android 3.0 (API 级别 11) 及更高版本，是显示上下文操作（如果可用）的首选方法。如果应用支持低于 3.0 版本的系统，则应在这些设备上回退到浮动上下文菜单。

创建浮动上下文菜单

要提供浮动上下文菜单，请执行以下操作：

1. 通过调用 `registerForContextMenu()`，注册应与上下文菜单关联的 `View` 并将其传递给 `View`。

如果 Activity 使用 `ListView` 或 `GridView` 且您希望每个项目均提供相同的上下文菜单，请通过将 `ListView` 或 `GridView` 传递给 `registerForContextMenu()`，为上下文菜单注册所有项目。

2. 在 `Activity` 或 `Fragment` 中实现 `onCreateContextMenu()` 方法。

当注册后的视图收到长按事件时，系统将调用您的 `onCreateContextMenu()` 方法。在此方法中，您通常可通过扩充菜单资源来定义菜单项。例如：

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
                               ContextMenuItemInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.context_menu, menu);  
}
```

`MenuInflater` 允许您通过[菜单资源](#)扩充上下文菜单。回调方法参数包括用户所选的 `View`，以及一个提供有关所选项的附加信息的 `ContextMenu.ContextMenuItemInfo` 对象。如果 Activity 有多个视图，每个视图均提供不同的上下文菜单，则可使用这些参数确定要扩充的上下文菜单。

3. 实现 `onContextItemSelected()`。

用户选择菜单项时，系统将调用此方法，以便您能够执行适当的操作。例如：

```
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();  
    switch (item.getItemId()) {  
        case R.id.edit:  
            editNote(info.id);  
            return true;  
        case R.id.delete:  
            deleteNote(info.id);  
            return true;  
        default:  
            return super.onContextItemSelected(item);  
    }  
}
```

`getItemId()` 方法将查询所选菜单项的 ID，您应使用 `android:id` 属性将此 ID 分配给 XML 中的每个菜单项，如[使用 XML 定义菜单](#)部分所示。

成功处理菜单项后，系统将返回 `true`。如果未处理菜单项，则应将菜单项传递给超类实现。如果 Activity 包括片段，则 Activity 将先收到此回调。通过在未处理的情况下调用超类，系统会将事件逐一传递给每个片段中相应的回调方法（按照每个片段的添加顺序），直到返回 `true` 或 `false` 为止。[（Activity 和 android.app.Fragment 的默认实现返回 false，因此您始终应在未处理的情况下调用超类。）](#)

使用上下文操作模式

上下文操作模式是 `ActionMode` 的一种系统实现，它将用户交互的重点转到执行上下文操作上。用户通过选择项目启用此模式时，屏幕顶部将出现一个“上下文操作栏”，显示用户可对当前所选项执行的操作。启用此模式后，用户可以选择多个项目（若您允许）、取消选择项目以及继续在 Activity 内导航（在您允许的最大范围内）。当用户取消选择所有项目、按“返回”按钮或选择操作栏左侧的“完成”操作时，该操作模式将会停用，且上下文操作栏将会消失。

注：上下文操作栏不一定与应用栏相关联。尽管表面上看来上下文操作栏取代了应用栏的位置，但事实上二者独立运行。

对于提供上下文操作的视图，当出现以下两个事件（或之一）时，您通常应调用上下文操作模式：

- 用户长按视图。
- 用户选中复选框或视图内的类似 UI 组件。

应用如何调用上下文操作模式以及如何定义每个操作的行为，具体取决于您的设计。设计基本上分为两种：

- 针对单个任意视图的上下文操作。
- 针对 `ListView` 或 `GridView` 中项目组的批处理上下文操作（允许用户选择多个项目并针对所有项目执行操作）。

下文介绍每种场景所需的设置。

为单个视图启用上下文操作模式

如果希望仅当用户选择特定视图时才调用上下文操作模式，则应：

1. 实现 `ActionMode.Callback` 接口。在其回调方法中，您既可以为上下文操作栏指定操作，又可以响应操作项目的点击事件，还可以处理操作模式的其他生命周期事件。
2. 当需要显示操作栏时（例如，用户长按视图），请调用 `startActionMode()`。

例如：

1. 实现 `ActionMode.Callback` 接口：

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {  
  
    // Called when the action mode is created; startActionMode() was called  
    @Override  
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {  
        // Inflate a menu resource providing context menu items  
        MenuInflater inflater = mode.getMenuInflater();  
        inflater.inflate(R.menu.context_menu, menu);  
        return true;  
    }  
  
    // Called each time the action mode is shown. Always called after onCreateActionMode, but  
    // may be called multiple times if the mode is invalidated.  
    @Override  
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {  
        return false; // Return false if nothing is done  
    }  
  
    // Called when the user selects a contextual menu item  
    @Override  
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {  
        switch (item.getItemId()) {  
            case R.id.menu_share:  
                shareCurrentItem();  
                mode.finish(); // Action picked, so close the CAB  
                return true;  
            default:  
                return false;  
        }  
    }  
  
    // Called when the user exits the action mode  
    @Override  
    public void onDestroyActionMode(ActionMode mode) {  
        mActionMode = null;  
    }  
};
```

请注意，这些事件回调与[选项菜单](#)的回调几乎完全相同，只是其中每个回调还会传递与事件相关联的 `ActionMode` 对象。您可以使用 `ActionMode` API 对 CAB 进行各种更改，例如：使用 `setTitle()` 和 `setSubtitle()`（这对指示要选择多少个项目非常有用）修改标题和副标题。

另请注意，操作模式被销毁时，上述示例会将 `mActionMode` 变量设置为 `null`。在下一步中，您将了解如何初始化该变量，以及保存 Activity 或片段中的成员变量有何作用。

2. 调用 `startActionMode()` 以便适时启用上下文操作模式，例如：响应对 `View` 的长按操作：

```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Called when the user long-clicks on someView
    public boolean onLongClick(View view) {
        if (mActionMode != null) {
            return false;
        }

        // Start the CAB using the ActionMode.Callback defined above
        mActionMode = getActivity().startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

当您调用 `startActionMode()` 时，系统将返回已创建的 `ActionMode`。通过将其保存在成员变量中，您可以更改上下文操作栏来响应其他事件。在上述示例中，`ActionMode` 用于在启动操作模式之前检查成员是否为空，以确保当 `ActionMode` 实例已激活时不再重建该实例。

在 ListView 或 GridView 中启用批处理上下文操作

如果您在 `ListView` 或 `GridView` 中有一组项目（或 `AbsListView` 的其他扩展），且需要允许用户执行批处理操作，则应：

- 实现 `AbsListView.MultiChoiceModeListener` 接口，并使用 `setMultiChoiceModeListener()` 为视图组设置该接口。在侦听器的回调方法中，您既可以为上下文操作栏指定操作，也可以响应操作项目的点击事件，还可以处理从 `ActionMode.Callback` 接口继承的其他回调。
- 使用 `CHOICE_MODE_MULTIPLE_MODAL` 参数调用 `setChoiceMode()`。

例如：

```

ListView listView = getListView();
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position,
                                         long id, boolean checked) {
        // Here you can do something when items are selected/de-selected,
        // such as update the title in the CAB
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        // Respond to clicks on the actions in the CAB
        switch (item.getItemId()) {
            case R.id.menu_delete:
                deleteSelectedItems();
                mode.finish(); // Action picked, so close the CAB
                return true;
            default:
                return false;
        }
    }

    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate the menu for the CAB
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context, menu);
        return true;
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) {
        // Here you can make any necessary updates to the activity when
        // the CAB is removed. By default, selected items are deselected/unchecked.
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        // Here you can perform updates to the CAB due to
        // an invalidate() request
        return false;
    }
});
```

就这么简单。现在，当用户通过长按选择项目时，系统即会调用 `onCreateActionMode()` 方法，并显示包含指定操作的上下文操作栏。当上下文操作栏可见时，用户可以选择其他项目。

在某些情况下，如果上下文操作提供常用的操作项目，则您可能需要添加一个复选框或类似的 UI 元素来支持用户选择项目，这是因为他们可能没有发现长按行为。用户选中该复选框时，您可以通过使用 `setItemChecked()` 将相应的列表项设置为选中状态，以此调用上下文操作模式。

创建弹出菜单

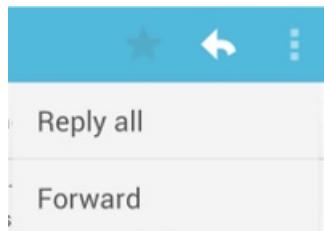


图 4. Gmail 应用中的弹出菜单，锚定到右上角的溢出按钮。

`PopupMenu` 是锚定到 `View` 的模态菜单。如果空间足够，它将显示在定位视图下方，否则显示在其上方。它适用于：

- 为与特定内容密切相关的操作提供溢出样式菜单（例如，Gmail的电子邮件标头，如图4所示）。

注：这与上下文菜单不同，后者通常用于影响所选内容的操作。对于影响所选内容的操作，请使用[上下文操作模式](#)或[浮动上下文菜单](#)。

- 提供命令语句的另一部分（例如，标记为“添加”且使用不同的“添加”选项生成弹出菜单的按钮）。
- 提供类似于[Spinner](#)且不保留永久选择的下拉菜单。

注：PopupMenu在API级别11及更高版本中可用。

如果使用XML定义菜单，则显示弹出菜单的方法如下：

- 实例化PopupMenu及其构造函数，该函数将提取当前应用的Context以及菜单应锚定到的View。
- 使用MenuInflater将菜单资源扩充到PopupMenu.getMenu()返回的Menu对象中。
- 调用PopupMenu.show()。

例如，以下是一个使用android:onClick属性显示弹出菜单的按钮：

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_overflow_holo_dark"  
    android:contentDescription="@string/descr_overflow_button"  
    android:onClick="showPopup" />
```

稍后，Activity可按照如下方式显示弹出菜单：

```
public void showPopup(View v) {  
    PopupMenu popup = new PopupMenu(this, v);  
    MenuInflater inflater = popup.getMenuInflater();  
    inflater.inflate(R.menu.actions, popup.getMenu());  
    popup.show();  
}
```

在API级别14及更高版本中，您可以将两行合并在一起，使用PopupMenu.inflate()扩充菜单。

当用户选择项目或触摸菜单以外的区域时，系统即会清除此菜单。您可使用PopupMenu.OnDismissListener侦听清除事件。

处理点击事件

要在用户选择菜单项时执行操作，您必须实现PopupMenu.OnMenuItemClickListener接口，并通过调用setOnMenuItemClickListener()将其注册到PopupMenu。用户选择项目时，系统会在接口中调用onMenuItemClick()回调。

例如：

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

创建菜单组

菜单组是指一系列具有某些共同特征的菜单项。通过菜单组，您可以：

- 使用 `setGroupVisible()` 显示或隐藏所有项目
- 使用 `setGroupEnabled()` 启用或禁用所有项目
- 使用 `setGroupCheckable()` 指定所有项目是否可选中

通过将 `<item>` 元素嵌套在菜单资源中的 `<group>` 元素内，或者通过使用 `add()` 方法指定组 ID，您可以创建组。

以下是包含组的菜单资源示例：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/menu_save"
          android:title="@string/menu_save" />
    <!-- menu group -->
    <group android:id="@+id/group_delete">
        <item android:id="@+id/menu_archive"
              android:title="@string/menu_archive" />
        <item android:id="@+id/menu_delete"
              android:title="@string/menu_delete" />
    </group>
</menu>
```

组中的项目出现在与第一项相同的级别，即：菜单中的所有三项均为同级。但是，您可以通过引用组 ID 并使用上面列出的方法，修改组中两项的特征。此外，系统也绝不会分离已分组的项目。例如，如果为每个项目声明 `android:showAsAction="ifRoom"`，则它们会同时显示在操作栏或操作溢出菜单中。

使用可选中的菜单项

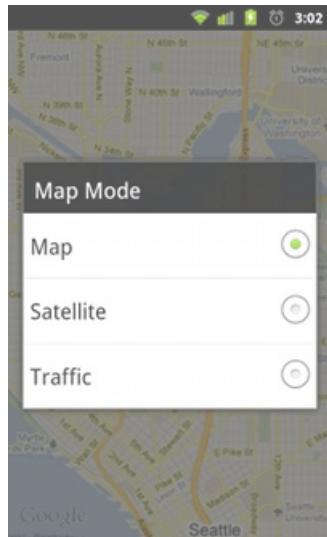


图 5. 含可选中项目的子菜单的屏幕截图。

作为启用/禁用选项的接口，菜单非常实用，既可针对独立选项使用复选框，也可针对互斥选项组使用单选按钮。图 5 显示了一个子菜单，其中的项目可使用单选按钮选中。

注：“图标菜单”（在选项菜单中）的菜单项无法显示复选框或单选按钮。如果您选择使“图标菜单”中的项目可选中，则必须在选中状态每次发生变化时交换图标和/或文本，手动指出该状态。

您可以使用 `<item>` 元素中的 `android:checkable` 属性为各个菜单项定义可选中的行为，或者使用 `<group>` 元素中的 `android:checkableBehavior` 属性为整个组定义可选中的行为。例如，此菜单组中的所有项目均可使用单选按钮选中：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item android:id="@+id/red"
              android:title="@string/red" />
        <item android:id="@+id/blue"
              android:title="@string/blue" />
    </group>
</menu>
```

`android:checkableBehavior` 属性接受以下任一选项：

`single`

组中只有一个项目可以选中（单选按钮）

`all`

所有项目均可选中（复选框）

`none`

所有项目均无法选中

您可以使用 `<item>` 元素中的 `android:checked` 属性将默认的选中状态应用于项目，并可使用 `setChecked()` 方法在代码中更改此默认状态。

选择可选中项目后，系统将调用所选项目的相应回调方法（例如，`onOptionsItemSelected()`）。此时，您必须设置复选框的状态，因为复选框或单选按钮不会自动更改其状态。您可以使用 `isChecked()` 查询项目的当前状态（正如用户选择该项目之前一样），然后使用 `setChecked()` 设置选中状态。例如：

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.vibrate:
        case R.id.dont_vibrate:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

如果未通过这种方式设置选中状态，则项目的可见状态（复选框或单选按钮）不会因为用户选择它而发生变化。如果已设置该状态，则 Activity 会保留项目的选中状态。这样一来，当用户稍后打开菜单时，您设置的选中状态将会可见。

注：可选中菜单项的使用往往因会话而异，且在应用销毁后不予保存。如果您想为用户保存某些应用设置，则应使用[共享首选项](#)存储数据。

添加基于 Intent 的菜单项

有时，您希望菜单项通过使用 `Intent` 启动 Activity（无论该 Activity 是位于您的应用还是其他应用中）。如果您知道自己要使用的 Intent，且具有启动 Intent 的特定菜单项，则可在相应的 on-item-selected 回调方法（例如，`onOptionsItemSelected()` 回调）期间使用 `startActivity()` 执行 Intent。

但是，如果不确定用户的设备是否包含可处理 Intent 的应用，则添加调用 Intent 的菜单项可能会导致该菜单项无法正常工作，这是因为 Intent 可能无法解析为 Activity。为了解决这一问题，当 Android 在设备上找到可处理 Intent 的 Activity 时，则允许您向菜单动态添加菜单项。

要根据接受 Intent 的可用 Activity 添加菜单项，请执行以下操作：

1. 使用类别 `CATEGORY_ALTERNATIVE` 和/或 `CATEGORY_SELECTED_ALTERNATIVE` 以及任何其他要求定义 Intent。
2. 调用 `Menu.addIntentOptions()`。Android 随后即会搜索能够执行 Intent 的所有应用，并将其添加到菜单中。

如果未安装可处理 Intent 的应用，则不会添加任何菜单项。

注：`CATEGORY_SELECTED_ALTERNATIVE` 用于处理屏幕上当前所选的元素。因此，只有在 `onCreateContextMenu()` 中创建菜单时，才能使用它。

例如：

```
@Override
public boolean onCreateOptionsMenu(Menu menu){
    super.onCreateOptionsMenu(menu);

    // Create an Intent that describes the requirements to fulfill, to be included
    // in our menu. The offering app must include a category value of Intent.CATEGORY_ALTERNATIVE.
    Intent intent = new Intent(null, dataUri);
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);

    // Search and populate the menu with acceptable offering applications.
    menu.addIntentOptions(
        R.id.intent_group, // Menu group to which new items will be added
        0, // Unique item ID (none)
        0, // Order for the items (none)
        this.getComponentName(), // The current activity name
        null, // Specific items to place first (none)
        intent, // Intent created above that describes our requirements
        0, // Additional flags to control items (none)
        null); // Array of MenuItem objects that correlate to specific items (none)

    return true;
}
```

如果发现 Activity 提供的 Intent 过滤器与定义的 Intent 匹配，则会添加菜单项，并使用 Intent 过滤器 `android:label` 中的值作为菜单项标

题，使用应用图标作为菜单项图标。`addIntentOptions()` 方法将返回已添加的菜单项数量。

注：调用 `addIntentOptions()` 方法时，它将使用第一个参数中指定的菜单组替代所有菜单项。

允许将 Activity 添加到其他菜单中

此外，您还可以为其他应用提供您的 Activity 服务，以便您的应用能够包含在其他应用的菜单中（与上述角色相反）。

要包含在其他应用菜单中，您需要按常规方式定义 Intent 过滤器，但请确保为 Intent 过滤器类别添加 `CATEGORY_ALTERNATIVE` 和/或 `CATEGORY_SELECTED_ALTERNATIVE` 值。例如：

```
<intent-filter label="@string/resize_image">
    ...
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    ...
</intent-filter>
```

请仔细阅读 [Intent 和 Intent 过滤器](#) 文档中更多有关编写 Intent 过滤器的内容。

有关使用此方法的应用示例，请参阅[记事本](#)示例代码。

设置

本文内容

- › [概览](#)
- › [首选项](#)
- › [使用 XML 定义首选项](#)
- › [创建设置组](#)
- › [使用 Intent](#)
- › [创建首选项 Activity](#)
- › [使用首选项片段](#)
- › [设置默认值](#)
- › [使用首选项标头](#)
 - › [创建标头文件](#)
 - › [显示标头](#)
 - › [使用首选项标头支持旧版本](#)
- › [读取首选项](#)
 - › [侦听首选项变更](#)
 - › [管理网络使用情况](#)
 - › [构建自定义首选项](#)
 - › [指定用户界面](#)
 - › [保存设置的值](#)
 - › [初始化当前值](#)
 - › [提供默认值](#)
 - › [保存和恢复首选项的状态](#)

关键类

- › [Preference](#)
- › [PreferenceActivity](#)
- › [PreferenceFragment](#)

另请参阅

- › [设置设计指南](#)

应用通常包括允许用户修改应用特性和行为的设置。例如，有些应用允许用户指定是否启用通知，或指定应用与云端同步数据的频率。

若要为应用提供设置，您应该使用 Android 的 [Preference API](#) 构建一个与其他 Android 应用中的用户体验一致的界面（包括系统设置）。本文旨在介绍如何使用 [Preference API](#) 构建应用设置。

设置设计

有关如何设计设置的信息，请阅读[设置设计指南](#)。

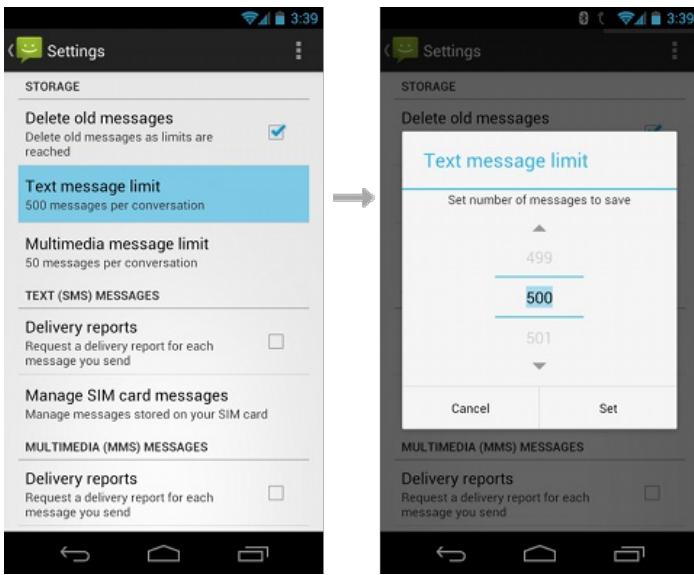


图 1. 来自 Android 信息应用的设置的屏幕截图。选择由 `Preference` 定义的项目将打开一个用于更改设置的界面。

概览

设置是使用您在 XML 文件中声明的 `Preference` 类的各种子类构建而成，而不是使用 `View` 对象构建用户界面。

`Preference` 对象是单个设置的构建基块。每个 `Preference` 均作为项目显示在列表中，并提供适当的 UI 供用户修改设置。例如，`CheckBoxPreference` 可创建一个列表项用于显示复选框，`ListPreference` 可创建一个项目用于打开包含选择列表的对话框。

您添加的每个 `Preference` 都有一个相应的键值对，可供系统用来将设置保存在应用设置的默认 `SharedPreferences` 文件中。当用户更改设置时，系统会为您更新 `SharedPreferences` 文件中的相应值。您只应在需要读取值以根据用户设置确定应用的行为时，才与关联的 `SharedPreferences` 文件直接交互。

为每个设置保存在 `SharedPreferences` 中的值可能是以下数据类型之一：

- 布尔值
- 浮点型
- 整型
- 长整型
- 字符串
- 字符串 `Set`

由于应用的设置 UI 是使用 `Preference` 对象（而非 `View` 对象）构建而成，因此您需要使用专门的 `Activity` 或 `Fragment` 子类显示列表设置：

- 如果应用支持早于 3.0 (API 级别 10 及更低级别) 的 Android 版本，则您必须将 Activity 构建为 `PreferenceActivity` 类的扩展。
- 对于 Android 3.0 及更高版本，您应改用传统 `Activity`，以托管可显示应用设置的 `PreferenceFragment`。但是，如果您拥有多组设置，则还可以使用 `PreferenceActivity` 为大屏幕创建双窗格布局。

创建首选项 `Activity` 和使用首选项片段部分将讨论如何设置 `PreferenceActivity` 以及 `PreferenceFragment` 实例。

首选项

所有应用设置均由 `Preference` 类的特定子类表示。每个子类均包括一组核心属性，允许您指定设置标题和默认值等内容。此外，每个子类还提供自己的专用属性和用户界面。例如，图 1 显示的是“信息”应用的设置屏幕截图。设置屏幕中的每个列表项均由不同的 `Preference` 对象提供支持。

一些最常用的首选项如下：

`CheckBoxPreference`

显示一个包含已启用或已停用设置复选框的项目。保存的值是布尔型（如果选中则为 `true`）。

ListPreference

打开一个包含单选按钮列表的对话框。保存的值可以是任一受支持的值类型（如上所列）。

EditTextPreference

打开一个包含 `EditText` 小部件的对话框。保存的值是 `String`。

有关所有其他子类及其对应属性的列表，请参阅 [Preference](#) 类。

当然，内置类不能满足所有需求，您的应用可能需要更专业化的内容。例如，该平台目前不提供用于选取数字或日期的 [Preference](#) 类。因此，您可能需要定义自己的 [Preference](#) 子类。如需有关执行此操作的帮助，请参阅[构建自定义首选项](#)部分。

使用 XML 定义首选项

虽然您可以在运行时实例化新的 [Preference](#) 对象，不过您还是应该使用 [Preference](#) 对象的层次结构在 XML 中定义设置列表。使用 XML 文件定义设置的集合是首选方法，因为该文件提供了一个便于更新的易读结构。此外，应用的设置通常是预先确定的，不过您仍可在运行时修改此集合。

每个 [Preference](#) 子类均可以使用与类名（如 `<CheckBoxPreference>`）匹配的 XML 元素来声明。

您必须将 XML 文件保存在 `res/xml/` 目录中。尽管您可以随意命名该文件，但它通常命名为 `preferences.xml`。您通常只需一个文件，因为层次结构中的分支（可打开各自的设置列表）是使用 `PreferenceScreen` 的嵌套实例声明的。

注：若要为设置创建多窗格布局，则需要为每个片段提供单独的 XML 文件。

XML 文件的根节点必须是一个 `PreferenceScreen` 元素。您可以在此元素内添加每个 [Preference](#)。在 `<PreferenceScreen>` 元素内添加的每个子项均将作为单独的项目显示在设置列表中。

例如：

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="pref_sync"
        android:title="@string/pref_sync"
        android:summary="@string/pref_sync_summ"
        android:defaultValue="true" />
    <ListPreference
        android:dependency="pref_sync"
        android:key="pref_syncConnectionType"
        android:title="@string/pref_syncConnectionType"
        android:dialogTitle="@string/pref_syncConnectionType"
        android:entries="@array/pref_syncConnectionTypes_entries"
        android:entryValues="@array/pref_syncConnectionTypes_values"
        android:defaultValue="@string/pref_syncConnectionTypes_default" />
</PreferenceScreen>
```

在此示例中，有一个 `CheckBoxPreference` 和 `ListPreference`。这两项均包括以下三个属性：

android:key

对于要保留数据值的首选项，必须拥有此属性。它指定系统在将此设置的值保存在 [SharedPreferences](#) 中时所用的唯一键（字符串）。

不需要此属性的唯一情形是：首选项是 `PreferenceCategory` 或 `PreferenceScreen`，或者首选项指定要调用的 `Intent`（使用 `<intent>` 元素）或要显示的 `Fragment`（使用 `android:fragment` 属性）。

android:title

此属性为设置提供用户可见的名称。

android:defaultValue

此属性指定系统应该在 [SharedPreferences](#) 文件中设置的初始值。您应该为所有设置提供默认值。

有关所有其他受支持属性的信息，请参阅 [Preference](#)（和相应子类）文档。

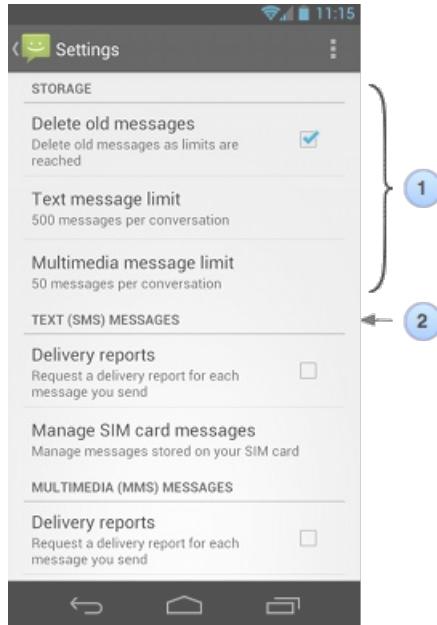


图 2. 带标题的设置类别。

1. 类别由 `<PreferenceCategory>` 元素指定。
2. 标题由 `android:title` 属性指定。

当设置列表超过 10 项时，您可能需要添加标题来定义设置组或在单独的屏幕中显示这些组。下文将介绍这些选项。

创建设置组

如果您提供的列表包含 10 项或更多设置，则用户可能难以浏览、理解和处理这些设置。若要弥补这一点，您可以将部分或全部设置分成若干组，从而有效地将一个长列表转化为多个短列表。可以通过下列两种方法之一提供一组相关设置：

- [使用标题](#)
- [使用子屏幕](#)

您可以使用其中一种或两种分组方法来组织应用的设置。决定要使用的方法以及如何拆分设置时，应遵循 Android 设计的[设置指南](#)中的准则。

使用标题

若要以分隔线分隔两组设置并为其提供标题（如图 2 所示），请将每组 [Preference](#) 对象放入 [PreferenceCategory](#) 内。

例如：

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory>
        android:title="@string/pref_sms_storage_title"
        android:key="pref_key_storage_settings">
            <CheckBoxPreference
                android:key="pref_key_auto_delete"
                android:summary="@string/pref_summary_auto_delete"
                android:title="@string/pref_title_auto_delete"
                android:defaultValue="false" ... />
            <Preference
                android:key="pref_key_sms_delete_limit"
                android:dependency="pref_key_auto_delete"
                android:summary="@string/pref_summary_delete_limit"
                android:title="@string/pref_title_sms_delete" ... />
            <Preference
                android:key="pref_key_mms_delete_limit"
                android:dependency="pref_key_auto_delete"
                android:summary="@string/pref_summary_delete_limit"
                android:title="@string/pref_title_mms_delete" ... />
        </PreferenceCategory>
        ...
    </PreferenceScreen>
```

使用子屏幕

若要将设置组放入子屏幕（如图 3 所示），请将 `Preference` 对象组放入 `PreferenceScreen` 内。

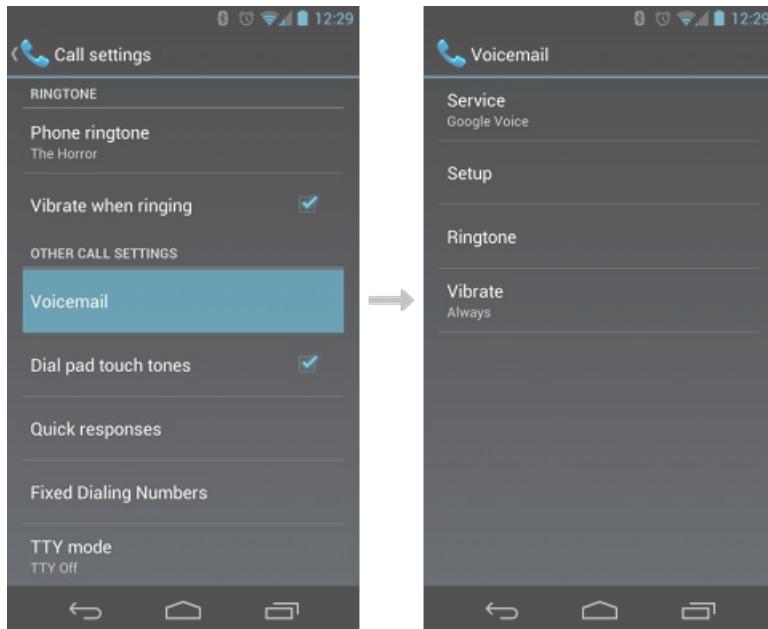


图 3. 设置子屏幕。`<PreferenceScreen>` 元素创建的项目选中后，即会打开一个单独的列表来显示嵌套设置。

例如：

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- opens a subscreen of settings -->
    <PreferenceScreen
        android:key="button_voicemail_category_key"
        android:title="@string/voicemail"
        android:persistent="false">
        <ListPreference
            android:key="button_voicemail_provider_key"
            android:title="@string/voicemail_provider" ... />
        <!-- opens another nested subscreen -->
        <PreferenceScreen
            android:key="button_voicemail_setting_key"
            android:title="@string/voicemail_settings"
            android:persistent="false">
            ...
        </PreferenceScreen>
        <RingtonePreference
            android:key="button_voicemail_ringtone_key"
            android:title="@string/voicemail_ringtone_title"
            android:ringtoneType="notification" ... />
        ...
    </PreferenceScreen>
    ...
</PreferenceScreen>
```

使用 Intent

在某些情况下，您可能需要首选项来打开不同的 Activity（而不是网络浏览器等设置屏幕）或查看网页。要在用户选择首选项时调用 [Intent](#)，请将`<intent>`元素添加为相应`<Preference>`元素的子元素。

例如，您可以按如下方法使用首选项打开网页：

```
<Preference android:title="@string/prefs_web_page" >
    <intent android:action="android.intent.action.VIEW"
        android:data="http://www.example.com" />
</Preference>
```

您可以使用以下属性创建隐式和显式 Intent：

`android:action`

要分配的操作（按照[setAction\(\)](#)方法）。

`android:data`

要分配的数据（按照[setData\(\)](#)方法）。

`android:mimeType`

要分配的 MIME 类型（按照[setType\(\)](#)方法）。

`android:targetClass`

组件名称的类部分（按照[setComponent\(\)](#)方法）。

`android:targetPackage`

组件名称的软件包部分（按照[setComponent\(\)](#)方法）。

创建首选项 Activity

要在 Activity 中显示您的设置，请扩展 [PreferenceActivity](#) 类。这是传统 [Activity](#) 类的扩展，该类根据 [Preference](#) 对象的层次结构显示设置列表。当用户进行更改时，[PreferenceActivity](#) 会自动保留与每个 [Preference](#) 相关的设置。

注：如果您在开发针对 Android 3.0 及更高版本的应用，则应改为使用 `PreferenceFragment`。转到下文有关[使用首选项片段](#)的部分。

请记住最重要的一点，就是不要在 `onCreate()` 回调期间加载视图的布局。相反，请调用 `addPreferencesFromResource()` 以将在 XML 文件中声明的首选项添加到 Activity。例如，一个能够正常工作的 `PreferenceActivity` 至少需要如下代码：

```
public class SettingsActivity extends PreferenceActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.preferences);  
    }  
}
```

实际上，对于某些应用而言，此代码就已足够，因为用户修改某首选项后，系统会立即将所做的更改保存到默认 `SharedPreferences` 文件中，如需检查用户的设置，可以使用您的其他应用组件读取该文件。不过，许多应用需要的代码要稍微多一点，以侦听首选项发生的变化。有关侦听 `SharedPreferences` 文件变化的信息，请参阅[读取首选项](#)部分。

使用首选项片段

如果您在开发针对 Android 3.0 (API 级别 11) 及更高版本的应用，则应使用 `PreferenceFragment` 显示 `Preference` 对象的列表。您可以将 `PreferenceFragment` 添加到任何 `Activity`，而不必使用 `PreferenceActivity`。

与仅使用上述 `Activity` 相比，无论您在构建何种 `Activity`，[片段](#)都可为应用提供一个更加灵活的体系结构。因此，我们建议您尽可能使用 `PreferenceFragment` 控制设置的显示，而不是使用 `PreferenceActivity`。

`PreferenceFragment` 的实现就像定义 `onCreate()` 方法以使用 `addPreferencesFromResource()` 加载首选项文件一样简单。例如：

```
public static class SettingsFragment extends PreferenceFragment {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Load the preferences from an XML resource  
        addPreferencesFromResource(R.xml.preferences);  
    }  
    ...  
}
```

然后，正如您对其他任何 `Fragment` 的处理一样，您可以将此片段添加到 `Activity`。例如：

```
public class SettingsActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Display the fragment as the main content.  
        getSupportFragmentManager().beginTransaction()  
            .replace(android.R.id.content, new SettingsFragment())  
            .commit();  
    }  
}
```

注：`PreferenceFragment` 没有自己的 `Context` 对象。如需 `Context` 对象，您可以调用 `getActivity()`。但请注意，只应在该片段附加到 `Activity` 时才调用 `getActivity()`。如果片段尚未附加，或在其生命周期结束期间分离，则 `getActivity()` 将返回 null。

设置默认值

您创建的首选项可能会为应用定义一些重要行为，因此在用户首次打开应用时，您有必要使用每个 `Preference` 的默认值初始化相关的 `SharedPreferences` 文件。

首先，您必须使用 `android:defaultValue` 属性为 XML 文件中的每个 `Preference` 对象指定默认值。该值可以是适合相应 `Preference` 对象的任意数据类型。例如：

```
<!-- default value is a boolean -->
<CheckBoxPreference
    android:defaultValue="true"
    ... />

<!-- default value is a string -->
<ListPreference
    android:defaultValue="@string/pref_syncConnectionTypes_default"
    ... />
```

然后，通过应用的主 Activity（以及用户首次进入应用所藉由的任何其他 Activity）中的 `onCreate()` 方法调用 `setDefaultValues()`：

```
PreferenceManager.setDefaultValues(this, R.xml.advanced_preferences, false);
```

在 `onCreate()` 期间调用此方法可确保使用默认设置正确初始化应用，而应用可能需要读取这些设置以确定某些行为（例如，是否在蜂窝网络中下载数据）。

此方法采用三个参数：

- 应用 `Context`。
- 要为其设置默认值的首选项 XML 文件的资源 ID。
- 一个布尔值，用于指示是否应该多次设置默认值。

如果该值为 `false`，则仅当过去从未调用此方法时（或者默认值共享首选项文件中的 `KEY_HAS_SET_DEFAULT_VALUES` 为 `false` 时），系统才会设置默认值。

只要将第三个参数设置为 `false`，您便可在每次启动 Activity 时安全地调用此方法，而不必通过重置为默认值来替代用户已保存的首选项。但是，如果将它设置为 `true`，则需要使用默认值替代之前的所有值。

使用首选项标头

在极少数情况下，您可能需要设计设置，使第一个屏幕仅显示子屏幕的列表（例如在系统“设置”应用中，如图 4 和图 5 所示）。在开发针对 Android 3.0 及更高版本的此类设计时，您应该使用“标头”功能，而非使用嵌套的 `PreferenceScreen` 元素构建子屏幕。

要使用标头构建设置，您需要：

1. 将每组设置分成单独的 `PreferenceFragment` 实例。即，每组设置均需要一个单独的 XML 文件。
2. 创建 XML 标头文件，其中列出每个设置组并声明哪个片段包含对应的设置列表。
3. 扩展 `PreferenceActivity` 类以托管设置。
4. 实现 `onBuildHeaders()` 回调以指定标头文件。

使用此设计的一大好处是，在大屏幕上运行时，`PreferenceActivity` 会自动提供双窗格布局（如图 4 所示）。

即使您的应用支持早于 3.0 的 Android 版本，您仍可将应用设计为使用 `PreferenceFragment` 在较新版本的设备上呈现双窗格布局，同时仍支持较旧版本设备上传统的多屏幕层次结构（请参阅[使用首选项标头支持旧版本](#)部分）。

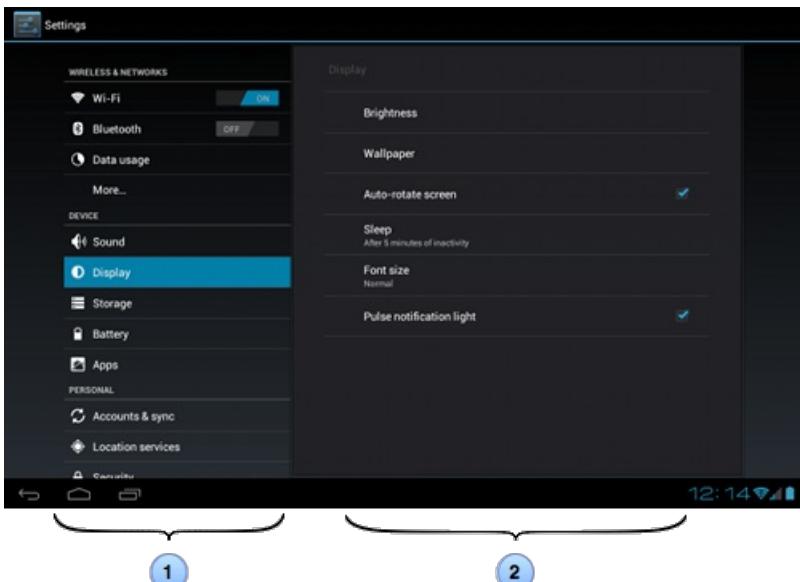


图 4. 带标头的双窗格布局。

1. 标头用 XML 标头文件定义。
2. 每组设置均由 `PreferenceFragment` (通过标头文件中的 `<header>` 元素指定) 定义。

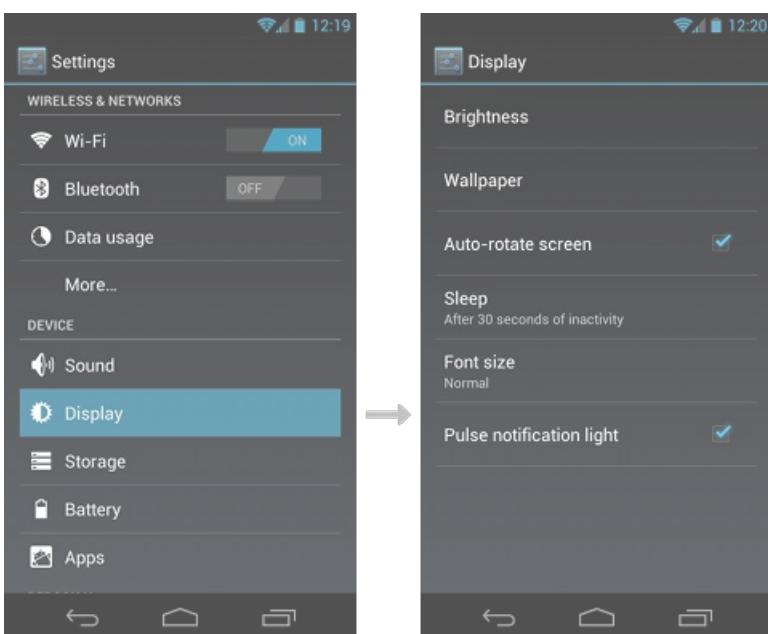


图 5. 带设置标头的手机设备。选择项目后，相关的 `PreferenceFragment` 将替换标头。

创建标头文件

标头列表中的每组设置均由根 `<preference-headers>` 元素内的单个 `<header>` 元素指定。例如：

```

<?xml version="1.0" encoding="utf-8"?>
<preference-headers xmlns:android="http://schemas.android.com/apk/res/android">
    <header
        android:fragment="com.example.prefs.SettingsActivity$SettingsFragmentOne"
        android:title="@string/prefs_category_one"
        android:summary="@string/prefs_summ_category_one" />
    <header
        android:fragment="com.example.prefs.SettingsActivity$SettingsFragmentTwo"
        android:title="@string/prefs_category_two"
        android:summary="@string/prefs_summ_category_two" >
        <!-- key/value pairs can be included as arguments for the fragment. -->
        <extra android:name="someKey" android:value="someHeaderValue" />
    </header>
</preference-headers>

```

每个标头均可使用 `android:fragment` 属性声明在用户选择该标头时应打开的 `PreferenceFragment` 实例。

`<extras>` 元素允许您使用 `Bundle` 将键值对传递给片段。该片段可以通过调用 `getArguments()` 检索参数。您向该片段传递参数的原因可能有很多，不过一个重要原因是，要对每个组重复使用 `PreferenceFragment` 的相同子类，而且要使用参数来指定该片段应加载哪些首选项 XML 文件。

例如，当每个标头均使用 "settings" 键定义 `<extra>` 参数时，则可以对多个设置组重复使用以下片段：

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String settings = getArguments().getString("settings");
        if ("notifications".equals(settings)) {
            addPreferencesFromResource(R.xml.settings_wifi);
        } else if ("sync".equals(settings)) {
            addPreferencesFromResource(R.xml.settings_sync);
        }
    }
}
```

显示标头

要显示首选项标头，您必须实现 `onBuildHeaders()` 回调方法并调用 `loadHeadersFromResource()`。例如：

```
public class SettingsActivity extends PreferenceActivity {
    @Override
    public void onBuildHeaders(List<Header> target) {
        loadHeadersFromResource(R.xml.preference_headers, target);
    }
}
```

当用户从标头列表中选择一个项目时，系统会打开相关的 `PreferenceFragment`。

注：使用首选项标头时，`PreferenceActivity` 的子类无需实现 `onCreate()` 方法，因为 Activity 唯一所需执行的任务就是加载标头。

使用首选项标头支持旧版本

如果您的应用支持早于 3.0 的 Android 版本，则在 Android 3.0 及更高版本系统上运行时，您仍可使用标头提供双窗格数据。为此，您只需另外创建一个使用基本 `<Preference>` 元素的首选项 XML 文件即可，这些基本元素的行为方式与标头项目类似（供较旧版本的 Android 系统使用）。

但是，每个 `<Preference>` 元素均会向 `PreferenceActivity` 发送一个 `Intent`，指定要加载哪个首选项 XML 文件，而不是打开新的 `PreferenceScreen`。

例如，下面就是一个用于 Android 3.0 及更高版本系统的首选项标头 XML 文件 (`res/xml/preference_headers.xml`)：

```
<preference-headers xmlns:android="http://schemas.android.com/apk/res/android">
    <header
        android:fragment="com.example.prefs.SettingsFragmentOne"
        android:title="@string/prefs_category_one"
        android:summary="@string/prefs_summ_category_one" />
    <header
        android:fragment="com.example.prefs.SettingsFragmentTwo"
        android:title="@string/prefs_category_two"
        android:summary="@string/prefs_summ_category_two" />
</preference-headers>
```

下面是为早于 Android 3.0 版本的系统提供相同标头的首选项文件 (`res/xml/preference_headers_legacy.xml`)：

```

<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <Preference
        android:title="@string/prefs_category_one"
        android:summary="@string/prefs_summ_category_one"  >
        <intent
            android:targetPackage="com.example.prefs"
            android:targetClass="com.example.prefs.SettingsActivity"
            android:action="com.example.prefs.PREFS_ONE" />
    </Preference>
    <Preference
        android:title="@string/prefs_category_two"
        android:summary="@string/prefs_summ_category_two" >
        <intent
            android:targetPackage="com.example.prefs"
            android:targetClass="com.example.prefs.SettingsActivity"
            android:action="com.example.prefs.PREFS_TWO" />
    </Preference>
</PreferenceScreen>

```

由于是从 Android 3.0 开始方添加对 `<preference-headers>` 的支持，因此只有在 Androd 3.0 或更高版本中运行时，系统才会在您的 `PreferenceActivity` 中调用 `onBuildHeaders()`。要加载“旧版”标头文件 (`preference_headers_legacy.xml`)，您必须检查 Android 版本，如果版本低于 Android 3.0 (`HONEYCOMB`)，请调用 `addPreferencesFromResource()` 来加载旧版标头文件。例如：

```

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

    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
        // Load the legacy preferences headers
        addPreferencesFromResource(R.xml.preference_headers_legacy);
    }
}

// Called only on Honeycomb and later
@Override
public void onBuildHeaders(List<Header> target) {
    loadHeadersFromResource(R.xml.preference_headers, target);
}

```

最后要做的就是处理传入 Activity 的 `Intent`，以确定要加载的首选项文件。因此，请检索 `Intent` 的操作，并将其与在首选项 XML 的 `<intent>` 标记中使用的已知操作字符串进行比较。

```

final static String ACTION_PREFS_ONE = "com.example.prefs.PREFS_ONE";
...

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

    String action = getIntent().getAction();
    if (action != null && action.equals(ACTION_PREFS_ONE)) {
        addPreferencesFromResource(R.xml.preferences);
    }
    ...

    else if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
        // Load the legacy preferences headers
        addPreferencesFromResource(R.xml.preference_headers_legacy);
    }
}

```

值得注意的是，连续调用 `addPreferencesFromResource()` 会将所有首选项堆叠在一个列表中，因此请将条件与 `else-if` 语句链接在一起，确保它只调用一次。

读取首选项

默认情况下，应用的所有首选项均保存到一个可通过调用静态方法 `PreferenceManager.getDefaultSharedPreferences()` 从应用内的任何位置访问的文件中。这将返回 `SharedPreferences` 对象，其中包含与 `PreferenceActivity` 中所用 `Preference` 对象相关的所有键值对。

例如，从应用中的任何其他 Activity 读取某个首选项值的方法如下：

```
SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
String syncConnPref = sharedPref.getString(SettingsActivity.KEY_PREF_SYNC_CONN, "");
```

侦听首选项变更

出于某些原因，您可能希望在用户更改任一首选项时立即收到通知。要在任一首选项发生更改时收到回调，请实现 `SharedPreference.OnSharedPreferenceChangeListener` 接口，并通过调用 `registerOnSharedPreferenceChangeListener()` 为 `SharedPreferences` 对象注册侦听器。

该接口只有 `onSharedPreferenceChanged()` 一种回调方法，而且您可能会发现在 Activity 过程中实现该接口最为简单。例如：

```
public class SettingsActivity extends PreferenceActivity
    implements OnSharedPreferenceChangeListener {
    public static final String KEY_PREF_SYNC_CONN = "pref_syncConnectionType";
    ...

    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
        String key) {
        if (key.equals(KEY_PREF_SYNC_CONN)) {
            Preference connectionPref = findPreference(key);
            // Set summary to be the user-description for the selected value
            connectionPref.setSummary(sharedPreferences.getString(key, ""));
        }
    }
}
```

在此示例中，该方法检查更改的设置是否是针对已知的首选项键。它调用 `findPreference()` 来获取已更改的 `Preference` 对象，以便能够将项目摘要修改为对用户选择的说明。即，如果设置为 `ListPreference` 或其他多选设置时，则当设置更改为显示当前状态（例如，图 5 所示的“Sleep”设置）时，您应调用 `setSummary()`。

注：正如 Android 设计有关[设置](#)的文档中所述，我们建议您在用户每次更改首选项时更新 `ListPreference` 的摘要，以描述当前设置。

若要妥善管理 Activity 生命周期，我们建议您在 `onResume()` 和 `onPause()` 回调期间分别注册和注销 `SharedPreference.OnSharedPreferenceChangeListener`。

```
@Override
protected void onResume() {
    super.onResume();
    getPreferenceScreen().getSharedPreferences()
        .registerOnSharedPreferenceChangeListener(this);
}

@Override
protected void onPause() {
    super.onPause();
    getPreferenceScreen().getSharedPreferences()
        .unregisterOnSharedPreferenceChangeListener(this);
}
```

注意：目前，首选项管理器不会在您调用 `registerOnSharedPreferenceChangeListener()` 时存储对侦听器的强引用。但是，您必须存储对侦听器的强引用，否则它将很容易被当作垃圾回收。我们建议您将对侦听器的引用保存在只要您需要侦听器就会存在的对象的数据中。

例如，在以下代码中，调用方未保留对侦听器的引用。因此，侦听器将容易被当作垃圾回收，并在将来某个不确定的时间失败：

```
prefs.registerOnSharedPreferenceChangeListener(  
    // Bad! The listener is subject to garbage collection!  
    new SharedPreferences.OnSharedPreferenceChangeListener() {  
        public void onSharedPreferenceChanged(SharedPreferences prefs, String key) {  
            // listener implementation  
        }  
    });
```

有鉴于此，请将对监听器的引用存储在只要需要监听器就会存在的对象的实例数据字段中：

```
SharedPreferences.OnSharedPreferenceChangeListener listener =  
    new SharedPreferences.OnSharedPreferenceChangeListener() {  
        public void onSharedPreferenceChanged(SharedPreferences prefs, String key) {  
            // listener implementation  
        }  
    };  
prefs.registerOnSharedPreferenceChangeListener(listener);
```

管理网络使用情况

从 Android 4.0 开始，通过系统的“设置”应用，用户可以了解自己的应用在前台和后台使用的网络数据量。然后，用户可以据此禁止具体的应用使用后台数据。为了避免用户禁止您的应用从后台访问数据，您应该有效地使用数据连接，并允许用户通过应用设置优化应用的数据使用。

例如，您可以允许用户控制应用同步数据的频率，控制应用是否仅在有 Wi-Fi 时才执行上传/下载操作，以及控制应用能否在漫游时使用数据，等等。为用户提供这些控件后，即使数据使用量接近他们在系统“设置”中设置的限制，他们也不大可能禁止您的应用访问数据，因为他们可以精确地控制应用使用的数据量。

在 [PreferenceActivity](#) 中添加必要的首选项来控制应用的数据使用习惯后，您应立即在清单文件中为 `ACTION_MANAGE_NETWORK_USAGE` 添加 Intent 过滤器。例如：

```
<activity android:name="SettingsActivity" ... >  
    <intent-filter>  
        <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>
```

此 Intent 过滤器指示系统此 Activity 控制应用的数据使用情况。因此，当用户从系统的“设置”应用检查应用所使用的数据量时，可以使用“查看应用设置”按钮启动 [PreferenceActivity](#)，这样，用户就能够优化应用使用的数据量。

构建自定义首选项

Android 框架包括各种 [Preference](#) 子类，您可以使用它们为各种不同类型的设置构建 UI。不过，您可能会发现自己需要的设置没有内置解决方案，例如，数字选取器或日期选取器。在这种情况下，您将需要通过扩展 [Preference](#) 类或其他子类之一来创建自定义首选项。

扩展 [Preference](#) 类时，您需要执行以下几项重要操作：

- 指定在用户选择设置时显示的用户界面。
- 适时保存设置的值。
- 使用显示的当前（默认）值初始化 [Preference](#)。
- 在系统请求时提供默认值。
- 如果 [Preference](#) 提供自己的 UI（例如对话框），请保存并恢复状态以处理生命周期变更（例如，用户旋转屏幕）。

下文介绍如何完成所有这些任务。

指定用户界面

如果您要直接扩展 `Preference` 类，则需要实现 `onClick()` 来定义在用户选择该项时发生的操作。不过，大多数自定义设置都会扩展 `DialogPreference` 以显示对话框，从而简化这一过程。扩展 `DialogPreference` 时，必须在类构造函数中调用 `setDialogLayoutResourcs()` 来指定对话框的布局。

例如，自定义 `DialogPreference` 可以使用下面的构造函数来声明布局并为默认的肯定和否定对话框按钮指定文本：

```
public class NumberPickerPreference extends DialogPreference {  
    public NumberPickerPreference(Context context, AttributeSet attrs) {  
        super(context, attrs);  
  
        setDialogLayoutResource(R.layout.numberpicker_dialog);  
        setPositiveButton(android.R.string.ok);  
        setNegativeButton(android.R.string.cancel);  
  
        setDialogIcon(null);  
    }  
    ...  
}
```

保存设置的值

如果设置的值为整型数或是用于保存布尔值的 `persistBoolean()`，则可通过调用 `Preference` 类的一个 `persist*` 方法（如 `persistInt()`）随时保存该值。

注：每个 `Preference` 均只能保存一种数据类型，因此您必须使用适合自定义 `Preference` 所用数据类型的 `persist*` 方法。

至于何时选择保留设置，则可能取决于要扩展的 `Preference` 类。如果扩展 `DialogPreference`，则只能在对话框因肯定结果（用户选择“确定”按钮）而关闭时保留该值。

当 `DialogPreference` 关闭时，系统会调用 `onDialogClosed()` 方法。该方法包括一个布尔参数，用于指定用户结果是否为“肯定”；如果值为 `true`，则表示用户选择的是肯定按钮且您应该保存新值。例如：

```
@Override  
protected void onDialogClosed(boolean positiveResult) {  
    // When the user selects "OK", persist the new value  
    if (positiveResult) {  
        persistInt(mNewValue);  
    }  
}
```

在此示例中，`mNewValue` 是一个类成员，可存放设置的当前值。调用 `persistInt()` 会将该值保存到 `SharedPreferences` 文件（自动使用在此 `Preference` 的 XML 文件中指定的键）。

初始化当前值

系统将 `Preference` 添加到屏幕时，会调用 `onSetInitialValue()` 来通知您设置是否具有保留值。如果没有保留值，则此调用将为您提供默认值。

`onSetInitialValue()` 方法传递一个布尔值 (`restorePersistedValue`)，以指示是否已为该设置保留值。如果值为 `true`，则应通过调用 `Preference` 类的一个 `getPersisted*` 方法（如整型值对应的 `getPersistedInt()`）来检索保留值。通常，您会需要检索保留值，以便能够正确更新 UI 来反映之前保存的值。

如果 `restorePersistedValue` 为 `false`，则应使用在第二个参数中传递的默认值。

```
@Override  
protected void onSetInitialValue(boolean restorePersistedValue, Object defaultValue) {  
    if (restorePersistedValue) {  
        // Restore existing state  
        mCurrentValue = this.getInt(DEFAULT_VALUE);  
    } else {  
        // Set default state from the XML attribute  
        mCurrentValue = (Integer) defaultValue;  
        persistInt(mCurrentValue);  
    }  
}
```

每种 `getPersisted*()` 方法均采用一个参数，用于指定实际上没有保留值或该键不存在时所要使用的默认值。在上述示例中，当 `getPersistedInt()` 不能返回保留值时，局部常量用于指定默认值。

注意：您不能使用 `defaultValue` 作为 `getPersisted*()` 方法中的默认值，因为当 `restorePersistedValue` 为 `true` 时，其值始终为 `null`。

提供默认值

如果 `Preference` 类的实例指定一个默认值（使用 `android:defaultValue` 属性），则在实例化对象以检索该值时，系统会调用 `onGetDefaultValue()`。您必须实现此方法，系统才能将默认值保存在 `SharedPreferences` 中。例如：

```
@Override  
protected Object onGetDefaultValue(TypedArray a, int index) {  
    return a.getInteger(index, DEFAULT_VALUE);  
}
```

方法参数可提供您所需的一切：属性的数组和 `android:defaultValue`（必须检索的值）的索引位置。之所以必须实现此方法以从该属性中提取默认值，是因为您必须为此属性指定在未定义属性值时所要使用的局部默认值。

保存和恢复首选项的状态

正如布局中的 `View` 一样，在重启 Activity 或片段时（例如，用户旋转屏幕），`Preference` 子类也负责保存并恢复其状态。要正确保存并恢复 `Preference` 类的状态，您必须实现生命周期回调方法 `onSaveInstanceState()` 和 `onRestoreInstanceState()`。

`Preference` 的状态由实现 `Parcelable` 接口的对象定义。Android 框架为您提供此类对象，作为定义状态对象 (`Preference.BaseSavedState` 类) 的起点。

要定义 `Preference` 类保存其状态的方式，您应该扩展 `Preference.BaseSavedState` 类。您只需重写几种方法并定义 `CREATOR` 对象。

对于大多数应用，如果 `Preference` 子类保存除整型数以外的其他数据类型，则可复制下列实现并直接更改处理 `value` 的行。

```
private static class SavedState extends BaseSavedState {
    // Member that holds the setting's value
    // Change this data type to match the type saved by your Preference
    int value;

    public SavedState(Parcelable superState) {
        super(superState);
    }

    public SavedState(Parcel source) {
        super(source);
        // Get the current preference's value
        value = source.readInt(); // Change this to read the appropriate data type
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        super.writeToParcel(dest, flags);
        // Write the preference's value
        dest.writeInt(value); // Change this to write the appropriate data type
    }

    // Standard creator object using an instance of this class
    public static final Parcelable.Creator<SavedState> CREATOR =
        new Parcelable.Creator<SavedState>() {

            public SavedState createFromParcel(Parcel in) {
                return new SavedState(in);
            }

            public SavedState[] newArray(int size) {
                return new SavedState[size];
            }
        };
}
```

如果将上述 `Preference.BaseSavedState` 实现添加到您的应用（通常，作为 `Preference` 子类的子类），则需要为 `Preference` 子类实现 `onSaveInstanceState()` 和 `onRestoreInstanceState()` 方法。

例如：

```
@Override
protected Parcelable onSaveInstanceState() {
    final Parcelable superState = super.onSaveInstanceState();
    // Check whether this Preference is persistent (continually saved)
    if (isPersistent()) {
        // No need to save instance state since it's persistent,
        // use superclass state
        return superState;
    }

    // Create instance of custom BaseSavedState
    final SavedState myState = new SavedState(superState);
    // Set the state's value with the class member that holds current
    // setting value
    myState.value = mNewValue;
    return myState;
}

@Override
protected void onRestoreInstanceState(Parcelable state) {
    // Check whether we saved the state in onSaveInstanceState
    if (state == null || !state.getClass().equals(SavedState.class)) {
        // Didn't save the state, so call superclass
        super.onRestoreInstanceState(state);
        return;
    }

    // Cast state to custom BaseSavedState and pass to superclass
    SavedState myState = (SavedState) state;
    super.onRestoreInstanceState(myState.getSuperState());

    // Set this Preference's widget to reflect the restored state
    mNumberPicker.setValue(myState.value);
}
```

对话框

本文内容

- › [创建对话框片段](#)
- › [构建提醒对话框](#)
 - › [添加按钮](#)
 - › [添加列表](#)
 - › [创建自定义布局](#)
- › [将事件传递回对话框的宿主](#)
- › [显示对话框](#)
- › [全屏显示对话框或将其显示为嵌入式片段](#)
 - › [将 Activity 显示为大屏幕上的对话框](#)
- › [清除对话框](#)

关键类

- › [DialogFragment](#)
- › [AlertDialog](#)

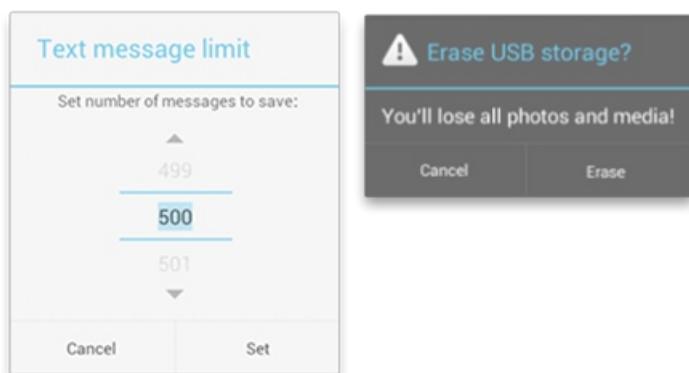
另请参阅

- › [对话框设计指南](#)
- › [选取器（日期/时间对话框）](#)

对话框是提示用户作出决定或输入额外信息的小窗口。 对话框不会填充屏幕，通常用于需要用户采取行动才能继续执行的模式事件。

对话框设计

如需了解有关如何设计对话框的信息（包括语言建议），请阅读[对话框设计指南](#)。



[Dialog](#) 类是对话框的基类，但您应该避免直接实例化 [Dialog](#)，而是使用下列子类之一：

[AlertDialog](#)

此对话框可显示标题、最多三个按钮、可选择项列表或自定义布局。

[DatePickerDialog](#) 或 [TimePickerDialog](#)

此对话框带有允许用户选择日期或时间的预定义 UI。

这些类定义您的对话框的样式和结构，但您应该将 `DialogFragment` 用作对话框的容器。`DialogFragment` 避免使用 `ProgressDialog` 和管理其外观所需的所有控件，而不是调用 `Dialog` 对象上的方法。

使用 `DialogFragment` 管理对话框可确保它能正确处理生命周期事件，如用户按“返回”按钮或旋转屏幕时。此外，`DialogFragment` 类还允许您将对话框的 UI 作为嵌入式组件在较大 UI 中重复使用，就像传统 `Fragment` 一样（例如，当您想让对话框 UI 在大屏幕和小屏幕上具有不同外观时）。

Android 包括另一种名为 `ProgressDialog` 的对话框类，可显示具有进度条的对话框。不过，如需指示加载进度或不确定的进度，则应改为遵循 [进度和 Activity](#) 的设计指南，并在您的布局中使用 `ProgressBar`。

本指南的后文将描述如何将 `DialogFragment` 与 `AlertDialog` 对象结合使用。如果您想创建一个日期或时间选取器，应改为阅读[选取器指南](#)。

注：由于 `DialogFragment` 类最初是通过 Android 3.0 (API 级别 11) 添加的，因此本文描述的是如何使用[支持库](#)附带的 `DialogFragment` 类。通过将该库添加到您的应用，您可以在运行 Android 1.6 或更高版本的设备上使用 `DialogFragment` 以及各种其他 API。如果您的应用支持的最低版本是 API 级别 11 或更高版本，则可使用 `DialogFragment` 的框架版本，但请注意，本文中的链接适用于支持库 API。使用支持库时，请确保您导入的是 `android.support.v4.app.DialogFragment` 类，而不是 `android.app.DialogFragment`。

创建对话框片段

您可以完成各种对话框设计—包括自定义布局以及[对话框设计指南](#)中描述的布局—通过扩展 `DialogFragment` 并在 `onCreateDialog()` 回调方法中创建 `AlertDialog`。

例如，以下是一个在 `DialogFragment` 内管理的基础 `AlertDialog`：

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
            .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // FIRE ZE MISSILES!
                }
            })
            .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // User cancelled the dialog
                }
            });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

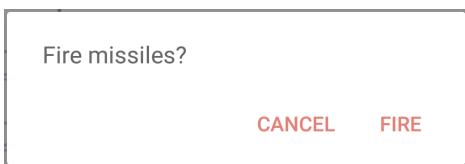


图 1. 一个包含消息和两个操作按钮的对话框。

现在，当您创建此类的实例并调用该对象上的 `show()` 时，对话框将如图 1 所示。

下文将详细描述如何使用 `AlertDialog.Builder` API 创建对话框。

根据对话框的复杂程度，您可以在 `DialogFragment` 中实现各种其他回调方法，包括所有基础[片段生命周期方法](#)。

构建提醒对话框

您可以通过 `AlertDialog` 类构建各种对话框设计，并且该类通常是您需要的唯一对话框类。如图 2 所示，提醒对话框有三个区域：

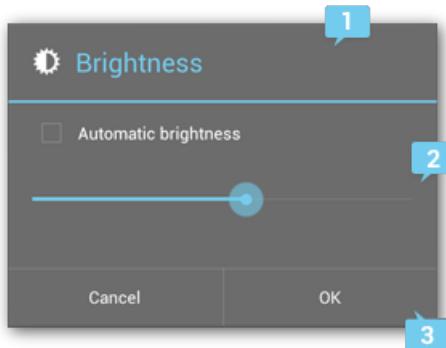


图 2. 对话框的布局。

1. 标题

这是可选项，只应在内容区域被详细消息、列表或自定义布局占据时使用。如需陈述的是一条简单消息或问题（如图 1 中的对话框），则不需要标题。

2. 内容区域

它可以显示消息、列表或其他自定义布局。

3. 操作按钮

对话框中的操作按钮不应超过三个。

`AlertDialog.Builder` 类提供的 API 允许您创建具有这几种内容（包括自定义布局）的 `AlertDialog`。

要想构建 `AlertDialog`，请执行以下操作：

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
    .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

以下主题介绍如何使用 `AlertDialog.Builder` 类定义各种对话框属性。

添加按钮

要想添加如图 2 所示的操作按钮，请调用 `setPositiveButton()` 和 `setNegativeButton()` 方法：

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// Add the buttons
builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User clicked OK button
    }
});
builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User cancelled the dialog
    }
});
// Set other dialog properties
...

// Create the AlertDialog
AlertDialog dialog = builder.create();
```

`set...Button()` 方法需要一个按钮标题（由字符串资源提供）和一个 `DialogInterface.OnClickListener`，后者用于定义用户按下该按钮时执行的操作。

您可以添加三种不同的操作按钮：

肯定

您应该使用此按钮来接受并继续执行操作（“确定”操作）。

否定

您应该使用此按钮来取消操作。

中性

您应该在用户可能不想继续执行操作，但也不一定想要取消操作时使用此按钮。它出现在肯定按钮和否定按钮之间。例如，实际操作可能是“稍后提醒我”。

对于每种按钮类型，您只能为 [AlertDialog](#) 添加一个该类型的按钮。也就是说，您不能添加多个“肯定”按钮。

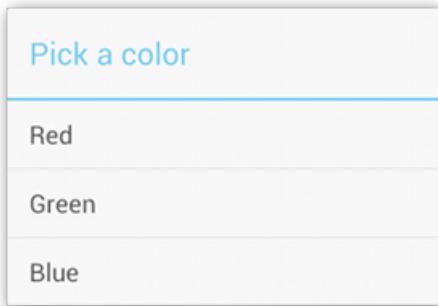


图 3. 一个包含标题和列表的对话框。

添加列表

可通过 [AlertDialog](#) API 提供三种列表：

- 传统单选列表
- 永久性单选列表（单选按钮）
- 永久性多选列表（复选框）

要想创建如图 3 所示的单选列表，请使用 [setItems\(\)](#) 方法：

```
@Override  
public Dialog onCreateDialog(Bundle savedInstanceState) {  
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
    builder.setTitle(R.string.pick_color)  
        .setItems(R.array.colors_array, new DialogInterface.OnClickListener() {  
            public void onClick(DialogInterface dialog, int which) {  
                // The 'which' argument contains the index position  
                // of the selected item  
            }  
        });  
    return builder.create();  
}
```

由于列表出现在对话框的内容区域，因此对话框无法同时显示消息和列表，您应该通过 [setTitle\(\)](#) 为对话框设置标题。要想指定列表项，请调用 [setItems\(\)](#) 来传递一个数组。或者，您也可以使用 [setAdapter\(\)](#) 指定一个列表。这样一来，您就可以使用 [ListAdapter](#) 以动态数据（如来自数据库的数据）支持列表。

如果您选择通过 [ListAdapter](#) 支持列表，请务必使用 [Loader](#)，以便内容以异步方式加载。[使用适配器构建布局](#)和[加载程序](#)指南中对此做了进一步描述。

注：默认情况下，触摸列表项会清除对话框，除非您使用的是下列其中一种永久性选择列表。

Pick your toppings

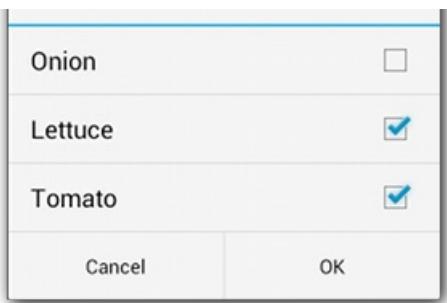


图 4. 多选项列表。

添加永久性多选列表或单选列表

要想添加多选项（复选框）或单选项（单选按钮）列表，请分别使用 `setMultiChoiceItems()` 或 `setSingleChoiceItems()` 方法。

例如，以下示例展示了如何创建如图 4 所示的多选列表，将选定项保存在一个 `ArrayList` 中：

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mSelectedItems = new ArrayList(); // Where we track the selected items
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Set the dialog title
    builder.setTitle(R.string.pick_toppings)
    // Specify the list array, the items to be selected by default (null for none),
    // and the listener through which to receive callbacks when items are selected
    .setMultiChoiceItems(R.array.toppings, null,
        new DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which,
                boolean isChecked) {
                if (isChecked) {
                    // If the user checked the item, add it to the selected items
                    mSelectedItems.add(which);
                } else if (mSelectedItems.contains(which)) {
                    // Else, if the item is already in the array, remove it
                    mSelectedItems.remove(Integer.valueOf(which));
                }
            }
        })
    // Set the action buttons
    .setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int id) {
            // User clicked OK, so save the mSelectedItems results somewhere
            // or return them to the component that opened the dialog
            ...
        }
    })
    .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int id) {
            ...
        }
    });
    return builder.create();
}
```

尽管传统列表和具有单选按钮的列表都能提供“单选”操作，但如果想持久保存用户的选择，则应使用 `setSingleChoiceItems()`。也就是说，如果稍后再次打开对话框时系统应指示用户的当前选择，那么您就需要创建一个具有单选按钮的列表。

创建自定义布局

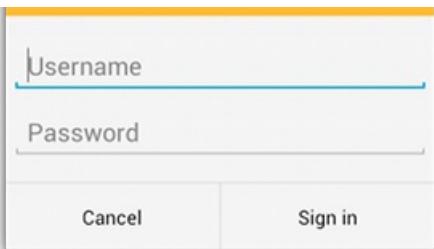


图 5. 自定义对话框布局。

如果您想让对话框具有自定义布局，请创建一个布局，然后通过调用 `AlertDialog.Builder` 对象上的 `setView()` 将其添加到 `AlertDialog`。

默认情况下，自定义布局会填充对话框窗口，但您仍然可以使用 `AlertDialog.Builder` 方法来添加按钮和标题。

例如，以下是图 5 中对话框的布局文件：

res/layout/dialog_signin.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:src="@drawable/header_logo"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:scaleType="center"
        android:background="#FFFFBB33"
        android:contentDescription="@string/app_name" />
    <EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/username" />
    <EditText
        android:id="@+id/password"
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:fontFamily="sans-serif"
        android:hint="@string/password"/>
</LinearLayout>
```

提示：默认情况下，当您将 `EditText` 元素设置为使用 `"textPassword"` 输入类型时，字体系列将设置为固定宽度。因此，您应该将其字体系列更改为 `"sans-serif"`，以便两个文本字段都使用匹配的字体样式。

要扩展 `DialogFragment` 中的布局，请通过 `getLayoutInflater()` 获取一个 `LayoutInflater` 并调用 `inflate()`，其中第一个参数是布局资源 ID，第二个参数是布局的父视图。然后，您可以调用 `setView()` 将布局放入对话框。

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Get the layout inflator
    LayoutInflator inflater = getActivity().getLayoutInflater();

    // Inflate and set the layout for the dialog
    // Pass null as the parent view because its going in the dialog layout
    builder.setView(inflater.inflate(R.layout.dialog_signin, null))
    // Add action buttons
        .setPositiveButton(R.string.signin, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                // sign in the user ...
            }
        })
        .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                LoginDialogFragment.this.getDialog().cancel();
            }
        });
    return builder.create();
}
```

提示：如果您想要自定义对话框，可以改用对话框的形式显示 `Activity`，而不是使用 `Dialog` API。只需创建一个 `Activity`，并在 `<activity>` 清单文件元素中将其主题设置为 `Theme.Holo.Dialog`：

```
<activity android:theme="@android:style/Theme.Holo.Dialog" >
```

就这么简单。`Activity` 现在会显示在一个对话框窗口中，而非全屏显示。

将事件传递回对话框的宿主

当用户触摸对话框的某个操作按钮或从列表中选择某一项时，您的 `DialogFragment` 可能会自行执行必要的操作，但通常您想将事件传递给打开该对话框的 `Activity` 或片段。为此，请定义一个界面，为每种点击事件定义一种方法。然后在从该对话框接收操作事件的宿主组件中实现该界面。

例如，以下 `DialogFragment` 定义了一个界面，通过该界面将事件传回给宿主 `Activity`：

```

public class NoticeDialogFragment extends DialogFragment {

    /* The activity that creates an instance of this dialog fragment must
     * implement this interface in order to receive event callbacks.
     * Each method passes the DialogFragment in case the host needs to query it. */
    public interface NoticeDialogListener {
        public void onDialogPositiveClick(DialogFragment dialog);
        public void onDialogNegativeClick(DialogFragment dialog);
    }

    // Use this instance of the interface to deliver action events
    NoticeDialogListener mListener;

    // Override the Fragment.onAttach() method to instantiate the NoticeDialogListener
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        // Verify that the host activity implements the callback interface
        try {
            // Instantiate the NoticeDialogListener so we can send events to the host
            mListener = (NoticeDialogListener) activity;
        } catch (ClassCastException e) {
            // The activity doesn't implement the interface, throw exception
            throw new ClassCastException(activity.toString()
                    + " must implement NoticeDialogListener");
        }
    }
    ...
}

```

对话框的宿主 Activity 会通过对对话框片段的构造函数创建一个对话框实例，并通过实现的 `NoticeDialogListener` 界面接收对话框的事件：

```

public class MainActivity extends FragmentActivity
    implements NoticeDialogFragment.NoticeDialogListener{
    ...

    public void showNoticeDialog() {
        // Create an instance of the dialog fragment and show it
        DialogFragment dialog = new NoticeDialogFragment();
        dialog.show(getSupportFragmentManager(), "NoticeDialogFragment");
    }

    // The dialog fragment receives a reference to this Activity through the
    // Fragment.onAttach() callback, which it uses to call the following methods
    // defined by the NoticeDialogFragment.NoticeDialogListener interface
    @Override
    public void onDialogPositiveClick(DialogFragment dialog) {
        // User touched the dialog's positive button
        ...
    }

    @Override
    public void onDialogNegativeClick(DialogFragment dialog) {
        // User touched the dialog's negative button
        ...
    }
}

```

由于宿主 Activity 会实现 `NoticeDialogListener`—由以上显示的 `onAttach()` 回调方法强制执行—因此对话框片段可以使用界面回调方法向 Activity 传递点击事件：

```
public class NoticeDialogFragment extends DialogFragment {
    ...
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Build the dialog and set up the button click handlers
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
            .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // Send the positive button event back to the host activity
                    mListener.onDialogPositiveClick(NoticeDialogFragment.this);
                }
            })
            .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // Send the negative button event back to the host activity
                    mListener.onDialogNegativeClick(NoticeDialogFragment.this);
                }
            });
        return builder.create();
    }
}
```

显示对话框

如果您想显示对话框，请创建一个 `DialogFragment` 实例并调用 `show()`，以传递对话框片段的 `FragmentManager` 和标记名称。

您可以通过从 `FragmentActivity` 调用 `getSupportFragmentManager()` 或从 `Fragment` 调用 `getFragmentManager()` 来获取 `FragmentManager`。例如：

```
public void confirmFireMissiles() {
    DialogFragment newFragment = new FireMissilesDialogFragment();
    newFragment.show(getSupportFragmentManager(), "missiles");
}
```

第二个参数 `"missiles"` 是系统用于保存片段状态并在必要时进行恢复的唯一标记名称。该标记还允许您通过调用 `findFragmentByTag()` 获取片段的句柄。

全屏显示对话框或将其显示为嵌入式片段

您可能采用以下 UI 设计：您想让一部分 UI 在某些情况下显示为对话框，但在其他情况下全屏显示或显示为嵌入式片段（也许取决于设备使用大屏幕还是小屏幕）。`DialogFragment` 类便具有这种灵活性，因为它仍然可以充当嵌入式 `Fragment`。

但在这种情况下，您不能使用 `AlertDialog.Builder` 或其他 `Dialog` 对象来构建对话框。如果您想让 `DialogFragment` 具有嵌入能力，则必须在布局中定义对话框的 UI，然后在 `onCreateView()` 回调中加载布局。

以下示例 `DialogFragment` 可以显示为对话框或嵌入式片段（使用名为 `purchase_items.xml` 的布局）：

```

public class CustomDialogFragment extends DialogFragment {
    /** The system calls this to get the DialogFragment's layout, regardless
     * of whether it's being displayed as a dialog or an embedded fragment. */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout to use as dialog or embedded fragment
        return inflater.inflate(R.layout.purchase_items, container, false);
    }

    /** The system calls this only when creating the layout in a dialog. */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // The only reason you might override this method when using onCreateView() is
        // to modify any dialog characteristics. For example, the dialog includes a
        // title by default, but your custom layout might not need it. So here you can
        // remove the dialog title, but you must call the superclass to get the Dialog.
        Dialog dialog = super.onCreateDialog(savedInstanceState);
        dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
        return dialog;
    }
}

```

以下代码可根据屏幕尺寸决定将片段显示为对话框还是全屏 UI：

```

public void showDialog() {
    FragmentManager fragmentManager = getSupportFragmentManager();
    CustomDialogFragment newFragment = new CustomDialogFragment();

    if (mIsLargeLayout) {
        // The device is using a large layout, so show the fragment as a dialog
        newFragment.show(fragmentManager, "dialog");
    } else {
        // The device is smaller, so show the fragment fullscreen
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        // For a little polish, specify a transition animation
        transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
        // To make it fullscreen, use the 'content' root view as the container
        // for the fragment, which is always the root view for the activity
        transaction.add(android.R.id.content, newFragment)
            .addToBackStack(null).commit();
    }
}

```

如需了解有关执行片段事务的详细信息，请参阅[片段](#)指南。

在本示例中，`mIsLargeLayout` 布尔值指定当前设备是否应该使用应用的大布局设计（进而将此片段显示为对话框，而不是全屏显示）。设置这种布尔值的最佳方法是声明一个[布尔资源值](#)，其中包含适用于不同屏幕尺寸的[备用资源](#)值。例如，以下两个版本的布尔资源适用于不同的屏幕尺寸：

res/values/bools.xml

```

<!-- Default boolean values -->
<resources>
    <bool name="large_layout">false</bool>
</resources>

```

res/values-large/bools.xml

```

<!-- Large screen boolean values -->
<resources>
    <bool name="large_layout">true</bool>
</resources>

```

然后，您可以在 Activity 的 `onCreate()` 方法执行期间初始化 `mIsLargeLayout` 值：

```
boolean mIsLargeLayout;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mIsLargeLayout = getResources().getBoolean(R.bool.large_layout);
}
```

将 Activity 显示为大屏幕上的对话框

相对于在小屏幕上将对话框显示为全屏 UI，您可以通过在大屏幕上将 `Activity` 显示为对话框来达到相同的效果。您选择的方法取决于应用设计，但当应用已经针对小屏幕进行设计，而您想要通过将短生存期 Activity 显示为对话框来改善平板电脑体验时，将 Activity 显示为对话框往往很有帮助。

要想仅在大屏幕上将 Activity 显示为对话框，请将 `Theme.Holo.DialogWhenLarge` 主题应用于 `<activity>` 清单文件元素：

```
<activity android:theme="@android:style/Theme.Holo.DialogWhenLarge" >
```

如需了解有关通过主题设置 Activity 样式的详细信息，请参阅[样式和主题指南](#)。

清除对话框

当用户触摸使用 `AlertDialog.Builder` 创建的任何操作按钮时，系统会为您清除对话框。

系统还会在用户触摸某个对话框列表项时清除对话框，但列表使用单选按钮或复选框时除外。否则，您可以通过在 `DialogFragment` 上调用 `dismiss()` 来手动清除对话框。

如需在对话框消失时执行特定操作，则可以在您的 `DialogFragment` 中实现 `onDismiss()` 方法。

您还可以取消对话框。这是一个特殊事件，它表示用户显式离开对话框，而不完成任务。如果用户按“返回”按钮，触摸对话框区域外部的屏幕，或者您在 `Dialog` 上显式调用 `cancel()`（例如，为了响应对话框中的“取消”按钮），就会发生这种情况。

如上例所示，您可以通过在您的 `DialogFragment` 类中实现 `onCancel()` 来响应取消事件。

注：系统会在每个调用 `onCancel()` 回调的事件发生时立即调用 `onDismiss()`。不过，如果您调用 `Dialog.dismiss()` 或 `DialogFragment.dismiss()`，系统会调用 `onDismiss()`，而不会调用 `onCancel()`。因此，当用户在对话框中按“肯定”按钮，从视图中移除对话框时，您通常应该调用 `dismiss()`。

通知

本文内容

- › [设计注意事项](#)
- › [创建通知](#)
 - › [必需的通知内容](#)
 - › [可选通知内容和设置](#)
 - › [通知操作](#)
 - › [通知优先级](#)
 - › [创建简单通知](#)
 - › [将扩展布局应用于通知](#)
 - › [处理兼容性](#)
- › [管理通知](#)
 - › [更新通知](#)
 - › [删除通知](#)
- › [启动 Activity 时保留导航](#)
 - › [设置常规 Activity PendingIntent](#)
 - › [设置特殊 Activity PendingIntent](#)
- › [在通知中显示进度](#)
 - › [显示持续时间固定的进度指示器](#)
 - › [显示持续 Activity 指示器](#)
- › [通知元数据](#)
- › [浮动通知](#)
- › [锁定屏幕通知](#)
- › [设置可见性](#)
- › [在锁定屏幕上控制媒体播放](#)
- › [自定义通知布局](#)

关键类

- › [NotificationManager](#)
- › [NotificationCompat](#)

视频

- › [4.1 中的通知](#)

另请参阅

- › [Android 设计：通知](#)

通知是您可以在应用的常规 UI 外部向用户显示的消息。当您告知系统发出通知时，它将先以图标的形式显示在**通知区域**中。用户可以打开**抽屉式通知栏**查看通知的详细信息。通知区域和抽屉式通知栏均是由系统控制的区域，用户可以随时查看。



图 1. 通知区域中的通知。

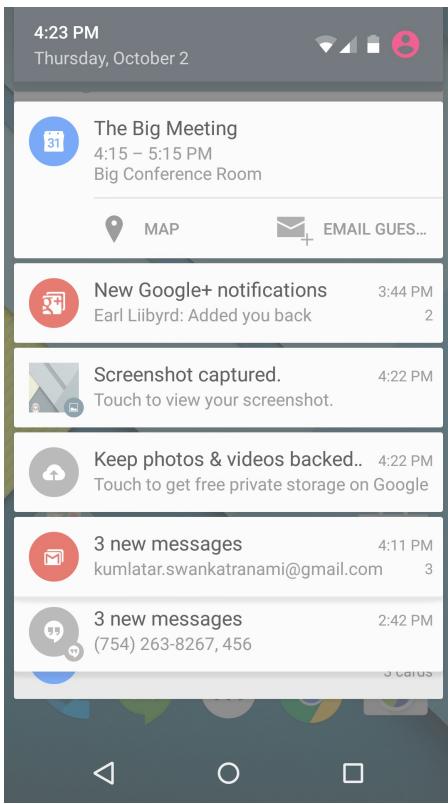


图 2. 抽屉式通知栏中的通知。

注：除非特别注明，否则本指南均引用版本 4 支持库中的 `NotificationCompat.Builder` 类。Android 3.0 (API 级别 11) 中已添加类 `Notification.Builder`。

设计注意事项

作为 Android 用户界面的一个重要组成部分，通知具有自己的设计指导方针。Android 5.0 (API 级别 21) 中引入的 Material Design 变更尤为重要，您应查阅 [Material Design](#) 培训资料了解详细信息。要了解如何设计通知及其交互，请阅读 [通知设计指南](#)。

创建通知

您可以在 `NotificationCompat.Builder` 对象中为通知指定 UI 信息和操作。要创建通知，请调用 `NotificationCompat.Builder.build()`，它将返回包含您的具体规范的 `Notification` 对象。要发出通知，请通过调用 `NotificationManager.notify()` 将 `Notification` 对象传递给系统。

必需的通知内容

`Notification` 对象必须包含以下内容：

- 小图标，由 `setSmallIcon()` 设置
- 标题，由 `setContentTitle()` 设置
- 详细文本，由 `setContentText()` 设置

可选通知内容和设置

所有其他通知设置和内容都是可选的。如需了解有关它们的更多详情，请参阅 `NotificationCompat.Builder` 参考文档。

通知操作

尽管通知操作都是可选的，但是您至少应向通知添加一个操作。操作允许用户直接从通知转到应用中的 `Activity`，他们可在其中查看一个或多个事件或执行进一步的操作。

一个通知可以提供多个操作。您应该始终定义一个当用户点击通知时会触发的操作；通常，此操作会在应用中打开 `Activity`。您也可以向通

知添加按钮来执行其他操作，例如，暂停闹铃或立即答复短信；此功能自 Android 4.1 起可用。如果使用其他操作按钮，则您还必须使这些按钮的功能在应用的 [Activity](#) 中可用；请参阅[处理兼容性](#)部分，以了解更多详情。

在 [Notification](#) 内部，操作本身由 [PendingIntent](#) 定义，后者包含在应用中启动 [Activity](#) 的 [Intent](#)。要将 [PendingIntent](#) 与手势相关联，请调用 [NotificationCompat.Builder](#) 的适当方法。例如，如果您要在用户点击抽屉式通知栏中的通知文本时启动 [Activity](#)，则可通过调用 [setContentIntent\(\)](#) 来添加 [PendingIntent](#)。

在用户点击通知时启动 [Activity](#) 是最常见的操作场景。此外，您还可以在用户清除通知时启动 [Activity](#)。在 Android 4.1 及更高版本中，您可以通过操作按钮启动 [Activity](#)。如需了解更多信息，请阅读参考指南的 [NotificationCompat.Builder](#) 部分。

通知优先级

您可以根据需要设置通知的优先级。优先级充当一个提示，提醒设备 UI 应该如何显示通知。要设置通知的优先级，请调用 [NotificationCompat.Builder.setPriority\(\)](#) 并传入一个 [NotificationCompat](#) 优先级常量。有五个优先级别，范围从 [PRIORITY_MIN](#) (-2) 到 [PRIORITY_MAX](#) (2)；如果未设置，则优先级默认为 [PRIORITY_DEFAULT](#) (0)。

有关设置适当优先级别的信息，请参阅[通知设计指南](#)中的“正确设置和管理通知优先级”。

创建简单通知

以下代码段说明了一个指定某项 [Activity](#) 在用户点击通知时打开的简单通知。请注意，该代码将创建 [TaskStackBuilder](#) 对象并使用它来为操作创建 [PendingIntent](#)。[启动 Activity 时保留导航](#)部分对此模式做了更详尽的阐述：

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());
```

就这么简单。您的用户现已收到通知。

将扩展布局应用于通知

要使通知出现在展开视图中，请先创建一个带有所需普通视图选项的 [NotificationCompat.Builder](#) 对象。接下来，调用以扩展布局对象作为其参数的 [Builder.setStyle\(\)](#)。

请记住，扩展通知在 Android 4.1 之前的平台上不可用。要了解如何处理针对 Android 4.1 及更早版本平台的通知，请阅读[处理兼容性](#)部分。

例如，以下代码段演示了如何更改在前面的代码段中创建的通知，以便使用扩展布局：

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Event tracker")
    .setContentText("Events received")
NotificationCompat.InboxStyle inboxStyle =
    new NotificationCompat.InboxStyle();
String[] events = new String[6];
// Sets a title for the Inbox in expanded layout
inboxStyle.setBigContentTitle("Event tracker details:");
...
// Moves events into the expanded layout
for (int i=0; i < events.length; i++) {

    inboxStyle.addLine(events[i]);
}
// Moves the expanded layout object into the notification object.
mBuilder.setStyle(inboxStyle);
...
// Issue the notification here.
```

处理兼容性

并非所有通知功能都可用于某特定版本，即便用于设置这些功能的方法位于支持库类 `NotificationCompat.Builder` 中也是如此。例如，依赖于扩展通知的操作按钮仅会显示在 Android 4.1 及更高版本的系统中，这是因为扩展通知本身仅在 Android 4.1 及更高版本的系统中可用。

为了确保最佳兼容性，请使用 `NotificationCompat` 及其子类（特别是 `NotificationCompat.Builder`）创建通知。此外，在实现通知时，请遵循以下流程：

1. 为所有用户提供通知的全部功能，无论他们使用何种版本的 Android 系统。为此，请验证是否可从应用的 `Activity` 中获得所有功能。要执行此操作，您可能需要添加新的 `Activity`。
例如，若要使用 `addAction()` 提供停止和启动媒体播放的控件，请先在应用的 `Activity` 中实现此控件。
2. 确保所有用户均可通过点击通知启动 `Activity` 来获得该 `Activity` 中的功能。为此，请为 `Activity` 创建 `PendingIntent`。调用 `setContentIntent()` 以将 `PendingIntent` 添加到通知。
3. 现在，将要使用的扩展通知功能添加到通知。请记住，您添加的任何功能还必须在用户点击通知时启动的 `Activity` 中可用。

管理通知

当您需要为同一类型的事件多次发出同一通知时，应避免创建全新的通知，而是应考虑通过更改之前通知的某些值和/或为其添加某些值来更新通知。

例如，Gmail 通过增加未读消息计数并将每封电子邮件的摘要添加到通知，通知用户收到了新的电子邮件。这称为“堆叠”通知；[通知设计指南](#)对此进行了更详尽的描述。

注：此 Gmail 功能需要“收件箱”扩展布局，该布局是自 Android 4.1 版本起可用的扩展通知功能的一部分。

下文介绍如何更新和删除通知。

更新通知

要将通知设置为能够更新，请通过调用 `NotificationManager.notify()` 发出带有通知 ID 的通知。要在发出之后更新此通知，请更新或创建 `NotificationCompat.Builder` 对象，从该对象构建 `Notification` 对象，并发出与之前所用 ID 相同的 `Notification`。如果之前的通知仍然可见，则系统会根据 `Notification` 对象的内容更新该通知。相反，如果之前的通知已被清除，系统则会创建一个新通知。

以下代码段演示了经过更新以反映所发生事件数量的通知。它将通知堆叠并显示摘要：

```
mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
// Because the ID remains unchanged, the existing notification is
// updated.
mNotificationManager.notify(
    notifyID,
    mNotifyBuilder.build());
...

```

删除通知

除非发生以下情况之一，否则通知仍然可见：

- 用户单独或通过使用“全部清除”清除了该通知（如果通知可以清除）。
- 用户点击通知，且您在创建通知时调用了 `setAutoCancel()`。
- 您针对特定的通知 ID 调用了 `cancel()`。此方法还会删除当前通知。
- 您调用了 `cancelAll()` 方法，该方法将删除之前发出的所有通知。

启动 Activity 时保留导航

从通知中启动 `Activity` 时，您必须保留用户的预期导航体验。点击“返回”应该使用户将应用的正常工作流回退到主屏幕，而点击“最新动态”则应将 `Activity` 显示为单独的任务。要保留导航体验，您应该在全新任务中启动 `Activity`。如何设置 `PendingIntent` 以获得全新任务取决于正在启动的 `Activity` 的性质。一般有两种情况：

常规 Activity

您要启动的 `Activity` 是应用的正常工作流的一部分。在这种情况下，请设置 `PendingIntent` 以启动全新任务并为 `PendingIntent` 提供返回栈，这将重现应用的正常“返回”行为。

Gmail 应用中的通知演示了这一点。点击一封电子邮件消息的通知时，您将看到消息具体内容。触摸“返回”将使您从 Gmail 回退到主屏幕，就好像您是从主屏幕（而不是通知）进入 Gmail 一样。

无论您触摸通知时处于哪个应用，都会发生这种情况。例如，如果您在 Gmail 中撰写消息时点击了一封电子邮件的通知，则会立即转到该电子邮件。触摸“返回”会依次转到收件箱和主屏幕，而不是转到您在撰写的邮件。

特殊 Activity

仅当从通知中启动时，用户才会看到此 `Activity`。从某种意义上说，`Activity` 是通过提供很难显示在通知本身中的信息来扩展通知。对于这种情况，请将 `PendingIntent` 设置为在全新任务中启动。但是，由于启动的 `Activity` 不是应用 `Activity` 流程的一部分，因此无需创建返回栈。点击“返回”仍会将用户带到主屏幕。

设置常规 Activity PendingIntent

要设置可启动直接进入 `Activity` 的 `PendingIntent`，请执行以下步骤：

1. 在清单文件中定义应用的 `Activity` 层次结构。
 - a. 添加对 Android 4.0.3 及更低版本的支持。为此，请通过添加 `<meta-data>` 元素作为 `<activity>` 的子项来指定正在启动的 `Activity` 的父项。

对于此元素，请设置 `android:name="android.support.PARENT_ACTIVITY"`。设置 `android:value="<parent_activity_name>"`，其中，`<parent_activity_name>` 是父 `<activity>` 元素的 `android:name` 值。请参阅下面的 XML 例子。

- b. 同样添加对 Android 4.1 及更高版本的支持。为此，请将 `android:parentActivityName` 属性添加到正在启动的 `Activity` 的 `<activity>` 元素中。

最终的 XML 应如下所示：

```
<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. 根据可启动 `Activity` 的 `Intent` 创建返回栈：

- 创建 `Intent` 以启动 `Activity`。
- 通过调用 `TaskStackBuilder.create()` 创建堆栈生成器。
- 通过调用 `addParentStack()` 将返回栈添加到堆栈生成器。对于在清单文件中所定义层次结构内的每个 `Activity`，返回栈均包含可启动 `Activity` 的 `Intent` 对象。此方法还会添加一些可在全新任务中启动堆栈的标志。
- 通过调用 `addNextIntent()`，添加可从通知中启动 `Activity` 的 `Intent`。将在第一步中创建的 `Intent` 作为 `addNextIntent()` 的参数传递。
- 如需，请通过调用 `TaskStackBuilder.editIntentAt()` 向堆栈中的 `Intent` 对象添加参数。有时，需要确保目标 `Activity` 在用户使用“返回”导航回它时会显示有意义的数据。
- 通过调用 `getPendingIntent()` 获得此返回栈的 `PendingIntent`。然后，您可以使用此 `PendingIntent` 作为 `setContentIntent()` 的参数。

注：尽管 `addParentStack()` 的参数是对已启动 `Activity` 的引用，但是方法调用不会添加可启动 `Activity` 的 `Intent`，而是留待下一步进行处理。

以下代码段演示了该流程：

```
...
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());
```

设置特殊 `Activity PendingIntent`

下文介绍如何设置特殊 Activity `PendingIntent`。

特殊 `Activity` 无需返回栈，因此您不必在清单文件中定义其 `Activity` 层次结构，也不必调用 `addParentStack()` 来构建返回栈。取而代之的是，您可使用清单文件设置 `Activity` 任务选项，并通过调用 `getActivity()` 创建 `PendingIntent`：

1. 在清单文件中，将以下属性添加到 `Activity` 的 `<activity>` 元素

```
android:name="activityclass"
```

Activity 的完全限定类名。

```
android:taskAffinity=""
```

与您在代码中设置的 `FLAG_ACTIVITY_NEW_TASK` 标志相结合，这可确保此 `Activity` 不会进入应用的默认任务。任何具有应用默认关联的现有任务均不受影响。

```
android:excludeFromRecents="true"
```

将新任务从“最新动态”中排除，这样用户就不会在无意中导航回它。

以下代码段显示了该元素：

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

2. 构建并发出通知：

- 创建可启动 `Activity` 的 `Intent`。
- 通过使用 `FLAG_ACTIVITY_NEW_TASK` 和 `FLAG_ACTIVITY_CLEAR_TASK` 标志调用 `setFlags()`，将 `Activity` 设置为在新的空任务中启动。
- 为 `Intent` 设置所需的任何其他选项。
- 通过调用 `getActivity()` 从 `Intent` 中创建 `PendingIntent`。然后，您可以使用此 `PendingIntent` 作为 `setContentIntent()` 的参数。

以下代码段演示了该流程：

```
// Instantiate a Builder object.
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
// Creates an Intent for the Activity
Intent notifyIntent =
    new Intent(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 notifyPendingIntent =
    PendingIntent.getActivity(
        this,
        0,
        notifyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyPendingIntent);
// 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());
```

在通知中显示进度

通知可能包括动画形式的进度指示器，向用户显示正在进行的操作状态。如果您可以估计操作所需的时间以及任意时刻的完成进度，则使用“限定”形式的指示器（进度栏）。如果无法估计操作的时长，则使用“非限定”形式的指示器（Activity 指示器）。

平台的 `ProgressBar` 类实现中显示有进度指示器。

要在 Android 4.0 及更高版本的平台上使用进度指示器，需调用 `setProgress()`。对于早期版本，您必须创建包括 `ProgressBar` 视图的自定义通知布局。

下文介绍如何使用 `setProgress()` 在通知中显示进度。

显示持续时间固定的进度指示器

要显示限定形式的进度栏，请通过调用 `setProgress(max, progress, false)` 将进度栏添加到通知，然后发出通知。随着操作继续进行，递增 `progress` 并更新通知。操作结束时，`progress` 应该等于 `max`。调用 `setProgress()` 的常见方法是将 `max` 设置为 100，然后将 `progress` 作为操作的“完成百分比”值递增。

您可以在操作完成后仍保留显示进度栏，也可以将其删除。无论哪种情况，都请记住更新通知文本以显示操作已完成。要删除进度栏，请调用 `setProgress(0, 0, false)`。例如：

```

...
mNotifyManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                // Sets the progress indicator to a max value, the
                // current completion percentage, and "determinate"
                // state
                mBuilder.setProgress(100, incr, false);
                // Displays the progress bar for the first time.
                mNotifyManager.notify(0, mBuilder.build());
                // Sleeps the thread, simulating an operation
                // that takes time
                try {
                    // Sleep for 5 seconds
                    Thread.sleep(5*1000);
                } catch (InterruptedException e) {
                    Log.d(TAG, "sleep failure");
                }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
            // Removes the progress bar
            .setProgress(0,0,false);
            mNotifyManager.notify(ID, mBuilder.build());
        }
    }
)
// Starts the thread by calling the run() method in its Runnable
).start();

```

显示持续 Activity 指示器

要显示非限定形式的 Activity 指示器，请使用 `setProgress(0, 0, true)` 将其添加到通知（忽略前两个参数），然后发出通知。这样一来，指示器的样式就与进度栏相同，只是其动画还在继续。

在操作开始之际发出通知。除非您修改通知，否则动画将一直运行。操作完成后，调用 `setProgress(0, 0, false)`，然后更新通知以删除 Activity 指示器。请务必这样做；否则，即使操作完成，动画仍将运行。同时，请记得更改通知文本，以表明操作已完成。

要了解 Activity 指示器的工作方式，请参阅上述代码段。找到以下几行：

```

// Sets the progress indicator to a max value, the current completion
// percentage, and "determinate" state
mBuilder.setProgress(100, incr, false);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());

```

将找到的这几行替换为以下几行：

```

// Sets an activity indicator for an operation of indeterminate length
mBuilder.setProgress(0, 0, true);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());

```

通知元数据

通知可根据您使用以下 `NotificationCompat.Builder` 方法分配的元数据进行排序：

- 当设备处于“优先”模式时，`setCategory()` 会告知系统如何处理应用通知（例如，通知代表传入呼叫、即时消息还是闹铃）。
- 如果优先级字段设置为 `PRIORITY_MAX` 或 `PRIORITY_HIGH` 的通知还有声音或振动，则 `setPriority()` 会将其显示在小型浮动窗口中。
- `addPerson()` 允许您向通知添加人员名单。您的应用可以使用此名单指示系统将指定人员发出的通知归成一组，或者将这些人员发出的通知视为更重要的通知。



图 3. 显示浮动通知的全屏 Activity

浮动通知

对于 Android 5.0 (API 级别 21)，当设备处于活动状态时（即，设备未锁定且其屏幕已打开），通知可以显示在小型浮动窗口中（也称为“浮动通知”）。这些通知看上去类似于精简版的通知，只是浮动通知还显示操作按钮。用户可以在不离开当前应用的情况下处理或清除浮动通知。

可能触发浮动通知的条件示例包括：

- 用户的 Activity 处于全屏模式中（应用使用 `fullScreenIntent`），或者
- 通知具有较高的优先级并使用铃声或振动

锁定屏幕通知

随着 Android 5.0 (API 级别 21) 的发布，通知现在还可显示在锁定屏幕上。您的应用可以使用此功能提供媒体播放控件以及其他常用操作。用户可以通过“设置”选择是否将通知显示在锁定屏幕上，并且您可以指定您应用中的通知在锁定屏幕上是否可见。

设置可见性

您的应用可以控制在安全锁定屏幕上显示的通知中可见的详细级别。调用 `setVisibility()` 并指定以下值之一：

- `VISIBILITY_PUBLIC` 显示通知的完整内容。
- `VISIBILITY_SECRET` 不会在锁定屏幕上显示此通知的任何部分。
- `VISIBILITY_PRIVATE` 显示通知图标和内容标题等基本信息，但是隐藏通知的完整内容。

设置 `VISIBILITY_PRIVATE` 后，您还可以提供其中隐藏了某些详细信息的替换版本通知内容。例如，短信应用可能会显示一条通知，指出“您有 3 条新短信”，但是隐藏了短信内容和发件人。要提供此替换版本的通知，请先使用 `NotificationCompat.Builder` 创建替换通知。创建专用通知对象时，请通过 `setPublicVersion()` 方法为其附加替换通知。

在锁定屏幕上控制媒体播放

在 Android 5.0 (API 级别 21) 中，锁定屏幕不再基于 `RemoteControlClient` (现已弃用) 显示媒体控件。取而代之的是，将 `Notification.MediaStyle` 模板与 `addAction()` 方法结合使用，后者可将操作转换为可点击的图标。

注：该模板和 `addAction()` 方法未包含在支持库中，因此这些功能只能在 Android 5.0 及更高版本的系统上运行。

要在 Android 5.0 系统的锁定屏幕上显示媒体播放控件，请将可见性设置为 `VISIBILITY_PUBLIC`，如上文所述。然后，添加操作并设置 `Notification.MediaStyle` 模板，如以下示例代码中所述：

```
Notification notification = new Notification.Builder(context)
    // Show controls on lock screen even when user hides sensitive content.
    .setVisibility(Notification.VISIBILITY_PUBLIC)
    .setSmallIcon(R.drawable.ic_stat_player)
    // Add media control buttons that invoke intents in your media service
    .addAction(R.drawable.ic_prev, "Previous", prevPendingIntent) // #0
    .addAction(R.drawable.ic_pause, "Pause", pausePendingIntent) // #1
    .addAction(R.drawable.ic_next, "Next", nextPendingIntent) // #2
    // Apply the media style template
    ..setStyle(new Notification.MediaStyle())
    .setShowActionsInCompactView(1 /* #1: pause button */)
    .setMediaSession(mMediaSession.getSessionToken())
    .setContentTitle("Wonderful music")
    .setContentText("My Awesome Band")
    .setLargeIcon(albumArtBitmap)
    .build();
```

注：弃用 `RemoteControlClient` 会对控制媒体产生进一步的影响。如需了解有关用于管理媒体会话和控制播放的新 API 的详细信息，请参阅[媒体播放控件](#)。

自定义通知布局

您可以利用通知框架定义自定义通知布局，由该布局定义通知在 `RemoteViews` 对象中的外观。自定义布局通知类似于常规通知，但是它们是基于 XML 布局文件中所定义的 `RemoteViews`。

自定义通知布局的可用高度取决于通知视图。普通视图布局限制为 64 dp，扩展视图布局限制为 256 dp。

要定义自定义通知布局，请首先实例化 `RemoteViews` 对象来扩充 XML 布局文件。然后，调用 `setContent()`，而不是调用 `setContentTitle()` 等方法。要在自定义通知中设置内容详细信息，请使用 `RemoteViews` 中的方法设置视图子项的值：

1. 在单独的文件中为通知创建 XML 布局。您可以根据需要使用任何文件名，但必须使用扩展名 `.xml`。
2. 在您的应用中，使用 `RemoteViews` 方法定义通知的图标和文本。通过调用 `setContent()` 将此 `RemoteViews` 对象放入 `NotificationCompat.Builder` 中。避免在 `RemoteViews` 对象上设置背景 `Drawable`，因为文本颜色可能使文本变得难以阅读。

此外，`RemoteViews` 类中还有一些方法可供您轻松将 `Chronometer` 或 `ProgressBar` 添加到通知布局。如需了解有关为通知创建自定义布局的详细信息，请参阅 `RemoteViews` 参考文档。

注意：使用自定义通知布局时，要特别注意确保自定义布局适用于不同的设备方向和分辨率。尽管这条建议适用于所有“视图”布局，但对通知尤为重要，因为抽屉式通知栏中的空间非常有限。不要让自定义布局过于复杂，同时确保在各种配置中对其进行测试。

对自定义通知文本使用样式资源

始终对自定义通知的文本使用样式资源。通知的背景颜色可能因设备和系统版本的不同而异，使用样式资源有助于您充分考虑到这一点。从 Android 2.3 开始，系统定义了标准通知布局文本的样式。若要在面向 Android 2.3 或更高版本系统的多个应用中使用相同样式，则应确保文本在显示背景上可见。

Toasts

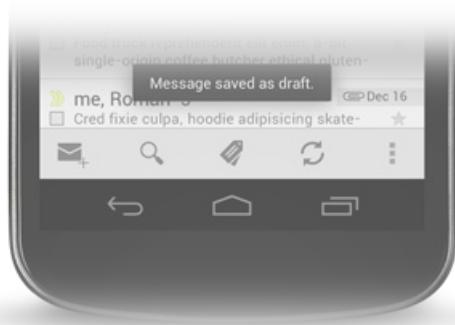
In this document

- [The Basics](#)
- [Positioning your Toast](#)
- [Creating a Custom Toast View](#)

Key classes

- [Toast](#)

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.



If user response to a status message is required, consider instead using a [Notification](#).

The Basics

First, instantiate a `Toast` object with one of the `makeText()` methods. This method takes three parameters: the application `Context`, the text message, and the duration for the toast. It returns a properly initialized `Toast` object. You can display the toast notification with `show()`, as shown in the following example:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

This example demonstrates everything you need for most toast notifications. You should rarely need anything else. You may, however, want to position the toast differently or even use your own layout instead of a simple text message. The following sections describe how you can do these things.

You can also chain your methods and avoid holding on to the `Toast` object, like this:

```
Toast.makeText(context, text, duration).show();
```

Positioning your Toast

POSITIONING YOUR TOAST

A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the `setGravity(int, int, int)` method. This accepts three parameters: a `Gravity` constant, an x-position offset, and a y-position offset.

For example, if you decide that the toast should appear in the top-left corner, you can set the gravity like this:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

If you want to nudge the position to the right, increase the value of the second parameter. To nudge it down, increase the value of the last parameter.

Creating a Custom Toast View

If a simple text message isn't enough, you can create a customized layout for your toast notification. To create a custom layout, define a View layout, in XML or in your application code, and pass the root `View` object to the `setView(View)` method.

For example, you can create the layout for the toast visible in the screenshot to the right with the following XML (saved as `layout/custom_toast.xml`):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/custom_toast_container"  
    android:orientation="horizontal"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:padding="8dp"  
    android:background="#DAAA"  
    >  
    <ImageView android:src="@drawable/droid"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_marginRight="8dp"  
        />  
    <TextView android:id="@+id/text"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:textColor="#FFF"  
        />  
</LinearLayout>
```

Notice that the ID of the `LinearLayout` element is "custom_toast_container". You must use this ID and the ID of the XML layout file "custom_toast" to inflate the layout, as shown here:

```
LayoutInflater inflater = getLayoutInflater();  
View layout = inflater.inflate(R.layout.custom_toast,  
    (ViewGroup) findViewById(R.id.custom_toast_container));  
  
TextView text = (TextView) layout.findViewById(R.id.text);  
text.setText("This is a custom toast");  
  
Toast toast = new Toast(getApplicationContext());  
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);  
toast.setDuration(Toast.LENGTH_LONG);  
toast.setView(layout);  
toast.show();
```

First, retrieve the `LayoutInflater` with `getLayoutInflater()` (or `getSystemService()`), and then inflate the layout from XML using `inflate(int, ViewGroup)`. The first parameter is the layout resource ID and the second is the root View. You can use this inflated layout to find more View objects in the layout, so now capture and define the content for the `ImageView` and `TextView` elements. Finally, create a new `Toast` with `Toast(Context)` and set some properties of the toast, such as the gravity and duration. Then call `setView(View)` and pass it the inflated layout. You can now display the toast with your custom layout by calling `show()`.

Note: Do not use the public constructor for a `Toast` unless you are going to define the layout with `setView(View)`. If you do not have a

custom layout to use, you must use `makeText(Context, int, int)` to create the Toast.

多窗口支持

本文内容

- › [概览](#)
- › [多窗口生命周期](#)
- › [针对多窗口模式配置应用](#)
- › [在多窗口模式中运行应用](#)
- › [测试应用的多窗口支持](#)

另请参阅

- › [多窗口 Playground 示例应用](#)
- › [在 Android N 为多窗口做准备的五条建议](#)

Android N 添加了对同时显示多个应用窗口的支持。在手持设备上，两个应用可以在“分屏”模式中左右并排或上下并排显示。在电视设备上，应用可以使用“画中画”模式，在用户与另一个应用交互的同时继续播放视频。

如果您使用 N Preview SDK 构建应用，则可以配置应用处理多窗口显示的方法。例如，您可以指定 Activity 的最小允许尺寸。您还可以禁用应用的多窗口显示，确保系统仅以全屏模式显示应用。

概览

Android N 允许多个应用同时共享屏幕。例如，用户可以分屏显示应用，在左边查看网页，同时在右边写邮件。用户体验取决于设备：

- 运行 Android N 的手持设备具有分屏模式。在此模式中，系统以左右并排或上下并排的方式分屏显示两个应用。用户可以拖动两个应用之间的分界线，放大其中一个应用，同时缩小另一个。
- 在运行 Android N 的 Nexus Player 上，应用能以[画中画模式](#)显示，即在用户浏览网页或其他应用交互的同时继续显示内容。
- 较大设备的制造商可选择启用自由形状模式，在该模式中，用户可以自由调整各 Activity 的尺寸。若制造商启用此功能，设备将同时具有自由形状模式和分屏模式。

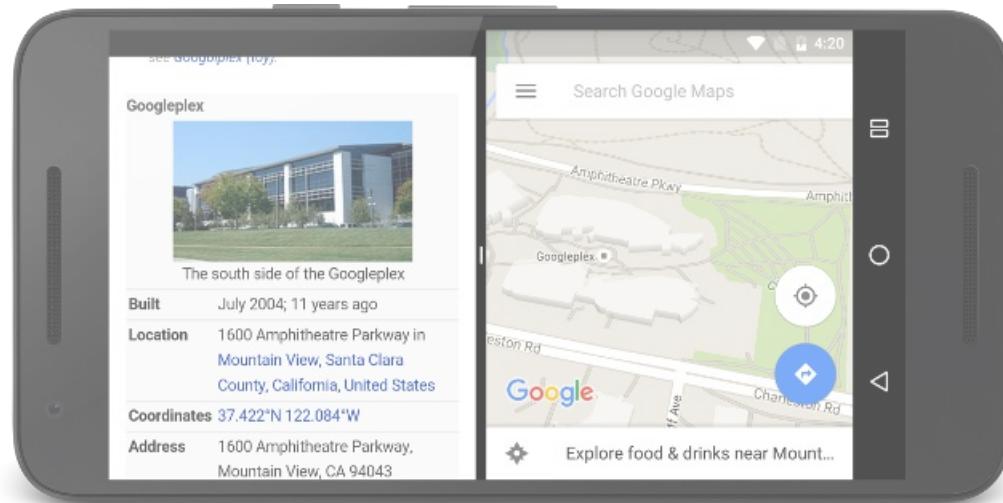


图 1. 两个应用在分屏模式中左右并排显示。

用户可以通过以下方式切换到多窗口模式：

- 若用户打开 [Overview 屏幕](#) 并长按 Activity 标题，则可以拖动该 Activity 至屏幕突出显示的区域，使 Activity 进入多窗口模式。
- 若用户长按 Overview 按钮，设备上的当前 Activity 将进入多窗口模式，同时将打开 Overview 屏幕，用户可在该屏幕中选择要共享屏幕的另一个 Activity。

用户可以在两个 Activity 共享屏幕的同时在这两个 Activity 之间[拖放](#)数据（在此之前，用户只能在一个 Activity 内部拖放数据）。

多窗口生命周期

多窗口模式不会更改 [Activity 生命周期](#)。

在多窗口模式中，在指定时间只有最近与用户交互过的 Activity 为活动状态。该 Activity 将被视为顶级 Activity。所有其他 Activity 虽然可见，但均处于暂停状态。但是，这些已暂停但可见的 Activity 在系统中享有比不可见 Activity 更高的优先级。如果用户与其中一个暂停的 Activity 交互，该 Activity 将恢复，而之前的顶级 Activity 将暂停。

注：在多窗口模式中，用户仍可以看到处于暂停状态的应用。应用在暂停状态下可能仍需要继续其操作。例如，处于暂停模式但可见的视频播放应用应继续显示视频。因此，我们建议播放视频的 Activity 不要暂停其 `onPause()` 处理程序中的视频。应暂停 `onStop()` 中的视频，并恢复 `onStart()` 中的视频播放。

如[处理运行时变更](#)中所述，用户使用多窗口模式显示应用时，系统将通知 Activity 发生配置变更。这也会发生在当用户调整应用大小，或将应用恢复到全屏模式时。该变更与系统通知应用设备从纵向模式切换到横向模式时的 Activity 生命周期影响基本相同，但设备不仅仅是交换尺寸，而是会变更尺寸。如[处理运行时变更](#)中所述，您的 Activity 可以自行处理配置变更，或允许系统销毁 Activity，并以新的尺寸重新创建该 Activity。

如果用户调整窗口大小，并在任意维度放大窗口尺寸，系统将调整 Activity 以匹配用户操作，同时根据需要发布[运行时变更](#)。如果应用在新公开区域的绘制滞后，系统将使用 `windowBackground` 属性或默认 `windowBackgroundFallback` 样式属性指定的颜色暂时填充该区域。

针对多窗口模式配置应用

如果您的应用面向 Android N，您可以对应用的 Activity 是否支持多窗口显示以及显示方式进行配置。您可以在清单文件中设置属性，以控制大小和布局。根 Activity 的属性设置适用于其任务栈中的所有 Activity。例如，如果根 Activity 已 `android:resizeableActivity` 设定为 `true`，则任务栈中的所有 Activity 都将可以调整大小。

注：如果您使用低于 Android N 版本的 SDK 构建多向应用，则用户在多窗口模式中使用应用时，系统将强制调整应用大小。系统将显示对话框，提醒用户应用可能会发生异常。系统不会调整定向应用的大小；如果用户尝试在多窗口模式下打开定向应用，应用将全屏显示。

android:resizeableActivity

在清单的 `<activity>` 或 `<application>` 节点中设置该属性，启用或禁用多窗口显示：

```
android:resizeableActivity=["true" | "false"]
```

如果该属性设置为 `true`，Activity 将能以分屏和自由形状模式启动。如果此属性设置为 `false`，Activity 将不支持多窗口模式。如果该值为 `false`，且用户尝试在多窗口模式下启动 Activity，该 Activity 将全屏显示。

如果您的应用面向 Android N，但未对该属性指定值，则该属性的值默认设为 `true`。

android:supportsPictureInPicture

在清单文件的 `<activity>` 节点中设置该属性，指明 Activity 是否支持画中画显示。如果 `android:resizeableActivity` 为 `false`，将忽略该属性。

```
android:supportsPictureInPicture=["true" | "false"]
```

布局属性

对于 Android N，`<layout>` 清单元素支持以下几种属性，这些属性影响 Activity 在多窗口模式中的行为：

android:defaultWidth

以自由形状模式启动时 Activity 的默认宽度。

`android:defaultHeight`

以自由形状模式启动时 Activity 的默认高度。

`android:gravity`

以自由形状模式启动时 Activity 的初始位置。请参阅 [Gravity](#) 参考资料，了解合适的值设置。

`android:minimalHeight`、`android:minimalWidth`

分屏和自由形状模式中 Activity 的最小高度和最小宽度。如果用户在分屏模式中移动分界线，使 Activity 尺寸低于指定的最小值，系统会将 Activity 裁剪为用户请求的尺寸。

例如，以下节点显示了如何指定 Activity 在自由形状模式中显示时 Activity 的默认大小、位置和最小尺寸：

```
<activity android:name=".MyActivity">
    <layout android:defaultHeight="500dp"
        android:defaultWidth="600dp"
        android:gravity="top|end"
        android:minimalHeight="450dp"
        android:minimalWidth="300dp" />
</activity>
```

在多窗口模式中运行应用

Android N 添加了新功能，以支持可在多窗口模式中运行的应用。

多窗口模式中被禁用的功能

在设备处于多窗口模式中时，某些功能会被禁用或忽略，因为这些功能对与其他 Activity 或应用共享设备屏幕的 Activity 而言没有意义。此类功能包括：

- 某些[系统 UI](#)自定义选项将被禁用；例如，在非全屏模式中，应用无法隐藏状态栏。
- 系统将忽略对 `android:screenOrientation` 属性所作的更改。

多窗口变更通知和查询

`Activity` 类中添加了以下新方法，以支持多窗口显示。有关各方法的详细信息，请参阅 [N Preview SDK 参考](#)。

`Activity.isInMultiWindowMode()`

调用该方法以确认 Activity 是否处于多窗口模式。

`Activity.isInPictureInPictureMode()`

调用该方法以确认 Activity 是否处于画中画模式。

注：画中画模式是多窗口模式的特例。如果 `myActivity.isInPictureInPictureMode()` 返回 `true`，则 `myActivity.isInMultiWindowMode()` 也返回 `true`。

`Activity.onMultiWindowModeChanged()`

Activity 进入或退出多窗口模式时系统将调用此方法。在 Activity 进入多窗口模式时，系统向该方法传递 `true` 值，在退出多窗口模式时，则传递 `false` 值。

`Activity.onPictureInPictureModeChanged()`

Activity 进入或退出画中画模式时系统将调用此方法。在 Activity 进入画中画模式时，系统向该方法传递 `true` 值，在退出画中画模式时，

则传递 `false` 值。

每个方法还有 `Fragment` 版本，例如 `Fragment.isInMultiWindowMode()`。

进入画中画模式

如需在画中画模式中启动 Activity，请调用新方法 `Activity.enterPictureInPictureMode()`。如果设备不支持画中画模式，则此方法无效。如需了解详细信息，请参阅[画中画](#)文档。

在多窗口模式中启动新 Activity

在启动新 Activity 时，用户可以提示系统如果可能，应将新 Activity 显示在当前 Activity 旁边。要执行此操作，可使用标志 `Intent.FLAG_ACTIVITY_LAUNCH_TO_ADJACENT`。传递此标志将请求以下行为：

- 如果设备处于分屏模式，系统会尝试在启动系统的 Activity 旁创建新 Activity，这样两个 Activity 将共享屏幕。系统并不一定能实现此操作，但如果可以，系统将使两个 Activity 处于相邻的位置。
- 如果设备不处于分屏模式，则该标志无效。

如果设备处于自由形状模式，则在启动新 Activity 时，用户可通过调用 `ActivityOptions.setLaunchBounds()` 指定新 Activity 的尺寸和屏幕位置。如果设备不处于多窗口模式，则该方法无效。

注：如果您在任务栈中启动 Activity，该 Activity 将替换屏幕上的 Activity，并继承其所有的多窗口属性。如果要在多窗口模式中以单独的窗口启动新 Activity，则必须在新的任务栈中启动此 Activity。

支持拖放

用户可以在两个 Activity 共享屏幕的同时在这两个 Activity 之间**拖放**数据（在此之前，用户只能在一个 Activity 内部拖放数据）。因此，如果您的应用目前不支持拖放功能，您可以在其中添加此功能。

N Preview SDK 扩展了 `android.view` 软件包，以支持跨应用拖放。有关以下类和方法的详细信息，请参阅 [N Preview SDK 参考](#)。

`android.view.DropPermissions`

令牌对象，负责指定对接收拖放数据的应用授予的权限。

`View.startDragAndDrop()`

`View.startDrag()` 的新别名。要启用跨 Activity 拖放，请传递新标志 `View.DRAG_FLAG_GLOBAL`。如需对接收拖放数据的 Activity 授予 URI 权限，可根据情况传递新标志 `View.DRAG_FLAG_GLOBAL_URI_READ` 或 `View.DRAG_FLAG_GLOBAL_URI_WRITE`。

`View.cancelDragAndDrop()`

取消当前正在进行的拖动操作。只能由发起拖动操作的应用调用。

`View.updateDragShadow()`

替换当前正在进行的拖动操作的拖动阴影。只能由发起拖动操作的应用调用。

`Activity.requestDropPermissions()`

请求使用 `DragEvent` 中包含的 `ClipData` 传递的内容 URI 的权限。

测试应用的多窗口支持

无论您是否针对 Android N 更新应用，都应验证应用在多窗口模式下的行为，以防用户尝试在运行 Android N 的设备上以多窗口模式启动应用。

配置测试设备

如果在设备上安装 Android N，则将自动支持分屏模式。

如果应用并非使用 N Preview SDK 构建

如果您的应用不是使用 N Preview SDK 构建的，则用户尝试在多窗口模式中使用应用时，系统将强制调整应用大小，除非应用进行了定向声明。

如果您的应用没有进行定向声明，则应在运行 Android N 的设备上启动应用，并尝试将应用切换到分屏模式。验证并确保在强制调整应用大小时用户体验可接受。

如果应用进行了定向声明，则应尝试将应用切换到多窗口模式。验证并确保执行此操作后，应用仍保持全屏模式。

如果支持多窗口模式

如果您的应用是使用 N Preview SDK 构建的，且未禁用多窗口支持，则分别在分屏和自由形状模式下验证以下行为。

- 在全屏模式下启动应用，然后通过长按 Overview 按钮切换到多窗口模式。验证并确保应用正常切换。
- 直接在多窗口模式中启动应用，验证并确保应用正常启动。您可以按一下 Overview 按钮，再长按应用的标题栏，并将其拖动到屏幕上任一突出显示的区域，从而在多窗口模式中启动应用。
- 拖动分界线，在分屏模式中调整应用的大小。验证并确保应用正常调整大小且未崩溃，并且必要的 UI 元素仍可见。
- 如果您指定了应用的最小尺寸，请尝试将应用尺寸调整到低于最小值。验证并确保无法将应用尺寸调整到低于指定最小值。
- 完成所有测试后，验证并确保应用性能可以接受。例如，验证并确保调整应用大小后更新 UI 没有长时间的滞后。

测试检查单

要在多窗口模式中验证应用性能，请执行以下操作。除非另有说明，否则请分别在分屏和多窗口模式中执行以下操作。

- 进入和退出多窗口模式。
- 从您的应用切换到另一个应用，验证并确保应用在非活动但可见的状态下正常运行。例如，如果您的应用在播放视频，则验证并确保在用户与另一个应用交互时视频仍在继续播放。
- 在分屏模式中，尝试移动分界线，放大或缩小应用。分别在左右和上下并排显示模式中尝试这些操作。验证并确保应用不会崩溃，主要功能可见，且调整操作不需要过长时间。
- 快速连续执行几次调整操作。验证并确保应用不会崩溃或出现内存泄漏。有关检查应用内存使用率的信息，请参阅[查看内存使用率](#)。
- 在多个不同窗口配置中正常使用应用，验证并确保应用正常运行。验证并确保文本可读，且 UI 元素大小正常，不影响交互。

如果已禁用多窗口支持

如果您通过设置 `android:resizableActivity="false"` 禁用了多窗口支持，则应在运行 Android N 的设备上启动应用，并尝试将应用切换到自由形状和分屏模式。验证并确保执行此操作后，应用仍保持全屏模式。

拖放

内容快览

- › 允许用户使用图形化手势在您的活动布局中移动数据。
- › 支持数据移动以外的操作。
- › 仅可在单一应用中使用。
- › 需要 API 11。

本文内容

- › [概览](#)
 - › [拖放过程](#)
 - › [拖拽事件侦听器和回调方法](#)
 - › [拖拽事件](#)
 - › [拖拽阴影](#)
- › [设计拖放操作](#)
 - › [开始拖拽](#)
 - › [响应拖拽开始](#)
 - › [在拖拽过程中处理事件](#)
 - › [响应放下](#)
 - › [响应拖拽结束](#)
 - › [响应拖拽事件：示例](#)

关键类

- › [View](#)
- › [OnLongClickListener](#)
- › [OnDragListener](#)
- › [DragEvent](#)
- › [DragShadowBuilder](#)
- › [ClipData](#)
- › [ClipDescription](#)

相关示例

- › [Honeycomb Gallery](#)。
- › [Api 演示中的 DragAndDropDemo.java 和 DraggableDot.java](#)。

另请参阅

- › [内容提供程序](#)
- › [输入事件](#)

借助 Android 拖放框架，可允许您的用户使用图形化拖放手势，将数据从当前布局中的一个视图移到另一个视图。该框架包括拖拽事件类、拖拽侦听器以及帮助程序方法和类。

尽管该框架主要为数据移动而设计，但也可以将其用于其他 UI 操作。例如，您可以创建一个应用，在用户将一个颜色图标拖到另一个图标上面时进行颜色混合。不过，本主题的其余部分将从数据移动方面介绍该框架。

概览

当用户做出某种您识别为开始拖拽数据的手势时，拖放操作开始。作为响应，您的应用会告知系统正在开始拖拽。系统回调应用，以获取正在拖拽的数据的表示。用户手指在当前布局上移动此表示（“拖拽阴影”）的过程中，系统将拖拽事件发送到与布局中的 `View` 对象相关联的拖拽事件监听器对象和拖拽事件回调方法。用户释放拖拽阴影后，系统立即结束拖拽操作。

从实现 `View.OnDragListener` 的类创建拖拽事件监听器对象（“listeners”）。使用视图对象的 `setOnDragListener()` 方法为视图设置拖拽事件监听器对象。每个视图对象还有一个 `onDragEvent()` 回调方法。在 [拖拽事件监听器和回调方法](#) 部分对以上两者进行了更详细的说明。

注：为简便起见，以下部分将接收拖拽事件的例程称为“拖拽事件监听器”，即便它实际可能是一个回调方法也是如此。

在开始拖拽时，将您想要移动的数据和描述此数据的元数据均包含在系统调用中。在拖拽期间，系统会将拖拽事件发送到布局中每个视图的拖拽事件监听器或回调方法。这些监听器或回调方法可使用元数据来确定其在数据被放下时是否想要接受这些数据。如果用户将数据放到某个视图对象上，并且该视图对象的监听器或回调方法之前已告知系统它想要接受放下的数据，则系统会将该数据发送到拖拽事件中的监听器或回调方法。

您的应用通过调用 `startDrag()` 方法告知系统开始拖拽。这将告知系统开始发送拖拽事件。该方法还会发送正在拖拽的数据。

您可以为当前布局中任意已连接的视图调用 `startDrag()`。系统仅使用视图对象获取布局中的全局设置访问权限。

应用调用 `startDrag()` 之后，该过程的剩余部分将使用系统发送给当前布局中的视图对象的事件。

拖放过程

拖放过程基本包含四个步骤或状态：

开始

为响应用户开始拖拽的手势，您的应用将调用 `startDrag()`，告知系统开始拖拽。参数 `startDrag()` 提供了将要拖拽的数据、此数据的元数据，以及用于绘制拖拽阴影的回调。

系统首先通过回调应用进行响应，以获取拖拽阴影。然后在设备上显示拖拽阴影。

接下来，系统将具有操作类型 `ACTION_DRAG_STARTED` 的拖拽事件发送到当前布局中所有视图对象的拖拽事件监听器。要继续接收拖拽事件（包括可能的放下事件），拖拽事件监听器必须返回 `true`。这将在系统中注册该监听器。只有已注册的监听器才能继续接收拖拽事件。此时，监听器也可以更改其视图对象的外观，以表明该监听器可以接受放下事件。

如果拖拽事件监听器返回 `false`，则它将不会接收当前操作的拖拽事件，直至系统发送具有操作类型 `ACTION_DRAG_ENDED` 的拖拽事件为止。通过发送 `false`，监听器告知系统，它对拖拽操作不感兴趣，不想接受拖拽的数据。

继续

用户继续拖拽。在拖拽阴影与某个视图对象的边界框相交时，系统将向视图对象的拖拽事件监听器（如果该监听器已注册接收事件）发送一个或多个拖拽事件。监听器可以选择更改其视图对象的外观以响应该事件。例如，如果该事件指示拖拽阴影已进入视图的边界框（操作类型 `ACTION_DRAG_ENTERED`），则监听器可通过突出显示其视图来做出反应。

放下

用户在可接受数据的视图的边界框内释放拖拽阴影。系统向视图对象的监听器发送具有操作类型 `ACTION_DROP` 的拖拽事件。该拖拽事件包含在启动操作的 `startDrag()` 调用中传递给系统的数据。如果成功执行了用于接受放下事件的代码，监听器预期将向系统返回布尔值 `true`。

请注意，仅当用户在其监听器已注册接收拖拽事件的视图的边界框内放下拖拽阴影时，才会发生这一步。如果用户在其他任何情况下释放拖拽阴影，将不会发送任何 `ACTION_DROP` 拖拽事件。

结束

在用户释放拖拽阴影并且系统发出（如果有必要）具有操作类型 `ACTION_DROP` 的拖拽事件后，系统将发出具有操作类型 `ACTION_DRAG_ENDED` 的拖拽事件，指示拖拽操作结束。不论用户在哪里释放拖拽阴影都会执行此操作。该事件将被发送到已注册接收拖拽事件的每个监听器（即使该监听器接收过 `ACTION_DROP` 事件）。

[设计拖放操作](#) 部分对以上四个步骤分别进行了更详尽的说明。

拖拽事件监听器和回调方法

视图使用实现 `View.OnDragListener` 的拖拽事件监听器，或者使用其 `onDragEvent(DragEvent)` 回调函数来接收拖拽事件。当系统调用该方法或监听器时，会向其传递一个 `DragEvent` 对象。

在多数情况下，您可能希望使用监听器。在设计 UI 时，通常不会将视图类划入子类，但使用回调方法会迫使您这样做，以便重写该方法。相比之下，您可以实现一个监听器类，然后将其与多个不同的视图对象配合使用。也可以将其实现为匿名内联类。要设置视图对象的监听器，请调用 `setOnDragListener()`。

您可能同时拥有视图对象的监听器和回调方法。如果出现这种情况，系统会首先调用监听器。除非监听器返回 `false`，否则系统不会调用回调方法。

`onDragEvent(DragEvent)` 方法和 `View.OnDragListener` 的组合与用于触摸事件的 `onTouchEvent()` 和 `View.OnTouchListener` 组合类似。

拖拽事件

系统以 `DragEvent` 对象的形式发出拖拽事件。该对象包含的操作类型会告知监听器拖放过程中所发生的情况。根据操作类型，该对象还包含其他数据。

要获取操作类型，监听程序可调用 `getAction()`。可能的值有六个，由 `DragEvent` 类中的常量定义。[表 1](#) 中列出了这些值。

`DragEvent` 对象还包含您的应用在 `startDrag()` 调用中提供给系统的数据。其中一些数据仅对特定的操作类型有效。[表 2](#) 中概括了每种操作类型的有效数据。在[设计拖放操作](#)部分也详尽描述了该数据及其适用的事件。

表 1. DragEvent 操作类型

getAction() 值	含义
<code>ACTION_DRAG_STARTED</code>	视图对象的拖拽事件监听器在应用调用 <code>startDrag()</code> 并获得拖拽阴影之后立即收到此事件操作类型。
<code>ACTION_DRAG_ENTERED</code>	视图对象的拖拽事件监听器在拖拽阴影刚刚进入视图的边界框时收到此事件操作类型。这是监听器在拖拽阴影进入边界框时收到的第一个事件操作类型。如果监听器想要继续接收此操作的拖拽事件，必须向系统返回布尔值 <code>true</code> 。
<code>ACTION_DRAG_LOCATION</code>	视图对象的拖拽事件监听器在收到 <code>ACTION_DRAG_ENTERED</code> 事件后并且拖拽阴影仍在该视图的边界框内时收到此事件操作类型。
<code>ACTION_DRAG_EXITED</code>	视图对象的拖拽事件监听器在收到 <code>ACTION_DRAG_ENTERED</code> 和至少一个 <code>ACTION_DRAG_LOCATION</code> 事件，并且用户已将拖拽阴影移到该视图的边界框以外后收到此事件操作类型。
<code>ACTION_DROP</code>	视图对象的拖拽事件监听器在用户将拖拽阴影释放到视图对象上面时收到此事件操作类型。仅当视图对象的监听器在响应 <code>ACTION_DRAG_STARTED</code> 拖拽事件时返回了布尔值 <code>true</code> ，才会将此操作类型发送至该监听器。如果用户将拖拽阴影释放到未注册监听器的视图上或用户将拖拽阴影释放到不属于当前布局的任何视图上，则不会发送此操作类型。 监听器如果成功处理了放下操作，预期将返回布尔值 <code>true</code> 。否则，它应该返回 <code>false</code> 。
<code>ACTION_DRAG_ENDED</code>	视图对象的拖拽事件监听器在系统结束拖拽操作时收到此事件操作类型。此操作类型不一定在 <code>ACTION_DROP</code> 事件之后。如果系统发送了 <code>ACTION_DROP</code> ，收到 <code>ACTION_DRAG_ENDED</code> 操作类型并不表示放下操作成功。监听器必须调用 <code>getResult()</code> 以获得响应 <code>ACTION_DROP</code> 时所返回的值。如果没有发送 <code>ACTION_DROP</code> 事件，则 <code>getResult()</code> 将返回 <code>false</code> 。

表 2. 按操作类型列出的有效 DragEvent 数据

getAction() 值	<code>getClipDescription()</code> 值	<code>getLocalState()</code> 值	<code>getX()</code> 值	<code>getY()</code> 值	<code>getClipData()</code> 值	<code>getResult()</code> 值
<code>ACTION_DRAG_STARTED</code>	X	X	X			
<code>ACTION_DRAG_ENTERED</code>	X	X	X	X		
<code>ACTION_DRAG_LOCATION</code>	X	X	X	X		
<code>ACTION_DRAG_EXITED</code>	X	X				
<code>ACTION_DROP</code>	X	X	X	X	X	

`getAction()`、`describeContents()`、`writeToParcel()` 和 `toString()` 方法始终返回有效数据。

如果某个方法不包含特定操作类型的有效数据，则根据其结果类型，将会返回 `null` 或 0。

拖拽阴影

在拖放操作期间，系统会显示用户拖拽的图像。对于数据移动，此图像表示正在拖拽的数据。对于其他操作，此图像表示拖拽操作的某个方面。

此图像被称为拖拽阴影。您使用为 `View.DragShadowBuilder` 对象声明的方法创建拖拽阴影，然后在使用 `startDrag()` 开始拖拽时将其传递给系统。作为系统对 `startDrag()` 的响应的一部分，系统会调用您在 `View.DragShadowBuilder` 中定义的回调方法以获取拖拽阴影。

`View.DragShadowBuilder` 类有两个构造函数：

`View.DragShadowBuilder(View)`

此构造函数可接受应用的任何 `View` 对象。该构造函数在 `View.DragShadowBuilder` 对象中存储视图对象，因此在回调期间，您可以在构造拖拽阴影时访问它。它不一定必须与用户选择开始拖拽操作的视图（如果有）相关联。

如果使用此构造函数，则无需扩展 `View.DragShadowBuilder` 或重写其方法。默认情况下，您将获得外观与您作为参数传递的视图相同的拖拽阴影，并且中心点位于用户触摸屏幕的位置。

`View.DragShadowBuilder()`

如果使用此构造函数，则 `View.DragShadowBuilder` 对象中没有任何可用的视图对象（该字段被设为 `null`）。如果使用此构造函数并且没有扩展 `View.DragShadowBuilder` 或重写其方法，您将获得不可见的拖拽阴影。系统不会给出错误。

`View.DragShadowBuilder` 类有两个方法：

`onProvideShadowMetrics()`

系统会在您调用 `startDrag()` 后立即调用此方法。使用此方法将拖拽阴影的尺寸和触摸点发送给系统。此方法具有两个参数：

dimensions

一个 `Point` 对象。拖拽阴影的宽度存储在 `x` 中，高度存储在 `y` 中。

touch_point

一个 `Point` 对象。触摸点是在拖拽操作期间，拖拽阴影内应该处于用户手指下面的位置。其 `X` 位置存储在 `x` 中，`Y` 位置存储在 `y` 中。

`onDrawShadow()`

在调用 `onProvideShadowMetrics()` 之后，系统会立即调用 `onDrawShadow()` 以获得拖拽阴影本身。该方法只有一个参数，即系统从您在 `onProvideShadowMetrics()` 中提供的参数构建的 `Canvas` 对象。使用此方法在提供的 `Canvas` 对象中绘制拖拽阴影。

为提高性能，应保持较小的拖拽阴影大小。对于单一项，您可能希望使用图标。对于多项选择，您可能希望使用堆栈中的图标，而不是在屏幕上展开完整的图像。

设计拖放操作

本部分展示有关如何开始拖拽、如何在拖拽期间响应事件、如何响应放下事件以及如何结束拖放操作的分步说明。

开始拖拽

用户使用拖拽手势（通常是长按视图对象）开始拖拽。作为响应，您应该执行以下操作：

1. 如果有必要，为正在移动的数据创建 `ClipData` 和 `ClipData.Item`。作为 `ClipData` 对象的一部分，提供存储在 `ClipData` 内的 `ClipDescription` 对象中的元数据。对于不提供数据移动的拖放操作，您可能需要使用 `null` 而不是实际对象。

例如，以下代码片段显示了如何通过创建包含 ImageView 标记或标签的 ClipData 对象来响应对 ImageView 的长按操作。此片段之后的下一个片段显示了如何重写 View.DragShadowBuilder 中的方法：

```
// Create a string for the ImageView label
private static final String IMAGEVIEW_TAG = "icon bitmap"

// Creates a new ImageView
ImageView imageView = new ImageView(this);

// Sets the bitmap for the ImageView from an icon bit map (defined elsewhere)
imageView.setImageBitmap(mIconBitmap);

// Sets the tag
imageView.setTag(IMAGEVIEW_TAG);

...

// Sets a long click listener for the ImageView using an anonymous listener object that
// implements the OnLongClickListener interface
imageView.setOnLongClickListener(new View.OnLongClickListener() {

    // Defines the one method for the interface, which is called when the View is long-clicked
    public boolean onLongClick(View v) {

        // Create a new ClipData.
        // This is done in two steps to provide clarity. The convenience method
        // ClipData.newPlainText() can create a plain text ClipData in one step.

        // Create a new ClipData.Item from the ImageView object's tag
        ClipData.Item item = new ClipData.Item(v.getTag());

        // Create a new ClipData using the tag as a label, the plain text MIME type, and
        // the already-created item. This will create a new ClipDescription object within the
        // ClipData, and set its MIME type entry to "text/plain"
        ClipData dragData = new ClipData(v.getTag(), ClipData.MIMETYPE_TEXT_PLAIN, item);

        // Instantiates the drag shadow builder.
        View.DragShadowBuilder myShadow = new MyDragShadowBuilder(imageView);

        // Starts the drag

            v.startDrag(dragData, // the data to be dragged
                        myShadow, // the drag shadow builder
                        null, // no need to use local data
                        0 // flags (not currently used, set to 0)
            );
    }
})
```

2. 以下代码片段定义了 myDragShadowBuilder 它会创建灰色小方框形式的拖拽阴影用于拖拽 TextView：

```

private static class MyDragShadowBuilder extends View.DragShadowBuilder {

    // The drag shadow image, defined as a drawable thing
    private static Drawable shadow;

    // Defines the constructor for myDragShadowBuilder
    public MyDragShadowBuilder(View v) {

        // Stores the View parameter passed to myDragShadowBuilder.
        super(v);

        // Creates a draggable image that will fill the Canvas provided by the system.
        shadow = new ColorDrawable(Color.LTGRAY);
    }

    // Defines a callback that sends the drag shadow dimensions and touch point back to the
    // system.
    @Override
    public void onProvideShadowMetrics (Point size, Point touch) {
        // Defines local variables
        private int width, height;

        // Sets the width of the shadow to half the width of the original View
        width = getView().getWidth() / 2;

        // Sets the height of the shadow to half the height of the original View
        height = getView().getHeight() / 2;

        // The drag shadow is a ColorDrawable. This sets its dimensions to be the same as the
        // Canvas that the system will provide. As a result, the drag shadow will fill the
        // Canvas.
        shadow.setBounds(0, 0, width, height);

        // Sets the size parameter's width and height values. These get back to the system
        // through the size parameter.
        size.set(width, height);

        // Sets the touch point's position to be in the middle of the drag shadow
        touch.set(width / 2, height / 2);
    }

    // Defines a callback that draws the drag shadow in a Canvas that the system constructs
    // from the dimensions passed in onProvideShadowMetrics().
    @Override
    public void onDrawShadow(Canvas canvas) {

        // Draws the ColorDrawable in the Canvas passed in from the system.
        shadow.draw(canvas);
    }
}

```

注：请记住，您无需扩展 `View.DragShadowBuilder`。构造函数 `View.DragShadowBuilder(View)` 以拖拽阴影的中心为触摸点，创建与传递给它的视图参数大小相同的默认拖拽阴影。

响应拖拽开始

在拖拽操作期间，系统会将拖拽事件分发到当前布局中的视图对象的拖拽事件侦听器。侦听器应通过调用 `getAction()` 做出反应，以获取操作类型。拖拽开始时，此方法将返回 `ACTION_DRAG_STARTED`。

在响应具有操作类型 `ACTION_DRAG_STARTED` 的事件时，侦听器应执行以下操作：

1. 调用 `getClipDescription()` 以获取 `ClipDescription`。使用 `ClipDescription` 中 MIME 类型的方法查看侦听器能否接受正在拖拽的数据。
如果拖放操作不表示数据移动，这可能不是必要的。
2. 如果侦听器可接受放下操作，则应返回 `true`。这将告知系统继续向侦听器发送拖拽事件。如果它不能接受放下操作，则应返回 `false`，系统将停止发送拖拽事件，直至其发出 `ACTION_DRAG_ENDED`。

请注意，对于 `ACTION_DRAG_STARTED` 事件，以下这些 `DragEvent` 方法全部无效：`getClipData()`、`getX()`、`getY()` 和 `getResult()`。

在拖拽过程中处理事件

在拖拽期间，返回 `true` 以响应 `ACTION_DRAG_STARTED` 拖拽事件的侦听器会继续接收拖拽事件。 侦听器在拖拽期间接收的拖拽事件类型取决于拖拽阴影的位置和侦听器视图的可见性。

在拖拽期间，侦听器主要使用拖拽事件来确定其是否应更改其视图的外观。

在拖拽期间，`getAction()` 将返回以下三个值中的一个：

- `ACTION_DRAG_ENTERED`：侦听器在触摸点（用户手指下面的屏幕点）进入侦听器视图的边界框时收到此事件。
- `ACTION_DRAG_LOCATION`：在侦听器收到 `ACTION_DRAG_ENTERED` 事件之后以及收到 `ACTION_DRAG_EXITED` 事件之前，它会在触摸点每次移动时收到新的 `ACTION_DRAG_LOCATION` 事件。`getX()` 和 `getY()` 方法会返回触摸点的 X 和 Y 坐标。
- `ACTION_DRAG_EXITED`：此事件在拖拽阴影不再在侦听器视图的边界框内之后，被发送到之前收到 `ACTION_DRAG_ENTERED` 的侦听器。

该侦听器不需要对以上任何操作类型做出反应。如果侦听器向系统返回值，该值将被忽略。以下是响应上述各个操作类型时的一些准则：

- 在响应 `ACTION_DRAG_ENTERED` 或 `ACTION_DRAG_LOCATION` 时，侦听器可以更改视图的外观，以指示它将要接收放下操作。
- 具有操作类型 `ACTION_DRAG_LOCATION` 的事件包含对应于触摸点位置的 `getX()` 和 `getY()` 的有效数据。 侦听器可能希望使用此信息来更改位于触摸点的视图部分外观。 侦听器也可以使用此信息来确定用户计划将拖拽阴影拖到的确切位置。
- 在响应 `ACTION_DRAG_EXITED` 时，侦听器应重置其在响应 `ACTION_DRAG_ENTERED` 或 `ACTION_DRAG_LOCATION` 时所应用的任何外观更改。 这向用户指明，该视图不再是迫在眉睫的放下目标。

响应放下

当用户将拖拽阴影释放到应用中的某个视图上并且该视图之前已报告它能接受所拖拽的内容时，系统会向该视图分发具有操作类型 `ACTION_DROP` 的拖拽事件。 侦听器应执行以下操作：

1. 调用 `getClipData()` 以获取最初在 `startDrag()` 调用中提供的 `ClipData` 对象并存储该对象。 如果拖放操作不表示数据移动，这可能是必要的。
2. 返回布尔值 `true` 指示已成功处理放下操作，或者，如果处理失败，则返回布尔值 `false`。 返回的值将成为 `getResult()` 针对 `ACTION_DRAG_ENDED` 事件返回的值。

请注意，如果系统没有发出 `ACTION_DROP` 事件，则 `ACTION_DRAG_ENDED` 事件的 `getResult()` 值为 `false`。

对于 `ACTION_DROP` 事件，`getX()` 和 `getY()` 将使用收到放下操作的视图的坐标系，返回拖拽点在放下时刻的 X 和 Y 位置。

系统允许用户将拖拽阴影释放到侦听器未接收拖拽事件的视图上。 它也允许用户在空的应用 UI 区域或在应用以外的区域释放拖拽阴影。 上述所有情况下，系统都不会发送具有操作类型 `ACTION_DROP` 的事件，不过它会发出 `ACTION_DRAG_ENDED` 事件。

响应拖拽结束

系统会在用户释放拖放阴影后，立即向应用中的所有拖拽事件侦听器发送具有操作类型 `ACTION_DRAG_ENDED` 的拖拽事件。 此事件指示拖拽操作结束。

每个侦听器应执行以下操作：

1. 如果侦听器在操作期间更改了其视图对象的外观，则应该将视图重置为其默认外观。 这是通知用户操作结束的视觉指示。
2. 侦听器可以选择调用 `getResult()` 以了解关于该操作的更多信息。 如果侦听器在响应 `ACTION_DROP` 操作类型的事件时返回了 `true`，则 `getResult()` 将返回布尔值 `true`。 在其他所有情况下，`getResult()` 均返回布尔值 `false`，包括系统未发出 `ACTION_DROP` 事件的任何情况。
3. 侦听器应该向系统返回布尔值 `true`。

响应拖拽事件：示例

所有拖拽事件最初都由拖拽事件方法或监听器接收。以下代码片段是在监听器中对拖拽事件做出反应的简单示例：

```
// Creates a new drag event listener
mDragListen = new myDragEventListener();

View imageView = new ImageView(this);

// Sets the drag event listener for the View
imageView.setOnDragListener(mDragListen);

...

protected class myDragEventListener implements View.OnDragListener {

    // This is the method that the system calls when it dispatches a drag event to the
    // listener.
    public boolean onDrag(View v, DragEvent event) {

        // Defines a variable to store the action type for the incoming event
        final int action = event.getAction();

        // Handles each of the expected events
        switch(action) {

            case DragEvent.ACTION_DRAG_STARTED:

                // Determines if this View can accept the dragged data
                if (event.getClipDescription().hasMimeType(ClipDescription.MIMETYPE_TEXT_PLAIN)) {

                    // As an example of what your application might do,
                    // applies a blue color tint to the View to indicate that it can accept
                    // data.
                    v.setColorFilter(Color.BLUE);

                    // Invalidate the view to force a redraw in the new tint
                    v.invalidate();

                    // returns true to indicate that the View can accept the dragged data.
                    return true;
                }

                // Returns false. During the current drag and drop operation, this View will
                // not receive events again until ACTION_DRAG_ENDED is sent.
                return false;
            case DragEvent.ACTION_DRAG_ENTERED:

                // Applies a green tint to the View. Return true; the return value is ignored.

                v.setColorFilter(Color.GREEN);

                // Invalidate the view to force a redraw in the new tint
                v.invalidate();

                return true;
            case DragEvent.ACTION_DRAG_LOCATION:

                // Ignore the event
                return true;
            case DragEvent.ACTION_DRAG_EXITED:

                // Re-sets the color tint to blue. Returns true; the return value is ignored.
                v.setColorFilter(Color.BLUE);

                // Invalidate the view to force a redraw in the new tint
                v.invalidate();
        }
    }
}
```

```
        return true;

    case DragEvent.ACTION_DROP:

        // Gets the item containing the dragged data
        ClipData.Item item = event.getClipData().getItemAt(0);

        // Gets the text data from the item.
        dragData = item.getText();

        // Displays a message containing the dragged data.
        Toast.makeText(this, "Dragged data is " + dragData, Toast.LENGTH_LONG);

        // Turns off any color tints
        v.clearColorFilter();

        // Invalidates the view to force a redraw
        v.invalidate();

        // Returns true. DragEvent.getResult() will return true.
        return true;

    case DragEvent.ACTION_DRAG_ENDED:

        // Turns off any color tinting
        v.clearColorFilter();

        // Invalidates the view to force a redraw
        v.invalidate();

        // Does a getResult(), and displays what happened.
        if (event.getResult()) {
            Toast.makeText(this, "The drop was handled.", Toast.LENGTH_LONG);

        } else {
            Toast.makeText(this, "The drop didn't work.", Toast.LENGTH_LONG);
        }

        // returns true; the value is ignored.
        return true;

    // An unknown action type was received.
    default:
        Log.e("DragDrop Example", "Unknown action type received by OnDragListener.");
        break;
    }

    return false;
}
};
```



样式和主题

本文内容

- › [定义样式](#)
- › [继承](#)
- › [样式属性](#)
- › [对 UI 应用样式和主题](#)
- › [对视图应用样式](#)
- › [对 Activity 或应用应用主题](#)
- › [根据平台版本选择主题](#)
- › [使用平台样式和主题](#)

另请参阅

- › [样式和主题资源](#)
- › [适用于 Android 样式和主题的 R.style](#)
- › [适用于所有样式属性的 R.attr](#)

样式是指为 `View` 或窗口指定外观和格式的属性集合。样式可以指定高度、填充、字体颜色、字号、背景色等许多属性。样式是在与指定布局的 XML 不同的 XML 资源中进行定义。

Android 中的样式与网页设计中层叠样式表的原理类似 — 您可以通过它将设计与内容分离。

例如，通过使用样式，您可以将以下布局 XML：

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textColor="#00FF00"  
    android:typeface="monospace"  
    android:text="@string/hello" />
```

简化成这个样子：

```
<TextView  
    style="@style/CodeFont"  
    android:text="@string/hello" />
```

布局 XML 中所有与样式有关的属性都已移除，并置于一个名为 `CodeFont` 的样式定义内，然后通过 `style` 属性加以应用。您会在下文中看到对该样式的定义。

主题是指对整个 `Activity` 或应用而不是对单个 `View`（如上例所示）应用的样式。以主题形式应用样式时，`Activity` 或应用中的每个视图都将应用其支持的每个样式属性。例如，您可以 `Activity` 主题形式应用同一 `CodeFont` 样式，之后该 `Activity` 内的所有文本都将具有绿色固定宽度字体。

定义样式

要创建一组样式，请在您的项目的 `res/values/` 目录中保存一个 XML 文件。可任意指定该 XML 文件的名称，但它必须使用 `.xml` 扩展名，并且必须保存在 `res/values/` 文件夹内。

该 XML 文件的根节点必须是 `<resources>`。

对于您想创建的每个样式，向该文件添加一个 `<style>` 元素，该元素带有对样式进行唯一标识的 `name` 属性（该属性为必需属性）。然后为该样式的每个属性添加一个 `<item>` 元素，该元素带有声明样式属性以及属性值的 `name`（该属性为必需属性）。根据样式属性，`<item>` 的值可以是关键字字符串、十六进制颜色值、对另一资源类型的引用或其他值。以下是一个包含单个样式的示例文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

`<resources>` 元素的每个子项都会在编译时转换成一个应用资源对象，该对象可由 `<style>` 元素的 `name` 属性中的值引用。可从 XML 布局以 `@style/CodeFont` 形式引用该示例样式（如上文引言中所示）。

`<style>` 元素中的 `parent` 属性是可选属性，它指定应作为此样式所继承属性来源的另一样式的资源 ID。如果愿意，您可在随后替换这些继承的样式属性。

切记，在 XML 中定义您想用作 Activity 或应用主题的样式与定义视图样式的方法完全相同。诸如上文所定义的样式可作为单个视图的样式加以应用，也可作为整个 Activity 或应用的主题加以应用。后文将阐述如何为单个视图应用样式或如何以应用主题形式应用样式。

继承

您可以通过 `<style>` 元素中的 `parent` 属性指定应作为您的样式所继承属性来源的样式。您可以利用它来继承现有样式的属性，然后只定义您想要更改或添加的属性。您可以从自行创建的样式或平台内建的样式继承属性。（如需了解有关从 Android 平台定义的样式继承属性的信息，请参阅下文的[使用平台样式和主题](#)。）例如，您可以继承 Android 平台的默认文本外观，然后对其进行修改：

```
<style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>
```

如果您想从自行定义的样式继承属性，则不必使用 `parent` 属性，而是只需将您想继承的样式的名称以前缀形式添加到新样式的名称之中，并以句点进行分隔。例如，要创建一个继承上文定义的 `CodeFont` 样式的新样式，但将颜色设置为红色，您可以按如下方式创建这个新样式：

```
<style name="CodeFont.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

请注意，`<style>` 标记中没有 `parent` 属性，但由于 `name` 属性以 `CodeFont` 样式名称（这是您创建的一个样式）开头，因此这个样式会继承该样式的所有样式属性。这个样式随后会替换 `android:textColor` 属性，将文本设置为红色。您可以 `@style/CodeFont.Red` 形式引用这个新样式。

您可以通过使用句点链接名称继续进行这样的继承，次数不限。例如，您可以通过以下代码将 `CodeFont.Red` 扩大：

```
<style name="CodeFont.Red.Big">
    <item name="android:textSize">30sp</item>
</style>
```

这段代码同时从 `CodeFont` 和 `CodeFont.Red` 样式继承，然后添加 `android:textSize` 属性。

注：这种通过将名称链接起来的继承方法只适用于由您自己的资源定义的样式。您无法通过这种方法继承 Android 内建样式。要引用内建样式（例如 `TextAppearance`），您必须使用 `parent` 属性。

样式属性

既然您已了解了样式是如何定义的，就需要了解什么类型的样式属性（由 `<item>` 元素定义）可以使用。您多半已经熟悉了其中的一些，例如 `layout_width` 和 `textColor`。当然，还有许多其他样式属性可供您使用。

相应的类引用最便于查找适用于特定 `View` 的属性，其中列出了所有支持的 XML 属性。例如，[TextView XML 属性](#) 表中所列的所有属性都可在

`TextView` 元素（或其其中一个子类）的样式定义中使用。该引用中列出的其中一个属性是 `android:inputType`，因此，如果您正常情况下会在 `<EditText>` 元素中放置 `android:inputType` 属性，如下所示：

```
<EditText  
    android:inputType="number"  
    ... />
```

您就可以改为给包括该属性的 `EditText` 元素创建一个样式：

```
<style name="Numbers">  
    <item name="android:inputType">number</item>  
    ...  
</style>
```

这样您的布局 XML 现在便可实现这个样式：

```
<EditText  
    style="@style/Numbers"  
    ... />
```

这个简单示例可能显得工作量更大，但如果您添加更多样式属性并将能够在各种地方重复使用样式这一因素考虑在内，就会发现回报可能很丰厚。

如需查看所有可用样式属性的参考资料，请参阅 [R.attr](#) 参考资料。切记，所有 View 对象仍然不接受样式属性，因此正常情况下您应该引用所支持样式属性的具体 `View` 类。不过，如果您应用样式的 View 不支持所有样式属性，该 View 将只应用那些受支持的属性，并直接忽略其他属性。

不过，某些样式属性任何 View 元素都不提供支持，只能以主题形式应用。这些样式属性应用于整个窗口而非任何类型的 View。例如，主题的样式属性可以隐藏应用标题、隐藏状态栏或更改窗口的背景。这些类型的样式属性不属于任何 View 对象。要发现这些仅主题样式属性，请在 [R.attr](#) 参考资料中查看有关以 `window` 开头的属性的内容。例如，`windowNoTitle` 和 `windowBackground` 是只有在样式以主题形式应用于 Activity 或应用时才起作用的样式属性。请参阅下文有关以主题形式应用样式的信息。

注：别忘了使用 `android:` 命名空间为每个 `<item>` 元素中的属性名称添加前缀。例如：`<item name="android:inputType">`。

对 UI 应用样式和主题

设置样式的方法有两种：

- 如果是对单个视图应用样式，请为布局 XML 中的 View 元素添加 `style` 属性。
- 或者，如果是对整个 Activity 或应用来应用样式，请为 Android 清单中的 `<activity>` 或 `<application>` 元素添加 `android:theme` 属性。

当您对布局中的单个 `View` 应用样式时，该样式定义的属性只应用于该 `View`。如果对 `ViewGroup` 应用样式，子 `View` 元素将**不会**继承样式属性 — 只有被您直接应用样式的元素才会应用其属性。不过，您可以通过以主题形式应用样式，使所应用的样式作用于所有 `View` 元素。

要以主题形式应用样式定义，您必须在 Android 清单中将样式应用于 `Activity` 或应用。如果您这样做，Activity 或应用内的每个 `View` 都将应用其支持的每个属性。例如，如果您对某个 Activity 应用前面示例中的 `CodeFont` 样式，则所有支持这些文本样式属性的 View 元素也会应用这些属性。任何不支持这些属性的 View 都会忽略这些属性。如果某个 View 仅支持部分属性，将只应用这些属性。

对视图应用样式

为 XML 布局中的视图设置样式的方法如下：

```
<TextView  
    style="@style/CodeFont"  
    android:text="@string/hello" />
```

现在该 `TextView` 将按照名为 `CodeFont` 的样式的定义设置样式（请参阅上文[定义样式](#)中的示例）。

注：style 属性不使用 android: 命名空间前缀。

对 Activity 或应用应用主题

要为您的应用的所有 Activity 设置主题，请打开 `AndroidManifest.xml` 文件并编辑 `<application>` 标记，在其中加入带样式名称的 `android:theme` 属性。例如：

```
<application android:theme="@style/CustomTheme">
```

如果您只想对应用中的一个 Activity 应用主题，则改为给 `<activity>` 标记添加 `android:theme` 属性。

正如 Android 提供了其他内建资源一样，有许多预定义主题可供您使用，可免于自行编写。例如，您可以使用 `Dialog` 主题，为您的 Activity 赋予类似对话框的外观：

```
<activity android:theme="@android:style/Theme.Dialog">
```

或者，如果您希望背景是透明的，则可使用 `Translucent` 主题：

```
<activity android:theme="@android:style/Theme.Translucent">
```

如果您喜欢某个主题，但想做些调整，只需将该主题添加为您的自定义主题的 `parent`。例如，您可以像下面这样对传统明亮主题进行修改，使用您自己的颜色：

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

(请注意，此处颜色需要以单独资源形式提供，因为 `android:windowBackground` 属性仅支持对另一资源的引用；不同于 `android:colorBackground`，无法为其提供颜色字面量。)

现在，在 Android 清单内使用 `CustomTheme` 替代 `Theme.Light`：

```
<activity android:theme="@style/CustomTheme">
```

根据平台版本选择主题

新版本的 Android 可为应用提供更多主题，您可能希望在这些平台上运行时可以使用这些新增主题，同时仍可兼容旧版本。您可以通过自定义主题来实现这一目的，该主题根据平台版本利用资源选择在不同父主题之间切换。

例如，以下这个声明所对应的自定义主题就是标准的平台默认明亮主题。它位于 `res/values` 之下的一个 XML 文件（通常是 `res/values/styles.xml`）中：

```
<style name="LightThemeSelector" parent="android:Theme.Light">
    ...
</style>
```

为了让该主题在应用运行在 Android 3.0 (API 级别 11) 或更高版本系统上时使用更新的全息主题，您可以在 `res/values-v11` 下的 XML 文件中加入一个替代主题声明，但将父主题设置为全息主题：

```
<style name="LightThemeSelector" parent="android:Theme.Holo.Light">
    ...
</style>
```

现在像您使用任何其他主题那样使用该主题，您的应用将在其运行于 Android 3.0 或更高版本的系统上时自动切换到全息主题。

`R.styleable.Theme` 提供了可在主题中使用的标准属性的列表。

如需了解有关根据平台版本或其他设备配置提供备用资源（例如主题和布局）的详细信息，请参阅[提供资源](#)文档。

使用平台样式和主题

Android 平台提供了庞大的样式和主题集合，供您在应用中使用。您可以在 [R.style](#) 类中找到所有可用样式的参考资料。要使用此处所列样式，请将样式名称中的所有下划线替换为句点。例如，您可以使用 "@android:style/Theme.NoTitleBar" 应用 [Theme_NoTitleBar](#) 主题。

不过，[R.style](#) 参考资料并不完备，未对样式做全面说明，因此查看这些样式和主题的实际源代码可让您更清楚地了解每个样式提供的样式属性。如需查看更详实的 Android 样式和主题参考资料，请参阅以下源代码：

- [Android 样式 \(styles.xml\)](#)
- [Android 主题 \(themes.xml\)](#)

这些文件有助于您通过示例进行学习。例如，在 Android 主题源代码中，您可以找到 `<style name="Theme.Dialog">` 的声明。在该定义中，您可以看到用来为 Android 框架使用的对话框设置样式的所有属性。

如需了解有关 XML 中样式和主题语法的详细信息，请参阅[样式资源](#)文档。

如需查看您可用来定义样式或主题的可用样式属性（例如“windowBackground”或“textAppearance”）的参考资料，请参阅[R.attr](#) 或您创建的样式所对应的 View 类。

Custom Components

In this document

- [The Basic Approach](#)
- [Fully Customized Components](#)
- [Compound Controls](#)
- [Modifying an Existing View Type](#)

Android offers a sophisticated and powerful componentized model for building your UI, based on the fundamental layout classes: [View](#) and [ViewGroup](#). To start with, the platform includes a variety of prebuilt View and ViewGroup subclasses — called widgets and layouts, respectively — that you can use to construct your UI.

A partial list of available widgets includes [Button](#), [TextView](#), [EditText](#), [ListView](#), [CheckBox](#), [RadioButton](#), [Gallery](#), [Spinner](#), and the more special-purpose [AutoCompleteTextView](#), [ImageSwitcher](#), and [TextSwitcher](#).

Among the layouts available are [LinearLayout](#), [FrameLayout](#), [RelativeLayout](#), and others. For more examples, see [Common Layout Objects](#).

If none of the prebuilt widgets or layouts meets your needs, you can create your own View subclass. If you only need to make small adjustments to an existing widget or layout, you can simply subclass the widget or layout and override its methods.

Creating your own View subclasses gives you precise control over the appearance and function of a screen element. To give an idea of the control you get with custom views, here are some examples of what you could do with them:

- You could create a completely custom-rendered View type, for example a "volume control" knob rendered using 2D graphics, and which resembles an analog electronic control.
- You could combine a group of View components into a new single component, perhaps to make something like a ComboBox (a combination of popup list and free entry text field), a dual-pane selector control (a left and right pane with a list in each where you can re-assign which item is in which list), and so on.
- You could override the way that an EditText component is rendered on the screen (the [Notepad Tutorial](#) uses this to good effect, to create a lined-notepad page).
- You could capture other events like key presses and handle them in some custom way (such as for a game).

The sections below explain how to create custom Views and use them in your application. For detailed reference information, see the [View](#) class.

The Basic Approach

Here is a high level overview of what you need to know to get started in creating your own View components:

1. Extend an existing [View](#) class or subclass with your own class.
2. Override some of the methods from the superclass. The superclass methods to override start with 'on', for example, [onDraw\(\)](#), [onMeasure\(\)](#), and [onKeyDown\(\)](#). This is similar to the [on...](#) events in [Activity](#) or [ListActivity](#) that you override for lifecycle and other functionality hooks.
3. Use your new extension class. Once completed, your new extension class can be used in place of the view upon which it was based.

Tip: Extension classes can be defined as inner classes inside the activities that use them. This is useful because it controls access to them but isn't necessary (perhaps you want to create a new public View for wider use in your application).

Fully Customized Components

Fully customized components can be used to create graphical components that appear however you wish. Perhaps a graphical VU meter that looks like an old analog gauge, or a sing-a-long text view where a bouncing ball moves along the words so you can sing along with a karaoke machine. Either way, you want something that the built-in components just won't do, no matter how you combine them.

Fortunately, you can easily create components that look and behave in any way you like, limited perhaps only by your imagination, the size of the screen, and the available processing power (remember that ultimately your application might have to run on something with significantly less power than your desktop workstation).

To create a fully customized component:

1. The most generic view you can extend is, unsurprisingly, [View](#), so you will usually start by extending this to create your new super component.
2. You can supply a constructor which can take attributes and parameters from the XML, and you can also consume your own such attributes and parameters (perhaps the color and range of the VU meter, or the width and damping of the needle, etc.)
3. You will probably want to create your own event listeners, property accessors and modifiers, and possibly more sophisticated behavior in your component class as well.
4. You will almost certainly want to override `onMeasure()` and are also likely to need to override `onDraw()` if you want the component to show something. While both have default behavior, the default `onDraw()` will do nothing, and the default `onMeasure()` will always set a size of 100x100 — which is probably not what you want.
5. Other `on...` methods may also be overridden as required.

Extend `onDraw()` and `onMeasure()`

The `onDraw()` method delivers you a [Canvas](#) upon which you can implement anything you want: 2D graphics, other standard or custom components, styled text, or anything else you can think of.

Note: This does not apply to 3D graphics. If you want to use 3D graphics, you must extend [SurfaceView](#) instead of View, and draw from a separate thread. See the [GLSurfaceViewActivity](#) sample for details.

`onMeasure()` is a little more involved. `onMeasure()` is a critical piece of the rendering contract between your component and its container. `onMeasure()` should be overridden to efficiently and accurately report the measurements of its contained parts. This is made slightly more complex by the requirements of limits from the parent (which are passed in to the `onMeasure()` method) and by the requirement to call the `setMeasuredDimension()` method with the measured width and height once they have been calculated. If you fail to call this method from an overridden `onMeasure()` method, the result will be an exception at measurement time.

At a high level, implementing `onMeasure()` looks something like this:

1. The overridden `onMeasure()` method is called with width and height measure specifications (`widthMeasureSpec` and `heightMeasureSpec` parameters, both are integer codes representing dimensions) which should be treated as requirements for the restrictions on the width and height measurements you should produce. A full reference to the kind of restrictions these specifications can require can be found in the reference documentation under [View.onMeasure\(int, int\)](#) (this reference documentation does a pretty good job of explaining the whole measurement operation as well).
2. Your component's `onMeasure()` method should calculate a measurement width and height which will be required to render the component. It should try to stay within the specifications passed in, although it can choose to exceed them (in this case, the parent can choose what to do, including clipping, scrolling, throwing an exception, or asking the `onMeasure()` to try again, perhaps with different measurement specifications).
3. Once the width and height are calculated, the `setMeasuredDimension(int width, int height)` method must be called with the calculated measurements. Failure to do this will result in an exception being thrown.

Here's a summary of some of the other standard methods that the framework calls on views:

Category	Methods	Description
----------	---------	-------------

Category	Methods	Description
		This is a form of the constructor that are called when the view is created from code and a form that is called when the view is inflated from a layout file. The second form should parse and apply any attributes defined in the layout file.
	<code>onFinishInflate()</code>	Called after a view and all of its children has been inflated from XML.
Layout	<code>onMeasure(int, int)</code>	Called to determine the size requirements for this view and all of its children.
	<code>onLayout(boolean, int, int, int, int)</code>	Called when this view should assign a size and position to all of its children.
	<code>onSizeChanged(int, int, int, int)</code>	Called when the size of this view has changed.
Drawing	<code>onDraw(Canvas)</code>	Called when the view should render its content.
Event processing	<code>onKeyDown(int, KeyEvent)</code>	Called when a new key event occurs.
	<code>onKeyUp(int, KeyEvent)</code>	Called when a key up event occurs.
	<code>onTrackballEvent(MotionEvent)</code>	Called when a trackball motion event occurs.
	<code>onTouchEvent(MotionEvent)</code>	Called when a touch screen motion event occurs.
Focus	<code>onFocusChanged(boolean, int, Rect)</code>	Called when the view gains or loses focus.
	<code>onWindowFocusChanged(boolean)</code>	Called when the window containing the view gains or loses focus.
Attaching	<code>onAttachedToWindow()</code>	Called when the view is attached to a window.
	<code>onDetachedFromWindow()</code>	Called when the view is detached from its window.
	<code>onWindowVisibilityChanged(int)</code>	Called when the visibility of the window containing the view has changed.

A Custom View Example

The CustomView sample in the [API Demos](#) provides an example of a customized View. The custom View is defined in the [LabelView](#) class.

The LabelView sample demonstrates a number of different aspects of custom components:

- Extending the View class for a completely custom component.
- Parameterized constructor that takes the view inflation parameters (parameters defined in the XML). Some of these are passed through to the View superclass, but more importantly, there are some custom attributes defined and used for LabelView.
- Standard public methods of the type you would expect to see for a label component, for example `setText()`, `setTextSize()`, `setTextColor()` and so on.
- An overridden `onMeasure` method to determine and set the rendering size of the component. (Note that in LabelView, the real work is done by a private `measureWidth()` method.)
- An overridden `onDraw()` method to draw the label onto the provided canvas.

You can see some sample usages of the LabelView custom View in [custom_view_1.xml](#) from the samples. In particular, you can see a mix of both `android:` namespace parameters and custom `app:` namespace parameters. These `app:` parameters are the custom ones that the LabelView recognizes and works with, and are defined in a styleable inner class inside of the samples R resources definition class.

Compound Controls

If you don't want to create a completely customized component, but instead are looking to put together a reusable component that consists of a group of existing controls, then creating a Compound Component (or Compound Control) might fit the bill. In a nutshell, this brings together a number of more atomic controls (or views) into a logical group of items that can be treated as a single thing. For example, a Combo Box can be thought of as a combination of a single line EditText field and an adjacent button with an attached PopupList. If you press the button and select something from the list, it populates the EditText field, but the user can also type something directly into the EditText if they prefer.

In Android, there are actually two other Views readily available to do this: [Spinner](#) and [AutoCompleteTextView](#), but regardless, the concept

of a Combo Box makes an easy-to-understand example.

To create a compound component:

1. The usual starting point is a Layout of some kind, so create a class that extends a Layout. Perhaps in the case of a Combo box we might use a LinearLayout with horizontal orientation. Remember that other layouts can be nested inside, so the compound component can be arbitrarily complex and structured. Note that just like with an Activity, you can use either the declarative (XML-based) approach to creating the contained components, or you can nest them programmatically from your code.
2. In the constructor for the new class, take whatever parameters the superclass expects, and pass them through to the superclass constructor first. Then you can set up the other views to use within your new component; this is where you would create the EditText field and the PopupList. Note that you also might introduce your own attributes and parameters into the XML that can be pulled out and used by your constructor.
3. You can also create listeners for events that your contained views might generate, for example, a listener method for the List Item Click Listener to update the contents of the EditText if a list selection is made.
4. You might also create your own properties with accessors and modifiers, for example, allow the EditText value to be set initially in the component and query for its contents when needed.
5. In the case of extending a Layout, you don't need to override the `onDraw()` and `onMeasure()` methods since the layout will have default behavior that will likely work just fine. However, you can still override them if you need to.
6. You might override other `on...` methods, like `onKeyDown()`, to perhaps choose certain default values from the popup list of a combo box when a certain key is pressed.

To summarize, the use of a Layout as the basis for a Custom Control has a number of advantages, including:

- You can specify the layout using the declarative XML files just like with an activity screen, or you can create views programmatically and nest them into the layout from your code.
- The `onDraw()` and `onMeasure()` methods (plus most of the other `on...` methods) will likely have suitable behavior so you don't have to override them.
- In the end, you can very quickly construct arbitrarily complex compound views and re-use them as if they were a single component.

Examples of Compound Controls

In the API Demos project that comes with the SDK, there are two List examples — Example 4 and Example 6 under Views/Lists demonstrate a SpeechView which extends LinearLayout to make a component for displaying Speech quotes. The corresponding classes in the sample code are `List4.java` and `List6.java`.

Modifying an Existing View Type

There is an even easier option for creating a custom View which is useful in certain circumstances. If there is a component that is already very similar to what you want, you can simply extend that component and just override the behavior that you want to change. You can do all of the things you would do with a fully customized component, but by starting with a more specialized class in the View hierarchy, you can also get a lot of behavior for free that probably does exactly what you want.

For example, the SDK includes a [NotePad application](#) in the samples. This demonstrates many aspects of using the Android platform, among them is extending an EditText View to make a lined notepad. This is not a perfect example, and the APIs for doing this might change from this early preview, but it does demonstrate the principles.

If you haven't done so already, import the NotePad sample into Android Studio (or just look at the source using the link provided). In particular look at the definition of `MyEditText` in the `NoteEditor.java` file.

Some points to note here

1. The Definition

The class is defined with the following line:

```
public static class MyEditText extends EditText
```

- It is defined as an inner class within the `NoteEditor` activity, but it is public so that it could be accessed as `NoteEditor.MyEditText`

from outside of the `NoteEditor` class if desired.

- It is `static`, meaning it does not generate the so-called "synthetic methods" that allow it to access data from the parent class, which in turn means that it really behaves as a separate class rather than something strongly related to `NoteEditor`. This is a cleaner way to create inner classes if they do not need access to state from the outer class, keeps the generated class small, and allows it to be used easily from other classes.
- It extends `EditText`, which is the View we have chosen to customize in this case. When we are finished, the new class will be able to substitute for a normal `EditText` view.

2. Class Initialization

As always, the super is called first. Furthermore, this is not a default constructor, but a parameterized one. The `EditText` is created with these parameters when it is inflated from an XML layout file, thus, our constructor needs to both take them and pass them to the superclass constructor as well.

3. Overridden Methods

In this example, there is only one method to be overridden: `onDraw()` — but there could easily be others needed when you create your own custom components.

For the NotePad sample, overriding the `onDraw()` method allows us to paint the blue lines on the `EditText` view canvas (the canvas is passed into the overridden `onDraw()` method). The `super.onDraw()` method is called before the method ends. The superclass method should be invoked, but in this case, we do it at the end after we have painted the lines we want to include.

4. Use the Custom Component

We now have our custom component, but how can we use it? In the NotePad example, the custom component is used directly from the declarative layout, so take a look at `note_editor.xml` in the `res/layout` folder.

```
<view  
    class="com.android.notepad.NoteEditor$MyEditText"  
    id="@+id/note"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:background="@android:drawable/empty"  
    android:padding="10dip"  
    android:scrollbars="vertical"  
    android:fadingEdge="vertical" />
```

- The custom component is created as a generic view in the XML, and the class is specified using the full package. Note also that the inner class we defined is referenced using the `NoteEditor$MyEditText` notation which is a standard way to refer to inner classes in the Java programming language.

If your custom View component is not defined as an inner class, then you can, alternatively, declare the View component with the XML element name, and exclude the `class` attribute. For example:

```
<com.android.notepad.MyEditText  
    id="@+id/note"  
    ... />
```

Notice that the `MyEditText` class is now a separate class file. When the class is nested in the `NoteEditor` class, this technique will not work.

- The other attributes and parameters in the definition are the ones passed into the custom component constructor, and then passed through to the `EditText` constructor, so they are the same parameters that you would use for an `EditText` view. Note that it is possible to add your own parameters as well, and we will touch on this again below.

And that's all there is to it. Admittedly this is a simple case, but that's the point — creating custom components is only as complicated as you need it to be.

A more sophisticated component may override even more `on...` methods and introduce some of its own helper methods, substantially customizing its properties and behavior. The only limit is your imagination and what you need the component to do.



布局

本文内容

- › 编写 XML
- › 加载 XML 资源
- › 属性
 - › ID
 - › 布局参数
- › 布局位置
- › 尺寸、内边距和外边距
- › 常见布局
- › 使用适配器构建布局
 - › 使用数据填充适配器视图
 - › 处理点击事件

关键类

- › View
- › ViewGroup
- › ViewGroup.LayoutParams

另请参阅

- › 构建简单的用户界面

布局定义用户界面的视觉结构，如Activity或应用小部件的 UI。您可以通过两种方式声明布局：

- 在 XML 中声明 UI 元素。Android 提供了对应于 View 类及其子类的简明 XML 词汇，如用于小部件和布局的词汇；
- 运行时实例化布局元素。您的应用可以通过编程创建 View 对象和 ViewGroup 对象（并操纵其属性）。

Android 框架让您灵活地使用以下一种或两种方法来声明和管理应用的 UI。例如，您可以在 XML 中声明应用的默认布局，包括将出现在布局中的屏幕元素及其属性。然后，您可以在应用中添加可在运行时修改屏幕对象（包括那些已在 XML 中声明的对象）状态的代码。

在 XML 中声明 UI 的优点在于，您可以更好地将应用的外观与控制应用行为的代码隔离。您的 UI 描述位于应用代码外部，这意味着您在修改或调整描述时无需修改您的源代码并重新编译。例如，您可以创建适用于不同屏幕方向、不同设备屏幕尺寸和不同语言的 XML 布局。此外，在 XML 中声明布局还能更轻松地显示 UI 的结构，从而简化问题调试过程。因此，本文将侧重于示范如何在 XML 中声明布局。如果您对在运行时实例化 View 对象感兴趣，请参阅 ViewGroup 类和 View 类的参考资料。

一般而言，用于声明 UI 元素的 XML 词汇严格遵循类和方法的结构和命名方式，其中元素名称对应于类名称，属性名称对应于方法。实际上，这种对应关系往往非常直接，让您可以猜到对应于类方法的 XML 属性，或对应于给定 XML 元素的类。但请注意，并非所有词汇都完全相同。在某些情况下，在命名上略有差异。例如，EditText 元素具有的 text 属性对应的类方法是 EditText.setText()。

• 您还应尝试使用层次结构查看器工具来调试布局—当您在模拟器或设备上进行调试时，它会显示布局属性值、绘制具有内边距/外边距指示符的线框以及完整渲染视图。

• 您可以利用 layoutopt 工具快速分析布局和层次结构中是否存在低效环节或其他问题。

提示：如需了解有关不同布局类型的更多信息，请参阅常见布局对象。

编写 XML

您可以利用 Android 的 XML 词汇，按照在 HTML 中创建包含一系列嵌套元素的网页的相同方式快速设计 UI 布局及其包含的屏幕元素。

每个布局文件都必须只包含一个根元素，并且该元素必须是视图对象或 ViewGroup 对象。定义根元素之后，即可再以子元素的形式添加其他布局对象或小部件，从而逐步构建定义布局的视图层次结构。例如，以下这个 XML 布局使用垂直 `LinearLayout` 来储存一个 `TextView` 和一个 `Button`：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

在 XML 中声明布局后，请在您的 Android 项目 `res/layout/` 目录中以 `.xml` 扩展名保存文件，以便其能够正确编译。

[布局资源](#) 文档中提供了有关布局 XML 文件语法的更多信息。

加载 XML 资源

当您编译应用时，每个 XML 布局文件都会编译到一个 `View` 资源中。您应该在 `Activity.onCreate()` 回调实现中从您的应用代码加载布局资源。请通过调用 `setContentView()`，以 `R.layout.layout_file_name` 形式向其传递对布局资源的引用执行此操作。例如，如果您的 XML 布局保存为 `main_layout.xml`，则需要像下面这样为您的 Activity 加载该布局：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

启动您的 Activity 时，Android 框架会调用 Activity 中的 `onCreate()` 回调方法（请参阅[Activity](#)文档中有关生命周期的阐述）。

属性

每个视图对象和 ViewGroup 对象都支持各自的各类 XML 属性。某些属性是视图对象的专用属性（例如，`TextView` 支持 `textSize` 属性），但这些属性也会被任何可以扩展此类的视图对象继承。某些属性通用于所有 `View` 对象，因为它们继承自根 `View` 类（如 `id` 属性）。此外，其他属性被视为“布局参数”，即描述 `View` 对象特定布局方向的属性，如该对象的父 `ViewGroup` 对象所定义的属性。

ID

任何视图对象都可能具有关联的整型 ID，此 ID 用于在结构树中对 `View` 对象进行唯一标识。编译应用后，此 ID 将作为整型数引用，但在布局 XML 文件中，通常会在 `id` 属性中为该 ID 赋予字符串值。这是所有 `View` 对象共用的 XML 属性（由 `View` 类定义），您会经常用到它。XML 标记内部的 ID 语法是：

```
android:id="@+id/my_button"
```

字符串开头处的 @ 符号指示 XML 解析程序应该解析并展开 ID 字符串的其余部分，并将其标识为 ID 资源。加号 (+) 表示这是一个新的资源名称，必须创建该名称并将其添加到我们的资源（在 `R.java` 文件中）内。Android 框架还提供了许多其他 ID 资源。引用 Android 资源 ID 时，不需要加号，但必须添加 `android` 软件包命名空间，如下所示：

```
        android:id="@+id/empty"
```

添加 `android` 软件包命名空间之后，现在，我们将从 `android.R` 资源类而非本地资源类引用 ID。

要想创建视图并从应用中引用它们，常见的模式是：

1. 在布局文件中定义一个视图/小部件，并为其分配一个唯一的 ID：

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

2. 然后创建一个 `View` 对象实例，并从布局中捕获它（通常使用 `onCreate()` 方法）：

```
Button myButton = (Button) findViewById(R.id.my_button);
```

创建 `RelativeLayout` 时，为 `View` 对象定义 ID 非常重要。在相对布局中，同级视图可以定义其相对于其他同级视图的布局，同级视图通过唯一的 ID 进行引用。

ID 不需要在整个结构树中具有唯一性，但在您要搜索的结构树部分应具有唯一性（要搜索的部分往往是整个结构树，因此最好尽可能具有全局唯一性）。

布局参数

名为 `layout_something` 的 XML 布局属性可为视图定义与其所在的 `ViewGroup` 相适的布局参数。

每个 `ViewGroup` 类都会实现一个扩展 `ViewGroup.LayoutParams` 的嵌套类。此子类包含的属性类型会根据需要为视图组的每个子视图定义尺寸和位置。正如您在图 1 中所见，父视图组为每个子视图（包括子视图组）定义布局参数。

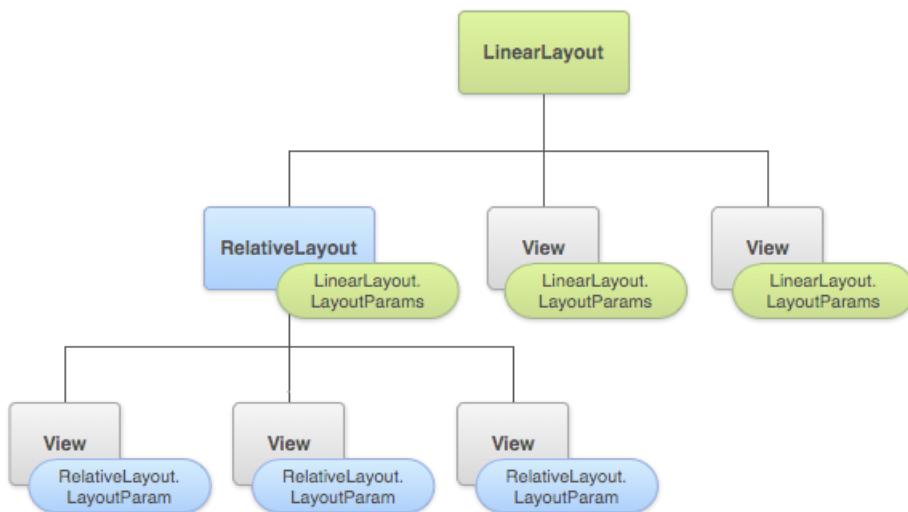


图 1. 以可视化方式表示的视图层次结构，其中包含与每个视图关联的布局参数。

请注意，每个 `LayoutParams` 子类都有自己的值设置语法。每个子元素都必须定义适合其父元素的 `LayoutParams`，但父元素也可为其子元素定义不同的 `LayoutParams`。

所有视图组都包括宽度和高度 (`layout_width` 和 `layout_height`)，并且每个视图都必须定义它们。许多 `LayoutParams` 还包括可选的外边距和边框。

您可以指定具有确切尺寸的宽度和高度，但您多半不想经常这样做。在更多的情况下，您会使用以下常量之一来设置宽度或高度：

- `wrap_content` 指示您的视图将其大小调整为内容所需的尺寸。
- `match_parent` 指示您的视图尽可能采用其父视图组所允许的最大尺寸。

一般而言，建议不要使用绝对单位（如像素）来指定布局宽度和高度，而是使用相对测量单位，如密度无关像素单位 (`dp`)、`wrap_content` 或 `match_parent`，这种方法更好，因为它有助于确保您的应用在各类尺寸的设备屏幕上正确显示。[可用资源](#) 文档中定义了可接受的测量单位类

型。

布局位置

视图的几何形状就是矩形的几何形状。视图具有一个位置（以一对水平向左和垂直向上坐标表示）和两个尺寸（以宽度和高度表示）。位置和尺寸的单位是像素。

可以通过调用方法 `getLeft()` 和方法 `getTop()` 来检索视图的位置。前者会返回表示视图的矩形的水平向左（或称 X 轴）坐标。后者会返回表示视图的矩形的垂直向上（或称 Y 轴）坐标。这些方法都会返回视图相对于其父项的位置。例如，如果 `getLeft()` 返回 20，则意味着视图位于其直接父项左边缘向右 20 个像素处。

此外，系统还提供了几种便捷方法来避免不必要的计算，即 `getRight()` 和 `getBottom()`。这些方法会返回表示视图的矩形的右边缘和下边缘的坐标。例如，调用 `getRight()` 类似于进行以下计算：`getLeft() + getWidth()`。

尺寸、内边距和外边距

视图的尺寸通过宽度和高度表示。视图实际上具有两对宽度和高度值。

第一对称为 **测量宽度** 和 **测量高度**。这些尺寸定义视图想要在其父项内具有的大小。这些测量尺寸可以通过调用 `getMeasuredWidth()` 和 `getMeasuredHeight()` 来获得。

第二对简称为 **宽度** 和 **高度**，有时称为 **绘制宽度** 和 **绘制高度**。这些尺寸定义视图在绘制时和布局后在屏幕上的实际尺寸。这些值可以（但不必）与测量宽度和测量高度不同。宽度和高度可以通过调用 `getWidth()` 和 `getHeight()` 来获得。

要想测量其尺寸，视图需要将其内边距考虑在内。内边距以视图左侧、顶部、右侧和底部各部分的像素数表示。内边距可用于以特定数量的像素弥补视图的内容。例如，左侧内边距为 2，会将视图的内容从左边缘向右推 2 个像素。可以使用 `setPadding(int, int, int, int)` 方法设置内边距，并通过调用 `getPaddingLeft()`、`getPaddingTop()`、`getPaddingRight()` 和 `getPaddingBottom()` 进行查询。

尽管视图可以定义内边距，但它并不支持外边距。不过，视图组可以提供此类支持。如需了解更多信息，请参阅 [ViewGroup](#) 和 [ViewGroup.MarginLayoutParams](#)。

如需了解有关尺寸的详细信息，请参阅 [尺寸值](#)。

常见布局

[ViewGroup](#) 类的每个子类都提供了一种独特的方式来显示您在其中嵌套的视图。以下是 Android 平台中内置的一些较为常见的布局类型。

注：尽管您可以通过将一个或多个布局嵌套在另一个布局内来实现您的 UI 设计，但应该使您的布局层次结构尽可能简略。布局的嵌套布局越少，绘制速度越快（扁平的视图层次结构优于深层的视图层次结构）。

线性布局



一种使用单个水平行或垂直行来组织子项的布局。它会在窗口长度超出屏幕长度时创建一个滚动条。

相对布局



让您能够指定子对象彼此之间的相对位置（子对象 A 在子对象 B 左侧）或子对象与父对象的相对位置（与父对象顶部对齐）。

网页视图



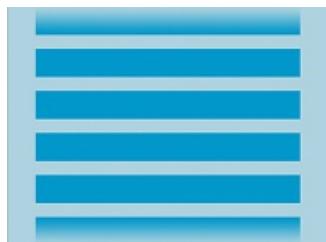
显示网页。

使用适配器构建布局

如果布局的内容是属于动态或未预先确定的内容，您可以使用这样一种布局：在运行时通过子类 `AdapterView` 用视图填充布局。`AdapterView` 类的子类使用 `Adapter` 将数据与其布局绑定。`Adapter` 充当数据源与 `AdapterView` 布局之间的中间人—`Adapter`（从数组或数据库查询等来源）检索数据，并将每个条目转换为可以添加到 `AdapterView` 布局中的视图。

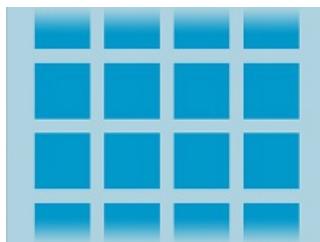
适配器支持的常见布局包括：

列表视图



显示滚动的单列列表。

网格视图



显示滚动的行列网格。

使用数据填充适配器视图

您可以通过将 `AdapterView` 实例与 `Adapter` 绑定来填充 `AdapterView`（如 `ListView` 或 `GridView`），此操作会从外部来源检索数据，并创建表示每个数据条目的 `View`。

Android 提供了几个 `Adapter` 子类，用于检索不同种类的数据和构建 `AdapterView` 的视图。两种最常见的适配器是：

ArrayAdapter

请在数据源为数组时使用此适配器。默认情况下， `ArrayAdapter` 会通过在每个项目上调用 `toString()` 并将内容放入 `TextView` 来为每个数组项创建视图。

例如，如果您具有想要在 `ListView` 中显示的字符串数组，请使用构造函数初始化一个新的 `ArrayAdapter`，为每个字符串和字符串数组指定布局：

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
    android.R.layout.simple_list_item_1, myStringArray);
```

此构造函数的参数是：

- 您的应用 `Context`
- 包含数组中每个字符串的 `TextView` 的布局
- 字符串数组

然后，只需在您的 `ListView` 上调用 `setAdapter()`：

```
ListView listView = (ListView) findViewById(R.id.listview);  
listView.setAdapter(adapter);
```

要想自定义每个项的外观，您可以重写数组中各个对象的 `toString()` 方法。或者，要想为 `TextView` 之外的每个项创建视图（例如，如果您想为每个数组项创建一个 `ImageView`），请扩展 `ArrayAdapter` 类并重写 `getView()` 以返回您想要为每个项获取的视图类型。

SimpleCursorAdapter

请在数据来自 `Cursor` 时使用此适配器。使用 `SimpleCursorAdapter` 时，您必须指定要为 `Cursor` 中的每个行使用的布局，以及应该在哪些布局视图中插入 `Cursor` 中的哪些列。例如，如果您想创建人员姓名和电话号码列表，则可以执行一个返回 `Cursor`（包含对应每个人的行，以及对应姓名和号码的列）的查询。然后，您可以创建一个字符串数组，指定您想要在每个结果的布局中包含 `Cursor` 中的哪些列，并创建一个整型数组，指定应该将每个列放入的对应视图：

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
                        ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

当您实例化 `SimpleCursorAdapter` 时，请传递要用于每个结果的布局、包含结果的 `Cursor` 以及以下两个数组：

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
    R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);
ListView listView = getListView();
listView.setAdapter(adapter);
```

然后，`SimpleCursorAdapter` 会使用提供的布局，将每个 `fromColumns` 项插入对应的 `toViews` 视图，为 `Cursor` 中的每个行创建一个视图。

如果您在应用的生命周期中更改了适配器读取的底层数据，则应调用 `notifyDataSetChanged()`。此操作会通知附加的视图，数据发生了变化，它应该自行刷新。

处理点击事件

您可以通过实现 `AdapterView.OnItemClickListener` 界面来响应 `AdapterView` 中每一项上的点击事件。例如：

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position, long id) {
        // Do something in response to the click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```

线性布局

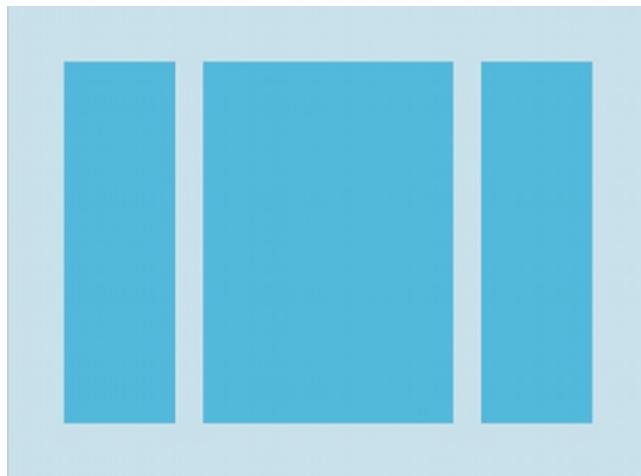
本文内容

- › [布局权重](#)
- › [示例](#)

关键类

- › [LinearLayout](#)
- › [LinearLayout.LayoutParams](#)

[LinearLayout](#) 是一个视图组，用于使所有子视图在单个方向（垂直或水平）保持对齐。您可以使用 `android:orientation` 属性指定布局方向。



[LinearLayout](#) 的所有子视图依次堆叠，因此无论子视图有多宽，垂直列表每行均只有一个子视图，水平列表将只有一行高（最高子视图的高度加上内边距）。[LinearLayout](#) 遵守子视图之间的“边距”以及每个子视图的“重力”（右对齐、居中对齐、左对齐）。

布局权重

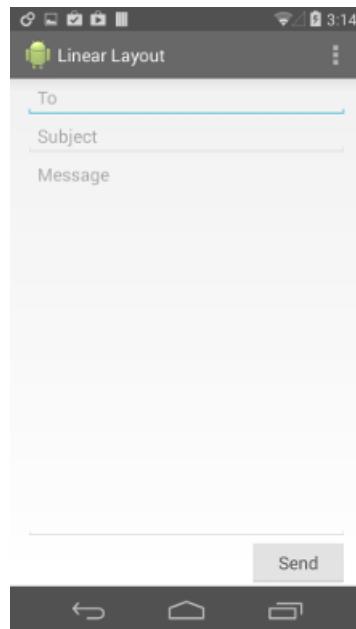
[LinearLayout](#) 还支持使用 `android:layout_weight` 属性为各个子视图分配权重。此属性根据视图应在屏幕上占据的空间量向视图分配“重要性”值。权重值更大的视图可以填充父视图中任何剩余的空间。子视图可以指定权重值，然后系统会按照子视图声明的权重值的比例，将视图组中的任何剩余空间分配给子视图。默认权重为零。

例如，如果有三个文本字段，其中两个声明权重为 1，另一个没有赋予权重，则没有权重的第三个文本字段将不会扩展，并且仅占据其内容所需的区域。另外两个文本字段将以同等幅度进行扩展，以填充所有三个字段都测量后还剩余的空间。如果为第三个字段提供权重 2（而非 0），那么相当于声明现在它比其他两个字段更为重要，因此，它将获得总剩余空间的一半，其他两个均享余下空间。

示例

权重相等的子视图

要创建一个线性布局，让每个子视图在屏幕上都占据相同的空间量，则将每个视图的 `android:layout_height` 均设置为 "`0dp`"（对于垂直布局），或将每个视图的 `android:layout_width` 均设置为 "`0dp`"（对于水平布局）。然后，将每个视图的 `android:layout_weight` 均设置为 "`1`"。



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

有关 [LinearLayout](#) 的每个子视图可用属性的详情，请参阅 [LinearLayout.LayoutParams](#)。

RelativeLayout

In this document

- › [Positioning Views](#)
- › [Example](#)

Key classes

- › [RelativeLayout](#)
- › [RelativeLayout.LayoutParams](#)

[RelativeLayout](#) is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent [RelativeLayout](#) area (such as aligned to the bottom, left or center).



A [RelativeLayout](#) is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. If you find yourself using several nested [LinearLayout](#) groups, you may be able to replace them with a single [RelativeLayout](#).

Positioning Views

[RelativeLayout](#) lets child views specify their position relative to the parent view or to each other (specified by ID). So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from [RelativeLayout.LayoutParams](#).

Some of the many layout properties available to views in a [RelativeLayout](#) include:

`android:layout_alignParentTop`

If "true", makes the top edge of this view match the top edge of the parent.

`android:layout_centerVertical`

If "true", centers this child vertically within its parent.

`android:layout_below`

Positions the top edge of this view below the view specified with a resource ID.

`android:layout_toRightOf`

Positions the left edge of this view to the right of the view specified with a resource ID.

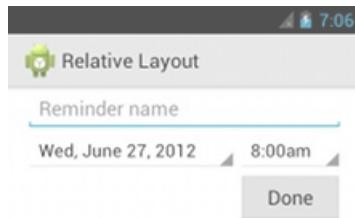
These are just a few examples. All layout attributes are documented at [RelativeLayout.LayoutParams](#).

The value for each layout property is either a boolean to enable a layout position relative to the parent [RelativeLayout](#) or an ID that references another view in the layout against which the view should be positioned.

In your XML layout, dependencies against other views in the layout can be declared in any order. For example, you can declare that "view1" be positioned below "view2" even if "view2" is the last view declared in the hierarchy. The example below demonstrates such a scenario.

Example

Each of the attributes that control the relative position of each view are emphasized.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@+id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

For details about all the layout attributes available to each child view of a `RelativeLayout`, see [RelativeLayout.LayoutParams](#).

列表视图

本文内容

- › [使用加载器](#)
- › [示例](#)

关键类

- › [ListView](#)
- › [Adapter](#)
- › [CursorLoader](#)

另请参阅

- › [加载器](#)

[ListView](#) 是一个显示一列可滚动项目的视图组。 系统使用 [Adapter](#) 自动将列表项目插入列表，适配器从来源（例如数组或数据库查询）提取内容，并将每个项目结果转换为视图放置到列表中。

有关如何使用适配器动态插入视图的介绍，请阅读[使用适配器构建布局](#)。



使用加载器

使用 [CursorLoader](#) 是以异步任务形式查询 [Cursor](#) 的标准方式，可避免查询阻塞应用的主线程。当 [CursorLoader](#) 接收到 [cursor](#) 结果时，[LoaderCallbacks](#) 会收到对 [onLoadFinished\(\)](#) 的回调，从中您可以使用新的 [Cursor](#) 更新 [Adapter](#)，然后列表视图会显示结果。

虽然 [CursorLoader](#) API 首先是在 Android 3.0 (API 级别 11) 中引入，但 [支持库](#) 中也有提供，因此您的应用可在使用这些 API 的同时，仍为 Android 1.6 或更高版本的设备提供支持。

如需了解有关使用 [Loader](#) 异步加载数据的详细信息，请参阅[加载器指南](#)。

示例

下例使用 [ListActivity](#)，该 Activity 包含一个 [ListView](#) 作为其仅有的默认布局元素。它对[联系人提供程序](#)执行查询，以获取姓名和电话号码清单。

Activity 实现 [LoaderCallbacks](#) 接口，以使用 [CursorLoader](#) 为列表视图动态加载数据。

```
public class ListViewLoader extends ListActivity
    implements LoaderManager.LoaderCallbacks<Cursor> {

    // This is the Adapter being used to display the list's data
    SimpleCursorAdapter mAdapter;

    // These are the Contacts rows that we will retrieve
    static final String[] PROJECTION = new String[] {ContactsContract.Data._ID,
        ContactsContract.Data.DISPLAY_NAME};

    // This is the select criteria
    static final String SELECTION = "((" +
        ContactsContract.Data.DISPLAY_NAME + " NOTNULL) AND (" +
        ContactsContract.Data.DISPLAY_NAME + " != '' ))";

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

        // Create a progress bar to display while the list loads
        ProgressBar progressBar = new ProgressBar(this);
        progressBar.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT, Gravity.CENTER));
        progressBar.setIndeterminate(true);
        getListView().setEmptyView(progressBar);

        // Must add the progress bar to the root of the layout
        ViewGroup root = (ViewGroup) findViewById(android.R.id.content);
        root.addView(progressBar);

        // For the cursor adapter, specify which columns go into which views
        String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME};
        int[] toViews = {android.R.id.text1}; // The TextView in simple_list_item_1

        // Create an empty adapter we will use to display the loaded data.
        // We pass null for the cursor, then update it in onLoadFinished()
        mAdapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_1, null,
            fromColumns, toViews, 0);
        setListAdapter(mAdapter);

        // Prepare the loader. Either re-connect with an existing one,
        // or start a new one.
        getLoaderManager().initLoader(0, null, this);
    }

    // Called when a new Loader needs to be created
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        // Now create and return a CursorLoader that will take care of
        // creating a Cursor for the data being displayed.
        return new CursorLoader(this, ContactsContract.Data.CONTENT_URI,
            PROJECTION, SELECTION, null, null);
    }

    // Called when a previously created loader has finished loading
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        // Swap the new cursor in. (The framework will take care of closing the
        // old cursor once we return.)
        mAdapter.swapCursor(data);
    }

    // Called when a previously created loader is reset, making the data unavailable
    public void onLoaderReset(Loader<Cursor> loader) {
        // This is called when the last Cursor provided to onLoadFinished()
        // above is about to be closed. We need to make sure we are no
        // longer using it.
        mAdapter.swapCursor(null);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Do something when a list item is clicked
    }
}
```

```
    }  
}
```

注：由于此示例在[联系人提供程序](#)中执行查询，因此，如果您想要试用此代码，您的应用必须在清单文件中请求 `READ_CONTACTS` 权限：

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

Grid View

In this document

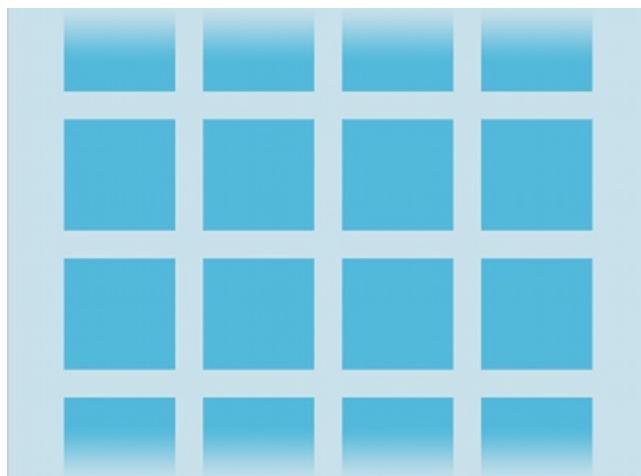
› [Example](#)

Key classes

› [GridView](#)
› [ImageView](#)
› [BaseAdapter](#)
› [AdapterView.OnItemClickListener](#)

[GridView](#) is a [ViewGroup](#) that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a [ListAdapter](#).

For an introduction to how you can dynamically insert views using an adapter, read [Building Layouts with an Adapter](#).



Example

In this tutorial, you'll create a grid of image thumbnails. When an item is selected, a toast message will display the position of the image.

1. Start a new project named *HelloGridView*.
2. Find some photos you'd like to use, or [download these sample images](#). Save the image files into the project's `res/drawable/` directory.
3. Open the `res/layout/main.xml` file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

This `GridView` will fill the entire screen. The attributes are rather self explanatory. For more information about valid attributes, see the [GridView reference](#).

4. Open `HelloGridView.java` and insert the following code for the `onCreate()` method:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v,
            int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position,
                Toast.LENGTH_SHORT).show();
        }
    });
}
```

After the `main.xml` layout is set for the content view, the `GridView` is captured from the layout with `findViewById(int)`. The `setAdapter()` method then sets a custom adapter (`ImageAdapter`) as the source for all items to be displayed in the grid. The `ImageAdapter` is created in the next step.

To do something when an item in the grid is clicked, the `setOnItemClickListener()` method is passed a new `AdapterView.OnItemClickListener`. This anonymous instance defines the `onItemClick()` callback method to show a `Toast` that displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

5. Create a new class called `ImageAdapter` that extends `BaseAdapter`:

```

public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {
            // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = {
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7
    };
}

```

First, this implements some required methods inherited from `BaseAdapter`. The constructor and `getCount()` are self-explanatory. Normally, `getItem(int)` should return the actual object at the specified position in the adapter, but it's ignored for this example. Likewise, `getItemId(int)` should return the row id of the item, but it's not needed here.

The first method necessary is `getView()`. This method creates a new `View` for each image added to the `ImageAdapter`. When this is called, a `View` is passed in, which is normally a recycled object (at least after this has been called once), so there's a check to see if the object is null. If it is null, an `ImageView` is instantiated and configured with desired properties for the image presentation:

- `setLayoutParams(ViewGroup.LayoutParams)` sets the height and width for the View—this ensures that, no matter the size of the drawable, each image is resized and cropped to fit in these dimensions, as appropriate.
- `setScaleType(ImageView.ScaleType)` declares that images should be cropped toward the center (if necessary).
- `setPadding(int, int, int, int)` defines the padding for all sides. (Note that, if the images have different aspect-ratios, then less padding will cause more cropping of the image if it does not match the dimensions given to the `ImageView`.)

If the `View` passed to `getView()` is *not* null, then the local `ImageView` is initialized with the recycled `View` object.

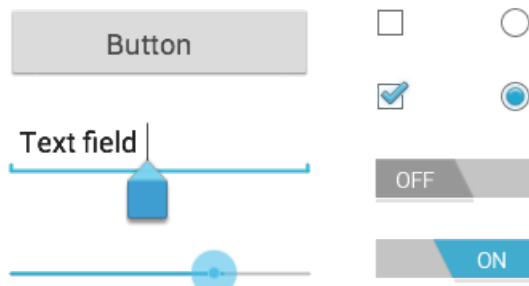
At the end of the `getView()` method, the `position` integer passed into the method is used to select an image from the `mThumbIds` array, which is set as the image resource for the `ImageView`.

All that's left is to define the `mThumbIds` array of drawable resources.

6. Run the application.

Try experimenting with the behaviors of the `GridView` and `ImageView` elements by adjusting their properties. For example, instead of using `setLayoutParams(ViewGroup.LayoutParams)`, try using `setAdjustViewBounds(boolean)`.

输入控件



输入控件是您的应用用户界面中的交互式组件。Android 提供了多种可在 UI 中使用的控件，如按钮、文本字段、定位栏、复选框、缩放按钮、切换按钮等。

向 UI 中添加输入控件与向 [XML 布局](#) 中添加 XML 元素一样简单。例如，以下是一个包含文本字段和按钮的布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button android:id="@+id/button_send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
</LinearLayout>
```

每个输入控件都支持一组特定的输入事件，以便您处理用户输入文本或触摸按钮等事件。

通用控件

以下列出了您可以在应用中使用的一些通用控件。点击链接可了解有关各控件用法的详情。

注：除了此处列出的控件外，Android 还提供了几种其他控件。浏览 [android.widget](#) 软件包可发现更多控件。如果您的应用需要特定类型的输入控件，则可以构建您自己的[自定义组件](#)。

控件类型	说明	相关类
按钮	可由用户按压或点击来执行操作的按钮。	Button
文本字段	一种可编辑的文本字段。您可以使用 AutoCompleteTextView 小部件创建提供自动完成建议的文本输入小部件	EditText 、 AutoCompleteTextView
复选框	可由用户切换的启用/禁用开关。您应该在向用户呈现一组不互斥的可选选项时使用复选框。	CheckBox
单选按钮	与复选框类似，不同的是只能选择组中的一个选项。	RadioGroup RadioButton
切换按钮	一种具有指示灯的开/关按钮。	ToggleButton
微调框	一种允许用户从值集中选择一个值的下拉列表。	Spinner

选取器

一种供用户通过使用向上/向下按钮或轻扫手势选择值集中单个值的对话框。使用 [DatePicker](#) 小部件输入日期（月、日、年）值，或使用 [TimePicker](#) 小部件输入时间（小时、分钟、上午/下午）值，系统将根据用户的语言区域自动设置所输入内容的格式。

[DatePicker](#)、[TimePicker](#)

Buttons

In this document

- › [Responding to Click Events](#)
 - › [Using an OnClickListener](#)
- › [Styling Your Button](#)
 - › [Borderless button](#)
 - › [Custom background](#)

Key classes

- › [Button](#)
- › [ImageButton](#)

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.



Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

- With text, using the [Button](#) class:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    ... />
```

- With an icon, using the [ImageButton](#) class:

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/button_icon"  
    ... />
```

- With text and an icon, using the [Button](#) class with the [android:drawableLeft](#) attribute:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    android:drawableLeft="@drawable/button_icon"  
    ... />
```

Responding to Click Events

When the user clicks a button, the [Button](#) object receives an on-click event.

To define the click event handler for a button, add the [android:onClick](#) attribute to the [`<Button>`](#) element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The [Activity](#) hosting the layout must then

implement the corresponding method.

For example, here's a layout with a button using `android:onClick`:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Within the `Activity` that hosts this layout, the following method handles the click event:

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a `View` as its only parameter (this will be the `View` that was clicked)

Using an OnClickListener

You can also declare the click event handler programmatically rather than in an XML layout. This might be necessary if you instantiate the `Button` at runtime or you need to declare the click behavior in a `Fragment` subclass.

To declare the event handler programmatically, create an `View.OnClickListener` object and assign it to the button by calling `setOnClickListener(View.OnClickListener)`. For example:

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Styling Your Button

The appearance of your button (background image and font) may vary from one device to another, because devices by different manufacturers often have different default styles for input controls.

You can control exactly how your controls are styled using a theme that you apply to your entire application. For instance, to ensure that all devices running Android 4.0 and higher use the Holo theme in your app, declare `android:theme="@android:style/Theme.Holo"` in your manifest's `<application>` element. Also read the blog post, [Holo Everywhere](#) for information about using the Holo theme while supporting older devices.

To customize individual buttons with a different background, specify the `android:background` attribute with a drawable or color resource. Alternatively, you can apply a `style` for the button, which works in a manner similar to HTML styles to define multiple style properties such as the background, font, size, and others. For more information about applying styles, see [Styles and Themes](#).

Borderless button

One design that can be useful is a "borderless" button. Borderless buttons resemble basic buttons except that they have no borders or background but still change appearance during different states, such as when clicked.

To create a borderless button, apply the `borderlessButtonStyle` style to the button. For example:

```
<Button  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage"  
    style="?android:attr/borderlessButtonStyle" />
```

Custom background

If you want to truly redefine the appearance of your button, you can specify a custom background. Instead of supplying a simple bitmap or color, however, your background should be a state list resource that changes appearance depending on the button's current state.

You can define the state list in an XML file that defines three different images or colors to use for the different button states.

To create a state list drawable for your button background:

1. Create three bitmaps for the button background that represent the default, pressed, and focused button states.

To ensure that your images fit buttons of various sizes, create the bitmaps as [Nine-patch](#) bitmaps.

2. Place the bitmaps into the `res/drawable/` directory of your project. Be sure each bitmap is named properly to reflect the button state that they each represent, such as `button_default.9.png`, `button_pressed.9.png`, and `button_focused.9.png`.
3. Create a new XML file in the `res/drawable/` directory (name it something like `button_custom.xml`). Insert the following XML:

```
<?xml version="1.0" encoding="utf-8"?>  
<selector xmlns:android="http://schemas.android.com/apk/res/android">  
    <item android:drawable="@drawable/button_pressed"  
        android:state_pressed="true" />  
    <item android:drawable="@drawable/button_focused"  
        android:state_focused="true" />  
    <item android:drawable="@drawable/button_default" />  
</selector>
```

This defines a single drawable resource, which will change its image based on the current state of the button.

- The first `<item>` defines the bitmap to use when the button is pressed (activated).
- The second `<item>` defines the bitmap to use when the button is focused (when the button is highlighted using the trackball or directional pad).
- The third `<item>` defines the bitmap to use when the button is in the default state (it's neither pressed nor focused).

Note: The order of the `<item>` elements is important. When this drawable is referenced, the `<item>` elements are traversed in-order to determine which one is appropriate for the current button state. Because the default bitmap is last, it is only applied when the conditions `android:state_pressed` and `android:state_focused` have both evaluated as false.

This XML file now represents a single drawable resource and when referenced by a [Button](#) for its background, the image displayed will change based on these three states.

4. Then simply apply the drawable XML file as the button background:

```
<Button  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage"  
    android:background="@drawable/button_custom" />
```

For more information about this XML syntax, including how to define a disabled, hovered, or other button states, read about [State List Drawable](#).

Specifying the Input Method Type

This lesson teaches you to

- [Specify the Keyboard Type](#)
- [Enable Spelling Suggestions and Other Behaviors](#)
- [Specify the Input Method Action](#)
- [Provide AutoComplete Suggestions](#)

Every text field expects a certain type of text input, such as an email address, phone number, or just plain text. So it's important that you specify the input type for each text field in your app so the system displays the appropriate soft input method (such as an on-screen keyboard).

Beyond the type of buttons available with an input method, you should specify behaviors such as whether the input method provides spelling suggestions, capitalizes new sentences, and replaces the carriage return button with an action button such as a **Done** or **Next**. This lesson shows how to specify these characteristics.

Specify the Keyboard Type

You should always declare the input method for your text fields by adding the `android:inputType` attribute to the `<EditText>` element.



Figure 1. The `phone` input type.

For example, if you'd like an input method for entering a phone number, use the "phone" value:

```
<EditText  
    android:id="@+id/phone"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/phone_hint"  
    android:inputType="phone" />
```

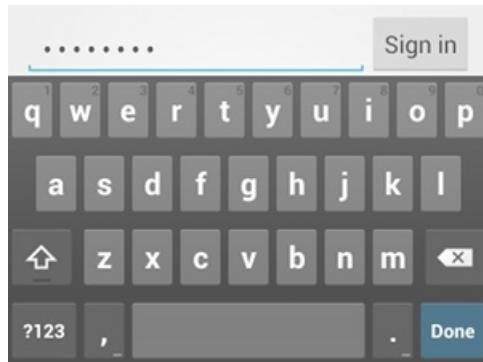


Figure 2. The `textPassword` input type.

Or if the text field is for a password, use the `"textPassword"` value so the text field conceals the user's input:

```
<EditText  
    android:id="@+id/password"  
    android:hint="@string/password_hint"  
    android:inputType="textPassword"  
    ... />
```

There are several possible values documented with the `android:inputType` attribute and some of the values can be combined to specify the input method appearance and additional behaviors.

Enable Spelling Suggestions and Other Behaviors

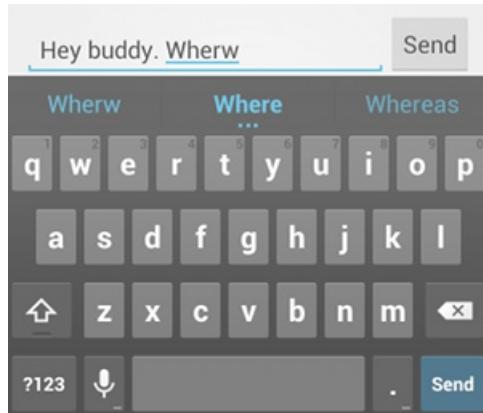


Figure 3. Adding `textAutoCorrect` provides auto-correction for misspellings.

The `android:inputType` attribute allows you to specify various behaviors for the input method. Most importantly, if your text field is intended for basic text input (such as for a text message), you should enable auto spelling correction with the `"textAutoCorrect"` value.

You can combine different behaviors and input method styles with the `android:inputType` attribute. For example, here's how to create a text field that capitalizes the first word of a sentence and also auto-corrects misspellings:

```
<EditText  
    android:id="@+id/message"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:inputType=  
        "textCapSentences|textAutoCorrect"  
    ... />
```

Specify the Input Method Action

Most soft input methods provide a user action button in the bottom corner that's appropriate for the current text field. By default, the system uses this button for either a **Next** or **Done** action unless your text field allows multi-line text (such as with `android:inputType="textMultiLine"`), in which case the action button is a carriage return. However, you can specify additional actions that might be more appropriate for your text field, such as **Send** or **Go**.

To specify the keyboard action button, use the `android:imeOptions` attribute with an action value such as `"actionSend"` or `"actionSearch"`. For example:



Figure 4. The Send button appears when you declare `android:imeOptions="actionSend"`.

```
<EditText  
    android:id="@+id/search"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/search_hint"  
    android:inputType="text"  
    android:imeOptions="actionSend" />
```

You can then listen for presses on the action button by defining a `TextView.OnEditorActionListener` for the `EditText` element. In your listener, respond to the appropriate IME action ID defined in the `EditorInfo` class, such as `IME_ACTION_SEND`. For example:

```
EditText editText = (EditText) findViewById(R.id.search);  
editText.setOnEditorActionListener(new OnEditorActionListener() {  
    @Override  
    public boolean onEditorAction(TextView v, int actionBarId, KeyEvent event) {  
        boolean handled = false;  
        if (actionBarId == EditorInfo.IME_ACTION_SEND) {  
            sendMessage();  
            handled = true;  
        }  
        return handled;  
    }  
});
```

Provide Auto-complete Suggestions

If you want to provide suggestions to users as they type, you can use a subclass of `EditText` called `AutoCompleteTextView`. To implement auto-complete, you must specify an `Adapter` that provides the text suggestions. There are several kinds of adapters available, depending on where the data is coming from, such as from a database or an array.

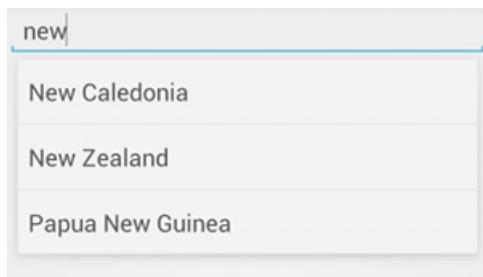


Figure 5. Example of `AutoCompleteTextView` with text suggestions.

The following procedure describes how to set up an `AutoCompleteTextView` that provides suggestions from an array, using `ArrayAdapter`:

1. Add the `AutoCompleteTextView` to your layout. Here's a layout with only the text field:

```
<?xml version="1.0" encoding="utf-8"?>
<AutoCompleteTextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

2. Define the array that contains all text suggestions. For example, here's an array of country names that's defined in an XML resource file (`res/values/strings.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Afghanistan</item>
        <item>Albania</item>
        <item>Algeria</item>
        <item>American Samoa</item>
        <item>Andorra</item>
        <item>Angola</item>
        <item>Anguilla</item>
        <item>Antarctica</item>
        ...
    </string-array>
</resources>
```

3. In your `Activity` or `Fragment`, use the following code to specify the adapter that supplies the suggestions:

```
// Get a reference to the AutoCompleteTextView in the layout
AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.autocomplete_country);
// Get the string array
String[] countries = getResources().getStringArray(R.array.countries_array);
// Create the adapter and set it to the AutoCompleteTextView
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, countries);
textView.setAdapter(adapter);
```

Here, a new `ArrayAdapter` is initialized to bind each item in the `countries_array` string array to a `TextView` that exists in the `simple_list_item_1` layout (this is a layout provided by Android that provides a standard appearance for text in a list).

4. Assign the adapter to the `AutoCompleteTextView` by calling `setAdapter()`.

Checkboxes

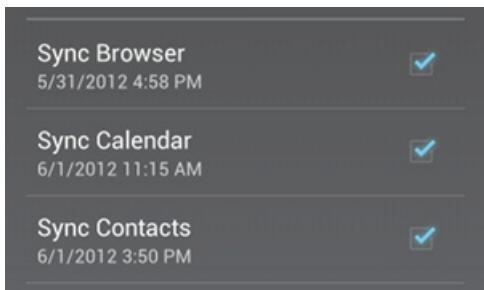
In this document

› [Responding to Click Events](#)

Key classes

› [CheckBox](#)

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.



To create each checkbox option, create a [CheckBox](#) in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

Responding to Click Events

When the user selects a checkbox, the [CheckBox](#) object receives an on-click event.

To define the click event handler for a checkbox, add the `android:onClick` attribute to the `<CheckBox>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The [Activity](#) hosting the layout must then implement the corresponding method.

For example, here are a couple [CheckBox](#) objects in a list:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Within the [Activity](#) that hosts this layout, the following method handles the click event for both checkboxes:

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a `View` as its only parameter (this will be the `View` that was clicked)

Tip: If you need to change the checkbox state yourself (such as when loading a saved `CheckBoxPreference`), use the `setChecked(boolean)` or `toggle()` method.

Radio Buttons

In this document

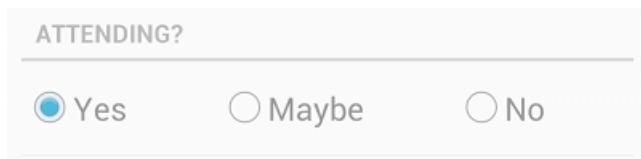
› [Responding to Click Events](#)

Key classes

› [RadioButton](#)

› [RadioGroup](#)

Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a [spinner](#) instead.



To create each radio button option, create a [RadioButton](#) in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a [RadioGroup](#). By grouping them together, the system ensures that only one radio button can be selected at a time.

Responding to Click Events

When the user selects one of the radio buttons, the corresponding [RadioButton](#) object receives an on-click event.

To define the click event handler for a button, add the `android:onClick` attribute to the `<RadioButton>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The [Activity](#) hosting the layout must then implement the corresponding method.

For example, here are a couple [RadioButton](#) objects:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

Note: The [RadioGroup](#) is a subclass of [LinearLayout](#) that has a vertical orientation by default.

Within the [Activity](#) that hosts this layout, the following method handles the click event for both radio buttons:

```
public void onRadioButtonClicked(View view) {  
    // Is the button now checked?  
    boolean checked = ((RadioButton) view).isChecked();  
  
    // Check which radio button was clicked  
    switch(view.getId()) {  
        case R.id.radio_pirates:  
            if (checked)  
                // Pirates are the best  
            break;  
        case R.id.radio_ninjas:  
            if (checked)  
                // Ninjas rule  
            break;  
    }  
}
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a `View` as its only parameter (this will be the `View` that was clicked)

Tip: If you need to change the radio button state yourself (such as when loading a saved `CheckBoxPreference`), use the `setChecked(boolean)` or `toggle()` method.

Toggle Buttons

In this document

» [Responding to Button Presses](#)

Key classes

» [ToggleButton](#)

» [Switch](#)

» [SwitchCompat](#)

» [CompoundButton](#)

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the [ToggleButton](#) object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a [Switch](#) object. [SwitchCompat](#) is a version of the Switch widget which runs on devices back to API 7.

If you need to change a button's state yourself, you can use the [CompoundButton.setChecked\(\)](#) or [CompoundButton.toggle\(\)](#) methods.



Toggle buttons



Switches (in Android 4.0+)

Responding to Button Presses

To detect when the user activates the button or switch, create an [CompoundButton.OnCheckedChangeListener](#) object and assign it to the button by calling [setOnCheckedChangeListener\(\)](#). For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

微调框

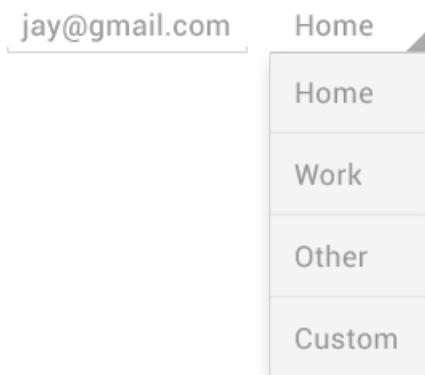
本文内容

- › [使用用户选择填充微调框](#)
- › [响应用户选择](#)

关键类

- › [Spinner](#)
- › [SpinnerAdapter](#)
- › [AdapterView.OnItemSelectedListener](#)

微调框提供一种方法，让用户可以从值集中快速选择一个值。默认状态下，微调框显示其当前所选的值。触摸微调框可显示下拉菜单，其中列有所有其他可用值，用户可从中选择一个新值。



您可以使用 [Spinner](#) 对象向您的布局中添加一个微调框。通常应在 XML 布局中使用 `<Spinner>` 元素来执行此操作。例如：

```
<Spinner  
    android:id="@+id/planets_spinner"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

要使用选择列表填充微调框，还需要在 [Activity](#) 或 [Fragment](#) 源代码中指定 [SpinnerAdapter](#)。

使用用户选择填充微调框

您为微调框提供的选择可来自任何来源，但必须通过 [SpinnerAdapter](#) 来提供，例如，如果选择可通过数组获取，则通过 [ArrayAdapter](#) 来提供，如果选择可通过数据库查询获取，则通过 [CursorAdapter](#) 来提供。

例如，如果预先确定了微调框的可用选择，则可通过[字符串资源文件](#)中定义的字符串数组来提供这些选择。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

对于上例所示数组，可在 `Activity` 或 `Fragment` 中使用以下代码，以使用 `ArrayAdapter` 实例为微调框提供该数组。

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

`createFromResource()` 方法允许从字符串数组创建 `ArrayAdapter`。此方法的第三个参数是布局资源，其定义所选选择如何显示在微调框控件中。`simple_spinner_item` 布局由平台提供，是默认布局，除非您想为微调框外观定义自己的布局，否则应使用此布局。

然后，应调用 `setDropDownViewResource(int)` 指定适配器应用于显示微调框选择列表的布局（`simple_spinner_dropdown_item` 是平台定义的另一标准布局）。

调用 `setAdapter()` 以将适配器应用到 `Spinner`。

响应用户选择

当用户从下拉菜单中选择一个项目时，`Spinner` 对象会收到一个 `on-item-selected` 事件。

要为微调框定义选择事件处理程序，请实现 `AdapterView.OnItemSelectedListener` 接口以及相应的 `onItemSelected()` 回调方法。例如，以下是 `Activity` 中的一个接口实现：

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {
    ...
    public void onItemSelected(AdapterView<?> parent, View view,
        int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }
    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

`AdapterView.OnItemSelectedListener` 需要 `onItemSelected()` 和 `onNothingSelected()` 回调方法。

然后，您需要调用 `setOnItemSelectedListener()` 来指定接口实现：

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

如果您使用 `Activity` 或 `Fragment` 来实现 `AdapterView.OnItemSelectedListener` 接口（如上例），则可以传递 `this` 作为接口实例。

Pickers

In this document

- › [Creating a Time Picker](#)
 - › [Extending DialogFragment for a time picker](#)
 - › [Showing the time picker](#)
- › [Creating a Date Picker](#)
 - › [Extending DialogFragment for a date picker](#)
 - › [Showing the date picker](#)

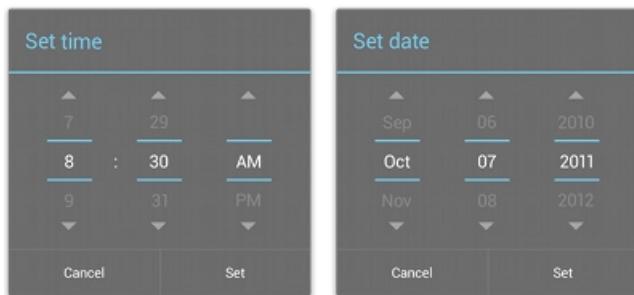
Key classes

- › [DatePickerDialog](#)
- › [TimePickerDialog](#)
- › [DialogFragment](#)

See also

- › [Fragments](#)

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.



We recommend that you use [DialogFragment](#) to host each time or date picker. The [DialogFragment](#) manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.

Although [DialogFragment](#) was first added to the platform in Android 3.0 (API level 11), if your app supports versions of Android older than 3.0—even as low as Android 1.6—you can use the [DialogFragment](#) class that's available in the [support library](#) for backward compatibility.

Note: The code samples below show how to create dialogs for a time picker and date picker using the [support library](#) APIs for [DialogFragment](#). If your app's `minSdkVersion` is 11 or higher, you can instead use the platform version of [DialogFragment](#).

Creating a Time Picker

To display a [TimePickerDialog](#) using [DialogFragment](#), you need to define a fragment class that extends [DialogFragment](#) and return a [TimePickerDialog](#) from the fragment's `onCreateDialog()` method.

Note: If your app supports versions of Android older than 3.0, be sure you've set up your Android project with the support library as described in [Setting Up a Project to Use a Library](#).

Extending DialogFragment for a time picker

To define a `DialogFragment` for a `TimePickerDialog`, you must:

- Define the `onCreateDialog()` method to return an instance of `TimePickerDialog`
- Implement the `TimePickerDialog.OnTimeSetListener` interface to receive a callback when the user sets the time.

Here's an example:

```
public static class TimePickerFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

See the `TimePickerDialog` class for information about the constructor arguments.

Now all you need is an event that adds an instance of this fragment to your activity.

Showing the time picker

Once you've defined a `DialogFragment` like the one shown above, you can display the time picker by creating an instance of the `DialogFragment` and calling `show()`.

For example, here's a button that, when clicked, calls a method to show the dialog:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />
```

When the user clicks this button, the system calls the following method:

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

This method calls `show()` on a new instance of the `DialogFragment` defined above. The `show()` method requires an instance of `FragmentManager` and a unique tag name for the fragment.

Caution: If your app supports versions of Android lower than 3.0, be sure that you call `getSupportFragmentManager()` to acquire an instance of `FragmentManager`. Also make sure that your activity that displays the time picker extends `FragmentActivity` instead of the standard `Activity` class.

Creating a Date Picker

Creating a [DatePickerDialog](#) is just like creating a [TimePickerDialog](#). The only difference is the dialog you create for the fragment.

To display a [DatePickerDialog](#) using [DialogFragment](#), you need to define a fragment class that extends [DialogFragment](#) and return a [DatePickerDialog](#) from the fragment's [onCreateDialog\(\)](#) method.

Extending DialogFragment for a date picker

To define a [DialogFragment](#) for a [DatePickerDialog](#), you must:

- Define the [onCreateDialog\(\)](#) method to return an instance of [DatePickerDialog](#)
- Implement the [DatePickerDialog.OnDateSetListener](#) interface to receive a callback when the user sets the date.

Here's an example:

```
public static class DatePickerFragment extends DialogFragment
        implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user
    }
}
```

See the [DatePickerDialog](#) class for information about the constructor arguments.

Now all you need is an event that adds an instance of this fragment to your activity.

Showing the date picker

Once you've defined a [DialogFragment](#) like the one shown above, you can display the date picker by creating an instance of the [DialogFragment](#) and calling [show\(\)](#).

For example, here's a button that, when clicked, calls a method to show the dialog:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_date"
    android:onClick="showDatePickerDialog" />
```

When the user clicks this button, the system calls the following method:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

This method calls [show\(\)](#) on a new instance of the [DialogFragment](#) defined above. The [show\(\)](#) method requires an instance of [FragmentManager](#) and a unique tag name for the fragment.

Search Overview

Topics

- › [Creating a Search Interface](#)
- › [Adding Recent Query Suggestions](#)
- › [Adding Custom Suggestions](#)

Reference

- › [Searchable Configuration](#)

Related samples

- › [Searchable Dictionary](#)

Search is a core user feature on Android. Users should be able to search any data that is available to them, whether the content is located on the device or the Internet. To help create a consistent search experience for users, Android provides a search framework that helps you implement search for your application.

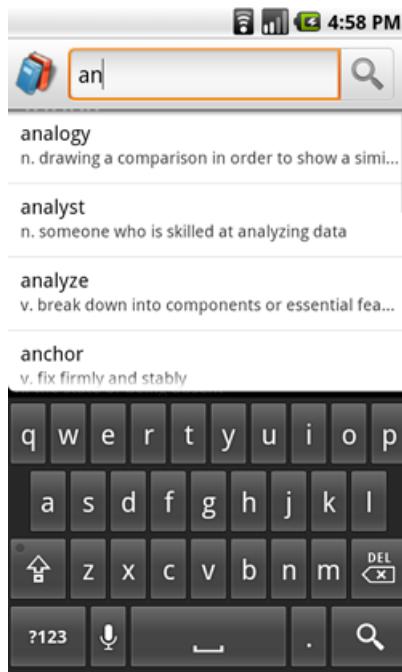


Figure 1. Screenshot of a search dialog with custom search suggestions.

The search framework offers two modes of search input: a search dialog at the top of the screen or a search widget ([SearchView](#)) that you can embed in your activity layout. In either case, the Android system will assist your search implementation by delivering search queries to a specific activity that performs searches. You can also enable either the search dialog or widget to provide search suggestions as the user types. Figure 1 shows an example of the search dialog with optional search suggestions.

Once you've set up either the search dialog or the search widget, you can:

- Enable voice search
- Provide search suggestions based on recent user queries

- Provide custom search suggestions that match actual results in your application data
- Offer your application's search suggestions in the system-wide Quick Search Box

Note: The search framework does *not* provide APIs to search your data. To perform a search, you need to use APIs appropriate for your data. For example, if your data is stored in an SQLite database, you should use the [android.database.sqlite](#) APIs to perform searches.

Also, there is no guarantee that a device provides a dedicated SEARCH button that invokes the search interface in your application. When using the search dialog or a custom interface, you must provide a search button in your UI that activates the search interface. For more information, see [Invoking the search dialog](#).

The following documents show you how to use Android's framework to implement search:

[Creating a Search Interface](#)

How to set up your application to use the search dialog or search widget.

[Adding Recent Query Suggestions](#)

How to provide suggestions based on queries previously used.

[Adding Custom Suggestions](#)

How to provide suggestions based on custom data from your application and also offer them in the system-wide Quick Search Box.

[Searchable Configuration](#)

A reference document for the searchable configuration file (though the other documents also discuss the configuration file in terms of specific behaviors).

Protecting User Privacy

When you implement search in your application, take steps to protect the user's privacy. Many users consider their activities on the phone—including searches—to be private information. To protect each user's privacy, you should abide by the following principles:

- **Don't send personal information to servers, but if you must, do not log it.**

Personal information is any information that can personally identify your users, such as their names, email addresses, billing information, or other data that can be reasonably linked to such information. If your application implements search with the assistance of a server, avoid sending personal information along with the search queries. For example, if you are searching for businesses near a zip code, you don't need to send the user ID as well; send only the zip code to the server. If you must send the personal information, you should not log it. If you must log it, protect that data very carefully and erase it as soon as possible.

- **Provide users with a way to clear their search history.**

The search framework helps your application provide context-specific suggestions while the user types. Sometimes these suggestions are based on previous searches or other actions taken by the user in an earlier session. A user might not wish for previous searches to be revealed to other device users, for instance, if the user shares the device with a friend. If your application provides suggestions that can reveal previous search activities, you should implement the ability for the user to clear the search history. If you are using [SearchRecentSuggestions](#), you can simply call the [clearHistory\(\)](#) method. If you are implementing custom suggestions, you'll need to provide a similar "clear history" method in your content provider that the user can execute.

Creating a Search Interface

In this document

- › [The Basics](#)
- › [Creating a Searchable Configuration](#)
- › [Creating a Searchable Activity](#)
 - › [Declaring a searchable activity](#)
 - › [Performing a search](#)
- › [Using the Search Dialog](#)
 - › [Invoking the search dialog](#)
 - › [The impact of the search dialog on your activity lifecycle](#)
 - › [Passing search context data](#)
- › [Using the Search Widget](#)
 - › [Configuring the search widget](#)
 - › [Other search widget features](#)
 - › [Using both the widget and the dialog](#)
- › [Adding Voice Search](#)
- › [Adding Search Suggestions](#)

Key classes

- › [SearchManager](#)
- › [SearchView](#)

Related samples

- › [Searchable Dictionary](#)
- › [SearchView in the Action Bar](#)
- › [SearchView filter mode](#)

Downloads

- › [Action Bar Icon Pack](#)

When you're ready to add search functionality to your application, Android helps you implement the user interface with either a search dialog that appears at the top of the activity window or a search widget that you can insert in your layout. Both the search dialog and the widget can deliver the user's search query to a specific activity in your application. This way, the user can initiate a search from any activity where the search dialog or widget is available, and the system starts the appropriate activity to perform the search and present results.

Other features available for the search dialog and widget include:

- Voice search
- Search suggestions based on recent queries
- Search suggestions that match actual results in your application data

This guide shows you how to set up your application to provide a search interface that's assisted by the Android system to deliver search queries, using either the search dialog or the search widget.

The Basics



Figure 1. Screenshot of an application's search dialog.

Before you begin, you should decide whether you'll implement your search interface using the search dialog or the search widget. Both provide the same search features, but in slightly different ways:

- The **search dialog** is a UI component that's controlled by the Android system. When activated by the user, the search dialog appears at the top of the activity, as shown in figure 1.
The Android system controls all events in the search dialog. When the user submits a query, the system delivers the query to the activity that you specify to handle searches. The dialog can also provide search suggestions while the user types.
- The **search widget** is an instance of [SearchView](#) that you can place anywhere in your layout. By default, the search widget behaves like a standard [EditText](#) widget and doesn't do anything, but you can configure it so that the Android system handles all input events, delivers queries to the appropriate activity, and provides search suggestions (just like the search dialog).

Note: If you want, you can handle all user input into the search widget yourself, using various callback methods and listeners. This document, however, focuses on how to integrate the search widget with the system for an assisted search implementation. If you want to handle all user input yourself, read the reference documentation for [SearchView](#) and its nested interfaces.

When the user executes a search from the search dialog or a search widget, the system creates an [Intent](#) and stores the user query in it. The system then starts the activity that you've declared to handle searches (the "searchable activity") and delivers it the intent. To set up your application for this kind of assisted search, you need the following:

- A searchable configuration

An XML file that configures some settings for the search dialog or widget. It includes settings for features such as voice search, search suggestion, and hint text for the search box.

- A searchable activity

The [Activity](#) that receives the search query, searches your data, and displays the search results.

- A search interface, provided by either:

- The search dialog

By default, the search dialog is hidden, but appears at the top of the screen when you call [onSearchRequested\(\)](#) (when the user presses your Search button).

- Or, a [SearchView](#) widget

Using the search widget allows you to put the search box anywhere in your activity. Instead of putting it in your activity layout, you

should usually use [SearchView](#) as an action view in the app bar.

The rest of this document shows you how to create the searchable configuration, searchable activity, and implement a search interface with either the search dialog or search widget.

Creating a Searchable Configuration

The first thing you need is an XML file called the searchable configuration. It configures certain UI aspects of the search dialog or widget and defines how features such as suggestions and voice search behave. This file is traditionally named `searchable.xml` and must be saved in the `res/xml/` project directory.

Note: The system uses this file to instantiate a [SearchableInfo](#) object, but you cannot create this object yourself at runtime—you must declare the searchable configuration in XML.

The searchable configuration file must include the `<searchable>` element as the root node and specify one or more attributes. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint" >
</searchable>
```

The `android:label` attribute is the only required attribute. It points to a string resource, which should be the application name. This label isn't actually visible to the user until you enable search suggestions for Quick Search Box. At that point, this label is visible in the list of Searchable items in the system Settings.

Though it's not required, we recommend that you always include the `android:hint` attribute, which provides a hint string in the search box before users enters a query. The hint is important because it provides important clues to users about what they can search.

Tip: For consistency among other Android applications, you should format the string for `android:hint` as "Search <content-or-product>". For example, "Search songs and artists" or "Search YouTube".

The `<searchable>` element accepts several other attributes. However, you don't need most attributes until you add features such as [search suggestions](#) and [voice search](#). For detailed information about the searchable configuration file, see the [Searchable Configuration](#) reference document.

Creating a Searchable Activity

A searchable activity is the [Activity](#) in your application that performs searches based on a query string and presents the search results.

When the user executes a search in the search dialog or widget, the system starts your searchable activity and delivers it the search query in an [Intent](#) with the `ACTION_SEARCH` action. Your searchable activity retrieves the query from the intent's `QUERY` extra, then searches your data and presents the results.

Because you may include the search dialog or widget in any other activity in your application, the system must know which activity is your searchable activity, so it can properly deliver the search query. So, you must first declare your searchable activity in the Android manifest file.

Declaring a searchable activity

If you don't have one already, create an [Activity](#) that will perform searches and present results. You don't need to implement the search functionality yet—just create an activity that you can declare in the manifest. Inside the manifest's `<activity>` element:

1. Declare the activity to accept the `ACTION_SEARCH` intent, in an `<intent-filter>` element.
2. Specify the searchable configuration to use, in a `<meta-data>` element.

For example:

```
<application ... >
    <activity android:name=".SearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
            android:resource="@xml/searchable"/>
    </activity>
    ...
</application>
```

The `<meta-data>` element must include the `android:name` attribute with a value of `"android.app.searchable"` and the `android:resource` attribute with a reference to the searchable configuration file (in this example, it refers to the `res/xml/searchable.xml` file).

Note: The `<intent-filter>` does not need a `<category>` with the `DEFAULT` value (which you usually see in `<activity>` elements), because the system delivers the `ACTION_SEARCH` intent explicitly to your searchable activity, using its component name.

Performing a search

Once you have declared your searchable activity in the manifest, performing a search in your searchable activity involves three steps:

1. Receiving the query
2. Searching your data
3. Presenting the results

Traditionally, your search results should be presented in a `ListView`, so you might want your searchable activity to extend `ListActivity`. It includes a default layout with a single `ListView` and provides several convenience methods for working with the `ListView`.

Receiving the query

When a user executes a search from the search dialog or widget, the system starts your searchable activity and sends it a `ACTION_SEARCH` intent. This intent carries the search query in the `QUERY` string extra. You must check for this intent when the activity starts and extract the string. For example, here's how you can get the search query when your searchable activity starts:

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

    // Get the intent, verify the action and get the query
    Intent intent = getIntent();
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        doMySearch(query);
    }
}
```

The `QUERY` string is always included with the `ACTION_SEARCH` intent. In this example, the query is retrieved and passed to a local `doMySearch()` method where the actual search operation is done.

Searching your data

The process of storing and searching your data is unique to your application. You can store and search your data in many ways, but this guide does not show you how to store your data and search it. Storing and searching your data is something you should carefully consider in terms of your needs and your data format. However, here are some tips you might be able to apply:

- If your data is stored in a SQLite database on the device, performing a full-text search (using FTS3, rather than a `LIKE` query) can provide a more robust search across text data and can produce results significantly faster. See sqlite.org for information about FTS3 and the `SQLiteDatabase` class for information about SQLite on Android. Also look at the [Searchable Dictionary](#) sample application to see a complete SQLite implementation that performs searches with FTS3.

- If your data is stored online, then the perceived search performance might be inhibited by the user's data connection. You might want to display a spinning progress wheel until your search returns. See [android.net](#) for a reference of network APIs and [Creating a Progress Dialog](#) for information about how to display a progress wheel.

Regardless of where your data lives and how you search it, we recommend that you return search results to your searchable activity with an [Adapter](#). This way, you can easily present all the search results in a [ListView](#). If your data comes from a SQLite database query, you can apply your results to a [ListView](#) using a [CursorAdapter](#). If your data comes in some other type of format, then you can create an extension of [BaseAdapter](#).

Presenting the results

As discussed above, the recommended UI for your search results is a [ListView](#), so you might want your searchable activity to extend [ListActivity](#). You can then call [setListAdapter\(\)](#), passing it an [Adapter](#) that is bound to your data. This injects all the search results into the activity [ListView](#).

For more help presenting your results in a list, see the [ListActivity](#) documentation.

Also see the [Searchable Dictionary](#) sample for an a complete demonstration of how to search an SQLite database and use an [Adapter](#) to provide results in a [ListView](#).

About Adapters

An [Adapter](#) binds each item from a set of data into a [View](#) object. When the [Adapter](#) is applied to a [ListView](#), each piece of data is inserted as an individual view into the list. [Adapter](#) is just an interface, so implementations such as [CursorAdapter](#) (for binding data from a [Cursor](#)) are needed. If none of the existing implementations work for your data, then you can implement your own from [BaseAdapter](#). Install the SDK Samples package for API Level 4 to see the original version of the Searchable Dictionary, which creates a custom adapter to read data from a file.

Using the Search Dialog

The search dialog provides a floating search box at the top of the screen, with the application icon on the left. The search dialog can provide search suggestions as the user types and, when the user executes a search, the system sends the search query to a searchable activity that performs the search. However, if you are developing your application for devices running Android 3.0, you should consider using the search widget instead (see [Using the Search Widget](#) section).

The search dialog is always hidden by default, until the user activates it. Your application can activate the search dialog by calling [onSearchRequested\(\)](#). However, this method doesn't work until you enable the search dialog for the activity.

To enable the search dialog, you must indicate to the system which searchable activity should receive search queries from the search dialog, in order to perform searches. For example, in the previous section about [Creating a Searchable Activity](#), a searchable activity named [SearchableActivity](#) was created. If you want a separate activity, named [OtherActivity](#), to show the search dialog and deliver searches to [SearchableActivity](#), you must declare in the manifest that [SearchableActivity](#) is the searchable activity to use for the search dialog in [OtherActivity](#).

To declare the searchable activity for an activity's search dialog, add a [`<meta-data>`](#) element inside the respective activity's [`<activity>`](#) element. The [`<meta-data>`](#) element must include the [android:value](#) attribute that specifies the searchable activity's class name and the [android:name](#) attribute with a value of `"android.app.default_searchable"`.

For example, here is the declaration for both a searchable activity, [SearchableActivity](#), and another activity, [OtherActivity](#), which uses [SearchableActivity](#) to perform searches executed from its search dialog:

```

<application ... >
    <!-- this is the searchable activity; it performs searches -->
    <activity android:name=".SearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
            android:resource="@xml/searchable"/>
    </activity>

    <!-- this activity enables the search dialog to initiate searches
        in the SearchableActivity -->
    <activity android:name=".OtherActivity" ... >
        <!-- enable the search dialog to send searches to SearchableActivity -->
        <meta-data android:name="android.app.default_searchable"
            android:value=".SearchableActivity" />
    </activity>
    ...
</application>

```

Because the `OtherActivity` now includes a `<meta-data>` element to declare which searchable activity to use for searches, the activity has enabled the search dialog. While the user is in this activity, the `onSearchRequested()` method activates the search dialog. When the user executes the search, the system starts `SearchableActivity` and delivers it the `ACTION_SEARCH` intent.

Note: The searchable activity itself provides the search dialog by default, so you don't need to add this declaration to `SearchableActivity`.

If you want every activity in your application to provide the search dialog, insert the above `<meta-data>` element as a child of the `<application>` element, instead of each `<activity>`. This way, every activity inherits the value, provides the search dialog, and delivers searches to the same searchable activity. (If you have multiple searchable activities, you can override the default searchable activity by placing a different `<meta-data>` declaration inside individual activities.)

With the search dialog now enabled for your activities, your application is ready to perform searches.

Invoking the search dialog

Although some devices provide a dedicated Search button, the behavior of the button may vary between devices and many devices do not provide a Search button at all. So when using the search dialog, you **must provide a search button in your UI** that activates the search dialog by calling `onSearchRequested()`.

For instance, you should add a Search button in your `Options Menu` or UI layout that calls `onSearchRequested()`. For consistency with the Android system and other apps, you should label your button with the Android Search icon that's available from the `Action Bar Icon Pack`.

Note: If your app uses an `app bar`, then you should not use the search dialog for your search interface. Instead, use the `search widget` as a collapsible view in the app bar.

You can also enable "type-to-search" functionality, which activates the search dialog when the user starts typing on the keyboard—the keystrokes are inserted into the search dialog. You can enable type-to-search in your activity by calling `setDefaultKeyMode(DEFAULT_KEYS_SEARCH_LOCAL)` during your activity's `onCreate()` method.

The impact of the search dialog on your activity lifecycle

The search dialog is a `Dialog` that floats at the top of the screen. It does not cause any change in the activity stack, so when the search dialog appears, no lifecycle methods (such as `onPause()`) are called. Your activity just loses input focus, as input focus is given to the search dialog.

If you want to be notified when the search dialog is activated, override the `onSearchRequested()` method. When the system calls this method, it is an indication that your activity has lost input focus to the search dialog, so you can do any work appropriate for the event (such as pause a game). Unless you are `passing search context data` (discussed below), you should end the method by calling the super class implementation. For example:

```

@Override
public boolean onSearchRequested() {
    pauseSomeStuff();
    return super.onSearchRequested();
}

```

If the user cancels search by pressing the *Back* button, the search dialog closes and the activity regains input focus. You can register to be notified when the search dialog is closed with `setOnDismissListener()` and/or `setOnCancelListener()`. You should need to register only the `OnDismissListener`, because it is called every time the search dialog closes. The `OnCancelListener` only pertains to events in which the user explicitly exited the search dialog, so it is not called when a search is executed (in which case, the search dialog naturally disappears).

If the current activity is not the searchable activity, then the normal activity lifecycle events are triggered once the user executes a search (the current activity receives `onPause()` and so forth, as described in the [Activities](#) document). If, however, the current activity is the searchable activity, then one of two things happens:

- By default, the searchable activity receives the `ACTION_SEARCH` intent with a call to `onCreate()` and a new instance of the activity is brought to the top of the activity stack. There are now two instances of your searchable activity in the activity stack (so pressing the *Back* button goes back to the previous instance of the searchable activity, rather than exiting the searchable activity).
- If you set `android:launchMode` to `"singleTop"`, then the searchable activity receives the `ACTION_SEARCH` intent with a call to `onNewIntent(Intent)`, passing the new `ACTION_SEARCH` intent here. For example, here's how you might handle this case, in which the searchable activity's launch mode is `"singleTop"`:

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.search);
    handleIntent(getIntent());
}

@Override
protected void onNewIntent(Intent intent) {
    setIntent(intent);
    handleIntent(intent);
}

private void handleIntent(Intent intent) {
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        doMySearch(query);
    }
}

```

Compared to the example code in the section about [Performing a Search](#), all the code to handle the search intent is now in the `handleIntent()` method, so that both `onCreate()` and `onNewIntent()` can execute it.

When the system calls `onNewIntent(Intent)`, the activity has not been restarted, so the `getIntent()` method returns the same intent that was received with `onCreate()`. This is why you should call `setIntent(Intent)` inside `onNewIntent(Intent)` (so that the intent saved by the activity is updated in case you call `getIntent()` in the future).

The second scenario using `"singleTop"` launch mode is usually ideal, because chances are good that once a search is done, the user will perform additional searches and it's a bad experience if your application creates multiple instances of the searchable activity. So, we recommend that you set your searchable activity to `"singleTop"` launch mode in the application manifest. For example:

```

<activity android:name=".SearchableActivity"
    android:launchMode="singleTop" >
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    <meta-data android:name="android.app.searchable"
        android:resource="@xml/searchable"/>
</activity>

```

Passing search context data

In some cases, you can make necessary refinements to the search query inside the searchable activity, for every search made. However, if you want to refine your search criteria based on the activity from which the user is performing a search, you can provide additional data in the intent that the system sends to your searchable activity. You can pass the additional data in the [APP_DATA Bundle](#), which is included in the [ACTION_SEARCH](#) intent.

To pass this kind of data to your searchable activity, override the [onSearchRequested\(\)](#) method for the activity from which the user can perform a search, create a [Bundle](#) with the additional data, and call [startSearch\(\)](#) to activate the search dialog. For example:

```
@Override  
public boolean onSearchRequested() {  
    Bundle appData = new Bundle();  
    appData.putBoolean(SearchableActivity.JARGON, true);  
    startSearch(null, false, appData, false);  
    return true;  
}
```

Returning "true" indicates that you have successfully handled this callback event and called [startSearch\(\)](#) to activate the search dialog. Once the user submits a query, it's delivered to your searchable activity along with the data you've added. You can extract the extra data from the [APP_DATA Bundle](#) to refine the search. For example:

```
Bundle appData = getIntent().getBundleExtra(SearchManager.APP_DATA);  
if (appData != null) {  
    boolean jargon = appData.getBoolean(SearchableActivity.JARGON);  
}
```

Caution: Never call the [startSearch\(\)](#) method from outside the [onSearchRequested\(\)](#) callback method. To activate the search dialog in your activity, always call [onSearchRequested\(\)](#). Otherwise, [onSearchRequested\(\)](#) is not called and customizations (such as the addition of [appData](#) in the above example) are missed.

Using the Search Widget



Figure 2. The [SearchView](#) widget as an "action view" in the Action Bar.

The [SearchView](#) widget is available in Android 3.0 and higher. If you're developing your application for Android 3.0 and have decided to use the search widget, we recommend that you insert the search widget as an action view in the app bar, instead of using the search dialog (and instead of placing the search widget in your activity layout). For example, figure 2 shows the search widget in the app bar.

The search widget provides the same functionality as the search dialog. It starts the appropriate activity when the user executes a search, and it can provide search suggestions and perform voice search. If it's not an option for you to put the search widget in the Action Bar, you can instead put the search widget somewhere in your activity layout.

Note: When you use the search widget as an action view, you still might need to support using the search dialog, for cases in which the search widget does not fit in the Action Bar. See the following section about [Using both the widget and the dialog](#).

Configuring the search widget

After you've created a [searchable configuration](#) and a [searchable activity](#), as discussed above, you need to enable assisted search for each [SearchView](#). You can do so by calling [setSearchableInfo\(\)](#) and passing it the [SearchableInfo](#) object that represents your searchable configuration.

You can get a reference to the [SearchableInfo](#) by calling [getSearchableInfo\(\)](#) on [SearchManager](#).

For example, if you're using a [SearchView](#) as an action view in the [app bar](#), you should enable the widget during the [onCreateOptionsMenu\(\)](#) callback:

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the options menu from XML
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager = (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView = (SearchView) menu.findItem(R.id.menu_search).getActionView();
    // Assumes current activity is the searchable activity
    searchView.setSearchableInfo(searchManager.getSearchableInfo(getApplicationContext()));
    searchView.setIconifiedByDefault(false); // Do not iconify the widget; expand it by default

    return true;
}

```

That's all you need. The search widget is now configured and the system will deliver search queries to your searchable activity. You can also enable [search suggestions](#) for the search widget.

Note: If you want to handle all user input yourself, you can do so with some callback methods and event listeners. For more information, see the reference documentation for [SearchView](#) and its nested interfaces for the appropriate event listeners.

For more information about action views in the Action Bar, see [Action Views and Action Providers](#).

Other search widget features

The [SearchView](#) widget allows for a few additional features you might want:

A submit button

By default, there's no button to submit a search query, so the user must press the "Return" key on the keyboard to initiate a search.

You can add a "submit" button by calling [setSubmitButtonEnabled\(true\)](#).

Query refinement for search suggestions

When you've enabled search suggestions, you usually expect users to simply select a suggestion, but they might also want to refine the suggested search query. You can add a button alongside each suggestion that inserts the suggestion in the search box for refinement by the user, by calling [setQueryRefinementEnabled\(true\)](#).

The ability to toggle the search box visibility

By default, the search widget is "iconified," meaning that it is represented only by a search icon (a magnifying glass), and expands to show the search box when the user touches it. As shown above, you can show the search box by default, by calling [setIconifiedByDefault\(false\)](#). You can also toggle the search widget appearance by calling [setIconified\(\)](#).

There are several other APIs in the [SearchView](#) class that allow you to customize the search widget. However, most of them are used only when you handle all user input yourself, instead of using the Android system to deliver search queries and display search suggestions.

Using both the widget and the dialog

If you insert the search widget in the Action Bar as an [action view](#), and you enable it to appear in the Action Bar "if there is room" (by setting `android:showAsAction="ifRoom"`), then there is a chance that the search widget will not appear as an action view, but the menu item will appear in the overflow menu. For example, when your application runs on a smaller screen, there might not be enough room in the Action Bar to display the search widget along with other action items or navigation elements, so the menu item will instead appear in the overflow menu. When placed in the overflow menu, the item works like an ordinary menu item and does not display the action view (the search widget).

To handle this situation, the menu item to which you've attached the search widget should activate the search dialog when the user selects it from the overflow menu. In order for it to do so, you must implement [onOptionsItemSelected\(\)](#) to handle the "Search" menu item and open the search dialog by calling [onSearchRequested\(\)](#).

For more information about how items in the Action Bar work and how to handle this situation, see the [Action Bar](#) developer guide.

Also see the [Searchable Dictionary](#) for an example implementation using both the dialog and the widget.

Adding Voice Search

You can add voice search functionality to your search dialog or widget by adding the `android:voiceSearchMode` attribute to your searchable configuration. This adds a voice search button that launches a voice prompt. When the user has finished speaking, the transcribed search query is sent to your searchable activity.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer" >
</searchable>
```

The value `showVoiceSearchButton` is required to enable voice search, while the second value, `launchRecognizer`, specifies that the voice search button should launch a recognizer that returns the transcribed text to the searchable activity.

You can provide additional attributes to specify the voice search behavior, such as the language to be expected and the maximum number of results to return. See the [Searchable Configuration](#) reference for more information about the available attributes.

Note: Carefully consider whether voice search is appropriate for your application. All searches performed with the voice search button are immediately sent to your searchable activity without a chance for the user to review the transcribed query. Sufficiently test the voice recognition and ensure that it understands the types of queries that the user might submit inside your application.

Adding Search Suggestions



Figure 3. Screenshot of a search dialog

with custom search suggestions.

Both the search dialog and the search widget can provide search suggestions as the user types, with assistance from the Android system. The system manages the list of suggestions and handles the event when the user selects a suggestion.

You can provide two kinds of search suggestions:

Recent query search suggestions

These suggestions are simply words that the user previously used as search queries in your application.

See [Adding Recent Query Suggestions](#).

Custom search suggestions

These are search suggestions that you provide from your own data source, to help users immediately select the correct spelling or item they are searching for. Figure 3 shows an example of custom suggestions for a dictionary application—the user can select a suggestion to instantly go to the definition.

See [Adding Custom Suggestions](#)

Adding Recent Query Suggestions

In this document

- › [The Basics](#)
- › [Creating a Content Provider](#)
- › [Modifying the Searchable Configuration](#)
- › [Saving Queries](#)
- › [Clearing the Suggestion Data](#)

Key classes

- › [SearchRecentSuggestions](#)
- › [SearchRecentSuggestionsProvider](#)

See also

- › [Searchable Configuration](#)

When using the Android search dialog or search widget, you can provide search suggestions based on recent search queries. For example, if a user previously searched for "puppies," then that query appears as a suggestion once he or she begins typing the same query. Figure 1 shows an example of a search dialog with recent query suggestions.

Before you begin, you need to implement the search dialog or a search widget for basic searches in your application. If you haven't, see [Creating a Search Interface](#).

The Basics

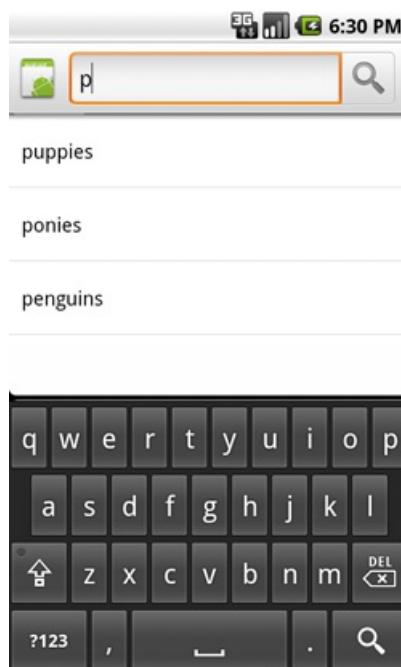


Figure 1. Screenshot of a search dialog with recent query suggestions.

Recent query suggestions are simply saved searches. When the user selects one of the suggestions, your searchable activity receives a `ACTION_SEARCH` intent with the suggestion as the search query, which your searchable activity already handles (as described in [Creating a Search Interface](#)).

To provide recent queries suggestions, you need to:

- Implement a searchable activity, as described in [Creating a Search Interface](#).
- Create a content provider that extends `SearchRecentSuggestionsProvider` and declare it in your application manifest.
- Modify the searchable configuration with information about the content provider that provides search suggestions.
- Save queries to your content provider each time a search is executed.

Just as the Android system displays the search dialog, it also displays the search suggestions below the dialog or search widget. All you need to do is provide a source from which the system can retrieve suggestions.

When the system identifies that your activity is searchable and provides search suggestions, the following procedure takes place as soon as the user begins typing a query:

1. The system takes the search query text (whatever has been typed so far) and performs a query to the content provider that contains your suggestions.
2. Your content provider returns a `Cursor` that points to all suggestions that match the search query text.
3. The system displays the list of suggestions provided by the Cursor.

Once the recent query suggestions are displayed, the following might happen:

- If the user types another key, or changes the query in any way, the aforementioned steps are repeated and the suggestion list is updated.
- If the user executes the search, the suggestions are ignored and the search is delivered to your searchable activity using the normal `ACTION_SEARCH` intent.
- If the user selects a suggestion, an `ACTION_SEARCH` intent is delivered to your searchable activity using the suggested text as the query.

The `SearchRecentSuggestionsProvider` class that you extend for your content provider automatically does the work described above, so there's actually very little code to write.

Creating a Content Provider

The content provider that you need for recent query suggestions must be an implementation of `SearchRecentSuggestionsProvider`. This class does practically everything for you. All you have to do is write a class constructor that executes one line of code.

For example, here's a complete implementation of a content provider for recent query suggestions:

```
public class MySuggestionProvider extends SearchRecentSuggestionsProvider {  
    public final static String AUTHORITY = "com.example.MySuggestionProvider";  
    public final static int MODE = DATABASE_MODE_QUERIES;  
  
    public MySuggestionProvider() {  
        setupSuggestions(AUTHORITY, MODE);  
    }  
}
```

The call to `setupSuggestions()` passes the name of the search authority and a database mode. The search authority can be any unique string, but the best practice is to use a fully qualified name for your content provider (package name followed by the provider's class name; for example, "com.example.MySuggestionProvider"). The database mode must include `DATABASE_MODE_QUERIES` and can optionally include `DATABASE_MODE_2LINES`, which adds another column to the suggestions table that allows you to provide a second line of text with each suggestion. For example, if you want to provide two lines in each suggestion:

```
public final static int MODE = DATABASE_MODE_QUERIES | DATABASE_MODE_2LINES;
```

Now declare the content provider in your application manifest with the same authority string used in your `SearchRecentSuggestionsProvider` class (and in the searchable configuration). For example:

```
<application>
    <provider android:name=".MySuggestionProvider"
              android:authorities="com.example.MySuggestionProvider" />
    ...
</application>
```

Modifying the Searchable Configuration

To configure the system to use your suggestions provider, you need to add the `android:searchSuggestAuthority` and `android:searchSuggestSelection` attributes to the `<searchable>` element in your searchable configuration file. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MySuggestionProvider"
    android:searchSuggestSelection=" ?" >
</searchable>
```

The value for `android:searchSuggestAuthority` should be a fully qualified name for your content provider that exactly matches the authority used in the content provider (the `AUTHORITY` string in the above example).

The value for `android:searchSuggestSelection` must be a single question mark, preceded by a space (" ?"), which is simply a placeholder for the SQLite selection argument (which is automatically replaced by the query text entered by the user).

Saving Queries

To populate your collection of recent queries, add each query received by your searchable activity to your `SearchRecentSuggestionsProvider`. To do this, create an instance of `SearchRecentSuggestions` and call `saveRecentQuery()` each time your searchable activity receives a query. For example, here's how you can save the query during your activity's `onCreate()` method:

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

    Intent intent = getIntent();

    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,
            MySuggestionProvider.AUTHORITY, MySuggestionProvider.MODE);
        suggestions.saveRecentQuery(query, null);
    }
}
```

The `SearchRecentSuggestionsProvider` constructor requires the same authority and database mode declared by your content provider.

The `saveRecentQuery()` method takes the search query string as the first parameter and, optionally, a second string to include as the second line of the suggestion (or null). The second parameter is only used if you've enabled two-line mode for the search suggestions with `DATABASE_MODE_2LINES`. If you have enabled two-line mode, then the query text is also matched against this second line when the system looks for matching suggestions.

Clearing the Suggestion Data

To protect the user's privacy, you should always provide a way for the user to clear the recent query suggestions. To clear the query history, call `clearHistory()`. For example:

```
SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,  
    HelloSuggestionProvider.AUTHORITY, HelloSuggestionProvider.MODE);  
suggestions.clearHistory();
```

Execute this from your choice of a "Clear Search History" menu item, preference item, or button. You should also provide a confirmation dialog to verify that the user wants to delete their search history.

Adding Custom Suggestions

In this document

- › [The Basics](#)
- › [Modifying the Searchable Configuration](#)
- › [Creating a Content Provider](#)
 - › [Handling a suggestion query](#)
 - › [Building a suggestion table](#)
- › [Declaring an Intent for Suggestions](#)
 - › [Declaring the intent action](#)
 - › [Declaring the intent data](#)
- › [Handling the Intent](#)
- › [Rewriting the Query Text](#)
- › [Exposing Search Suggestions to Quick Search Box](#)

Key classes

- › [SearchManager](#)
- › [SearchRecentSuggestionsProvider](#)
- › [ContentProvider](#)

Related samples

- › [Searchable Dictionary](#)

See also

- › [Searchable Configuration](#)
- › [Content Providers](#)

When using the Android search dialog or search widget, you can provide custom search suggestions that are created from data in your application. For example, if your application is a word dictionary, you can suggest words from the dictionary that match the text entered so far. These are the most valuable suggestions, because you can effectively predict what the user wants and provide instant access to it. Figure 1 shows an example of a search dialog with custom suggestions.

Once you provide custom suggestions, you can also make them available to the system-wide Quick Search Box, providing access to your content from outside your application.

Before you begin with this guide to add custom suggestions, you need to have implemented the Android search dialog or a search widget for searches in your application. If you haven't, see [Creating a Search Interface](#).

The Basics

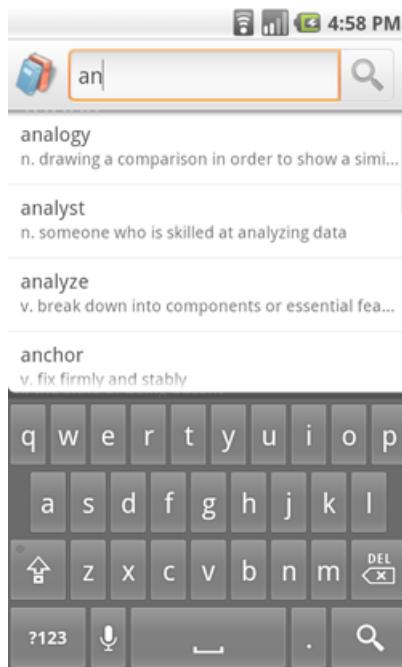


Figure 1. Screenshot of a search dialog with custom search suggestions.

When the user selects a custom suggestion, the Android system sends an [Intent](#) to your searchable activity. Whereas a normal search query sends an intent with the `ACTION_SEARCH` action, you can instead define your custom suggestions to use `ACTION_VIEW` (or any other intent action), and also include data that's relevant to the selected suggestion. Continuing the dictionary example, when the user selects a suggestion, your application can immediately open the definition for that word, instead of searching the dictionary for matches.

To provide custom suggestions, do the following:

- Implement a basic searchable activity, as described in [Creating a Search Interface](#).
- Modify the searchable configuration with information about the content provider that provides custom suggestions.
- Build a table (such as in an [SQLiteDatabase](#)) for your suggestions and format the table with required columns.
- Create a [Content Provider](#) that has access to your suggestions table and declare the provider in your manifest.
- Declare the type of [Intent](#) to be sent when the user selects a suggestion (including a custom action and custom data).

Just as the Android system displays the search dialog, it also displays your search suggestions. All you need is a content provider from which the system can retrieve your suggestions. If you're not familiar with creating content providers, read the [Content Providers](#) developer guide before you continue.

When the system identifies that your activity is searchable and provides search suggestions, the following procedure takes place when the user types a query:

1. The system takes the search query text (whatever has been typed so far) and performs a query to your content provider that manages your suggestions.
2. Your content provider returns a [Cursor](#) that points to all suggestions that are relevant to the search query text.
3. The system displays the list of suggestions provided by the Cursor.

Once the custom suggestions are displayed, the following might happen:

- If the user types another key, or changes the query in any way, the above steps are repeated and the suggestion list is updated as appropriate.
- If the user executes the search, the suggestions are ignored and the search is delivered to your searchable activity using the normal `ACTION_SEARCH` intent.
- If the user selects a suggestion, an intent is sent to your searchable activity, carrying a custom action and custom data so that your application can open the suggested content.

Modifying the searchable configuration

To add support for custom suggestions, add the `android:searchSuggestAuthority` attribute to the `<searchable>` element in your searchable configuration file. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider">
</searchable>
```

You might need some additional attributes, depending on the type of intent you attach to each suggestion and how you want to format queries to your content provider. The other optional attributes are discussed in the following sections.

Creating a Content Provider

Creating a content provider for custom suggestions requires previous knowledge about content providers that's covered in the [Content Provider](#) developer guide. For the most part, a content provider for custom suggestions is the same as any other content provider. However, for each suggestion you provide, the respective row in the `Cursor` must include specific columns that the system understands and uses to format the suggestions.

When the user starts typing into the search dialog or search widget, the system queries your content provider for suggestions by calling `query()` each time a letter is typed. In your implementation of `query()`, your content provider must search your suggestion data and return a `Cursor` that points to the rows you have determined to be good suggestions.

Details about creating a content provider for custom suggestions are discussed in the following two sections:

Handling the suggestion query

How the system sends requests to your content provider and how to handle them

Building a suggestion table

How to define the columns that the system expects in the `Cursor` returned with each query

Handling the suggestion query

When the system requests suggestions from your content provider, it calls your content provider's `query()` method. You must implement this method to search your suggestion data and return a `Cursor` pointing to the suggestions you deem relevant.

Here's a summary of the parameters that the system passes to your `query()` method (listed in order):

uri

Always a content `Uri`, formatted as:

```
content://your.authority/optional.suggest.path/SUGGEST_URI_PATH_QUERY
```

The default behavior is for system to pass this URI and append it with the query text. For example:

```
content://your.authority/optional.suggest.path/SUGGEST_URI_PATH_QUERY/puppies
```

The query text on the end is encoded using URI encoding rules, so you might need to decode it before performing a search.

The `optional.suggest.path` portion is only included in the URI if you have set such a path in your searchable configuration file with the `android:searchSuggestPath` attribute. This is only needed if you use the same content provider for multiple searchable activities, in which case, you need to disambiguate the source of the suggestion query.

Note: `SUGGEST_URI_PATH_QUERY` is not the literal string provided in the URI, but a constant that you should use if you need to refer to this path.

projection

Always null

selection

The value provided in the `android:searchSuggestSelection` attribute of your searchable configuration file, or null if you have not declared the `android:searchSuggestSelection` attribute. More about using this to [get the query](#) below.

selectionArgs

Contains the search query as the first (and only) element of the array if you have declared the `android:searchSuggestSelection` attribute in your searchable configuration. If you have not declared `android:searchSuggestSelection`, then this parameter is null. More about using this to [get the query](#) below.

sortOrder

Always null

The system can send you the search query text in two ways. The default manner is for the query text to be included as the last path of the content URI passed in the `uri` parameter. However, if you include a selection value in your searchable configuration's `android:searchSuggestSelection` attribute, then the query text is instead passed as the first element of the `selectionArgs` string array. Both options are summarized next.

Get the query in the Uri

By default, the query is appended as the last segment of the `uri` parameter (a `Uri` object). To retrieve the query text in this case, simply use `getLastPathSegment()`. For example:

```
String query = uri.getLastPathSegment().toLowerCase();
```

This returns the last segment of the `Uri`, which is the query text entered by the user.

Get the query in the selection arguments

Instead of using the URI, you might decide it makes more sense for your `query()` method to receive everything it needs to perform the look-up and you want the `selection` and `selectionArgs` parameters to carry the appropriate values. In such a case, add the `android:searchSuggestSelection` attribute to your searchable configuration with your SQLite selection string. In the selection string, include a question mark ("?") as a placeholder for the actual search query. The system calls `query()` with the selection string as the `selection` parameter and the search query as the first element in the `selectionArgs` array.

For example, here's how you might form the `android:searchSuggestSelection` attribute to create a full-text search statement:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:searchSuggestSelection="word MATCH ?">
</searchable>
```

With this configuration, your `query()` method delivers the `selection` parameter as "word MATCH ?" and the `selectionArgs` parameter as the search query. When you pass these to an SQLite `query()` method, as their respective arguments, they are synthesized together (the question mark is replaced with the query text). If you chose to receive suggestion queries this way and need to add wildcards to the query text, append (and/or prefix) them to the `selectionArgs` parameter, because this value is wrapped in quotes and inserted in place of the question mark.

Another new attribute in the example above is `android:searchSuggestIntentAction`, which defines the intent action sent with each intent when the user selects a suggestion. It is discussed further in the section about [Declaring an Intent for Suggestions](#).

Tip: If you don't want to define a selection clause in the `android:searchSuggestSelection` attribute, but would still like to receive the query text in the `selectionArgs` parameter, simply provide a non-null value for the `android:searchSuggestSelection` attribute. This triggers the query to be passed in `selectionArgs` and you can ignore the `selection` parameter. In this way, you can instead define the actual selection clause at a lower level so that your content provider doesn't have to handle it.

Building a suggestion table

When you return suggestions to the system with a `Cursor`, the system expects specific columns in each row. So, regardless of whether you decide to store your suggestion data in an SQLite database on the device, a database on a web server, or another format on the device or web, you must format the suggestions as rows in a table and present them with a `Cursor`. The system understands several columns, but only two are required:

`_ID`

A unique integer row ID for each suggestion. The system requires this in order to present suggestions in a [ListView](#).

`SUGGEST_COLUMN_TEXT_1`

The string that is presented as a suggestion.

The following columns are all optional (and most are discussed further in the following sections):

`SUGGEST_COLUMN_TEXT_2`

A string. If your Cursor includes this column, then all suggestions are provided in a two-line format. The string in this column is displayed as a second, smaller line of text below the primary suggestion text. It can be null or empty to indicate no secondary text.

`SUGGEST_COLUMN_ICON_1`

A drawable resource, content, or file URI string. If your Cursor includes this column, then all suggestions are provided in an icon-plus-text format with the drawable icon on the left side. This can be null or zero to indicate no icon in this row.

`SUGGEST_COLUMN_ICON_2`

A drawable resource, content, or file URI string. If your Cursor includes this column, then all suggestions are provided in an icon-plus-text format with the icon on the right side. This can be null or zero to indicate no icon in this row.

`SUGGEST_COLUMN_INTENT_ACTION`

An intent action string. If this column exists and contains a value at the given row, the action defined here is used when forming the suggestion's intent. If the element is not provided, the action is taken from the `android:searchSuggestIntentAction` field in your searchable configuration. If your action is the same for all suggestions, it is more efficient to specify the action using `android:searchSuggestIntentAction` and omit this column.

`SUGGEST_COLUMN_INTENT_DATA`

A data URI string. If this column exists and contains a value at the given row, this is the data that is used when forming the suggestion's intent. If the element is not provided, the data is taken from the `android:searchSuggestIntentData` field in your searchable configuration. If neither source is provided, the intent's data field is null. If your data is the same for all suggestions, or can be described using a constant part and a specific ID, it is more efficient to specify it using `android:searchSuggestIntentData` and omit this column.

`SUGGEST_COLUMN_INTENT_DATA_ID`

Creating a Cursor without a table

If your search suggestions are not stored in a table format (such as an SQLite table) using the columns required by the system, then you can search your suggestion data for matches and then format them into the necessary table on each request. To do so, create a `MatrixCursor` using the required column names and then add a row for each suggestion using `addRow(Object[])`. Return the final product from your Content Provider's `query()` method.

A URI path string. If this column exists and contains a value at the given row, then "/" and this value is appended to the data field in the intent. This should only be used if the data field specified by the `android:searchSuggestIntentData` attribute in the searchable configuration has already been set to an appropriate base string.

SUGGEST_COLUMN_INTENT_EXTRA_DATA

Arbitrary data. If this column exists and contains a value at a given row, this is the `extra` data used when forming the suggestion's intent. If not provided, the intent's extra data field is null. This column allows suggestions to provide additional data that is included as an extra in the intent's `EXTRA_DATA_KEY` key.

SUGGEST_COLUMN_QUERY

If this column exists and this element exists at the given row, this is the data that is used when forming the suggestion's query, included as an extra in the intent's `QUERY` key. Required if suggestion's action is `ACTION_SEARCH`, optional otherwise.

SUGGEST_COLUMN_SHORTCUT_ID

Only used when providing suggestions for Quick Search Box. This column indicates whether a search suggestion should be stored as a shortcut and whether it should be validated. Shortcuts are usually formed when the user clicks a suggestion from Quick Search Box. If missing, the result is stored as a shortcut and never refreshed. If set to `SUGGEST_NEVER_MAKE_SHORTCUT`, the result is not stored as a shortcut. Otherwise, the shortcut ID is used to check back for an up to date suggestion using `SUGGEST_URI_PATH_SHORTCUT`.

SUGGEST_COLUMN_SPINNER WHILE_REFRESHING

Only used when providing suggestions for Quick Search Box. This column specifies that a spinner should be shown instead of an icon from `SUGGEST_COLUMN_ICON_2` while the shortcut of this suggestion is being refreshed in Quick Search Box.

Some of these columns are discussed more in the following sections.

Declaring an Intent for Suggestions

When the user selects a suggestion from the list that appears below the search dialog or widget, the system sends a custom `Intent` to your searchable activity. You must define the action and data for the intent.

Declaring the intent action

The most common intent action for a custom suggestion is `ACTION_VIEW`, which is appropriate when you want to open something, like the definition for a word, a person's contact information, or a web page. However, the intent action can be any other action and can even be different for each suggestion.

Depending on whether you want all suggestions to use the same intent action, you can define the action in two ways:

- Use the `android:searchSuggestIntentAction` attribute of your searchable configuration file to define the action for all suggestions.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.Intent.action.VIEW" >
</searchable>
```

- Use the `SUGGEST_COLUMN_INTENT_ACTION` column to define the action for individual suggestions.

Add the `SUGGEST_COLUMN_INTENT_ACTION` column to your suggestions table and, for each suggestion, place in it the action to use (such as `"android.Intent.action.VIEW"`).

You can also combine these two techniques. For instance, you can include the `android:searchSuggestIntentAction` attribute with an action to be used with all suggestions by default, then override this action for some suggestions by declaring a different action in the

`SUGGEST_COLUMN_INTENT_ACTION` column. If you do not include a value in the `SUGGEST_COLUMN_INTENT_ACTION` column, then the intent provided in the `android:searchSuggestIntentAction` attribute is used.

Note: If you do not include the `android:searchSuggestIntentAction` attribute in your searchable configuration, then you *must* include a value in the `SUGGEST_COLUMN_INTENT_ACTION` column for every suggestion, or the intent will fail.

Declaring intent data

When the user selects a suggestion, your searchable activity receives the intent with the action you've defined (as discussed in the previous section), but the intent must also carry data in order for your activity to identify which suggestion was selected. Specifically, the data should be something unique for each suggestion, such as the row ID for the suggestion in your SQLite table. When the intent is received, you can retrieve the attached data with `getData()` or `getDataString()`.

You can define the data included with the intent in two ways:

- Define the data for each suggestion inside the `SUGGEST_COLUMN_INTENT_DATA` column of your suggestions table.

Provide all necessary data information for each intent in the suggestions table by including the `SUGGEST_COLUMN_INTENT_DATA` column and then populating it with unique data for each row. The data from this column is attached to the intent exactly as you define it in this column. You can then retrieve it with with `getData()` or `getDataString()`.

Tip: It's usually easiest to use the table's row ID as the Intent data, because it's always unique. And the easiest way to do that is by using the `SUGGEST_COLUMN_INTENT_DATA` column name as an alias for the row ID column. See the [Searchable Dictionary sample app](#) for an example in which `SQLiteQueryBuilder` creates a projection map of column names to aliases.

- Fragment a data URI into two pieces: the portion common to all suggestions and the portion unique to each suggestion. Place these parts into the `android:searchSuggestIntentData` attribute of the searchable configuration and the `SUGGEST_COLUMN_INTENT_DATA_ID` column of your suggestions table, respectively.

Declare the piece of the URI that is common to all suggestions in the `android:searchSuggestIntentData` attribute of your searchable configuration. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:searchSuggestIntentData="content://com.example/datatable" >
</searchable>
```

Then include the final path for each suggestion (the unique part) in the `SUGGEST_COLUMN_INTENT_DATA_ID` column of your suggestions table. When the user selects a suggestion, the system takes the string from `android:searchSuggestIntentData`, appends a slash ("/") and then adds the respective value from the `SUGGEST_COLUMN_INTENT_DATA_ID` column to form a complete content URI. You can then retrieve the `Uri` with with `getData()`.

Add more data

If you need to express even more information with your intent, you can add another table column, `SUGGEST_COLUMN_INTENT_EXTRA_DATA`, which can store additional information about the suggestion. The data saved in this column is placed in `EXTRA_DATA_KEY` of the intent's extra Bundle.

Handling the Intent

Now that you provide custom search suggestions with custom intents, you need your searchable activity to handle these intents when the user selects a suggestion. This is in addition to handling the `ACTION_SEARCH` intent, which your searchable activity already does. Here's an example of how you can handle the intents during your activity `onCreate()` callback:

```

Intent intent = getIntent();
if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
    // Handle the normal search query case
    String query = intent.getStringExtra(SearchManager.QUERY);
    doSearch(query);
} else if (Intent.ACTION_VIEW.equals(intent.getAction())) {
    // Handle a suggestions click (because the suggestions all use ACTION_VIEW)
    Uri data = intent.getData();
    showResult(data);
}

```

In this example, the intent action is `ACTION_VIEW` and the data carries a complete URI pointing to the suggested item, as synthesized by the `android:searchSuggestIntentData` string and `SUGGEST_COLUMN_INTENT_DATA_ID` column. The URI is then passed to the local `showResult()` method that queries the content provider for the item specified by the URI.

Note: You do *not* need to add an intent filter to your Android manifest file for the intent action you defined with the `android:searchSuggestIntentAction` attribute or `SUGGEST_COLUMN_INTENT_ACTION` column. The system opens your searchable activity by name to deliver the suggestion's intent, so the activity does not need to declare the accepted action.

Rewriting the query text

If the user navigates through the suggestions list using the directional controls (such as with a trackball or d-pad), the query text does not update, by default. However, you can temporarily rewrite the user's query text as it appears in the text box with a query that matches the suggestion currently in focus. This enables the user to see what query is being suggested (if appropriate) and then select the search box and edit the query before dispatching it as a search.

You can rewrite the query text in the following ways:

- Add the `android:searchMode` attribute to your searchable configuration with the "queryRewriteFromText" value. In this case, the content from the suggestion's `SUGGEST_COLUMN_TEXT_1` column is used to rewrite the query text.
- Add the `android:searchMode` attribute to your searchable configuration with the "queryRewriteFromData" value. In this case, the content from the suggestion's `SUGGEST_COLUMN_INTENT_DATA` column is used to rewrite the query text. This should only be used with URI's or other data formats that are intended to be user-visible, such as HTTP URLs. Internal URI schemes should not be used to rewrite the query in this way.
- Provide a unique query text string in the `SUGGEST_COLUMN_QUERY` column of your suggestions table. If this column is present and contains a value for the current suggestion, it is used to rewrite the query text (and override either of the previous implementations).

Exposing search suggestions to Quick Search Box

Once you configure your application to provide custom search suggestions, making them available to the globally accessible Quick Search Box is as easy as modifying your searchable configuration to include `android:includeInGlobalSearch` as "true".

The only scenario in which additional work is necessary is when your content provider demands a read permission. In which case, you need to add a special `<path-permission>` element for the provider to grant Quick Search Box read access to your content provider. For example:

```

<provider android:name="MySuggestionProvider"
          android:authorities="com.example.MyCustomSuggestionProvider"
          android:readPermission="com.example.provider.READ_MY_DATA"
          android:writePermission="com.example.provider.WRITE_MY_DATA">
    <path-permission android:pathPrefix="/search_suggest_query"
                   android:readPermission="android.permission.GLOBAL_SEARCH" />
</provider>

```

In this example, the provider restricts read and write access to the content. The `<path-permission>` element amends the restriction by granting read access to content inside the `"/search_suggest_query"` path prefix when the `"android.permission.GLOBAL_SEARCH"` permission exists. This grants access to Quick Search Box so that it may query your content provider for suggestions.

If your content provider does not enforce read permissions, then Quick Search Box can read it by default.

Enabling suggestions on a device

When your application is configured to provide suggestions in Quick Search Box, it is not actually enabled to provide suggestions in Quick Search Box, by default. It is the user's choice whether to include suggestions from your application in the Quick Search Box. To enable search suggestions from your application, the user must open "Searchable items" (in Settings > Search) and enable your application as a searchable item.

Each application that is available to Quick Search Box has an entry in the Searchable items settings page. The entry includes the name of the application and a short description of what content can be searched from the application and made available for suggestions in Quick Search Box. To define the description text for your searchable application, add the `android:searchSettingsDescription` attribute to your searchable configuration. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:includeInGlobalSearch="true"
    android:searchSettingsDescription="@string/search_description" >
</searchable>
```

The string for `android:searchSettingsDescription` should be as concise as possible and state the content that is searchable. For example, "Artists, albums, and tracks" for a music application, or "Saved notes" for a notepad application. Providing this description is important so the user knows what kind of suggestions are provided. You should always include this attribute when `android:includeInGlobalSearch` is "true".

Remember that the user must visit the settings menu to enable search suggestions for your application before your search suggestions appear in Quick Search Box. As such, if search is an important aspect of your application, then you might want to consider a way to convey that to your users — you might provide a note the first time they launch the app that instructs them how to enable search suggestions for Quick Search Box.

Managing Quick Search Box suggestion shortcuts

Suggestions that the user selects from Quick Search Box can be automatically made into shortcuts. These are suggestions that the system has copied from your content provider so it can quickly access the suggestion without the need to re-query your content provider.

By default, this is enabled for all suggestions retrieved by Quick Search Box, but if your suggestion data changes over time, then you can request that the shortcuts be refreshed. For instance, if your suggestions refer to dynamic data, such as a contact's presence status, then you should request that the suggestion shortcuts be refreshed when shown to the user. To do so, include the `SUGGEST_COLUMN_SHORTCUT_ID` in your suggestions table. Using this column, you can configure the shortcut behavior for each suggestion in one of the following ways:

- Have Quick Search Box re-query your content provider for a fresh version of the suggestion shortcut.

Provide a value in the `SUGGEST_COLUMN_SHORTCUT_ID` column and the suggestion is requeried for a fresh version each time the shortcut is displayed. The shortcut is quickly displayed with whatever data was most recently available until the refresh query returns, at which point the suggestion is refreshed with the new information. The refresh query is sent to your content provider with a URI path of `SUGGEST_URI_PATH_SHORTCUT` (instead of `SUGGEST_URI_PATH_QUERY`).

The `Cursor` you return should contain one suggestion using the same columns as the original suggestion, or be empty, indicating that the shortcut is no longer valid (in which case, the suggestion disappears and the shortcut is removed).

If a suggestion refers to data that could take longer to refresh, such as a network-based refresh, you can also add the `SUGGEST_COLUMN_SPINNER_WHILE_REFRESHING` column to your suggestions table with a value of "true" in order to show a progress spinner for the right hand icon until the refresh is complete. Any value other than "true" does not show the progress spinner.

- Prevent the suggestion from being copied into a shortcut at all.

Provide a value of `SUGGEST_NEVER_MAKE_SHORTCUT` in the `SUGGEST_COLUMN_SHORTCUT_ID` column. In this case, the suggestion is never copied into a shortcut. This should only be necessary if you absolutely do not want the previously copied suggestion to appear. (Recall that if you provide a normal value for the column, then the suggestion shortcut appears only until the refresh query returns.)

c. Allow the default shortcut behavior to apply.

Leave the `SUGGEST_COLUMN_SHORTCUT_ID` empty for each suggestion that will not change and can be saved as a shortcut.

If none of your suggestions ever change, then you do not need the `SUGGEST_COLUMN_SHORTCUT_ID` column at all.

Note: Quick Search Box ultimately decides whether or not to create a shortcut for a suggestion, considering these values as a strong request from your application—there is no guarantee that the behavior you have requested for your suggestion shortcuts will be honored.

About Quick Search Box suggestion ranking

Once you make your application's search suggestions available to Quick Search Box, the Quick Search Box ranking determines how the suggestions are surfaced to the user for a particular query. This might depend on how many other apps have results for that query, and how often the user has selected your results compared to those from other apps. There is no guarantee about how your suggestions are ranked, or whether your app's suggestions show at all for a given query. In general, you can expect that providing quality results increases the likelihood that your app's suggestions are provided in a prominent position and apps that provide low quality suggestions are more likely to be ranked lower or not displayed.

See the [Searchable Dictionary sample app](#) for a complete demonstration of custom search suggestions.



Searchable Configuration

See also

- [Creating a Search Interface](#)
- [Adding Recent Query Suggestions](#)
- [Adding Custom Suggestions](#)

In order to implement search with assistance from the Android system (to deliver search queries to an activity and provide search suggestions), your application must provide a search configuration in the form of an XML file.

This page describes the search configuration file in terms of its syntax and usage. For more information about how to implement search features for your application, begin with the developer guide about [Creating a Search Interface](#).

FILE LOCATION:

`res/xml/filename.xml`

Android uses the filename as the resource ID.

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="string resource"
    android:hint="string resource"
    android:searchMode=["queryRewriteFromData" | "queryRewriteFromText"]
    android:searchButtonText="string resource"
    android:inputType="inputType"
    android:imeOptions="imeOptions"
    android:searchSuggestAuthority="string"
    android:searchSuggestPath="string"
    android:searchSuggestSelection="string"
    android:searchSuggestIntentAction="string"
    android:searchSuggestIntentData="string"
    android:searchSuggestThreshold="int"
    android:includeInGlobalSearch=["true" | "false"]
    android:searchSettingsDescription="string resource"
    android:queryAfterZeroResults=["true" | "false"]
    android:voiceSearchMode=["showVoiceSearchButton" | "launchWebSearch" | "launchRecognizer"]
    android:voiceLanguageModel=["free-form" | "web_search"]
    android:voicePromptText="string resource"
    android:voiceLanguage="string"
    android:voiceMaxResults="int"
    >
    <actionkey
        android: keycode="KEYCODE"
        android: queryActionMsg="string"
        android: suggestActionMsg="string"
        android: suggestActionMsgColumn="string" />
</searchable>
```

ELEMENTS:

`<searchable>`

Defines all search configurations used by the Android system to provide assisted search.

attributes:

`android:label`

String resource. (Required.) The name of your application. It should be the same as the name applied to the `android:label` attribute of your `<activity>` or `<application>` manifest element. This label is only visible to the user when you set `android:includeInGlobalSearch` to "true", in which case, this label is used to identify your application as a searchable item in the system's search settings.

`android:hint`

String resource. (Recommended.) The text to display in the search text field when no text has been entered. It provides a hint to the user about what content is searchable. For consistency with other Android applications, you should format the string for `android:hint` as "Search <content-or-product>". For example, "Search songs and artists" or "Search YouTube".

`android:searchMode`

Keyword. Sets additional modes that control the search presentation. Currently available modes define how the query text should be rewritten when a custom suggestion receives focus. The following mode values are accepted:

Value	Description
<code>"queryRewriteFromText"</code>	Use the value from the <code>SUGGEST_COLUMN_TEXT_1</code> column to rewrite the query text.
<code>"queryRewriteFromData"</code>	Use the value from the <code>SUGGEST_COLUMN_INTENT_DATA</code> column to rewrite the query text. This should only be used when the values in <code>SUGGEST_COLUMN_INTENT_DATA</code> are suitable for user inspection and editing, typically HTTP URI's.

For more information, see the discussion about rewriting the query text in [Adding Custom Suggestions](#).

`android:searchButtonText`

String resource. The text to display in the button that executes search. By default, the button shows a search icon (a magnifying glass), which is ideal for internationalization, so you should not use this attribute to change the button unless the behavior is something other than a search (such as a URL request in a web browser).

`android:inputType`

Keyword. Defines the type of input method (such as the type of soft keyboard) to use. For most searches, in which free-form text is expected, you don't need this attribute. See [inputType](#) for a list of suitable values for this attribute.

`android:imeOptions`

Keyword. Supplies additional options for the input method. For most searches, in which free-form text is expected, you don't need this attribute. The default IME is "actionSearch" (provides the "search" button instead of a carriage return in the soft keyboard). See [imeOptions](#) for a list of suitable values for this attribute.

Search suggestion attributes

If you have defined a content provider to generate search suggestions, you need to define additional attributes that configure communications with the content provider. When providing search suggestions, you need some of the following `<searchable>` attributes:

`android:searchSuggestAuthority`

String. (Required to provide search suggestions.) This value must match the authority string provided in the `android:authorities` attribute of the Android manifest `<provider>` element.

`android:searchSuggestPath`

String. This path is used as a portion of the suggestions query `Uri`, after the prefix and authority, but before the standard suggestions path. This is only required if you have a single content provider issuing different types of suggestions (such as for different data types) and you need a way to disambiguate the suggestions queries when you receive them.

`android:searchSuggestSelection`

String. This value is passed into your query function as the `selection` parameter. Typically this is a WHERE clause for your database, and should contain a single question mark, which is a placeholder for the actual query string that has been typed by the user (for example, `"query=?"`). However, you can also use any non-null value to trigger the delivery of the query text via the `selectionArgs` parameter (and then ignore the `selection` parameter).

`android:searchSuggestIntentAction`

String. The default intent action to be used when a user clicks on a custom search suggestion (such as `"android.intent.action.VIEW"`). If this is not overridden by the selected suggestion (via the `SUGGEST_COLUMN_INTENT_ACTION` column), this value is placed in the action field of the `Intent` when the user clicks a suggestion.

`android:searchSuggestIntentData`

String. The default intent data to be used when a user clicks on a custom search suggestion. If not overridden by the selected suggestion (via the `SUGGEST_COLUMN_INTENT_DATA` column), this value is placed in the data field of the `Intent` when the user clicks a suggestion.

`android:searchSuggestThreshold`

Integer. The minimum number of characters needed to trigger a suggestion look-up. Only guarantees that the system will not query your content provider for anything shorter than the threshold. The default value is 0.

For more information about the above attributes for search suggestions, see the guides for [Adding Recent Query Suggestions](#) and [Adding Custom Suggestions](#).

Quick Search Box attributes

To make your custom search suggestions available to Quick Search Box, you need some of the following `<searchable>` attributes:

`android:includeInGlobalSearch`

Boolean. (Required to provide search suggestions in Quick Search Box.) Set to "true" if you want your suggestions to be included in the globally accessible Quick Search Box. The user must still enable your application as a searchable item in the system search settings before your suggestions will appear in Quick Search Box.

`android:searchSettingsDescription`

String. Provides a brief description of the search suggestions that you provide to Quick Search Box, which is displayed in the searchable items entry for your application. Your description should concisely describe the content that is searchable. For example, "Artists, albums, and tracks" for a music application, or "Saved notes" for a notepad application.

`android:queryAfterZeroResults`

Boolean. Set to "true" if you want your content provider to be invoked for supersets of queries that have returned zero results in the past. For example, if your content provider returned zero results for "bo", it should be required for "bob". If set to "false", supersets are ignored for a single session ("bob" does not invoke a requery). This lasts only for the life of the search dialog or the life of the activity when using the search widget (when the search dialog or activity is reopened, "bo" queries your content provider again). The default value is false.

Voice search attributes

To enable voice search, you'll need some of the following `<searchable>` attributes:

`android:voiceSearchMode`

Keyword. (Required to provide voice search capabilities.) Enables voice search, with a specific mode for voice search. (Voice search may not be provided by the device, in which case these flags have no effect.) The following mode values are accepted:

Value	Description
<code>"showVoiceSearchButton"</code>	Display a voice search button, if voice search is available on the device. If set, then either <code>"launchWebSearch"</code> or <code>"launchRecognizer"</code> must also be set (separated by the pipe character).
<code>"launchWebSearch"</code>	The voice search button takes the user directly to a built-in voice web search activity. Most applications don't need this flag, as it takes the user away from the activity in which search was invoked.
<code>"launchRecognizer"</code>	The voice search button takes the user directly to a built-in voice recording activity. This activity prompts the user to speak, transcribes the spoken text, and forwards the resulting query text to the searchable activity, just as if the user typed it into the search UI and clicked the search button.

`android:voiceLanguageModel`

Keyword. The language model that should be used by the voice recognition system. The following values are accepted:

Value	Description
<code>"free_form"</code>	Use free-form speech recognition for dictating queries. This is primarily optimized for English. This is the default.
<code>"web_search"</code>	Use web-search-term recognition for shorter, search-like phrases. This is available in more languages than <code>"free_form"</code> .

Also see [EXTRA_LANGUAGE_MODEL](#) for more information.

`android:voicePromptText`

String. An additional message to display in the voice input dialog.

`android:voiceLanguage`

String. The spoken language to be expected, expressed as the string value of a constants in `Locale` (such as `"de"` for German or `"fr"` for French). This is needed only if it is different from the current value of [Locale.getDefault\(\)](#).

`android:voiceMaxResults`

Integer. Forces the maximum number of results to return, including the "best" result which is always provided as the `ACTION_SEARCH` intent's primary query. Must be 1 or greater. Use [EXTRA_RESULTS](#) to get the results from the intent. If not provided, the recognizer chooses how many results to return.

`<actionkey>`

Defines a device key and behavior for a search action. A search action provides a special behavior at the touch of a button on the device, based on the current query or focused suggestion. For example, the Contacts application provides a search action to initiate a phone call to the currently focused contact suggestion at the press of the CALL button.

Not all action keys are available on every device, and not all keys are allowed to be overridden in this way. For example, the "Home" key cannot be used and must always return to the home screen. Also be sure not to define an action key for a key that's needed for typing a search query. This essentially limits the available and reasonable action keys to the call button and menu button. Also note that action keys are not generally discoverable, so you should not provide them as a core user feature.

You must define the `android: keycode` to define the key and at least one of the other three attributes in order to define the search action.

attributes:

`android: keycode`

String. (Required.) A key code from `KeyEvent` that represents the action key you wish to respond to (for example `"KEYCODE_CALL"`). This is added to the `ACTION_SEARCH` intent that is passed to your searchable activity. To examine the key code, use `getIntExtra(SearchManager.ACTION_KEY)`. Not all keys are supported for a search action, as many of them are used for typing, navigation, or system functions.

`android: queryActionMsg`

String. An action message to be sent if the action key is pressed while the user is entering query text. This is added to the `ACTION_SEARCH` intent that the system passes to your searchable activity. To examine the string, use `getStringExtra(SearchManager.ACTION_MSG)`.

`android: suggestActionMsg`

String. An action message to be sent if the action key is pressed while a suggestion is in focus. This is added to the intent that the system passes to your searchable activity (using the action you've defined for the suggestion). To examine the string, use `getStringExtra(SearchManager.ACTION_MSG)`. This should only be used if all your suggestions support this action key. If not all suggestions can handle the same action key, then you must instead use the following `android: suggestActionMsgColumn` attribute.

`android: suggestActionMsgColumn`

String. The name of the column in your content provider that defines the action message for this action key, which is to be sent if the user presses the action key while a suggestion is in focus. This attribute lets you control the action key on a suggestion-by-suggestion basis, because, instead of using the `android: suggestActionMsg` attribute to define the action message for all suggestions, each entry in your content provider provides its own action message.

First, you must define a column in your content provider for each suggestion to provide an action message, then provide the name of that column in this attribute. The system looks at your suggestion cursor, using the string provided here to select your action message column, and then select the action message string from the Cursor. That string is added to the intent that the system passes to your searchable activity (using the action you've defined for suggestions). To examine the string, use `getStringExtra(SearchManager.ACTION_MSG)`. If the data does not exist for the selected suggestion, the action key is ignored.

EXAMPLE:

XML file saved at `res/xml/searchable.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="dictionary"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:includeInGlobalSearch="true"
    android:searchSettingsDescription="@string/settings_description" >
</searchable>
```



Accessibility

Accessibility is an important part of any app. Whether you're developing a new app or improving an existing one, ensure that components are accessible to everyone.

In this document

- › [Accessibility Overview](#)
- › [The Impact](#)
- › [Development Resources](#)

Accessibility Overview

By integrating accessible components and services, you can improve your app's usability for all users, including those with disabilities.

The Impact

By integrating accessibility features in your app, you make it more inclusive:

- **Increase your app's reach.**

According to [The World Bank](#), 15% of the world's population has some type of disability. People with disabilities depend on accessible apps and services to communicate, learn, and work. By making your app accessible, you can reach more users.

- **Improve your app's versatility.**

Accessibility can make it easier for all users, not only those with disabilities, to interact with your app. For example, if someone is using your app while cooking, accessibility features let them use voice commands instead of the touchscreen to navigate.

Development Resources

To make your app accessible, follow these key principles of inclusive design, development, and testing:

- **Design your app to support accessibility needs.**

By following material design best practices, you allow all of your users, including users with disabilities, to navigate and interact with your app more easily.

- **Develop your app using accessibility best practices.**

As you create your app, follow basic accessibility principles that make a big difference to your users, including content labeling, touch target size, color contrast, and view attributes.

- **Test with accessibility in mind.**

Integrate accessibility with your app's testing cycle by incorporating both manual and automated accessibility testing. To make this process easier, learn about the different accessibility services that Android offers, such as [TalkBack](#) and [Switch Access](#).

By familiarizing yourself with these services and testing your app using them, you can better understand the experience of users with accessibility needs.

If you want to create a new interaction model to help people with a specific type of disability, or if your app's content has a particular accessibility requirement, you can create your own accessibility service. To learn more, see the [Building Accessibility Services](#) guide.

Making Apps More Accessible

In this document

- › [Labeling UI Elements](#)
- › [Grouping Content](#)
 - › [Organizing content into a single announcement](#)
 - › [Creating natural groupings](#)
- › [Making Touch Targets Large](#)
- › [Providing Adequate Color Contrast](#)
- › [Using Cues Other Than Color](#)
- › [Applying Ideas from Accessibility Resources](#)
 - › [Applying Accessibility Design Principles](#)
 - › [Activating Accessibility Settings](#)
 - › [Testing Your App's Accessibility](#)
 - › [Working with Custom Views](#)
 - › [Sample Accessibility App](#)

See also

- › [Accessibility Developer Checklist](#)
- › [Accessibility Testing Checklist](#)
- › [Material Design: Accessibility](#)
- › [Designing Effective Navigation](#)
- › [Training: Implementing Accessibility](#)

Android apps should be usable by everyone, including people with disabilities.

Common disabilities that affect a person's use of an Android device include blindness or low vision, color blindness, deafness or impaired hearing, and restricted motor skills. When you develop apps with accessibility in mind, you make the user experience better not only for users with these disabilities, but also for all of your other users.

This document presents guidelines for improving your app's accessibility. It also lists resources that provide additional details and information related to accessibility features in Android.

Labeling UI Elements

It's important to provide useful and descriptive labels that explain the meaning and purpose of each interactive element to users. These labels allow screen readers, such as TalkBack, to properly explain the function of a particular control to users who rely on these services.

You can provide labels for elements in the following two ways:

- When labeling *static* elements, which don't change appearance throughout an activity's lifecycle, add an attribute to the corresponding XML element within the activity's layout resource file.
- When labeling *dynamic* elements, which change appearance during an activity's lifetime, set the element's label in the dynamic logic that changes the element's appearance.

The actual attributes and methods that you use to apply the element's label depend on the type of element:

- When labeling graphical elements, such as `ImageView` and `ImageButton` objects, use the `android:contentDescription` XML attribute for static elements and the `setContentDescription()` method for dynamic elements.

For graphical elements that are purely decorative, set their respective `android:contentDescription` XML attributes to "`@null`". If your app only supports devices running Android 4.1 (API level 16) or higher, you can instead set these elements' `android:isImportantForAccessibility` XML attributes to "no".

- When labeling editable elements, such as `EditText` objects, use the `android:hint` XML attribute for static elements and the `setHint()` method for dynamic elements to indicate each element's purpose.
- If your app is installed on a device running Android 4.2 (API level 17) or higher, use the `android:labelFor` attribute when labeling `View` objects that serve as content labels for other `View` objects.

Note: Accessibility services automatically capture the text that appears in `TextView` objects, so you usually don't need to label these elements.

In the following example, a static `ImageButton` object designed to provide sharing functionality is given a label of "share" (defined in `values/strings.xml`) as the value:

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentBottom="true"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"  
    android:contentDescription="@string/share"  
    android:src="@drawable/ic_share" />
```

Note: Many accessibility services, such as TalkBack and BrailleBack, automatically announce an element's type after announcing its label, so you shouldn't include element types in your labels. For example, "`submit`" is a good label for a `Button` object, but "`submitButton`" isn't a good label.

This next example shows how to update a dynamic `ImageView` object that displays a play or pause icon within an activity:

```
ImageView playPauseImageView = new ImageView();  
boolean mediaCurrentlyPlaying = true;  
...  
private void updateImageButton() {  
    if (mediaCurrentlyPlaying) {  
        playPauseImageView.setImageResource(R.drawable.ic_pause);  
  
        // In res/values/strings.xml, "pause" contains a value of "Pause".  
        playPauseImageView.setContentDescription(getString(R.string.pause));  
    } else {  
        playPauseImageView.setImageResource(R.drawable.ic_play);  
  
        // In res/values/strings.xml, "play" contains a value of "Play".  
        playPauseImageView.setContentDescription(getString(R.string.play));  
    }  
}
```

When adding labels to the elements that appear in a given activity, make sure that each label is unique so that users can identify each element accurately. In particular, you should include additional text or contextual information in elements within reused layouts, such as `ListView` and `RecyclerView` objects, so that each child element is uniquely identified.

Grouping Content

You should arrange related content into groups so that accessibility services announce the content in a way that reflects its natural groupings. Users of assistive technology then don't need to swipe, scan, or wait as much to discover all information on the screen.

The two most effective methods of grouping related content are the following:

- For smaller or simpler groups of content, you can [organize all content](#) into a single announcement.
- For larger or more complex content structures, you can [create natural groupings](#) for the content.

Organizing content into a single announcement

If users should treat a set of elements as a single unit of information, you can group these elements in a focusable container. That way, accessibility services present the grouped content in a single announcement. In the following example, a `RelativeLayout` element contains pieces of content that relate to one another:

```
<RelativeLayout
    android:id="@+id/song_data_container"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:focusable="true">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/song_title"
        android:text="@string/song_title" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/singer"
        android:layout_toRightOf="@+id/song_title"
        android:text="@string/singer" />
    ...
</RelativeLayout>
```

Note: You can specify the `android:contentDescription` XML attribute on a container to override automatic grouping and ordering of contained items.

When grouping element labels, make sure that you don't create unnecessarily verbose announcements for accessibility services to present.

Creating natural groupings

If you need to convey more complex structures, such as tables, you can assign focus to one piece of the structure at a time, such as a single row. To define the proper focusing pattern for a set of related content, place each piece of the structure into its own focusable `ViewGroup`. In the following example, a structure comprising 6 `TextView` objects is divided into 3 pieces, which the `RelativeLayout` elements define:

```
<LinearLayout
    ...
    orientation="vertical">
    <RelativeLayout
        ...
        android:focusable="true">
        <TextView ... />
        <TextView ... />
    </RelativeLayout>
    <RelativeLayout
        ...
        android:focusable="true">
        <TextView ... />
        <TextView ... />
    </RelativeLayout>
    <RelativeLayout
        ...
        android:focusable="true">
        <TextView ... />
        <TextView ... />
    </RelativeLayout>
</LinearLayout>
```

Making Touch Targets Large

Many people have difficulty interacting with small touch targets on a device's screen. This could be because their fingers are large or because they have a motor or visual impairment. By providing larger touch targets, you make it substantially easier for users to navigate your app.

In general, you want the touchable area of focusable items to be a minimum of 48dp x 48dp. Larger is even better.

To ensure that each focusable item in your app has a large enough touch target, set the `android:minWidth` and `android:minHeight` attributes of each interactive element to 48dp or greater:

```
<ImageButton  
    ...  
    android:minWidth="48dp"  
    android:minHeight="48dp" />
```

You can also add padding or use the [TouchDelegate](#) API to increase the size of your elements' touch targets without increasing the size of the elements themselves.

Providing Adequate Color Contrast

People with low vision and those who use devices with dimmed displays can have difficulty reading information on the screen. By providing increased contrast ratios between the foreground and background colors in your app, you make it easier for users to navigate within and between screens.

To help developers use sufficient contrast ratios in their apps, The World Wide Web Consortium (W3C) has created a set of [color contrast accessibility guidelines](#):

- For large text, 18 points or higher for regular text and 14 points or higher for bold text, you should use a contrast ratio of at least **3.0 to 1**.
- For small text, smaller than 18 points for regular text and smaller than 14 points for bold text, you should use a contrast ratio of at least **4.5 to 1**.

Figure 1 shows two versions of an activity. One version uses a low contrast ratio between background and foreground colors, and the other version uses an increased contrast ratio:

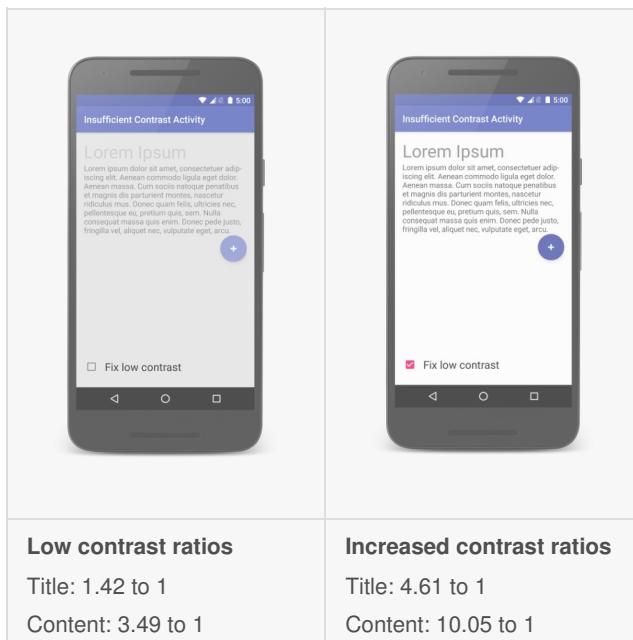


Figure 1: Example of low and increased contrast ratios between foreground and background colors

To check your contrast ratios, you can use the [Accessibility Scanner](#) or one of the many contrast checkers available online.

Note: If your app's color scheme uses partially transparent colors, keep in mind that these non-opaque colors might appear lighter than the color defined by their RGB values.

Using Cues Other Than Color

To assist users with color vision deficiencies, use cues other than color to distinguish UI elements within your app's screens. These techniques could include using different shapes or sizes, providing text or visual patterns, or adding audio- or touch-based (haptic) feedback to mark the elements' differences.

Figure 2 shows two versions of an activity. One version uses only color to distinguish between two possible actions in a workflow, and the other uses shapes and text to highlight the differences between the two options:

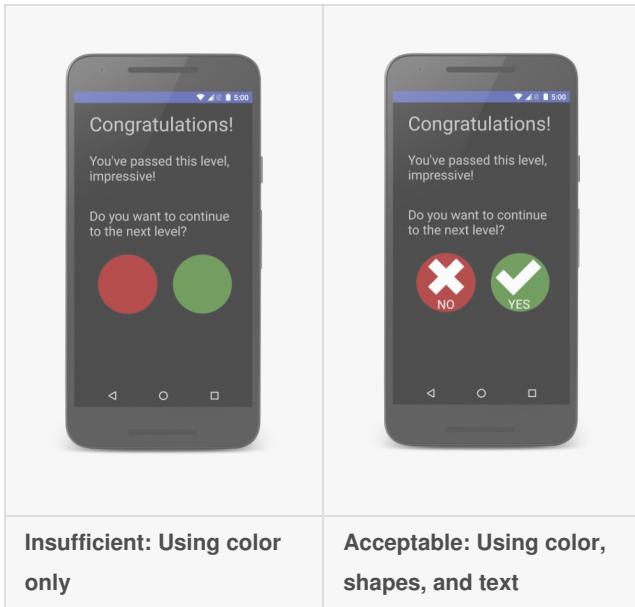


Figure 2: Examples of differentiating UI elements using color only and using color, shapes, and text

Presenting Media Content

If you're developing an app that includes media content, such as a video clip or an audio recording, make sure that users with different types of accessibility needs can understand this material, as well. In particular, you should make sure that you've provided the following accommodations:

- All video and audio materials should include controls that allow users to pause or stop the media, change the volume, and toggle subtitles (captions).
- If a video presents information that is vital to completing a workflow, you should provide the same content in an alternative format, such as a transcript.

Applying Ideas from Accessibility Resources

The following resources describe other accessibility features that Android supports. After you've applied the guidelines in the previous sections, use the following information to further improve your app's accessibility.

Applying Accessibility Design Principles

Google's material design guidelines include a set of recommendations for structuring your app's UI so that all users, including users with disabilities, can interact easily with your app. To view these guidelines, navigate to the [Accessibility](#) page within the material design site.

Activating Accessibility Settings

In addition to offering the accessibility services described in other sections on this page, the Android platform includes several [accessibility settings](#), such as increased font size and magnified screen area, that users can adjust. As you develop your app, you should adjust these settings yourself to ensure that your app's important UI elements remain fully visible and usable.

Testing Your App's Accessibility

As you develop your app, it's important to test its accessibility using a combination of manual tests, automated tests, and user tests. To learn more about the aspects of your app that you should test, see [Testing Your App's Accessibility](#).

Working with Custom Views

The UI components built into the Android framework have predefined accessibility capabilities. These components include useful metadata that accessibility services access to present these components successfully to users who use assistive technology.

When you create your own custom view, such as an animated bar graph widget, you need to manually define the accessibility metadata so that accessibility services can interpret the custom view properly. For more information about how to make custom views accessible, read [Build Accessible Custom Views](#).

Sample Accessibility App

The [Basic Accessibility](#) sample, available on GitHub, presents an app containing a variety of UI elements. It demonstrates how to add accessibility markup to each of these elements and how accessibility services respond to this markup.

Accessibility Developer Checklist

In this document

- › [Accessibility Requirements](#)
- › [Accessibility Recommendations](#)
- › [Special Cases and Considerations](#)

See also

- › [Android Design: Accessibility](#)
- › [Accessibility Testing Checklist](#)
- › [Training: Implementing Accessibility](#)
- › [Designing Effective Navigation](#)

Making an application accessible is about a deep commitment to usability, getting the details right and delighting your users. This document provides a checklist of accessibility requirements, recommendations and considerations to help you make sure your application is accessible. Following this checklist does not guarantee your application is accessible, but it's a good place to start.

Creating an accessible application is not just the responsibility of developers. Involve your design and testing folks as well, and make them are aware of the guidelines for these other stages of development:

- [Android Design: Accessibility](#)
- [Accessibility Testing Checklist](#)

In most cases, creating an accessible Android application does not require extensive code restructuring. Rather, it means working through the subtle details of how users interact with your application, so you can provide them with feedback they can sense and understand. This checklist helps you focus on the key development issues to get the details of accessibility right.

Accessibility Requirements

The following steps must be completed in order to ensure a minimum level of application accessibility.

1. **Describe user interface controls:** Provide content [descriptions](#) for user interface components that do not have visible text, particularly [ImageButton](#), [ImageView](#) and [CheckBox](#) components. Use the [android:contentDescription](#) XML layout attribute or the [setContentDescription\(CharSequence\)](#) method to provide this information for accessibility services. (Exception: [decorative graphics](#))
2. **Enable focus-based navigation:** Make sure [users can navigate](#) your screen layouts using hardware-based or software directional controls (D-pads, trackballs, keyboards and navigation gestures). In a few cases, you may need to make user interface components [focusable](#) or change the [focus order](#) to be more logical for user actions.
3. **Custom view controls:** If you build [custom interface controls](#) for your application, [implement accessibility interfaces](#) for your custom views and provide content descriptions. For custom controls that are intended to be compatible with versions of Android back to 1.6, use the [Support Library](#) to implement the latest accessibility features.
4. **No audio-only feedback:** Audio feedback must always have a secondary feedback mechanism to support users who are deaf or hard of hearing. For example, a sound alert for the arrival of a message must be accompanied by a system [Notification](#), haptic feedback (if available) or other visual alert.
5. **Test:** Test accessibility by navigating your application using directional controls, and using eyes-free navigation with TalkBack enabled. For more accessibility testing information, see the [Accessibility Testing Checklist](#).

Accessibility Recommendations

The following steps are recommended for ensuring the accessibility of your application. If you do not take these actions, it may impact the overall accessibility and quality of your application.

1. **Android Design Accessibility Guidelines:** Before building your layouts, review and follow the accessibility guidelines provided in the [Design guidelines](#).
2. **Framework-provided controls:** Use Android's built-in user interface controls whenever possible, as these components provide accessibility support by default.
3. **Temporary or self-hiding controls and notifications:** Avoid having user interface controls that fade out or disappear after a certain amount of time. If this behavior is important to your application, provide an alternative interface for these functions.

Special Cases and Considerations

The following list describes specific situations where action should be taken to ensure an accessible app. Review this list to see if any of these special cases and considerations apply to your application, and take the appropriate action.

1. **Text field hints:** For [EditText](#) fields, provide an [android:hint](#) attribute *instead* of a content description, to help users understand what content is expected when the text field is empty and allow the contents of the field to be spoken when it is filled.
2. **Custom controls with high visual context:** If your application contains a [custom control](#) with a high degree of visual context (such as a calendar control), default accessibility services processing may not provide adequate descriptions for users, and you should consider providing a [virtual view hierarchy](#) for your control using [AccessibilityNodeProvider](#).
3. **Custom controls and click handling:** If a custom control in your application performs specific handling of user touch interaction, such as listening with [onTouchEvent\(MotionEvent\)](#) for [MotionEvent.ACTION_DOWN](#) and [MotionEvent.ACTION_UP](#) and treating it as a click event, you must trigger an [AccessibilityEvent](#) equivalent to a click and provide a way for accessibility services to perform this action for users. For more information, see [Handling custom touch events](#).
4. **Controls that change function:** If you have buttons or other controls that change function during the normal activity of a user in your application (for example, a button that changes from **Play** to **Pause**), make sure you also change the [android:contentDescription](#) of the button appropriately.
5. **Prompts for related controls:** Make sure sets of controls which provide a single function, such as the [DatePicker](#), provide useful audio feedback when an user interacts with the individual controls.
6. **Video playback and captioning:** If your application provides video playback, it must support captioning and subtitles to assist users who are deaf or hard of hearing. Your video playback controls must also clearly indicate if captioning is available for a video and provide a clear way of enabling captions.
7. **Supplemental accessibility audio feedback:** Use only the Android accessibility framework to provide accessibility audio feedback for your app. Accessibility services such as [TalkBack](#) should be the only way your application provides accessibility audio prompts to users. Provide the prompting information with a [android:contentDescription](#) XML layout attribute or dynamically add it using accessibility framework APIs. For example, if your application takes action that you want to announce to a user, such as automatically turning the page of a book, use the [announceForAccessibility\(CharSequence\)](#) method to have accessibility services speak this information to the user.
8. **Custom controls with complex visual interactions:** For custom controls that provide complex or non-standard visual interactions, provide a [virtual view hierarchy](#) for your control using [AccessibilityNodeProvider](#) that allows accessibility services to provide a simplified interaction model for the user. If this approach is not feasible, consider providing an alternate view that is accessible.
9. **Sets of small controls:** If you have controls that are smaller than the minimum recommended touch size in your application screens, consider grouping these controls together using a [ViewGroup](#) and providing a [android:contentDescription](#) for the group.
10. **Decorative images and graphics:** Elements in application screens that are purely decorative and do not provide any content or enable a user action should not have accessibility content descriptions.

Building Accessibility Services

Topics

- › [Manifest Declarations and Permissions](#)
 - › [Accessibility service declaration](#)
 - › [Accessibility service configuration](#)
- › [Registering for Accessibility Events](#)
- › [AccessibilityService Methods](#)
- › [Getting Event Details](#)
- › [Taking Action for Users](#)
 - › [Listening for gestures](#)
 - › [Using accessibility actions](#)
 - › [Using focus types](#)
- › [Example Code](#)

Key classes

- › [AccessibilityService](#)
- › [AccessibilityServiceInfo](#)
- › [AccessibilityEvent](#)
- › [AccessibilityRecord](#)
- › [AccessibilityNodeInfo](#)

See also

- › [Training: Implementing Accessibility](#)

An accessibility service is an application that provides user interface enhancements to assist users with disabilities, or who may temporarily be unable to fully interact with a device. For example, users who are driving, taking care of a young child or attending a very loud party might need additional or alternative interface feedback.

Android provides standard accessibility services, including TalkBack, and developers can create and distribute their own services. This document explains the basics of building an accessibility service.

The ability for you to build and deploy accessibility services was introduced with Android 1.6 (API Level 4) and received significant improvements with Android 4.0 (API Level 14). The Android [Support Library](#) was also updated with the release of Android 4.0 to provide support for these enhanced accessibility features back to Android 1.6. Developers aiming for widely compatible accessibility services are encouraged to use the Support Library and develop for the more advanced accessibility features introduced in Android 4.0.

Manifest Declarations and Permissions

Applications that provide accessibility services must include specific declarations in their application manifests to be treated as an accessibility service by the Android system. This section explains the required and optional settings for accessibility services.

Accessibility service declaration

In order to be treated as an accessibility service, you must include a `service` element (rather than the `activity` element) within the `application` element in your manifest. In addition, within the `service` element, you must also include an accessibility service intent filter. For compatibility with Android 4.1 and higher, the manifest must also protect the service by adding the `BIND_ACCESSIBILITY_SERVICE`

permission to ensure that only the system can bind to it. Here's an example:

```
<application>
    <service android:name=".MyAccessibilityService"
        android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE"
        android:label="@string/accessibility_service_label">
        <intent-filter>
            <action android:name="android.accessibilityservice.AccessibilityService" />
        </intent-filter>
    </service>
</application>
```

These declarations are required for all accessibility services deployed on Android 1.6 (API Level 4) or higher.

Accessibility service configuration

Accessibility services must also provide a configuration which specifies the types of accessibility events that the service handles and additional information about the service. The configuration of an accessibility service is contained in the [AccessibilityServiceInfo](#) class. Your service can build and set a configuration using an instance of this class and [setServiceInfo\(\)](#) at runtime. However, not all configuration options are available using this method.

Beginning with Android 4.0, you can include a [<meta-data>](#) element in your manifest with a reference to a configuration file, which allows you to set the full range of options for your accessibility service, as shown in the following example:

```
<service android:name=".MyAccessibilityService">
    ...
    <meta-data
        android:name="android.accessibilityservice"
        android:resource="@xml/accessibility_service_config" />
</service>
```

This meta-data element refers to an XML file that you create in your application's resource directory (`<project_dir>/res/xml/accessibility_service_config.xml`). The following code shows example contents for the service configuration file:

```
<accessibility-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/accessibility_service_description"
    android:packageName="com.example.android.apis"
    android:accessibilityEventTypes="typeAllMask"
    android:accessibilityFlags="flagDefault"
    android:accessibilityFeedbackType="feedbackSpoken"
    android:notificationTimeout="100"
    android:canRetrieveWindowContent="true"
    android:settingsActivity="com.example.android.accessibility.ServiceSettingsActivity"
/>
```

For more information about the XML attributes which can be used in the accessibility service configuration file, follow these links to the reference documentation:

- [android:description](#)
- [android:packageName](#)
- [android:accessibilityEventTypes](#)
- [android:accessibilityFlags](#)
- [android:accessibilityFeedbackType](#)
- [android:notificationTimeout](#)
- [android:canRetrieveWindowContent](#)
- [android:settingsActivity](#)

For more information about which configuration settings can be dynamically set at runtime, see the [AccessibilityServiceInfo](#) reference documentation.

Registering for Accessibility Events

One of the most important functions of the accessibility service configuration parameters is to allow you to specify what types of accessibility events your service can handle. Being able to specify this information enables accessibility services to cooperate with each other, and allows you as a developer the flexibility to handle only specific events types from specific applications. The event filtering can include the following criteria:

- **Package Names** - Specify the package names of applications whose accessibility events you want your service to handle. If this parameter is omitted, your accessibility service is considered available to service accessibility events for any application. This parameter can be set in the accessibility service configuration files with the `android:packageNames` attribute as a comma-separated list, or set using the `AccessibilityServiceInfo.packageNames` member.
- **Event Types** - Specify the types of accessibility events you want your service to handle. This parameter can be set in the accessibility service configuration files with the `android:accessibilityEventTypes` attribute as a list separated by the `|` character (for example `accessibilityEventTypes="typeViewClicked|typeViewFocused"`), or set using the `AccessibilityServiceInfo.eventTypes` member.

When setting up your accessibility service, carefully consider what events your service is able to handle and only register for those events. Since users can activate more than one accessibility services at a time, your service must not consume events that it is not able to handle. Remember that other services may handle those events in order to improve a user's experience.

Note: The Android framework dispatches accessibility events to more than one accessibility service if the services provide different [feedback types](#). However, if two or more services provide the same feedback type, then only the first registered service receives the event.

AccessibilityService Methods

An accessibility service must extend the `AccessibilityService` class and override the following methods from that class. These methods are presented in the order in which they are called by the Android system, from when the service is started (`onServiceConnected()`), while it is running (`onAccessibilityEvent()`, `onInterrupt()`) to when it is shut down (`onUnbind()`).

- `onServiceConnected()` - (optional) This system calls this method when it successfully connects to your accessibility service. Use this method to do any one-time setup steps for your service, including connecting to user feedback system services, such as the audio manager or device vibrator. If you want to set the configuration of your service at runtime or make one-time adjustments, this is a convenient location from which to call `setServiceInfo()`.
- `onAccessibilityEvent()` - (required) This method is called back by the system when it detects an `AccessibilityEvent` that matches the event filtering parameters specified by your accessibility service. For example, when the user clicks a button or focuses on a user interface control in an application for which your accessibility service is providing feedback. When this happens, the system calls this method, passing the associated `AccessibilityEvent`, which the service can then interpret and use to provide feedback to the user. This method may be called many times over the lifecycle of your service.
- `onInterrupt()` - (required) This method is called when the system wants to interrupt the feedback your service is providing, usually in response to a user action such as moving focus to a different control. This method may be called many times over the lifecycle of your service.
- `onUnbind()` - (optional) This method is called when the system is about to shutdown the accessibility service. Use this method to do any one-time shutdown procedures, including de-allocating user feedback system services, such as the audio manager or device vibrator.

These callback methods provide the basic structure for your accessibility service. It is up to you to decide on how to process data provided by the Android system in the form of `AccessibilityEvent` objects and provide feedback to the user. For more information about getting information from an accessibility event, see the [Implementing Accessibility](#) training.

Getting Event Details

The Android system provides information to accessibility services about the user interface interaction through [AccessibilityEvent](#) objects. Prior to Android 4.0, the information available in an accessibility event, while providing a significant amount of detail about a user interface control selected by the user, offered limited contextual information. In many cases, this missing context information might be critical to understanding the meaning of the selected control.

An example of an interface where context is critical is a calendar or day planner. If the user selects a 4:00 PM time slot in a Monday to Friday day list and the accessibility service announces “4 PM”, but does not announce the weekday name, the day of the month, or the month name, the resulting feedback is confusing. In this case, the context of a user interface control is critical to a user who wants to schedule a meeting.

Android 4.0 significantly extends the amount of information that an accessibility service can obtain about an user interface interaction by composing accessibility events based on the view hierarchy. A view hierarchy is the set of user interface components that contain the component (its parents) and the user interface elements that may be contained by that component (its children). In this way, the Android system can provide much richer detail about accessibility events, allowing accessibility services to provide more useful feedback to users.

An accessibility service gets information about an user interface event through an [AccessibilityEvent](#) passed by the system to the service's [onAccessibilityEvent\(\)](#) callback method. This object provides details about the event, including the type of object being acted upon, its descriptive text and other details. Starting in Android 4.0 (and supported in previous releases through the [AccessibilityEventCompat](#) object in the Support Library), you can obtain additional information about the event using these calls:

- [AccessibilityEvent.getRecordCount\(\)](#) and [getRecord\(int\)](#) - These methods allow you to retrieve the set of [AccessibilityRecord](#) objects which contributed to the [AccessibilityEvent](#) passed to you by the system. This level of detail provides more context for the event that triggered your accessibility service.
- [AccessibilityEvent.getSource\(\)](#) - This method returns an [AccessibilityNodeInfo](#) object. This object allows you to request view layout hierarchy (parents and children) of the component that originated the accessibility event. This feature allows an accessibility service to investigate the full context of an event, including the content and state of any enclosing views or child views.

Important: The ability to investigate the view hierarchy from an [AccessibilityEvent](#) potentially exposes private user information to your accessibility service. For this reason, your service must request this level of access through the accessibility [service configuration XML](#) file, by including the `canRetrieveWindowContent` attribute and setting it to `true`. If you do not include this setting in your service configuration xml file, calls to [getSource\(\)](#) fail.

Note: In Android 4.1 (API Level 16) and higher, the [getSource\(\)](#) method, as well as [AccessibilityNodeInfo.getChildAt\(\)](#) and [getParent\(\)](#), return only view objects that are considered important for accessibility (views that draw content or respond to user actions). If your service requires all views, it can request them by setting the `flags` member of the service's [AccessibilityServiceInfo](#) instance to `FLAG_INCLUDE_NOT_IMPORTANT_VIEWS`.

Taking Action for Users

Starting with Android 4.0 (API Level 14), accessibility services can act on behalf of users, including changing the input focus and selecting (activating) user interface elements. In Android 4.1 (API Level 16) the range of actions has been expanded to include scrolling lists and interacting with text fields. Accessibility services can also take global actions, such as navigating to the Home screen, pressing the Back button, opening the notifications screen and recent applications list. Android 4.1 also includes a new type of focus, *Accessibility Focus*, which makes all visible elements selectable by an accessibility service.

These new capabilities make it possible for developers of accessibility services to create alternative navigation modes such as [gesture navigation](#), and give users with disabilities improved control of their Android devices.

Listening for gestures

Accessibility services can listen for specific gestures and respond by taking action on behalf of a user. This feature, added in Android 4.1 (API Level 16), and requires that your accessibility service request activation of the Explore by Touch feature. Your service can request this activation by setting the `flags` member of the service's [AccessibilityServiceInfo](#) instance to `FLAG_REQUEST_TOUCH_EXPLORATION_MODE`, as shown in the following example.

```
public class MyAccessibilityService extends AccessibilityService {  
    @Override  
    public void onCreate() {  
        getServiceInfo().flags = AccessibilityServiceInfo.FLAG_REQUEST_TOUCH_EXPLORATION_MODE;  
    }  
    ...  
}
```

Once your service has requested activation of Explore by Touch, the user must allow the feature to be turned on, if it is not already active. When this feature is active, your service receives notification of accessibility gestures through your service's `onGesture()` callback method and can respond by taking actions for the user.

Using accessibility actions

Accessibility services can take action on behalf of users to make interacting with applications simpler and more productive. The ability of accessibility services to perform actions was added in Android 4.0 (API Level 14) and significantly expanded with Android 4.1 (API Level 16).

In order to take actions on behalf of users, your accessibility service must `register` to receive events from a few or many applications and request permission to view the content of applications by setting the `android:canRetrieveWindowContent` to `true` in the [service configuration file](#). When events are received by your service, it can then retrieve the `AccessibilityNodeInfo` object from the event using `getSource()`. With the `AccessibilityNodeInfo` object, your service can then explore the view hierarchy to determine what action to take and then act for the user using `performAction()`.

```
public class MyAccessibilityService extends AccessibilityService {  
  
    @Override  
    public void onAccessibilityEvent(AccessibilityEvent event) {  
        // get the source node of the event  
        AccessibilityNodeInfo nodeInfo = event.getSource();  
  
        // Use the event and node information to determine  
        // what action to take  
  
        // take action on behalf of the user  
        nodeInfo.performAction(AccessibilityNodeInfo.ACTION_SCROLL_FORWARD);  
  
        // recycle the nodeInfo object  
        nodeInfo.recycle();  
    }  
    ...  
}
```

The `performAction()` method allows your service to take action within an application. If your service needs to perform a global action such as navigating to the Home screen, pressing the Back button, opening the notifications screen or recent applications list, then use the `performGlobalAction()` method.

Using focus types

Android 4.1 (API Level 16) introduces a new type of user interface focus called *Accessibility Focus*. Accessibility services can use this type of focus to select any visible user interface element and act on it. This focus type is different from the more well known *Input Focus*, which determines what on-screen user interface element receives input when a user types characters, presses **Enter** on a keyboard or pushes the center button of a D-pad control.

Accessibility Focus is completely separate and independent from Input Focus. In fact, it is possible for one element in a user interface to have Input Focus while another element has Accessibility Focus. The purpose of Accessibility Focus is to provide accessibility services with a method of interacting with any visible element on a screen, regardless of whether or not the element is input-focusable from a system perspective. You can see accessibility focus in action by testing accessibility gestures. For more information about testing this feature, see [Testing gesture navigation](#).

Note: Accessibility services that use Accessibility Focus are responsible for synchronizing the current Input Focus when an element is capable of this type of focus. Services that do not synchronize Input Focus with Accessibility Focus run the risk of causing problems in

applications that expect input focus to be in a specific location when certain actions are taken.

An accessibility service can determine what user interface element has Input Focus or Accessibility Focus using the `AccessibilityNodeInfo.findFocus()` method. You can also search for elements that can be selected with Input Focus using the `focusSearch()` method. Finally, your accessibility service can set Accessibility Focus using the `performAction(AccessibilityNodeInfo.ACTION_SET_ACCESSIBILITY_FOCUS)` method.

Example Code

The API Demo project contains two samples which can be used as a starting point for generating accessibility services (`<sdk>/samples/<platform>/ApiDemos/src/com/example/android/apis/accessibility`):

- **ClockBackService** - This service is based on the original implementation of `AccessibilityService` and can be used as a base for developing basic accessibility services that are compatible with Android 1.6 (API Level 4) and higher.
- **TaskBackService** - This service is based on the enhanced accessibility APIs introduced in Android 4.0 (API Level 14). However, you can use the Android [Support Library](#) to substitute classes introduced in later API levels (e.g., `AccessibilityRecord`, `AccessibilityNodeInfo`) with equivalent support package classes (e.g., `AccessibilityRecordCompat`, `AccessibilityNodeInfoCompat`) to make this example work with API versions back to Android 1.6 (API Level 4).



Animation and Graphics

Make your apps look and perform their best using Android's powerful graphics features such as OpenGL, hardware acceleration, and built-in UI animations.

BLOG ARTICLES

Android 4.0 Graphics and Animations

Earlier this year, Android 3.0 launched with a new 2D rendering pipeline designed to support hardware acceleration on tablets. With this new pipeline, all drawing operations performed by the UI toolkit are carried out using the GPU. You'll be happy to hear that Android 4.0, Ice Cream Sandwich, brings an improved version of the hardware-accelerated 2D rendering pipeline.

Introducing ViewPropertyAnimator

This new animation system makes it easy to animate any kind of property on any object, including the new properties added to the View class in 3.0. In the 3.1 release, we added a small utility class that makes animating these properties even easier.

Android 3.0 Hardware Acceleration

Hardware accelerated graphics is nothing new to the Android platform, it has always been used for windows composition or OpenGL games for instance, but with this new rendering pipeline applications can benefit from an extra boost in performance.

TRAINING

Displaying Bitmaps Efficiently

This class covers some common techniques for processing and loading Bitmap objects in a way that keeps your user interface (UI) components responsive and avoids exceeding your application memory limit.



Animation and Graphics Overview

Android provides a variety of powerful APIs for applying animation to UI elements and drawing custom 2D and 3D graphics. The sections below provide an overview of the APIs and system capabilities available and help you decide with approach is best for your needs.

Animation

The Android framework provides two animation systems: property animation and view animation. Both animation systems are viable options, but the property animation system, in general, is the preferred method to use, because it is more flexible and offers more features. In addition to these two systems, you can utilize Drawable animation, which allows you to load drawable resources and display them one frame after another.

Property Animation

Introduced in Android 3.0 (API level 11), the property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well.

View Animation

View Animation is the older system and can only be used for Views. It is relatively easy to setup and offers enough capabilities to meet many application's needs.

Drawable Animation

Drawable animation involves displaying [Drawable](#) resources one after another, like a roll of film. This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.

2D and 3D Graphics

When writing an application, it's important to consider exactly what your graphical demands will be. Varying graphical tasks are best accomplished with varying techniques. For example, graphics and animations for a rather static application should be implemented much differently than graphics and animations for an interactive game. Here, we'll discuss a few of the options you have for drawing graphics on Android and which tasks they're best suited for.

Canvas and Drawables

Android provides a set of [View](#) widgets that provide general functionality for a wide array of user interfaces. You can also extend these widgets to modify the way they look or behave. In addition, you can do your own custom 2D rendering using the various drawing methods contained in the [Canvas](#) class or create [Drawable](#) objects for things such as textured buttons or frame-by-frame animations.

Hardware Acceleration

Beginning in Android 3.0, you can hardware accelerate the majority of the drawing done by the Canvas APIs to further increase their performance.

OpenGL

Android supports OpenGL ES 1.0 and 2.0, with Android framework APIs as well as natively with the Native Development Kit (NDK). Using the framework APIs is desireable when you want to add a few graphical enhancements to your application that are not supported with the Canvas APIs, or if you desire platform independence and don't demand high performance. There is a performance hit in using the framework APIs compared to the NDK, so for many graphic intensive applications such as games, using the NDK is beneficial (It is

important to note though that you can still get adequate performance using the framework APIs. For example, the Google Body app is developed entirely using the framework APIs). OpenGL with the NDK is also useful if you have a lot of native code that you want to port over to Android. For more information about using the NDK, read the docs in the [docs/](#) directory of the [NDK download](#).



Property Animation

In this document

- [How Property Animation Works](#)
- [Animating with ValueAnimator](#)
- [Animating with ObjectAnimator](#)
- [Choreographing Multiple Animations with AnimatorSet](#)
- [Animation Listeners](#)
- [Using a TypeEvaluator](#)
- [Using Interpolators](#)
- [Specifying Keyframes](#)
- [Animating Layout Changes to ViewGroups](#)
- [Animating Views
 - \[ViewPropertyAnimator\]\(#\)](#)
- [Declaring Animations in XML](#)
- [Implications for UI performance](#)

Key classes

- [ValueAnimator](#)
- [ObjectAnimator](#)
- [TypeEvaluator](#)

Related samples

- [API Demos](#)

The property animation system is a robust framework that allows you to animate almost anything. You can define an animation to change any object property over time, regardless of whether it draws to the screen or not. A property animation changes a property's (a field in an object) value over a specified length of time. To animate something, you specify the object property that you want to animate, such as an object's position on the screen, how long you want to animate it for, and what values you want to animate between.

The property animation system lets you define the following characteristics of an animation:

- Duration: You can specify the duration of an animation. The default length is 300 ms.
- Time interpolation: You can specify how the values for the property are calculated as a function of the animation's current elapsed time.
- Repeat count and behavior: You can specify whether or not to have an animation repeat when it reaches the end of a duration and how many times to repeat the animation. You can also specify whether you want the animation to play back in reverse. Setting it to reverse plays the animation forwards then backwards repeatedly, until the number of repeats is reached.
- Animator sets: You can group animations into logical sets that play together or sequentially or after specified delays.
- Frame refresh delay: You can specify how often to refresh frames of your animation. The default is set to refresh every 10 ms, but the speed in which your application can refresh frames is ultimately dependent on how busy the system is overall and how fast the system can service the underlying timer.

How Property Animation Works

First, let's go over how an animation works with a simple example. Figure 1 depicts a hypothetical object that is animated with its `x` property, which represents its horizontal location on a screen. The duration of the animation is set to 40 ms and the distance to travel is 40 pixels. Every 10 ms, which is the default frame refresh rate, the object moves horizontally by 10 pixels. At the end of 40ms, the animation stops, and the object ends at horizontal position 40. This is an example of an animation with linear interpolation, meaning the object moves at a constant speed.

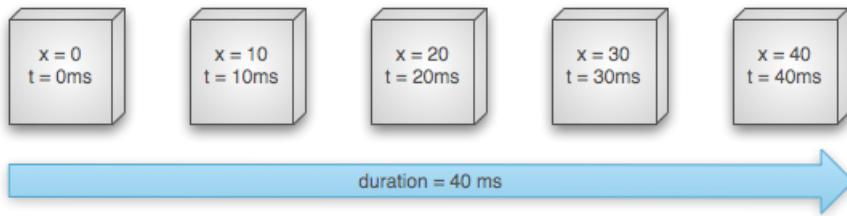


Figure 1. Example of a linear animation

You can also specify animations to have a non-linear interpolation. Figure 2 illustrates a hypothetical object that accelerates at the beginning of the animation, and decelerates at the end of the animation. The object still moves 40 pixels in 40 ms, but non-linearly. In the beginning, this animation accelerates up to the halfway point then decelerates from the halfway point until the end of the animation. As Figure 2 shows, the distance traveled at the beginning and end of the animation is less than in the middle.

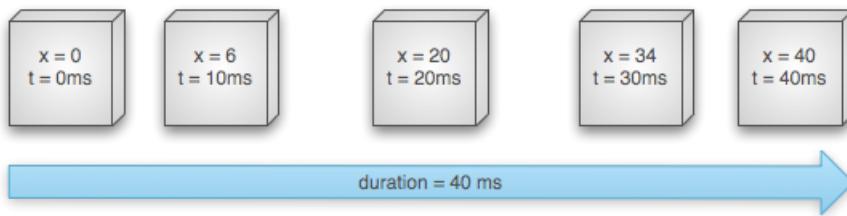


Figure 2. Example of a non-linear animation

Let's take a detailed look at how the important components of the property animation system would calculate animations like the ones illustrated above. Figure 3 depicts how the main classes work with one another.



Figure 3. How animations are calculated

The `ValueAnimator` object keeps track of your animation's timing, such as how long the animation has been running, and the current value of the property that it is animating.

The `ValueAnimator` encapsulates a `TimeInterpolator`, which defines animation interpolation, and a `TypeEvaluator`, which defines how to calculate values for the property being animated. For example, in Figure 2, the `TimeInterpolator` used would be `AccelerateDecelerateInterpolator` and the `TypeEvaluator` would be `IntEvaluator`.

To start an animation, create a `ValueAnimator` and give it the starting and ending values for the property that you want to animate, along with the duration of the animation. When you call `start()` the animation begins. During the whole animation, the `ValueAnimator` calculates an *elapsed fraction* between 0 and 1, based on the duration of the animation and how much time has elapsed. The elapsed fraction represents the percentage of time that the animation has completed, 0 meaning 0% and 1 meaning 100%. For example, in Figure 1, the elapsed fraction at $t = 10$ ms would be .25 because the total duration is $t = 40$ ms.

When the `ValueAnimator` is done calculating an elapsed fraction, it calls the `TimeInterpolator` that is currently set, to calculate an

interpolated fraction. An interpolated fraction maps the elapsed fraction to a new fraction that takes into account the time interpolation that is set. For example, in Figure 2, because the animation slowly accelerates, the interpolated fraction, about .15, is less than the elapsed fraction, .25, at t = 10 ms. In Figure 1, the interpolated fraction is always the same as the elapsed fraction.

When the interpolated fraction is calculated, [ValueAnimator](#) calls the appropriate [TypeEvaluator](#), to calculate the value of the property that you are animating, based on the interpolated fraction, the starting value, and the ending value of the animation. For example, in Figure 2, the interpolated fraction was .15 at t = 10 ms, so the value for the property at that time would be .15 X (40 - 0), or 6.

The `com.example.android.apis.animation` package in the [API Demos](#) sample project provides many examples on how to use the property animation system.

How Property Animation Differs from View Animation

The view animation system provides the capability to only animate [View](#) objects, so if you wanted to animate non-[View](#) objects, you have to implement your own code to do so. The view animation system is also constrained in the fact that it only exposes a few aspects of a [View](#) object to animate, such as the scaling and rotation of a View but not the background color, for instance.

Another disadvantage of the view animation system is that it only modified where the View was drawn, and not the actual View itself. For instance, if you animated a button to move across the screen, the button draws correctly, but the actual location where you can click the button does not change, so you have to implement your own logic to handle this.

With the property animation system, these constraints are completely removed, and you can animate any property of any object (Views and non-Views) and the object itself is actually modified. The property animation system is also more robust in the way it carries out animation. At a high level, you assign animators to the properties that you want to animate, such as color, position, or size and can define aspects of the animation such as interpolation and synchronization of multiple animators.

The view animation system, however, takes less time to setup and requires less code to write. If view animation accomplishes everything that you need to do, or if your existing code already works the way you want, there is no need to use the property animation system. It also might make sense to use both animation systems for different situations if the use case arises.

API Overview

You can find most of the property animation system's APIs in [android.animation](#). Because the view animation system already defines many interpolators in [android.view.animation](#), you can use those interpolators in the property animation system as well. The following tables describe the main components of the property animation system.

The [Animator](#) class provides the basic structure for creating animations. You normally do not use this class directly as it only provides minimal functionality that must be extended to fully support animating values. The following subclasses extend [Animator](#):

Table 1. Animators

Class	Description
ValueAnimator	The main timing engine for property animation that also computes the values for the property to be animated. It has all of the core functionality that calculates animation values and contains the timing details of each animation, information about whether an animation repeats, listeners that receive update events, and the ability to set custom types to evaluate. There are two pieces to animating properties: calculating the animated values and setting those values on the object and property that is being animated. ValueAnimator does not carry out the second piece, so you must listen for updates to values calculated by the ValueAnimator and modify the objects that you want to animate with your own logic. See the section about Animating with ValueAnimator for more information.
ObjectAnimator	A subclass of ValueAnimator that allows you to set a target object and object property to animate. This class updates the property accordingly when it computes a new value for the animation. You want to use ObjectAnimator most of the time, because it makes the process of animating values on target objects much easier. However, you sometimes want to use ValueAnimator directly because ObjectAnimator has a few more restrictions, such as requiring specific accessor methods to be present on the target object.
AnimatorSet	Provides a mechanism to group animations together so that they run in relation to one another. You can set animations to play together, sequentially, or after a specified delay. See the section about Choreographing multiple animations with Animator Sets for more information.

Evaluators tell the property animation system how to calculate values for a given property. They take the timing data that is provided by an [Animator](#) class, the animation's start and end value, and calculate the animated values of the property based on this data. The property animation system provides the following evaluators:

Table 2. Evaluators

Class/Interface	Description
IntEvaluator	The default evaluator to calculate values for <code>int</code> properties.
FloatEvaluator	The default evaluator to calculate values for <code>float</code> properties.
ArgbEvaluator	The default evaluator to calculate values for color properties that are represented as hexadecimal values.
TypeEvaluator	An interface that allows you to create your own evaluator. If you are animating an object property that is <i>not</i> an <code>int</code> , <code>float</code> , or color, you must implement the TypeEvaluator interface to specify how to compute the object property's animated values. You can also specify a custom TypeEvaluator for <code>int</code> , <code>float</code> , and color values as well, if you want to process those types differently than the default behavior. See the section about Using a TypeEvaluator for more information on how to write a custom evaluator.

A time interpolator defines how specific values in an animation are calculated as a function of time. For example, you can specify animations to happen linearly across the whole animation, meaning the animation moves evenly the entire time, or you can specify animations to use non-linear time, for example, accelerating at the beginning and decelerating at the end of the animation. Table 3 describes the interpolators that are contained in [android.view.animation](#). If none of the provided interpolators suits your needs, implement the [TimeInterpolator](#) interface and create your own. See [Using interpolators](#) for more information on how to write a custom interpolator.

Table 3. Interpolators

Class/Interface	Description
AccelerateDecelerateInterpolator	An interpolator whose rate of change starts and ends slowly but accelerates through the middle.
AccelerateInterpolator	An interpolator whose rate of change starts out slowly and then accelerates.
AnticipateInterpolator	An interpolator whose change starts backward then flings forward.
AnticipateOvershootInterpolator	An interpolator whose change starts backward, flings forward and overshoots the target value, then finally goes back to the final value.
BounceInterpolator	An interpolator whose change bounces at the end.
CycleInterpolator	An interpolator whose animation repeats for a specified number of cycles.
DecelerateInterpolator	An interpolator whose rate of change starts out quickly and then decelerates.
LinearInterpolator	An interpolator whose rate of change is constant.
OvershootInterpolator	An interpolator whose change flings forward and overshoots the last value then comes back.
TimeInterpolator	An interface that allows you to implement your own interpolator.

Animating with ValueAnimator

The [ValueAnimator](#) class lets you animate values of some type for the duration of an animation by specifying a set of `int`, `float`, or color values to animate through. You obtain a [ValueAnimator](#) by calling one of its factory methods: `ofInt()`, `ofFloat()`, or `ofObject()`. For example:

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 100f);
animation.setDuration(1000);
animation.start();
```

In this code, the [ValueAnimator](#) starts calculating the values of the animation, between 0 and 100, for a duration of 1000 ms, when the `start()` method runs.

You can also specify a custom type to animate by doing the following:

```
ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(), startPropertyValue, endPropertyValue);
animation.setDuration(1000);
animation.start();
```

In this code, the `ValueAnimator` starts calculating the values of the animation, between `startPropertyValue` and `endPropertyValue` using the logic supplied by `MyTypeEvaluator` for a duration of 1000 ms, when the `start()` method runs.

You can use the values of the animation by adding an `AnimatorUpdateListener` to the `ValueAnimator` object, as shown in the following code:

```
animation.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation) {
        // You can use the animated value in a property that uses the
        // same type as the animation. In this case, you can use the
        // float value in the translationX property.
        float animatedValue = (float)updatedAnimation.getAnimatedValue();
        textView.setTranslationX(animatedValue);
    }
});
```

In the `onAnimationUpdate()` method you can access the updated animation value and use it in a property of one of your views. For more information on listeners, see the section about [Animation Listeners](#)

Animating with ObjectAnimator

The `ObjectAnimator` is a subclass of the `ValueAnimator` (discussed in the previous section) and combines the timing engine and value computation of `ValueAnimator` with the ability to animate a named property of a target object. This makes animating any object much easier, as you no longer need to implement the `ValueAnimator.AnimatorUpdateListener`, because the animated property updates automatically.

Instantiating an `ObjectAnimator` is similar to a `ValueAnimator`, but you also specify the object and the name of that object's property (as a String) along with the values to animate between:

```
ObjectAnimator animation = ObjectAnimator.ofFloat(textView, "translationX", 100f);
animation.setDuration(1000);
animation.start();
```

To have the `ObjectAnimator` update properties correctly, you must do the following:

- The object property that you are animating must have a setter function (in camel case) in the form of `set<PropertyName>()`. Because the `ObjectAnimator` automatically updates the property during animation, it must be able to access the property with this setter method. For example, if the property name is `foo`, you need to have a `setFoo()` method. If this setter method does not exist, you have three options:
 - Add the setter method to the class if you have the rights to do so.
 - Use a wrapper class that you have rights to change and have that wrapper receive the value with a valid setter method and forward it to the original object.
 - Use `ValueAnimator` instead.
- If you specify only one value for the `values...` parameter in one of the `ObjectAnimator` factory methods, it is assumed to be the ending value of the animation. Therefore, the object property that you are animating must have a getter function that is used to obtain the starting value of the animation. The getter function must be in the form of `get<PropertyName>()`. For example, if the property name is `foo`, you need to have a `getFoo()` method.
- The getter (if needed) and setter methods of the property that you are animating must operate on the same type as the starting and ending values that you specify to `ObjectAnimator`. For example, you must have `targetObject.setPropName(float)` and

`targetObject.getPropName(float)` if you construct the following `ObjectAnimator`:

```
ObjectAnimator.ofFloat(targetObject, "propName", 1f)
```

- Depending on what property or object you are animating, you might need to call the `invalidate()` method on a View to force the screen to redraw itself with the updated animated values. You do this in the `onAnimationUpdate()` callback. For example, animating the color property of a Drawable object only causes updates to the screen when that object redraws itself. All of the property setters on View, such as `setAlpha()` and `setTranslationX()` invalidate the View properly, so you do not need to invalidate the View when calling these methods with new values. For more information on listeners, see the section about [Animation Listeners](#).

Choreographing Multiple Animations with AnimatorSet

In many cases, you want to play an animation that depends on when another animation starts or finishes. The Android system lets you bundle animations together into an `AnimatorSet`, so that you can specify whether to start animations simultaneously, sequentially, or after a specified delay. You can also nest `AnimatorSet` objects within each other.

The following sample code taken from the [Bouncing Balls](#) sample (modified for simplicity) plays the following `Animator` objects in the following manner:

- Plays `bounceAnim`.
- Plays `squashAnim1`, `squashAnim2`, `stretchAnim1`, and `stretchAnim2` at the same time.
- Plays `bounceBackAnim`.
- Plays `fadeAnim`.

```
AnimatorSet bouncer = new AnimatorSet();
bouncer.play(bounceAnim).before(squashAnim1);
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
bouncer.play(bounceBackAnim).after(stretchAnim2);
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(bouncer).before(fadeAnim);
animatorSet.start();
```

For a more complete example on how to use animator sets, see the [Bouncing Balls](#) sample in APIDemos.

Animation Listeners

You can listen for important events during an animation's duration with the listeners described below.

- `Animator.AnimatorListener`**
 - `onAnimationStart()` - Called when the animation starts.
 - `onAnimationEnd()` - Called when the animation ends.
 - `onAnimationRepeat()` - Called when the animation repeats itself.
 - `onAnimationCancel()` - Called when the animation is canceled. A cancelled animation also calls `onAnimationEnd()`, regardless of how they were ended.
- `ValueAnimator.AnimatorUpdateListener`**
 - `onAnimationUpdate()` - called on every frame of the animation. Listen to this event to use the calculated values generated by `ValueAnimator` during an animation. To use the value, query the `ValueAnimator` object passed into the event to get the current animated value with the `getAnimatedValue()` method. Implementing this listener is required if you use `ValueAnimator`.

Depending on what property or object you are animating, you might need to call `invalidate()` on a View to force that area of the screen to redraw itself with the new animated values. For example, animating the color property of a Drawable object only cause

updates to the screen when that object redraws itself. All of the property setters on View, such as `setAlpha()` and `setTranslationX()` invalidate the View properly, so you do not need to invalidate the View when calling these methods with new values.

You can extend the `AnimatorListenerAdapter` class instead of implementing the `Animator.AnimatorListener` interface, if you do not want to implement all of the methods of the `Animator.AnimatorListener` interface. The `AnimatorListenerAdapter` class provides empty implementations of the methods that you can choose to override.

For example, the [Bouncing Balls](#) sample in the API demos creates an `AnimatorListenerAdapter` for just the `onAnimationEnd()` callback:

```
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
fadeAnim.addListener(new AnimatorListenerAdapter() {
    public void onAnimationEnd(Animator animation) {
        balls.remove(((ObjectAnimator)animation).getTarget());
    }
})
```

Animating Layout Changes to ViewGroups

The property animation system provides the capability to animate changes to ViewGroup objects as well as provide an easy way to animate View objects themselves.

You can animate layout changes within a ViewGroup with the `LayoutTransition` class. Views inside a ViewGroup can go through an appearing and disappearing animation when you add them to or remove them from a ViewGroup or when you call a View's `setVisibility()` method with `VISIBLE`, `INVISIBLE`, or `GONE`. The remaining Views in the ViewGroup can also animate into their new positions when you add or remove Views. You can define the following animations in a `LayoutTransition` object by calling `setAnimator()` and passing in an `Animator` object with one of the following `LayoutTransition` constants:

- **APPEARING** - A flag indicating the animation that runs on items that are appearing in the container.
- **CHANGE_APPEARING** - A flag indicating the animation that runs on items that are changing due to a new item appearing in the container.
- **DISAPPEARING** - A flag indicating the animation that runs on items that are disappearing from the container.
- **CHANGE_DISAPPEARING** - A flag indicating the animation that runs on items that are changing due to an item disappearing from the container.

You can define your own custom animations for these four types of events to customize the look of your layout transitions or just tell the animation system to use the default animations.

The [LayoutAnimations](#) sample in API Demos shows you how to define animations for layout transitions and then set the animations on the View objects that you want to animate.

The `LayoutAnimationsByDefault` and its corresponding `layout_animations_by_default.xml` layout resource file show you how to enable the default layout transitions for ViewGroups in XML. The only thing that you need to do is to set the `android:animateLayoutChanges` attribute to `true` for the ViewGroup. For example:

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@+id/verticalContainer"
    android:animateLayoutChanges="true" />
```

Setting this attribute to true automatically animates Views that are added or removed from the ViewGroup as well as the remaining Views in the ViewGroup.

Using a TypeEvaluator

If you want to animate a type that is unknown to the Android system, you can create your own evaluator by implementing the `TypeEvaluator` interface. The types that are known by the Android system are `int`, `float`, or a color, which are supported by the `IntEvaluator`,

`FloatEvaluator`, and `ArgbEvaluator` type evaluators.

There is only one method to implement in the `TypeEvaluator` interface, the `evaluate()` method. This allows the animator that you are using to return an appropriate value for your animated property at the current point of the animation. The `FloatEvaluator` class demonstrates how to do this:

```
public class FloatEvaluator implements TypeEvaluator {  
  
    public Object evaluate(float fraction, Object startValue, Object endValue) {  
        float startFloat = ((Number) startValue).floatValue();  
        return startFloat + fraction * (((Number) endValue).floatValue() - startFloat);  
    }  
}
```

Note: When `ValueAnimator` (or `ObjectAnimator`) runs, it calculates a current elapsed fraction of the animation (a value between 0 and 1) and then calculates an interpolated version of that depending on what interpolator that you are using. The interpolated fraction is what your `TypeEvaluator` receives through the `fraction` parameter, so you do not have to take into account the interpolator when calculating animated values.

Using Interpolators

An interpolator define how specific values in an animation are calculated as a function of time. For example, you can specify animations to happen linearly across the whole animation, meaning the animation moves evenly the entire time, or you can specify animations to use non-linear time, for example, using acceleration or deceleration at the beginning or end of the animation.

Interpolators in the animation system receive a fraction from Animators that represent the elapsed time of the animation. Interpolators modify this fraction to coincide with the type of animation that it aims to provide. The Android system provides a set of common interpolators in the `android.view.animation package`. If none of these suit your needs, you can implement the `TimeInterpolator` interface and create your own.

As an example, how the default interpolator `AccelerateDecelerateInterpolator` and the `LinearInterpolator` calculate interpolated fractions are compared below. The `LinearInterpolator` has no effect on the elapsed fraction. The `AccelerateDecelerateInterpolator` accelerates into the animation and decelerates out of it. The following methods define the logic for these interpolators:

AccelerateDecelerateInterpolator

```
public float getInterpolation(float input) {  
    return (float)(Math.cos((input + 1) * Math.PI) / 2.0f) + 0.5f;  
}
```

LinearInterpolator

```
public float getInterpolation(float input) {  
    return input;  
}
```

The following table represents the approximate values that are calculated by these interpolators for an animation that lasts 1000ms:

ms elapsed	Elapsed fraction/Interpolated fraction (Linear)	Interpolated fraction (Accelerate/Decelerate)
0	0	0
200	.2	.1
400	.4	.345
600	.6	.8
800	.8	.9
1000	1	1

As the table shows, the `LinearInterpolator` changes the values at the same speed, .2 for every 200ms that passes. The

`AccelerateDecelerateInterpolator` changes the values faster than `LinearInterpolator` between 200ms and 600ms and slower between 600ms and 1000ms.

Specifying Keyframes

A `Keyframe` object consists of a time/value pair that lets you define a specific state at a specific time of an animation. Each keyframe can also have its own interpolator to control the behavior of the animation in the interval between the previous keyframe's time and the time of this keyframe.

To instantiate a `Keyframe` object, you must use one of the factory methods, `ofInt()`, `ofFloat()`, or `ofObject()` to obtain the appropriate type of `Keyframe`. You then call the `ofKeyframe()` factory method to obtain a `PropertyValuesHolder` object. Once you have the object, you can obtain an animator by passing in the `PropertyValuesHolder` object and the object to animate. The following code snippet demonstrates how to do this:

```
Keyframe kf0 = Keyframe.ofFloat(0f, 0f);
Keyframe kf1 = Keyframe.ofFloat(.5f, 360f);
Keyframe kf2 = Keyframe.ofFloat(1f, 0f);
PropertyValuesHolder pvhRotation = PropertyValuesHolder.ofKeyframe("rotation", kf0, kf1, kf2);
ObjectAnimator rotationAnim = ObjectAnimator.ofPropertyValuesHolder(target, pvhRotation)
rotationAnim.setDuration(5000ms);
```

For a more complete example on how to use keyframes, see the [MultiPropertyAnimation](#) sample in APIDemos.

Animating Views

The property animation system allow streamlined animation of View objects and offers a few advantages over the view animation system. The view animation system transformed View objects by changing the way that they were drawn. This was handled in the container of each View, because the View itself had no properties to manipulate. This resulted in the View being animated, but caused no change in the View object itself. This led to behavior such as an object still existing in its original location, even though it was drawn on a different location on the screen. In Android 3.0, new properties and the corresponding getter and setter methods were added to eliminate this drawback.

The property animation system can animate Views on the screen by changing the actual properties in the View objects. In addition, Views also automatically call the `invalidate()` method to refresh the screen whenever its properties are changed. The new properties in the `View` class that facilitate property animations are:

- `translationX` and `translationY`: These properties control where the View is located as a delta from its left and top coordinates which are set by its layout container.
- `rotation`, `rotationX`, and `rotationY`: These properties control the rotation in 2D (`rotation` property) and 3D around the pivot point.
- `scaleX` and `scaleY`: These properties control the 2D scaling of a View around its pivot point.
- `pivotX` and `pivotY`: These properties control the location of the pivot point, around which the rotation and scaling transforms occur. By default, the pivot point is located at the center of the object.
- `x` and `y`: These are simple utility properties to describe the final location of the View in its container, as a sum of the left and top values and `translationX` and `translationY` values.
- `alpha`: Represents the alpha transparency on the View. This value is 1 (opaque) by default, with a value of 0 representing full transparency (not visible).

To animate a property of a View object, such as its color or rotation value, all you need to do is create a property animator and specify the View property that you want to animate. For example:

```
ObjectAnimator.ofFloat(myView, "rotation", 0f, 360f);
```

For more information on creating animators, see the sections on animating with `ValueAnimator` and `ObjectAnimator`.

Animating with ViewPropertyAnimator

The [ViewPropertyAnimator](#) provides a simple way to animate several properties of a [View](#) in parallel, using a single underlying [Animator](#) object. It behaves much like an [ObjectAnimator](#), because it modifies the actual values of the view's properties, but is more efficient when animating many properties at once. In addition, the code for using the [ViewPropertyAnimator](#) is much more concise and easier to read. The following code snippets show the differences in using multiple [ObjectAnimator](#) objects, a single [ObjectAnimator](#), and the [ViewPropertyAnimator](#) when simultaneously animating the `x` and `y` property of a view.

Multiple ObjectAnimator objects

```
ObjectAnimator animX = ObjectAnimator.ofFloat(myView, "x", 50f);
ObjectAnimator animY = ObjectAnimator.ofFloat(myView, "y", 100f);
AnimatorSet animSetXY = new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

One ObjectAnimator

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvhY).start();
```

ViewPropertyAnimator

```
myView.animate().x(50f).y(100f);
```

For more detailed information about [ViewPropertyAnimator](#), see the corresponding Android Developers [blog post](#).

Declaring Animations in XML

The property animation system lets you declare property animations with XML instead of doing it programmatically. By defining your animations in XML, you can easily reuse your animations in multiple activities and more easily edit the animation sequence.

To distinguish animation files that use the new property animation APIs from those that use the legacy [view animation](#) framework, starting with Android 3.1, you should save the XML files for property animations in the `res/Animator/` directory.

The following property animation classes have XML declaration support with the following XML tags:

- [ValueAnimator](#) - `<animator>`
- [ObjectAnimator](#) - `<objectAnimator>`
- [AnimatorSet](#) - `<set>`

To find the attributes that you can use in your XML declaration, see [Animation Resources](#). The following example plays the two sets of object animations sequentially, with the first nested set playing two object animations together:

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

In order to run this animation, you must inflate the XML resources in your code to an `AnimatorSet` object, and then set the target objects for all of the animations before starting the animation set. Calling `setTarget()` sets a single target object for all children of the `AnimatorSet` as a convenience. The following code shows how to do this:

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,
    R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

You can also declare a `ValueAnimator` in XML, as shown in the following example:

```
<animator xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:valueType="floatType"
    android:valueFrom="0f"
    android:valueTo="-100f" />
```

To use the previous `ValueAnimator` in your code, you must inflate the object, add an `AnimatorUpdateListener`, get the updated animation value, and use it in a property of one of your views, as shown in the following code:

```
ValueAnimator xmlAnimator = (ValueAnimator) AnimatorInflater.loadAnimator(this,
    R.animator.animator);
xmlAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation) {
        float animatedValue = (float)updatedAnimation.getAnimatedValue();
        textView.setTranslationX(animatedValue);
    }
});
xmlAnimator.start();
```

For information about the XML syntax for defining property animations, see [Animation Resources](#).

Implications for UI performance

Animators that update the UI cause extra rendering work for every frame in which the animation runs. For this reason, using resource intensive animations can negatively impact the performance of your app.

Work required to animate your UI is added to the `animation stage` of the rendering pipeline. You can find out if your animations impact the performance of your app by enabling **Profile GPU Rendering** and monitoring the animation stage. For more information, see [Profile GPU Rendering Walkthrough](#).



View Animation

You can use the view animation system to perform tweened animation on Views. Tween animation calculates the animation with information such as the start point, end point, size, rotation, and other common aspects of an animation.

A tween animation can perform a series of simple transformations (position, size, rotation, and transparency) on the contents of a View object. So, if you have a [TextView](#) object, you can move, rotate, grow, or shrink the text. If it has a background image, the background image will be transformed along with the text. The [animation package](#) provides all the classes used in a tween animation.

A sequence of animation instructions defines the tween animation, defined by either XML or Android code. As with defining a layout, an XML file is recommended because it's more readable, reusable, and swappable than hard-coding the animation. In the example below, we use XML. (To learn more about defining an animation in your application code, instead of XML, refer to the [AnimationSet](#) class and other [Animation](#) subclasses.)

The animation instructions define the transformations that you want to occur, when they will occur, and how long they should take to apply. Transformations can be sequential or simultaneous - for example, you can have the contents of a TextView move from left to right, and then rotate 180 degrees, or you can have the text move and rotate simultaneously. Each transformation takes a set of parameters specific for that transformation (starting size and ending size for size change, starting angle and ending angle for rotation, and so on), and also a set of common parameters (for instance, start time and duration). To make several transformations happen simultaneously, give them the same start time; to make them sequential, calculate the start time plus the duration of the preceding transformation.

The animation XML file belongs in the `res/anim/` directory of your Android project. The file must have a single root element: this will be either a single `<alpha>`, `<scale>`, `<translate>`, `<rotate>`, interpolator element, or `<set>` element that holds groups of these elements (which may include another `<set>`). By default, all animation instructions are applied simultaneously. To make them occur sequentially, you must specify the `startOffset` attribute, as shown in the example below.

The following XML from one of the ApiDemos is used to stretch, then simultaneously spin and rotate a View object.

```

<set android:shareInterpolator="false">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
    <set android:interpolator="@android:anim/decelerate_interpolator">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="700"
            android:duration="400"
            android:fillBefore="false" />
        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="700"
            android:duration="400" />
    </set>
</set>

```

Screen coordinates (not used in this example) are (0,0) at the upper left hand corner, and increase as you go down and to the right.

Some values, such as pivotX, can be specified relative to the object itself or relative to the parent. Be sure to use the proper format for what you want ("50" for 50% relative to the parent, or "50%" for 50% relative to itself).

You can determine how a transformation is applied over time by assigning an [Interpolator](#). Android includes several Interpolator subclasses that specify various speed curves: for instance, [AccelerateInterpolator](#) tells a transformation to start slow and speed up. Each one has an attribute value that can be applied in the XML.

With this XML saved as `hyperspace_jump.xml` in the `res/anim/` directory of the project, the following code will reference it and apply it to an [ImageView](#) object from the layout.

```

ImageView spaceshipImage = (ImageView) findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation = AnimationUtils.loadAnimation(this, R.anim.hyperspace_jump);
spaceshipImage.startAnimation(hyperspaceJumpAnimation);

```

As an alternative to `startAnimation()`, you can define a starting time for the animation with `Animation.setStartTime()`, then assign the animation to the View with `View.setAnimation()`.

For more information on the XML syntax, available tags and attributes, see [Animation Resources](#).

Note: Regardless of how your animation may move or resize, the bounds of the View that holds your animation will not automatically adjust to accommodate it. Even so, the animation will still be drawn beyond the bounds of its View and will not be clipped. However, clipping *will occur* if the animation exceeds the bounds of the parent View.



Drawable Animation

Drawable animation lets you load a series of Drawable resources one after another to create an animation. This is a traditional animation in the sense that it is created with a sequence of different images, played in order, like a roll of film. The [AnimationDrawable](#) class is the basis for Drawable animations.

While you can define the frames of an animation in your code, using the [AnimationDrawable](#) class API, it's more simply accomplished with a single XML file that lists the frames that compose the animation. The XML file for this kind of animation belongs in the `res/drawable/` directory of your Android project. In this case, the instructions are the order and duration for each frame of the animation.

The XML file consists of an `<animation-list>` element as the root node and a series of child `<item>` nodes that each define a frame: a drawable resource for the frame and the frame duration. Here's an example XML file for a Drawable animation:

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

This animation runs for just three frames. By setting the `android:oneshot` attribute of the list to `true`, it will cycle just once then stop and hold on the last frame. If it is set `false` then the animation will loop. With this XML saved as `rocket_thrust.xml` in the `res/drawable/` directory of the project, it can be added as the background image to a View and then called to play. Here's an example Activity, in which the animation is added to an [ImageView](#) and then animated when the screen is touched:

```
AnimationDrawable rocketAnimation;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
    rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
    rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        rocketAnimation.start();
        return true;
    }
    return super.onTouchEvent(event);
}
```

It's important to note that the `start()` method called on the `AnimationDrawable` cannot be called during the `onCreate()` method of your Activity, because the `AnimationDrawable` is not yet fully attached to the window. If you want to play the animation immediately, without requiring interaction, then you might want to call it from the `onWindowFocusChanged()` method in your Activity, which will get called when Android brings your window into focus.

For more information on the XML syntax, available tags and attributes, see [Animation Resources](#).



Canvas and Drawables

In this document

- › [Draw with a canvas](#)
- › [Drawing on a view](#)
- › [Drawing on a SurfaceView](#)
- › [Drawables](#)
 - › [Creating drawables from resource images](#)
 - › [Creating drawables from XML resources](#)
 - › [Shape drawables](#)
 - › [NinePatch graphics](#)
 - › [Vector drawables](#)

See also

- › [OpenGL with the Framework APIs](#)
- › [RenderScript](#)

The Android framework provides a set of two-dimensional drawing APIs that allow you to render your own custom graphics onto a canvas or to modify existing views to customize their look and feel. You typically draw 2-D graphics in one of the following ways:

- a. Draw your graphics or animations on a [View](#) object in your layout. By using this option, the system's rendering pipeline handles your graphics—it's your responsibility to define the graphics inside the view.
- b. Draw your graphics in a [Canvas](#) object. To use this option, you pass your canvas to the appropriate class' [onDraw\(Canvas\)](#) method. You can also use the drawing methods in [Canvas](#). This option also puts you in control of any animation.

Drawing to a view is a good choice when you want to draw simple graphics that don't need to change dynamically and aren't part of a performance-intensive app, such as a game. For example, you should draw your graphics into a view when you want to display a static graphic or predefined animation, within an otherwise static app. For more information, read [Drawables](#).

Drawing to a canvas is better when your app needs to regularly redraw itself. Apps, such as video games, should draw to the canvas on their own. However, there's more than one way to do this:

- a. In your app's main thread, wherein you create a custom view component in your layout, call [invalidate\(\)](#) and then handle the [onDraw\(Canvas\)](#) callback.
- b. In a worker thread that manages a [SurfaceView](#), use drawing methods of the canvas. You don't need to call [invalidate\(\)](#).

Draw with a canvas

You can meet the requirements of an app that needs specialized drawing and/or control of the graphics animation by drawing to a canvas, which is represented by the [Canvas](#) class. A canvas serves as a pretense, or interface, to the actual surface upon which your graphics are drawn—you can perform your *draw* operations to the canvas. Via the canvas, your app draws to the underlying [Bitmap](#) object, which is placed into the window.

If you're drawing within the [onDraw\(Canvas\)](#) callback, the canvas is already provided and you only need to place your drawing calls upon it. If you're using a [SurfaceView](#) object, you can acquire a canvas from [lockCanvas\(\)](#). Both of these scenarios are discussed in the following sections.

If you need to create a new `Canvas` object, then you must define the underlying `Bitmap` object that is required to place the drawing into a window. The following code example shows how to set up a new canvas from a bitmap:

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

It's possible to use the bitmap in a different canvas by using one of the `drawBitmap()` methods. However, we recommend that you use a canvas provided by the `onDraw(Canvas)` callback or the `lockCanvas()` method. For more information, see [Drawing on a View](#) and [Drawing on a SurfaceView](#).

The `Canvas` class has its own set of drawing methods, including `drawBitmap()`, `drawRect()`, `drawText()`, and many more. Other classes that you might use also have `draw()` methods. For example, you probably have some `Drawable` objects that you want to put on the canvas. The `Drawable` class has its own `draw(Canvas)` method that takes your canvas as an argument.

Drawing on a view

If your app doesn't require a significant amount of processing or a high frame rate (for example a chess game, a snake game, or another slowly animated app), then you should consider [creating a custom view](#) and drawing with a canvas in the `View.onDraw(Canvas)` callback. The most convenient aspect of this is that the Android framework provides a predefined canvas that can perform your drawing operations.

To start, create a subclass of `View` and implement the `onDraw(Canvas)` callback, which the Android framework calls to draw the view. Then perform the draw operations through the `Canvas` object, which is provided by the framework.

The Android framework only calls `onDraw(Canvas)` when necessary. When it's time to redraw your app, you must invalidate the view by calling `invalidate()`. Calling `invalidate()` indicates that you'd like your view to be drawn. Android then calls your view's `onDraw(Canvas)` method, though the call isn't guaranteed to be instantaneous.

Inside your view's `onDraw(Canvas)` method, call drawing methods on the canvas or on other classes' methods that can take your canvas as an argument. Once your `onDraw()` finishes, the Android framework uses your canvas to draw a bitmap that is handled by the system.

Note: To invalidate a view from a thread other than the app's main thread, you must call `postInvalidate()` instead of `invalidate()`.

For information about extending the `View` class, read [Creating a view class](#).

Drawing on a SurfaceView

`SurfaceView` is a special subclass of `View` that offers a dedicated drawing surface within the view hierarchy. The goal is to offer this drawing surface to an app's worker thread. This way, the app isn't required to wait until the system's view hierarchy is ready to draw. Instead, a worker thread that has a reference to a `SurfaceView` object can draw to its own canvas at its own pace.

To begin, you need to create a new class that extends `SurfaceView`. This class should also implement the `SurfaceHolder.Callback` interface, which provides events that happen in the underlying `Surface` object, such as when it's created, changed, or destroyed. These events let you know when you can start drawing, whether you need to make adjustments based on new surface properties, and when to stop drawing and potentially terminate some tasks. The class that extends `SurfaceView` is also a good place to define your worker thread, which calls all the drawing procedures in your canvas.

Instead of handling the `Surface` object directly, you should handle it via a `SurfaceHolder`. After your `SurfaceView` object is initialized, you can get a `SurfaceHolder` object by calling `getHolder()`. You should register your `SurfaceView` object to receive notifications from the `SurfaceHolder` by calling `addCallback()`. Then implement each `SurfaceHolder.Callback` abstract method in your `SurfaceView` class.

You can draw to the surface canvas from a worker thread that has access to a `SurfaceHolder` object. Perform the following steps from inside the worker thread every time your app needs to redraw the surface:

1. Use `lockCanvas()` to retrieve the canvas.
2. Perform drawing operations on the canvas.
3. Unlock the canvas by calling `unlockCanvasAndPost(Canvas)` passing the `Canvas` object that you used for your drawing operations.

The surface draws the canvas considering all the drawing operations you performed on it.

Note: Every time you retrieve the canvas from the `SurfaceHolder`, the previous state of the canvas is retained. In order to properly animate your graphics, you must repaint the entire surface. For example, you can clear the previous state of the canvas by filling in a color using the `drawColor()` method or setting a background image using the `drawBitmap()` method. Otherwise, your canvas could show traces of previous drawings.

Drawables

The Android framework offers a custom 2-D graphics library for drawing shapes and images. The `android.graphics.drawable` package contains common classes used for drawing in two dimensions.

This section discusses the basics of using drawable objects to draw graphics and how to use a couple of subclasses of the `Drawable` class. For information on how to use drawables for frame-by-frame animation, see [Drawable Animation](#).

A `Drawable` is a general abstraction for *something that can be drawn*. The Android framework offers a set of `direct` and `indirect` subclasses of `Drawable` that you can use in a variety of scenarios. You can also extend these classes to define your own custom drawable objects that behave in unique ways.

There are two ways to define and instantiate a `Drawable` besides using the standard class constructors:

- a. Using an resource image saved in your project.
- b. Using an XML resource that defines the drawable properties.

Creating drawables from resource images

You can add graphics to your app by referencing an image file from your project resources. Supported file types are PNG (preferred), JPG (acceptable), and GIF (discouraged). App icons, logos, and other graphics, such as those used in games, are well suited for this technique.

To use an image resource, add your file to the `res/drawable/` directory of your project. Once in your project, you can reference the image resource from your code or your XML layout. Either way, it's referred to using a resource ID, which is the file name without the file type extension. For example, refer to `my_image.png` as `my_image`.

Note: Image resources placed in the `res/drawable/` directory may be automatically optimized with lossless image compression by the `aapt` tool during the build process. For example, a true-color PNG that doesn't require more than 256 colors may be converted to an 8-bit PNG with a color palette. This results in an image of equal quality but which requires less memory. As a result, the image binaries placed in this directory can change at build time. If you plan on reading an image as a bitstream in order to convert it to a bitmap, put your images in the `res/raw/` folder instead, where the `aapt` tool doesn't modify them.

The following code snippet demonstrates how to build an `ImageView` that uses an image created from a drawable resource and adds it to the layout:

```
LinearLayout mLinearLayout;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Create a LinearLayout in which to add the ImageView
    mLinearLayout = new LinearLayout(this);

    // Instantiate an ImageView and define its properties
    ImageView i = new ImageView(this);
    i.setImageResource(R.drawable.my_image);

    // set the ImageView bounds to match the Drawable's dimensions
    i.setAdjustViewBounds(true);
    i.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
        LayoutParams.WRAP_CONTENT));

    // Add the ImageView to the layout and set the layout as the content view
    mLinearLayout.addView(i);
    setContentView(mLinearLayout);
}
```

In other cases, you may want to handle your image resource as a `Drawable` object, as shown in the following example:

```
Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);
```

Note: Each unique resource in your project can maintain only one state, no matter how many different objects you instantiate for it. For example, if you instantiate two `Drawable` objects from the same image resource and change a property (such as the alpha) for one object, then it also affects the other. When dealing with multiple instances of an image resource, instead of directly transforming the `Drawable` object you should perform a [tween animation](#).

The XML snippet below shows how to add a drawable resource to an `ImageView` in the XML layout:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/my_image" />
```

For more information about using project resources, see [Resources and Assets](#).

Creating drawables from XML resources

If there is a `Drawable` object that you'd like to create, which isn't initially dependent on variables defined by your code or user interaction, then defining the `Drawable` in XML is a good option. Even if you expect your `Drawable` to change its properties during the user's interaction with your app, you should consider defining the object in XML, as you can modify properties after it's instantiated.

After you've defined your `Drawable` in XML, save the file in the `res/drawable/` directory of your project. The following example shows the XML that defines a `TransitionDrawable` resource, which inherits from `Drawable`:

```
<!-- res/drawable/expand_collapse.xml -->
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/image_expand">
    <item android:drawable="@drawable/imageCollapse">
</transition>
```

Then, retrieve and instantiate the object by calling `Resources.getDrawable()`, and passing the resource ID of your XML file. Any `Drawable` subclass that supports the `inflate()` method can be defined in XML and instantiated by your app. Each drawable class that supports XML inflation utilizes specific XML attributes that help define the object properties. The following code instantiates the `TransitionDrawable` and sets it as the content of an `ImageView` object:

```
Resources res = mContext.getResources();
TransitionDrawable transition =
    (TransitionDrawable) res.getDrawable(R.drawable.expand_collapse);

ImageView image = (ImageView) findViewById(R.id.toggle_image);
image.setImageDrawable(transition);

// Then you can call the TransitionDrawable object's methods
transition.startTransition(1000);
```

For more information about the XML attributes supported, refer to the classes listed above.

Shape drawables

A `ShapeDrawable` object can be a good option when you want to dynamically draw a two-dimensional graphic. You can programmatically draw primitive shapes on a `ShapeDrawable` object and apply the styles that your app needs.

`ShapeDrawable` is a subclass of `Drawable`. For this reason, you can use a `ShapeDrawable` wherever a `Drawable` is expected. For example, you can use a `ShapeDrawable` object to set the background of a view by passing it to the `setBackgroundDrawable()` method of the view. You can also draw your shape as its own custom view and add it to a layout in your app.

Because `ShapeDrawable` has its own `draw()` method, you can create a subclass of `View` that draws the `ShapeDrawable` object during the `onDraw()` event, as shown in the following code example:

```
public class CustomDrawableView extends View {
    private ShapeDrawable mDrawable;

    public CustomDrawableView(Context context) {
        super(context);

        int x = 10;
        int y = 10;
        int width = 300;
        int height = 50;

        mDrawable = new ShapeDrawable(new OvalShape());
        // If the color isn't set, the shape uses black as the default.
        mDrawable.getPaint().setColor(0xff74AC23);
        // If the bounds aren't set, the shape can't be drawn.
        mDrawable.setBounds(x, y, x + width, y + height);
    }

    protected void onDraw(Canvas canvas) {
        mDrawable.draw(canvas);
    }
}
```

You can use the `CustomDrawableView` class in the code sample above as you would use any other custom view. For example, you can programmatically add it to an activity in your app, as shown in the following example:

```
CustomDrawableView mCustomDrawableView;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mCustomDrawableView = new CustomDrawableView(this);

    setContentView(mCustomDrawableView);
}
```

If you want to use the custom view in the XML layout instead, then the `CustomDrawableView` class must override the `View(Context, AttributeSet)` constructor, which is called when the class is inflated from XML. The following example shows how to declare the `CustomDrawableView` in the XML layout:

```
<com.example.shapeddrawable.CustomDrawableView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

The `ShapeDrawable` class, like many other drawable types in the `android.graphics.drawable` package, allows you to define various properties of the object by using public methods. Some example properties you might want to adjust include alpha transparency, color filter, dither, opacity, and color.

You can also define primitive drawable shapes using XML resources. For more information, see [Shape Drawable](#) in [Drawable Resource Types](#).

NinePatch drawables

A `NinePatchDrawable` graphic is a stretchable bitmap image that you can use as the background of a view. Android automatically resizes the graphic to accommodate the contents of the view. An example use of a NinePatch image is the background used by standard Android buttons—buttons must stretch to accommodate strings of various lengths. A NinePatch graphic is a standard PNG image that includes an extra 1-pixel border. It must be saved with the `.9.png` extension in the `res/drawable/` directory of your project.

Use the border to define the stretchable and static areas of the image. You indicate a stretchable section by drawing one (or more) 1-pixel wide black line(s) in the left and top part of the border (the other border pixels should be fully transparent or white). You can have as many

stretchable sections as you want. The relative size of the stretchable sections stays the same, so the largest section always remains the largest.

You can also define an optional drawable section of the image (effectively, the padding lines) by drawing a line on the right and a line on the bottom. If a [View](#) object sets the NinePatch graphic as its background and then specifies the view's text, it stretches itself so that all the text occupies only the area designated by the right and bottom lines (if included). If the padding lines aren't included, Android uses the left and top lines to define this drawable area.

To clarify the difference between the lines, the left and top lines define which pixels of the image are allowed to be replicated in order to stretch the image. The bottom and right lines define the relative area within the image that the contents of the view are allowed to occupy.

Figure 1 shows an example of a NinePatch graphic used to define a button:

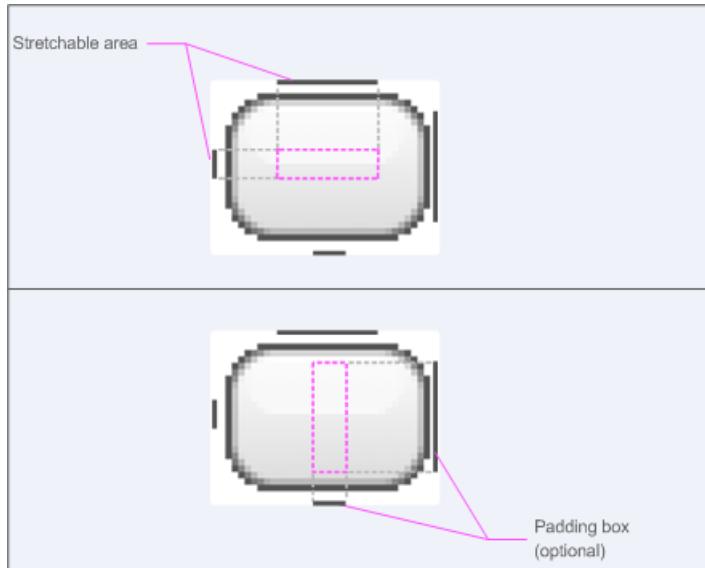


Figure 1: Example of a NinePatch graphic that defines a button

This NinePatch graphic defines one stretchable area with the left and top lines, and the drawable area with the bottom and right lines. In the top image, the dotted grey lines identify the regions of the image that are replicated in order to stretch the image. The pink rectangle in the bottom image identifies the region in which the contents of the view are allowed. If the contents don't fit in this region, then the image is stretched to make them fit.

The [Draw 9-patch](#) tool offers an extremely handy way to create your NinePatch images, using a WYSIWYG graphics editor. It even raises warnings if the region you've defined for the stretchable area is at risk of producing drawing artifacts as a result of the pixel replication.

The following sample layout XML demonstrates how to add a NinePatch graphic to a couple of buttons. The NinePatch image is saved to `res/drawable/my_button_background.9.png`.

```
<Button id="@+id/tiny"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerInParent="true"
    android:text="Tiny"
    android:textSize="8sp"
    android:background="@drawable/my_button_background"/>

<Button id="@+id/big"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerInParent="true"
    android:text="Biiiiig text!"
    android:textSize="30sp"
    android:background="@drawable/my_button_background"/>
```

Note that the `layout_width` and `layout_height` attributes are set to `wrap_content` to make the button fit neatly around the text.

Figure 2 shows the two buttons rendered from the XML and NinePatch image shown above. Notice how the width and height of the button

varies with the text, and the background image stretches to accommodate it.



Figure 2: Buttons rendered using an XML resource and a NinePatch graphic

Vector drawables

A vector drawable is a vector graphic defined in an XML file as a set of points, lines, and curves along with its associated color information. The Android framework provides the [VectorDrawable](#) and [AnimatedVectorDrawable](#) classes, which support vector graphics as drawable resources.

Apps that target Android versions lower than 5.0 (API level 21) can use the Support Library version 23.2 or higher to get support for vector drawables and animated vector drawables. For more information about using the vector drawable classes, see [Vector Drawable](#).

OpenGL ES

In this document

- › [The Basics](#)
- › [OpenGL ES packages](#)
- › [Declaring OpenGL Requirements](#)
- › [Mapping Coordinates for Drawn Objects](#)
- › [Projection and camera in ES 1.0](#)
- › [Projection and camera in ES 2.0 and higher](#)
- › [Shape Faces and Winding](#)
- › [OpenGL Versions and Device Compatibility](#)
 - › [Texture compression support](#)
 - › [Determining OpenGL extensions](#)
 - › [Checking OpenGL ES Version](#)
- › [Choosing an OpenGL API Version](#)

Key classes

- › [GLSurfaceView](#)
- › [GLSurfaceView.Renderer](#)

See also

- › [Displaying Graphics with OpenGL ES](#)
- › [OpenGL ES](#)
- › [OpenGL ES 1.x Specification](#)
- › [OpenGL ES 2.x specification](#)
- › [OpenGL ES 3.x specification](#)

Android includes support for high performance 2D and 3D graphics with the Open Graphics Library (OpenGL®), specifically, the OpenGL ES API. OpenGL is a cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware. OpenGL ES is a flavor of the OpenGL specification intended for embedded devices. Android supports several versions of the OpenGL ES API:

- OpenGL ES 1.0 and 1.1 - This API specification is supported by Android 1.0 and higher.
- OpenGL ES 2.0 - This API specification is supported by Android 2.2 (API level 8) and higher.
- OpenGL ES 3.0 - This API specification is supported by Android 4.3 (API level 18) and higher.
- OpenGL ES 3.1 - This API specification is supported by Android 5.0 (API level 21) and higher.

Caution: Support of the OpenGL ES 3.0 API on a device requires an implementation of this graphics pipeline provided by the device manufacturer. A device running Android 4.3 or higher *may not support* the OpenGL ES 3.0 API. For information on checking what version of OpenGL ES is supported at run time, see [Checking OpenGL ES Version](#).

Note: The specific API provided by the Android framework is similar to the J2ME JSR239 OpenGL ES API, but is not identical. If you are familiar with J2ME JSR239 specification, be alert for variations.

The Basics

Android supports OpenGL both through its framework API and the Native Development Kit (NDK). This topic focuses on the Android framework interfaces. For more information about the NDK, see the [Android NDK](#).

There are two foundational classes in the Android framework that let you create and manipulate graphics with the OpenGL ES API: [GLSurfaceView](#) and [GLSurfaceView.Renderer](#). If your goal is to use OpenGL in your Android application, understanding how to implement these classes in an activity should be your first objective.

[GLSurfaceView](#)

This class is a [View](#) where you can draw and manipulate objects using OpenGL API calls and is similar in function to a [SurfaceView](#).

You can use this class by creating an instance of [GLSurfaceView](#) and adding your [Renderer](#) to it. However, if you want to capture touch screen events, you should extend the [GLSurfaceView](#) class to implement the touch listeners, as shown in OpenGL training lesson, [Responding to Touch Events](#).

[GLSurfaceView.Renderer](#)

This interface defines the methods required for drawing graphics in a [GLSurfaceView](#). You must provide an implementation of this interface as a separate class and attach it to your [GLSurfaceView](#) instance using [GLSurfaceView.setRenderer\(\)](#).

The [GLSurfaceView.Renderer](#) interface requires that you implement the following methods:

- [onSurfaceCreated\(\)](#): The system calls this method once, when creating the [GLSurfaceView](#). Use this method to perform actions that need to happen only once, such as setting OpenGL environment parameters or initializing OpenGL graphic objects.
- [onDrawFrame\(\)](#): The system calls this method on each redraw of the [GLSurfaceView](#). Use this method as the primary execution point for drawing (and re-drawing) graphic objects.
- [onSurfaceChanged\(\)](#): The system calls this method when the [GLSurfaceView](#) geometry changes, including changes in size of the [GLSurfaceView](#) or orientation of the device screen. For example, the system calls this method when the device changes from portrait to landscape orientation. Use this method to respond to changes in the [GLSurfaceView](#) container.

OpenGL ES packages

Once you have established a container view for OpenGL ES using [GLSurfaceView](#) and [GLSurfaceView.Renderer](#), you can begin calling OpenGL APIs using the following classes:

- OpenGL ES 1.0/1.1 API Packages
 - [android.opengl](#) - This package provides a static interface to the OpenGL ES 1.0/1.1 classes and better performance than the [javax.microedition.khronos](#) package interfaces.
 - [GLES10](#)
 - [GLES10Ext](#)
 - [GLES11](#)
 - [GLES11Ext](#)
 - [javax.microedition.khronos.opengles](#) - This package provides the standard implementation of OpenGL ES 1.0/1.1.
 - [GL10](#)
 - [GL10Ext](#)
 - [GL11](#)
 - [GL11Ext](#)
 - [GL11ExtensionPack](#)
- OpenGL ES 2.0 API Class
 - [android.opengl.GLES20](#) - This package provides the interface to OpenGL ES 2.0 and is available starting with Android 2.2 (API level 8).

- OpenGL ES 3.0/3.1 API Packages
 - [android.opengl](#) - This package provides the interface to the OpenGL ES 3.0/3.1 classes. Version 3.0 is available starting with Android 4.3 (API level 18). Version 3.1 is available starting with Android 5.0 (API level 21).
 - [GLES30](#)
 - [GLES31](#)
 - [GLES31Ext \(Android Extension Pack\)](#)

If you want to start building an app with OpenGL ES right away, follow the [Displaying Graphics with OpenGL ES](#) class.

Declaring OpenGL Requirements

If your application uses OpenGL features that are not available on all devices, you must include these requirements in your [AndroidManifest.xml](#) file. Here are the most common OpenGL manifest declarations:

- **OpenGL ES version requirements** - If your application requires a specific version of OpenGL ES, you must declare that requirement by adding the following settings to your manifest as shown below.

For OpenGL ES 2.0:

```
<!-- Tell the system this app requires OpenGL ES 2.0. -->
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

Adding this declaration causes Google Play to restrict your application from being installed on devices that do not support OpenGL ES 2.0. If your application is exclusively for devices that support OpenGL ES 3.0, you can also specify this in your manifest:

For OpenGL ES 3.0:

```
<!-- Tell the system this app requires OpenGL ES 3.0. -->
<uses-feature android:glEsVersion="0x00030000" android:required="true" />
```

For OpenGL ES 3.1:

```
<!-- Tell the system this app requires OpenGL ES 3.1. -->
<uses-feature android:glEsVersion="0x00030001" android:required="true" />
```

Note: The OpenGL ES 3.x API is backwards-compatible with the 2.0 API, which means you can be more flexible with your implementation of OpenGL ES in your application. By declaring the OpenGL ES 2.0 API as a requirement in your manifest, you can use that API version as a default, check for the availability of the 3.x API at run time and then use OpenGL ES 3.x features if the device supports it. For more information about checking the OpenGL ES version supported by a device, see [Checking OpenGL ES Version](#).

- **Texture compression requirements** - If your application uses texture compression formats, you must declare the formats your application supports in your manifest file using [`<supports-gl-texture>`](#). For more information about available texture compression formats, see [Texture compression support](#).

Declaring texture compression requirements in your manifest hides your application from users with devices that do not support at least one of your declared compression types. For more information on how Google Play filtering works for texture compressions, see the [Google Play and texture compression filtering](#) section of the [`<supports-gl-texture>`](#) documentation.

Mapping Coordinates for Drawn Objects

One of the basic problems in displaying graphics on Android devices is that their screens can vary in size and shape. OpenGL assumes a square, uniform coordinate system and, by default, happily draws those coordinates onto your typically non-square screen as if it is perfectly square.

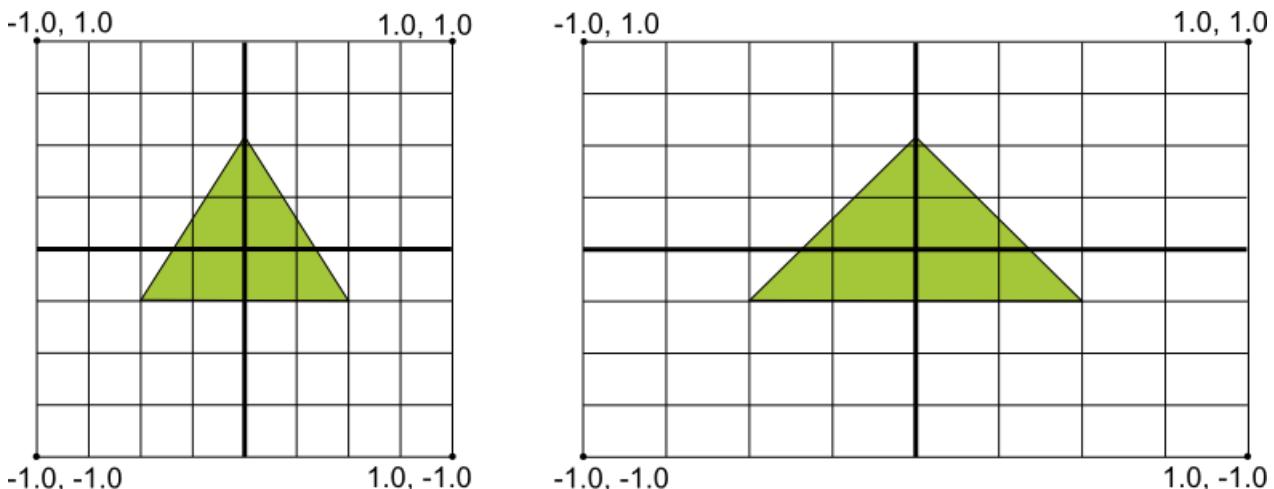


Figure 1. Default OpenGL coordinate system (left) mapped to a typical Android device screen (right).

The illustration above shows the uniform coordinate system assumed for an OpenGL frame on the left, and how these coordinates actually map to a typical device screen in landscape orientation on the right. To solve this problem, you can apply OpenGL projection modes and camera views to transform coordinates so your graphic objects have the correct proportions on any display.

In order to apply projection and camera views, you create a projection matrix and a camera view matrix and apply them to the OpenGL rendering pipeline. The projection matrix recalculates the coordinates of your graphics so that they map correctly to Android device screens. The camera view matrix creates a transformation that renders objects from a specific eye position.

Projection and camera view in OpenGL ES 1.0

In the ES 1.0 API, you apply projection and camera view by creating each matrix and then adding them to the OpenGL environment.

1. **Projection matrix** - Create a projection matrix using the geometry of the device screen in order to recalculate object coordinates so they are drawn with correct proportions. The following example code demonstrates how to modify the `onSurfaceChanged()` method of a `GLSurfaceView.Renderer` implementation to create a projection matrix based on the screen's aspect ratio and apply it to the OpenGL rendering environment.

```
public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);

    // make adjustments for screen ratio
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);           // set matrix to projection mode
    gl.glLoadIdentity();                          // reset the matrix to its default state
    gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7); // apply the projection matrix
}
```

2. **Camera transformation matrix** - Once you have adjusted the coordinate system using a projection matrix, you must also apply a camera view. The following example code shows how to modify the `onDrawFrame()` method of a `GLSurfaceView.Renderer` implementation to apply a model view and use the `GLU.gluLookAt()` utility to create a viewing transformation which simulates a camera position.

```
public void onDrawFrame(GL10 gl) {
    ...
    // Set GL_MODELVIEW transformation mode
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();                         // reset the matrix to its default state

    // When using GL_MODELVIEW, you must set the camera view
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 1.0f, 0.0f);
    ...
}
```

Projection and camera view in OpenGL ES 2.0 and higher

In the ES 2.0 and 3.0 APIs, you apply projection and camera view by first adding a matrix member to the vertex shaders of your graphics objects. With this matrix member added, you can then generate and apply projection and camera viewing matrices to your objects.

1. **Add matrix to vertex shaders** - Create a variable for the view projection matrix and include it as a multiplier of the shader's position. In the following example vertex shader code, the included `uMVPMatrix` member allows you to apply projection and camera viewing matrices to the coordinates of objects that use this shader.

```
private final String vertexShaderCode =  
  
    // This matrix member variable provides a hook to manipulate  
    // the coordinates of objects that use this vertex shader.  
    "uniform mat4 uMVPMatrix;  \n" +  
  
    "attribute vec4 vPosition;  \n" +  
    "void main(){  \n" +  
    // The matrix must be included as part of gl_Position  
    // Note that the uMVPMatrix factor *must be first* in order  
    // for the matrix multiplication product to be correct.  
    " gl_Position = uMVPMatrix * vPosition; \n" +  
  
    "}\n";
```

Note: The example above defines a single transformation matrix member in the vertex shader into which you apply a combined projection matrix and camera view matrix. Depending on your application requirements, you may want to define separate projection matrix and camera viewing matrix members in your vertex shaders so you can change them independently.

2. **Access the shader matrix** - After creating a hook in your vertex shaders to apply projection and camera view, you can then access that variable to apply projection and camera viewing matrices. The following code shows how to modify the `onSurfaceCreated()` method of a `GLSurfaceView.Renderer` implementation to access the matrix variable defined in the vertex shader above.

```
public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
    ...  
    mUMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram, "uMVPMatrix");  
    ...  
}
```

3. **Create projection and camera viewing matrices** - Generate the projection and viewing matrices to be applied the graphic objects. The following example code shows how to modify the `onSurfaceCreated()` and `onSurfaceChanged()` methods of a `GLSurfaceView.Renderer` implementation to create camera view matrix and a projection matrix based on the screen aspect ratio of the device.

```
public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
    ...  
    // Create a camera view matrix  
    Matrix.setLookAtM(mVMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0f, 0.0f);  
}  
  
public void onSurfaceChanged(GL10 unused, int width, int height) {  
    GLES20.glViewport(0, 0, width, height);  
  
    float ratio = (float) width / height;  
  
    // create a projection matrix from device screen geometry  
    Matrix.frustumM(mProjMatrix, 0, -ratio, ratio, -1, 1, 3, 7);  
}
```

4. **Apply projection and camera viewing matrices** - To apply the projection and camera view transformations, multiply the matrices together and then set them into the vertex shader. The following example code shows how modify the `onDrawFrame()` method of a `GLSurfaceView.Renderer` implementation to combine the projection matrix and camera view created in the code above and then apply it to the graphic objects to be rendered by OpenGL.

```

public void onDrawFrame(GL10 unused) {
    ...
    // Combine the projection and camera view matrices
    Matrix.multiplyMM(mMVPMatrix, 0, mProjMatrix, 0, mMMatrix, 0);

    // Apply the combined projection and camera view transformations
    GLES20.glUniformMatrix4fv(muMVPMatrixHandle, 1, false, mMVPMatrix, 0);

    // Draw objects
    ...
}

```

For a complete example of how to apply projection and camera view with OpenGL ES 2.0, see the [Displaying Graphics with OpenGL ES](#) class.

Shape Faces and Winding

In OpenGL, the face of a shape is a surface defined by three or more points in three-dimensional space. A set of three or more three-dimensional points (called vertices in OpenGL) have a front face and a back face. How do you know which face is front and which is the back? Good question. The answer has to do with winding, or, the direction in which you define the points of a shape.

```

float triangleCoords[] =
    {x1, y1, z1,
     x2, y2, z2,
     x3, y3, z3};

```

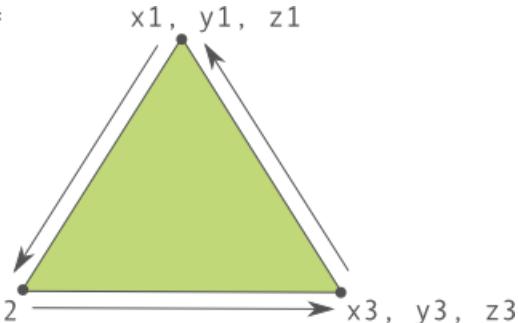


Figure 1. Illustration of a coordinate list which translates into a counterclockwise drawing order.

In this example, the points of the triangle are defined in an order such that they are drawn in a counterclockwise direction. The order in which these coordinates are drawn defines the winding direction for the shape. By default, in OpenGL, the face which is drawn counterclockwise is the front face. The triangle shown in Figure 1 is defined so that you are looking at the front face of the shape (as interpreted by OpenGL) and the other side is the back face.

Why is it important to know which face of a shape is the front face? The answer has to do with a commonly used feature of OpenGL, called face culling. Face culling is an option for the OpenGL environment which allows the rendering pipeline to ignore (not calculate or draw) the back face of a shape, saving time, memory and processing cycles:

```

// enable face culling feature
gl.glEnable(GL10.GL_CULL_FACE);
// specify which faces to not draw
gl.glCullFace(GL10.GL_BACK);

```

If you try to use the face culling feature without knowing which sides of your shapes are the front and back, your OpenGL graphics are going to look a bit thin, or possibly not show up at all. So, always define the coordinates of your OpenGL shapes in a counterclockwise drawing order.

Note: It is possible to set an OpenGL environment to treat the clockwise face as the front face, but doing so requires more code and is likely to confuse experienced OpenGL developers when you ask them for help. So don't do that.

OpenGL Versions and Device Compatibility

The OpenGL ES 1.0 and 1.1 API specifications have been supported since Android 1.0. Beginning with Android 2.2 (API level 8), the framework supports the OpenGL ES 2.0 API specification. OpenGL ES 2.0 is supported by most Android devices and is recommended for new applications being developed with OpenGL. OpenGL ES 3.0 is supported with Android 4.3 (API level 18) and higher, on devices that

provide an implementation of the OpenGL ES 3.0 API. For information about the relative number of Android-powered devices that support a given version of OpenGL ES, see the [OpenGL ES Version Dashboard](#).

Graphics programming with OpenGL ES 1.0/1.1 API is significantly different than using the 2.0 and higher versions. The 1.x version of the API has more convenience methods and a fixed graphics pipeline, while the OpenGL ES 2.0 and 3.0 APIs provide more direct control of the pipeline through use of OpenGL shaders. You should carefully consider the graphics requirements and choose the API version that works best for your application. For more information, see [Choosing an OpenGL API Version](#).

The OpenGL ES 3.0 API provides additional features and better performance than the 2.0 API and is also backward compatible. This means that you can potentially write your application targeting OpenGL ES 2.0 and conditionally include OpenGL ES 3.0 graphics features if they are available. For more information on checking for availability of the 3.0 API, see [Checking OpenGL ES Version](#)

Texture compression support

Texture compression can significantly increase the performance of your OpenGL application by reducing memory requirements and making more efficient use of memory bandwidth. The Android framework provides support for the ETC1 compression format as a standard feature, including a `ETC1Util` utility class and the `etc1tool` compression tool (located in the Android SDK at `<sdk>/tools/`). For an example of an Android application that uses texture compression, see the `CompressedTextureActivity` code sample in Android SDK (`<sdk>/samples/<version>/ApiDemos/src/com/example/android/apis/graphics/`).

Caution: The ETC1 format is supported by most Android devices, but it is not guaranteed to be available. To check if the ETC1 format is supported on a device, call the `ETC1Util.isETC1Supported()` method.

Note: The ETC1 texture compression format does not support textures with transparency (alpha channel). If your application requires textures with transparency, you should investigate other texture compression formats available on your target devices.

The ETC2/EAC texture compression formats are guaranteed to be available when using the OpenGL ES 3.0 API. This texture format offers excellent compression ratios with high visual quality and the format also supports transparency (alpha channel).

Beyond the ETC formats, Android devices have varied support for texture compression based on their GPU chipsets and OpenGL implementations. You should investigate texture compression support on the devices you are targeting to determine what compression types your application should support. In order to determine what texture formats are supported on a given device, you must [query the device](#) and review the *OpenGL extension names*, which identify what texture compression formats (and other OpenGL features) are supported by the device. Some commonly supported texture compression formats are as follows:

- **ATITC (ATC)** - ATI texture compression (ATITC or ATC) is available on a wide variety of devices and supports fixed rate compression for RGB textures with and without an alpha channel. This format may be represented by several OpenGL extension names, for example:
 - `GL_AMD_compressed_ATC_texture`
 - `GL_ATI_texture_compression_atitc`
- **PVRTC** - PowerVR texture compression (PVRTC) is available on a wide variety of devices and supports 2-bit and 4-bit per pixel textures with or without an alpha channel. This format is represented by the following OpenGL extension name:
 - `GL_IMG_texture_compression_pvrtc`
- **S3TC (DXTn/DXTc)** - S3 texture compression (S3TC) has several format variations (DXT1 to DXT5) and is less widely available. The format supports RGB textures with 4-bit alpha or 8-bit alpha channels. These formats are represented by the following OpenGL extension name:
 - `GL_EXT_texture_compression_s3tc`

Some devices only support the DXT1 format variation; this limited support is represented by the following OpenGL extension name:

- `GL_EXT_texture_compression_dxt1`

- **3DC** - 3DC texture compression (3DC) is a less widely available format that supports RGB textures with an alpha channel. This format is represented by the following OpenGL extension name:
 - `GL_AMD_compressed_3DC_texture`

Warning: These texture compression formats are *not supported* on all devices. Support for these formats can vary by manufacturer and device. For information on how to determine what texture compression formats are on a particular device, see the next section.

Note: Once you decide which texture compression formats your application will support, make sure you declare them in your manifest using `<supports-gl-texture>`. Using this declaration enables filtering by external services such as Google Play, so that your app is installed only on devices that support the formats your app requires. For details, see [OpenGL manifest declarations](#).

Determining OpenGL extensions

Implementations of OpenGL vary by Android device in terms of the extensions to the OpenGL ES API that are supported. These extensions include texture compressions, but typically also include other extensions to the OpenGL feature set.

To determine what texture compression formats, and other OpenGL extensions, are supported on a particular device:

1. Run the following code on your target devices to determine what texture compression formats are supported:

```
String extensions = javax.microedition.khronos.opengles.GL10.glGetString(GL10.GL_EXTENSIONS);
```

Warning: The results of this call *vary by device model!* You must run this call on several target devices to determine what compression types are commonly supported.

2. Review the output of this method to determine what OpenGL extensions are supported on the device.

Android Extension Pack (AEP)

The AEP ensures that your application supports a standardized set of OpenGL extensions above and beyond the core set described in the OpenGL 3.1 specification. Packaging these extensions together encourages a consistent set of functionality across devices, while allowing developers to take full advantage of the latest crop of mobile GPU devices.

The AEP also improves support for images, shader storage buffers, and atomic counters in fragment shaders.

For your app to be able to use the AEP, the app's manifest must declare that the AEP is required. In addition, the platform version must support it.

Declare the AEP requirement in the manifest as follows:

```
<uses feature android:name="android.hardware.opengles.aep"
    android:required="true" />
```

To verify that the platform version supports the AEP, use the `hasSystemFeature(String)` method, passing in `FEATURE_OPENGL_ES_EXTENSION_PACK` as the argument. The following code snippet shows an example of how to do so:

```
boolean deviceSupportsAEP = getPackageManager().hasSystemFeature(
    PackageManager.FEATURE_OPENGL_ES_EXTENSION_PACK);
```

If the method returns true, AEP is supported.

For more information about the AEP, visit its page at the [Khronos OpenGL ES Registry](#).

Checking the OpenGL ES Version

There are several versions of OpenGL ES available on Android devices. You can specify the minimum version of the API your application requires in your [manifest](#), but you may also want to take advantage of features in a newer API at the same time. For example, the OpenGL ES 3.0 API is backward-compatible with the 2.0 version of the API, so you may want to write your application so that it uses OpenGL ES 3.0 features, but falls back to the 2.0 API if the 3.0 API is not available.

Before using OpenGL ES features from a version higher than the minimum required in your application manifest, your application should check the version of the API available on the device. You can do this in one of two ways:

1. Attempt to create the higher-level OpenGL ES context ([EGLContext](#)) and check the result.
2. Create a minimum-supported OpenGL ES context and check the version value.

The following example code demonstrates how to check the available OpenGL ES version by creating an [EGLContext](#) and checking the

result. This example shows how to check for OpenGL ES 3.0 version:

```
private static double glVersion = 3.0;

private static class ContextFactory implements GLSurfaceView.EGLContextFactory {

    private static int EGL_CONTEXT_CLIENT_VERSION = 0x3098;

    public EGLContext createContext(
        EGL10 egl, EGLDisplay display, EGLConfig eglConfig) {

        Log.w(TAG, "creating OpenGL ES " + glVersion + " context");
        int[] attrib_list = {EGL_CONTEXT_CLIENT_VERSION, (int) glVersion,
            EGL10.EGL_NONE };
        // attempt to create a OpenGL ES 3.0 context
        EGLContext context = egl.eglCreateContext(
            display, eglConfig, EGL10.EGL_NO_CONTEXT, attrib_list);
        return context; // returns null if 3.0 is not supported;
    }
}
```

If the `createContext()` method shown above returns null, your code should create a OpenGL ES 2.0 context instead and fall back to using only that API.

The following code example demonstrates how to check the OpenGL ES version by creating a minimum supported context first, and then checking the version string:

```
// Create a minimum supported OpenGL ES context, then check:
String version = javax.microedition.khronos.opengles.GL10.glGetString(
    GL10.GL_VERSION);
Log.w(TAG, "Version: " + version );
// The version format is displayed as: "OpenGL ES <major>.<minor>"
// followed by optional content provided by the implementation.
```

With this approach, if you discover that the device supports a higher-level API version, you must destroy the minimum OpenGL ES context and create a new context with the higher available API version.

Choosing an OpenGL API Version

OpenGL ES 1.0 API version (and the 1.1 extensions), version 2.0, and version 3.0 all provide high performance graphics interfaces for creating 3D games, visualizations and user interfaces. Graphics programming for OpenGL ES 2.0 and 3.0 is largely similar, with version 3.0 representing a superset of the 2.0 API with additional features. Programming for the OpenGL ES 1.0/1.1 API versus OpenGL ES 2.0 and 3.0 differs significantly, and so developers should carefully consider the following factors before starting development with these APIs:

- **Performance** - In general, OpenGL ES 2.0 and 3.0 provide faster graphics performance than the ES 1.0/1.1 APIs. However, the performance difference can vary depending on the Android device your OpenGL application is running on, due to differences in hardware manufacturer's implementation of the OpenGL ES graphics pipeline.
- **Device Compatibility** - Developers should consider the types of devices, Android versions and the OpenGL ES versions available to their customers. For more information on OpenGL compatibility across devices, see the [OpenGL Versions and Device Compatibility](#) section.
- **Coding Convenience** - The OpenGL ES 1.0/1.1 API provides a fixed function pipeline and convenience functions which are not available in the OpenGL ES 2.0 or 3.0 APIs. Developers who are new to OpenGL ES may find coding for version 1.0/1.1 faster and more convenient.
- **Graphics Control** - The OpenGL ES 2.0 and 3.0 APIs provide a higher degree of control by providing a fully programmable pipeline through the use of shaders. With more direct control of the graphics processing pipeline, developers can create effects that would be very difficult to generate using the 1.0/1.1 API.
- **Texture Support** - The OpenGL ES 3.0 API has the best support for texture compression because it guarantees availability of the ETC2 compression format, which supports transparency. The 1.x and 2.0 API implementations usually include support for ETC1, however this texture format does not support transparency and so you must typically provide resources in other compression formats supported by the

devices you are targeting. For more information, see [Texture compression support](#).

While performance, compatibility, convenience, control and other factors may influence your decision, you should pick an OpenGL API version based on what you think provides the best experience for your users.

Hardware Acceleration

In this document

- › [Controlling Hardware Acceleration](#)
- › [Determining if a View is Hardware Accelerated](#)
- › [Android Drawing Models](#)
 - › [Software-based drawing model](#)
 - › [Hardware accelerated drawing model](#)
- › [Unsupported Drawing Operations](#)
- › [View Layers](#)
 - › [View Layers and Animations](#)
- › [Tips and Tricks](#)

See also

- › [OpenGL with the Framework APIs](#)
- › [RenderScript](#)

Beginning in Android 3.0 (API level 11), the Android 2D rendering pipeline supports hardware acceleration, meaning that all drawing operations that are performed on a [View](#)'s canvas use the GPU. Because of the increased resources required to enable hardware acceleration, your app will consume more RAM.

Hardware acceleration is enabled by default if your Target API level is ≥ 14 , but can also be explicitly enabled. If your application uses only standard views and [Drawables](#), turning it on globally should not cause any adverse drawing effects. However, because hardware acceleration is not supported for all of the 2D drawing operations, turning it on might affect some of your custom views or drawing calls. Problems usually manifest themselves as invisible elements, exceptions, or wrongly rendered pixels. To remedy this, Android gives you the option to enable or disable hardware acceleration at multiple levels. See [Controlling Hardware Acceleration](#).

If your application performs custom drawing, test your application on actual hardware devices with hardware acceleration turned on to find any problems. The [Unsupported drawing operations](#) section describes known issues with hardware acceleration and how to work around them.

Controlling Hardware Acceleration

You can control hardware acceleration at the following levels:

- Application
- Activity
- Window
- View

Application level

In your Android manifest file, add the following attribute to the [`<application>`](#) tag to enable hardware acceleration for your entire application:

```
<application android:hardwareAccelerated="true" ...>
```

Activity level

If your application does not behave properly with hardware acceleration turned on globally, you can control it for individual activities as well. To enable or disable hardware acceleration at the activity level, you can use the `android:hardwareAccelerated` attribute for the `<activity>` element. The following example enables hardware acceleration for the entire application but disables it for one activity:

```
<application android:hardwareAccelerated="true">
    <activity ... />
    <activity android:hardwareAccelerated="false" />
</application>
```

Window level

If you need even more fine-grained control, you can enable hardware acceleration for a given window with the following code:

```
getWindow().setFlags(
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
```

Note: You currently cannot disable hardware acceleration at the window level.

View level

You can disable hardware acceleration for an individual view at runtime with the following code:

```
myView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

Note: You currently cannot enable hardware acceleration at the view level. View layers have other functions besides disabling hardware acceleration. See [View layers](#) for more information about their uses.

Determining if a View is Hardware Accelerated

It is sometimes useful for an application to know whether it is currently hardware accelerated, especially for things such as custom views. This is particularly useful if your application does a lot of custom drawing and not all operations are properly supported by the new rendering pipeline.

There are two different ways to check whether the application is hardware accelerated:

- `View.isHardwareAccelerated()` returns `true` if the `View` is attached to a hardware accelerated window.
- `Canvas.isHardwareAccelerated()` returns `true` if the `Canvas` is hardware accelerated

If you must do this check in your drawing code, use `Canvas.isHardwareAccelerated()` instead of `View.isHardwareAccelerated()` when possible. When a view is attached to a hardware accelerated window, it can still be drawn using a non-hardware accelerated `Canvas`. This happens, for instance, when drawing a view into a bitmap for caching purposes.

Android Drawing Models

When hardware acceleration is enabled, the Android framework utilizes a new drawing model that utilizes *display lists* to render your application to the screen. To fully understand display lists and how they might affect your application, it is useful to understand how Android draws views without hardware acceleration as well. The following sections describe the software-based and hardware-accelerated drawing models.

Software-based drawing model

In the software drawing model, views are drawn with the following two steps:

1. Invalidate the hierarchy
2. Draw the hierarchy

Whenever an application needs to update a part of its UI, it invokes `invalidate()` (or one of its variants) on any view that has changed

content. The invalidation messages are propagated all the way up the view hierarchy to compute the regions of the screen that need to be redrawn (the dirty region). The Android system then draws any view in the hierarchy that intersects with the dirty region. Unfortunately, there are two drawbacks to this drawing model:

- First, this model requires execution of a lot of code on every draw pass. For example, if your application calls `invalidate()` on a button and that button sits on top of another view, the Android system redraws the view even though it hasn't changed.
- The second issue is that the drawing model can hide bugs in your application. Since the Android system redraws views when they intersect the dirty region, a view whose content you changed might be redrawn even though `invalidate()` was not called on it. When this happens, you are relying on another view being invalidated to obtain the proper behavior. This behavior can change every time you modify your application. Because of this, you should always call `invalidate()` on your custom views whenever you modify data or state that affects the view's drawing code.

Note: Android views automatically call `invalidate()` when their properties change, such as the background color or the text in a `TextView`.

Hardware accelerated drawing model

The Android system still uses `invalidate()` and `draw()` to request screen updates and to render views, but handles the actual drawing differently. Instead of executing the drawing commands immediately, the Android system records them inside display lists, which contain the output of the view hierarchy's drawing code. Another optimization is that the Android system only needs to record and update display lists for views marked dirty by an `invalidate()` call. Views that have not been invalidated can be redrawn simply by re-issuing the previously recorded display list. The new drawing model contains three stages:

1. Invalidate the hierarchy
2. Record and update display lists
3. Draw the display lists

With this model, you cannot rely on a view intersecting the dirty region to have its `draw()` method executed. To ensure that the Android system records a view's display list, you must call `invalidate()`. Forgetting to do so causes a view to look the same even after it has been changed.

Using display lists also benefits animation performance because setting specific properties, such as alpha or rotation, does not require invalidating the targeted view (it is done automatically). This optimization also applies to views with display lists (any view when your application is hardware accelerated.) For example, assume there is a `LinearLayout` that contains a `ListView` above a `Button`. The display list for the `LinearLayout` looks like this:

- `DrawDisplayList(ListView)`
- `DrawDisplayList(Button)`

Assume now that you want to change the `ListView`'s opacity. After invoking `setAlpha(0.5f)` on the `ListView`, the display list now contains this:

- `SaveLayerAlpha(0.5)`
- `DrawDisplayList(ListView)`
- `Restore`
- `DrawDisplayList(Button)`

The complex drawing code of `ListView` was not executed. Instead, the system only updated the display list of the much simpler `LinearLayout`. In an application without hardware acceleration enabled, the drawing code of both the list and its parent are executed again.

Unsupported Drawing Operations

When hardware accelerated, the 2D rendering pipeline supports the most commonly used `Canvas` drawing operations as well as many less-used operations. All of the drawing operations that are used to render applications that ship with Android, default widgets and layouts, and common advanced visual effects such as reflections and tiled textures are supported.

The following table describes the support level of various operations across API levels:

	First supported API level
Canvas	
drawBitmapMesh() (colors array)	18
drawPicture()	23
drawPosText()	16
drawTextOnPath()	16
drawVertices()	X
setDrawFilter()	16
clipPath()	18
clipRegion()	18
clipRect(Region.Op.XOR)	18
clipRect(Region.Op.Difference)	18
clipRect(Region.Op.ReverseDifference)	18
clipRect() with rotation/perspective	18
Paint	
setAntiAlias() (for text)	18
setAntiAlias() (for lines)	16
setFilterBitmap()	17
setLinearText()	X
setMaskFilter()	X
setPathEffect() (for lines)	X
setRasterizer()	X
setShadowLayer() (other than text)	X
setStrokeCap() (for lines)	18
setStrokeCap() (for points)	19
setSubpixelText()	X
Xfermode	
PorterDuff.Mode.DARKEN (framebuffer)	X
PorterDuff.Mode.LIGHTEN (framebuffer)	X
PorterDuff.Mode.OVERLAY (framebuffer)	X
Shader	
ComposeShader inside ComposeShader	X
Same type shaders inside ComposeShader	X
Local matrix on ComposeShader	18

Canvas Scaling

The hardware accelerated 2D rendering pipeline was built first to support unscaled drawing, with some drawing operations degrading quality significantly at higher scale values. These operations are implemented as textures drawn at scale 1.0, transformed by the GPU. In API level <17, using these operations will result in scaling artifacts increasing with scale.

The following table shows when implementation was changed to correctly handle large scales:

Drawing operation to be scaled	First supported API level
drawText()	18
drawPosText()	X
drawTextOnPath()	X
Simple Shapes*	17
Complex Shapes*	X
drawPath()	X
Shadow layer	X

Note: 'Simple' shapes are `drawRect()`, `drawCircle()`, `drawOval()`, `drawRoundRect()`, and `drawArc()` (with `useCenter=false`) commands issued with a Paint that doesn't have a PathEffect, and doesn't contain non-default joins (via `setStrokeJoin()` / `setStrokeMiter()`). Other instances of those draw commands fall under 'Complex,' in the above chart.

If your application is affected by any of these missing features or limitations, you can turn off hardware acceleration for just the affected portion of your application by calling `setLayerType(View.LAYER_TYPE_SOFTWARE, null)`. This way, you can still take advantage of hardware acceleration everywhere else. See [Controlling Hardware Acceleration](#) for more information on how to enable and disable hardware acceleration at different levels in your application.

View Layers

In all versions of Android, views have had the ability to render into off-screen buffers, either by using a view's drawing cache, or by using `Canvas.saveLayer()`. Off-screen buffers, or layers, have several uses. You can use them to get better performance when animating complex views or to apply composition effects. For instance, you can implement fade effects using `Canvas.saveLayer()` to temporarily render a view into a layer and then composite it back on screen with an opacity factor.

Beginning in Android 3.0 (API level 11), you have more control on how and when to use layers with the `View.setLayerType()` method. This API takes two parameters: the type of layer you want to use and an optional `Paint` object that describes how the layer should be composited. You can use the `Paint` parameter to apply color filters, special blending modes, or opacity to a layer. A view can use one of three layer types:

- `LAYER_TYPE_NONE`: The view is rendered normally and is not backed by an off-screen buffer. This is the default behavior.
- `LAYER_TYPE_HARDWARE`: The view is rendered in hardware into a hardware texture if the application is hardware accelerated. If the application is not hardware accelerated, this layer type behaves the same as `LAYER_TYPE_SOFTWARE`.
- `LAYER_TYPE_SOFTWARE`: The view is rendered in software into a bitmap.

The type of layer you use depends on your goal:

- **Performance**: Use a hardware layer type to render a view into a hardware texture. Once a view is rendered into a layer, its drawing code does not have to be executed until the view calls `invalidate()`. Some animations, such as alpha animations, can then be applied directly onto the layer, which is very efficient for the GPU to do.
- **Visual effects**: Use a hardware or software layer type and a `Paint` to apply special visual treatments to a view. For instance, you can draw a view in black and white using a `ColorMatrixColorFilter`.
- **Compatibility**: Use a software layer type to force a view to be rendered in software. If a view that is hardware accelerated (for instance, if your whole application is hardware accelerated), is having rendering problems, this is an easy way to work around limitations of the hardware rendering pipeline.

View layers and animations

Hardware layers can deliver faster and smoother animations when your application is hardware accelerated. Running an animation at 60 frames per second is not always possible when animating complex views that issue a lot of drawing operations. This can be alleviated by using hardware layers to render the view to a hardware texture. The hardware texture can then be used to animate the view, eliminating the need for the view to constantly redraw itself when it is being animated. The view is not redrawn unless you change the view's properties, which calls `invalidate()`, or if you call `invalidate()` manually. If you are running an animation in your application and do not obtain the smooth results you want, consider enabling hardware layers on your animated views.

When a view is backed by a hardware layer, some of its properties are handled by the way the layer is composited on screen. Setting these properties will be efficient because they do not require the view to be invalidated and redrawn. The following list of properties affect the way the layer is composited. Calling the setter for any of these properties results in optimal invalidation and no redrawing of the targeted view:

- `alpha`: Changes the layer's opacity
- `x, y, translationX, translationY`: Changes the layer's position
- `scaleX, scaleY`: Changes the layer's size
- `rotation, rotationX, rotationY`: Changes the layer's orientation in 3D space

- **pivotX**, **pivotY**: Changes the layer's transformations origin

These properties are the names used when animating a view with an `ObjectAnimator`. If you want to access these properties, call the appropriate setter or getter. For instance, to modify the alpha property, call `setAlpha()`. The following code snippet shows the most efficient way to rotate a view in 3D around the Y-axis:

```
view.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator.ofFloat(view, "rotationY", 180).start();
```

Because hardware layers consume video memory, it is highly recommended that you enable them only for the duration of the animation and then disable them after the animation is done. You can accomplish this using animation listeners:

```
View.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "rotationY", 180);
animator.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        view.setLayerType(View.LAYER_TYPE_NONE, null);
    }
});
animator.start();
```

For more information on property animation, see [Property Animation](#).

Tips and Tricks

Switching to hardware accelerated 2D graphics can instantly increase performance, but you should still design your application to use the GPU effectively by following these recommendations:

Reduce the number of views in your application

The more views the system has to draw, the slower it will be. This applies to the software rendering pipeline as well. Reducing views is one of the easiest ways to optimize your UI.

Avoid overdraw

Do not draw too many layers on top of each other. Remove any views that are completely obscured by other opaque views on top of it. If you need to draw several layers blended on top of each other, consider merging them into a single layer. A good rule of thumb with current hardware is to not draw more than 2.5 times the number of pixels on screen per frame (transparent pixels in a bitmap count!).

Don't create render objects in draw methods

A common mistake is to create a new `Paint` or a new `Path` every time a rendering method is invoked. This forces the garbage collector to run more often and also bypasses caches and optimizations in the hardware pipeline.

Don't modify shapes too often

Complex shapes, paths, and circles for instance, are rendered using texture masks. Every time you create or modify a path, the hardware pipeline creates a new mask, which can be expensive.

Don't modify bitmaps too often

Every time you change the content of a bitmap, it is uploaded again as a GPU texture the next time you draw it.

Use alpha with care

When you make a view translucent using `setAlpha()`, `AlphaAnimation`, or `ObjectAnimator`, it is rendered in an off-screen buffer which doubles the required fill-rate. When applying alpha on very large views, consider setting the view's layer type to `LAYER_TYPE_HARDWARE`.



Computation

RenderScript provides a platform-independent computation engine that operates at the native level. Use it to accelerate your apps that require extensive computational horsepower.

BLOG ARTICLES

Evolution of RenderScript Performance

It's been a year since the last blog post on RenderScript, and with the release of Android 4.2, it's a good time to talk about the performance work that we've done since then. One of the major goals of this past year was to improve the performance of common image-processing operations with RenderScript.

Levels in RenderScript

For ICS, RenderScript (RS) has been updated with several new features to simplify adding compute acceleration to your application. RS is interesting for compute acceleration when you have large buffers of data on which you need to do significant processing. In this example we will look at applying a levels/saturation operation on a bitmap.

RenderScript Part 2

In Introducing RenderScript I gave a brief overview of this technology. In this post I'll look at "compute" in more detail. In RenderScript we use "compute" to mean offloading of data processing from Dalvik code to RenderScript code which may run on the same or different processor(s).



RenderScript

In this document

- [Writing a RenderScript Kernel](#)
- [Accessing RenderScript APIs from Java](#)
 - [Setting Up Your Development Environment](#)
 - [Using RenderScript from Java Code](#)
 - [Single-Source RenderScript](#)
 - [Script Globals](#)
 - [Reduction Kernels in Depth](#)
 - [Writing a reduction kernel](#)
 - [Calling a reduction kernel from Java code](#)
 - [More example reduction kernels](#)

Related Samples

- [Hello Compute](#)

RenderScript is a framework for running computationally intensive tasks at high performance on Android. RenderScript is primarily oriented for use with data-parallel computation, although serial workloads can benefit as well. The RenderScript runtime parallelizes work across processors available on a device, such as multi-core CPUs and GPUs. This allows you to focus on expressing algorithms rather than scheduling work. RenderScript is especially useful for applications performing image processing, computational photography, or computer vision.

To begin with RenderScript, there are two main concepts you should understand:

- The *language* itself is a C99-derived language for writing high-performance compute code. [Writing a RenderScript Kernel](#) describes how to use it to write compute kernels.
- The *control API* is used for managing the lifetime of RenderScript resources and controlling kernel execution. It is available in three different languages: Java, C++ in Android NDK, and the C99-derived kernel language itself. [Using RenderScript from Java Code](#) and [Single-Source RenderScript](#) describe the first and the third options, respectively.

Writing a RenderScript Kernel

A RenderScript kernel typically resides in a `.rs` file in the `<project_root>/src/` directory; each `.rs` file is called a *script*. Every script contains its own set of kernels, functions, and variables. A script can contain:

- A pragma declaration (`#pragma version(1)`) that declares the version of the RenderScript kernel language used in this script. Currently, 1 is the only valid value.
- A pragma declaration (`#pragma rs java_package_name(com.example.app)`) that declares the package name of the Java classes reflected from this script. Note that your `.rs` file must be part of your application package, and not in a library project.
- Zero or more **invokable functions**. An invokable function is a single-threaded RenderScript function that you can call from your Java code with arbitrary arguments. These are often useful for initial setup or serial computations within a larger processing pipeline.
- Zero or more **script globals**. A script global is similar to a global variable in C. You can access script globals from Java code, and these are often used for parameter passing to RenderScript kernels. Script globals are explained in more detail [here](#).
- Zero or more **compute kernels**. A compute kernel is a function or collection of functions that you can direct the RenderScript runtime to

execute in parallel across a collection of data. There are two kinds of compute kernels: *mapping kernels* (also called *foreach kernels*) and *reduction kernels*.

A *mapping kernel* is a parallel function that operates on a collection of [Allocations](#) of the same dimensions. By default, it executes once for every coordinate in those dimensions. It is typically (but not exclusively) used to transform a collection of input [Allocations](#) to an output [Allocation](#) one [Element](#) at a time.

- Here is an example of a simple **mapping kernel**:

```
uchar4 RS_KERNEL invert(uchar4 in, uint32_t x, uint32_t y) {
    uchar4 out = in;
    out.r = 255 - in.r;
    out.g = 255 - in.g;
    out.b = 255 - in.b;
    return out;
}
```

In most respects, this is identical to a standard C function. The `RS_KERNEL` property applied to the function prototype specifies that the function is a RenderScript mapping kernel instead of an invokable function. The `in` argument is automatically filled in based on the input [Allocation](#) passed to the kernel launch. The arguments `x` and `y` are discussed [below](#). The value returned from the kernel is automatically written to the appropriate location in the output [Allocation](#). By default, this kernel is run across its entire input [Allocation](#), with one execution of the kernel function per [Element](#) in the [Allocation](#).

A mapping kernel may have one or more input [Allocations](#), a single output [Allocation](#), or both. The RenderScript runtime checks to ensure that all input and output Allocations have the same dimensions, and that the [Element](#) types of the input and output Allocations match the kernel's prototype; if either of these checks fails, RenderScript throws an exception.

NOTE: Before Android 6.0 (API level 23), a mapping kernel may not have more than one input [Allocation](#).

If you need more input or output [Allocations](#) than the kernel has, those objects should be bound to `rs_allocation` script globals and accessed from a kernel or invokable function via `rsGetElementAt_type()` or `rsSetElementAt_type()`.

NOTE: `RS_KERNEL` is a macro defined automatically by RenderScript for your convenience:

```
#define RS_KERNEL __attribute__((kernel))
```

A *reduction kernel* is a family of functions that operates on a collection of input [Allocations](#) of the same dimensions. By default, its [accumulator function](#) executes once for every coordinate in those dimensions. It is typically (but not exclusively) used to "reduce" a collection of input [Allocations](#) to a single value.

- Here is an [example](#) of a simple **reduction kernel** that adds up the [Elements](#) of its input:

```
#pragma rs reduce(addint) accumulator(addintAccum)

static void addintAccum(int *accum, int val) {
    *accum += val;
}
```

A reduction kernel consists of one or more user-written functions. `#pragma rs reduce` is used to define the kernel by specifying its name (`addint`, in this example) and the names and roles of the functions that make up the kernel (an [accumulator](#) function `addintAccum`, in this example). All such functions must be `static`. A reduction kernel always requires an [accumulator](#) function; it may also have other functions, depending on what you want the kernel to do.

A reduction kernel accumulator function must return `void` and must have at least two arguments. The first argument (`accum`, in this example) is a pointer to an *accumulator data item* and the second (`val`, in this example) is automatically filled in based on the input [Allocation](#) passed to the kernel launch. The accumulator data item is created by the RenderScript runtime; by default, it is initialized to zero. By default, this kernel is run across its entire input [Allocation](#), with one execution of the accumulator function per [Element](#) in the [Allocation](#). By default, the final value of the accumulator data item is treated as the result of the reduction, and is returned to Java. The RenderScript runtime checks to ensure that the [Element](#) type of the input Allocation matches the accumulator function's prototype; if it does not match, RenderScript throws an exception.

A reduction kernel has one or more input [Allocations](#) but no output [Allocations](#).

Reduction kernels are explained in more detail [here](#).

Reduction kernels are supported in Android 7.0 (API level 24) and later.

A mapping kernel function or a reduction kernel accumulator function may access the coordinates of the current execution using the [special arguments](#) `x`, `y`, and `z`, which must be of type `int` or `uint32_t`. These arguments are optional.

A mapping kernel function or a reduction kernel accumulator function may also take the optional special argument `context` of type `rs_kernel_context`. It is needed by a family of runtime APIs that are used to query certain properties of the current execution -- for example, `rsGetDimX`. (The `context` argument is available in Android 6.0 (API level 23) and later.)

- An optional `init()` function. The `init()` function is a special type of invokable function that RenderScript runs when the script is first instantiated. This allows for some computation to occur automatically at script creation.
- Zero or more **static script globals and functions**. A static script global is equivalent to a script global except that it cannot be accessed from Java code. A static function is a standard C function that can be called from any kernel or invokable function in the script but is not exposed to the Java API. If a script global or function does not need to be accessed from Java code, it is highly recommended that it be declared `static`.

Setting floating point precision

You can control the required level of floating point precision in a script. This is useful if full IEEE 754-2008 standard (used by default) is not required. The following pragmas can set a different level of floating point precision:

- `#pragma rs_fp_full` (default if nothing is specified): For apps that require floating point precision as outlined by the IEEE 754-2008 standard.
- `#pragma rs_fp_relaxed`: For apps that don't require strict IEEE 754-2008 compliance and can tolerate less precision. This mode enables flush-to-zero for denorms and round-towards-zero.
- `#pragma rs_fp_imprecise`: For apps that don't have stringent precision requirements. This mode enables everything in `rs_fp_relaxed` along with the following:
 - Operations resulting in -0.0 can return +0.0 instead.
 - Operations on INF and NAN are undefined.

Most applications can use `rs_fp_relaxed` without any side effects. This may be very beneficial on some architectures due to additional optimizations only available with relaxed precision (such as SIMD CPU instructions).

Accessing RenderScript APIs from Java

When developing an Android application that uses RenderScript, you can access its API from Java in one of two ways:

- `android.renderscript` - The APIs in this class package are available on devices running Android 3.0 (API level 11) and higher.
- `android.support.v8.renderscript` - The APIs in this package are available through a [Support Library](#), which allows you to use them on devices running Android 2.3 (API level 9) and higher.

Here are the tradeoffs:

- If you use the Support Library APIs, the RenderScript portion of your application will be compatible with devices running Android 2.3 (API level 9) and higher, regardless of which RenderScript features you use. This allows your application to work on more devices than if you use the native (`android.renderscript`) APIs.
- Certain RenderScript features are not available through the Support Library APIs.
- If you use the Support Library APIs, you will get (possibly significantly) larger APKs than if you use the native (`android.renderscript`) APIs.

Using the RenderScript Support Library APIs

In order to use the Support Library RenderScript APIs, you must configure your development environment to be able to access them. The following Android SDK tools are required for using these APIs:

- Android SDK Tools revision 22.2 or higher

- Android SDK Build-tools revision 18.1.0 or higher

Note that starting from Android SDK Build-tools 24.0.0, Android 2.2 (API level 8) is no longer supported.

You can check and update the installed version of these tools in the [Android SDK Manager](#).

To use the Support Library RenderScript APIs:

1. Make sure you have the required Android SDK version and Build Tools version installed.
2. Update the settings for the Android build process to include the RenderScript settings:
 - Open the `build.gradle` file in the app folder of your application module.
 - Add the following RenderScript settings to the file:

```
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    defaultConfig {
        minSdkVersion 9
        targetSdkVersion 19

        renderscriptTargetApi 18
        renderscriptSupportModeEnabled true

    }
}
```

The settings listed above control specific behavior in the Android build process:

- `renderscriptTargetApi` - Specifies the bytecode version to be generated. We recommend you set this value to the lowest API level able to provide all the functionality you are using and set `renderscriptSupportModeEnabled` to `true`. Valid values for this setting are any integer value from 11 to the most recently released API level. If your minimum SDK version specified in your application manifest is set to a different value, that value is ignored and the target value in the build file is used to set the minimum SDK version.
- `renderscriptSupportModeEnabled` - Specifies that the generated bytecode should fall back to a compatible version if the device it is running on does not support the target version.
- `buildToolsVersion` - The version of the Android SDK build tools to use. This value should be set to `18.1.0` or higher. If this option is not specified, the highest installed build tools version is used. You should always set this value to ensure the consistency of builds across development machines with different configurations.

3. In your application classes that use RenderScript, add an import for the Support Library classes:

```
import android.support.v8.renderscript.*;
```

Using RenderScript from Java Code

Using RenderScript from Java code relies on the API classes located in the `android.renderscript` or the `android.support.v8.renderscript` package. Most applications follow the same basic usage pattern:

1. **Initialize a RenderScript context.** The `RenderScript` context, created with `create(Context)`, ensures that RenderScript can be used and provides an object to control the lifetime of all subsequent RenderScript objects. You should consider context creation to be a potentially long-running operation, since it may create resources on different pieces of hardware; it should not be in an application's critical path if at all possible. Typically, an application will have only a single RenderScript context at a time.
2. **Create at least one Allocation to be passed to a script.** An `Allocation` is a RenderScript object that provides storage for a fixed amount of data. Kernels in scripts take `Allocation` objects as their input and output, and `Allocation` objects can be accessed in kernels using `rsGetElementAt_type()` and `rsSetElementAt_type()` when bound as script globals. `Allocation` objects allow arrays to be passed from Java code to RenderScript code and vice-versa. `Allocation` objects are typically created using `createTyped()` or `createFromBitmap()`.

3. **Create whatever scripts are necessary.** There are two types of scripts available to you when using RenderScript:
- **ScriptC:** These are the user-defined scripts as described in [Writing a RenderScript Kernel](#) above. Every script has a Java class reflected by the RenderScript compiler in order to make it easy to access the script from Java code; this class has the name `ScriptC_filename`. For example, if the mapping kernel above were located in `invert.rs` and a RenderScript context were already located in `mRenderScript`, the Java code to instantiate the script would be:
- ```
ScriptC_invert invert = new ScriptC_invert(mRenderScript);
```
- **ScriptIntrinsic:** These are built-in RenderScript kernels for common operations, such as Gaussian blur, convolution, and image blending. For more information, see the subclasses of [ScriptIntrinsic](#).
4. **Populate Allocations with data.** Except for Allocations created with `createFromBitmap()`, an Allocation is populated with empty data when it is first created. To populate an Allocation, use one of the "copy" methods in [Allocation](#). The "copy" methods are [synchronous](#).
5. **Set any necessary script globals.** You may set globals using methods in the same `ScriptC_filename` class named `set_globalname`. For example, in order to set an `int` variable named `threshold`, use the Java method `set_threshold(int)`; and in order to set an `rs_allocation` variable named `lookup`, use the Java method `set_lookup(Allocation)`. The `set` methods are [asynchronous](#).
6. **Launch the appropriate kernels and invokable functions.**
- Methods to launch a given kernel are reflected in the same `ScriptC_filename` class with methods named `forEach_mappingKernelName()` or `reduce_reductionKernelName()`. These launches are [asynchronous](#). Depending on the arguments to the kernel, the method takes one or more Allocations, all of which must have the same dimensions. By default, a kernel executes over every coordinate in those dimensions; to execute a kernel over a subset of those coordinates, pass an appropriate `Script.LaunchOptions` as the last argument to the `forEach` or `reduce` method.
- Launch invokable functions using the `invoke_functionName` methods reflected in the same `ScriptC_filename` class. These launches are [asynchronous](#).
7. **Retrieve data from Allocation objects and javaFutureType objects.** In order to access data from an [Allocation](#) from Java code, you must copy that data back to Java using one of the "copy" methods in [Allocation](#). In order to obtain the result of a reduction kernel, you must use the `javaFutureType.get()` method. The "copy" and `get()` methods are [synchronous](#).
8. **Tear down the RenderScript context.** You can destroy the RenderScript context with `destroy()` or by allowing the RenderScript context object to be garbage collected. This causes any further use of any object belonging to that context to throw an exception.

## Asynchronous execution model

The reflected `forEach`, `invoke`, `reduce`, and `set` methods are asynchronous -- each may return to Java before completing the requested action. However, the individual actions are serialized in the order in which they are launched.

The [Allocation](#) class provides "copy" methods to copy data to and from Allocations. A "copy" method is synchronous, and is serialized with respect to any of the asynchronous actions above that touch the same Allocation.

The reflected `javaFutureType` classes provide a `get()` method to obtain the result of a reduction. `get()` is synchronous, and is serialized with respect to the reduction (which is asynchronous).

## Single-Source RenderScript

Android 7.0 (API level 24) introduces a new programming feature called *Single-Source RenderScript*, in which kernels are launched from the script where they are defined, rather than from Java. This approach is currently limited to mapping kernels, which are simply referred to as "kernels" in this section for conciseness. This new feature also supports creating allocations of type `rs_allocation` from inside the script. It is now possible to implement a whole algorithm solely within a script, even if multiple kernel launches are required. The benefit is twofold: more readable code, because it keeps the implementation of an algorithm in one language; and potentially faster code, because of fewer transitions between Java and RenderScript across multiple kernel launches.

In Single-Source RenderScript, you write kernels as described in [Writing a RenderScript Kernel](#). You then write an invokable function that calls `rsForEach()` to launch them. That API takes a kernel function as the first parameter, followed by input and output allocations. A similar API `rsForEachWithOptions()` takes an extra argument of type `rs_script_call_t`, which specifies a subset of the elements from the input

and output allocations for the kernel function to process.

To start RenderScript computation, you call the invokable function from Java. Follow the steps in [Using RenderScript from Java Code](#). In the step [launch the appropriate kernels](#), call the invokable function using `invoke_function_name()`, which will start the whole computation, including launching kernels.

Allocations are often needed to save and pass intermediate results from one kernel launch to another. You can create them using `rsCreateAllocation()`. One easy-to-use form of that API is `rsCreateAllocation_<T><W>(...)`, where *T* is the data type for an element, and *W* is the vector width for the element. The API takes the sizes in dimensions X, Y, and Z as arguments. For 1D or 2D allocations, the size for dimension Y or Z can be omitted. For example, `rsCreateAllocation uchar4(16384)` creates a 1D allocation of 16384 elements, each of which is of type `uchar4`.

Allocations are managed by the system automatically. You do not have to explicitly release or free them. However, you can call `rsClearObject(rs_allocation* alloc)` to indicate you no longer need the handle `alloc` to the underlying allocation, so that the system can free up resources as early as possible.

The [Writing a RenderScript Kernel](#) section contains an example kernel that inverts an image. The example below expands that to apply more than one effect to an image, using Single-Source RenderScript. It includes another kernel, `greyscale`, which turns a color image into black-and-white. An invokable function `process()` then applies those two kernels consecutively to an input image, and produces an output image. Allocations for both the input and the output are passed in as arguments of type `rs_allocation`.

```
// File: singlesource.rs

#pragma version(1)
#pragma rs java_package_name(com.android.rssample)

static const float4 weight = {0.299f, 0.587f, 0.114f, 0.0f};

uchar4 RS_KERNEL invert(uchar4 in, uint32_t x, uint32_t y) {
 uchar4 out = in;
 out.r = 255 - in.r;
 out.g = 255 - in.g;
 out.b = 255 - in.b;
 return out;
}

uchar4 RS_KERNEL greyscale(uchar4 in) {
 const float4 inF = rsUnpackColor8888(in);
 const float4 outF = (float4){ dot(inF, weight) };
 return rsPackColorTo8888(outF);
}

void process(rs_allocation inputImage, rs_allocation outputImage) {
 const uint32_t imageWidth = rsAllocationGetDimX(inputImage);
 const uint32_t imageHeight = rsAllocationGetDimY(inputImage);
 rs_allocation tmp = rsCreateAllocation_uchar4(imageWidth, imageHeight);
 rsForEach(invert, inputImage, tmp);
 rsForEach(greyscale, tmp, outputImage);
}
```

You can call the `process()` function from Java as follows:

```
// File SingleSource.java

RenderScript RS = RenderScript.create(context);
ScriptC_singlesource script = new ScriptC_singlesource(RS);
Allocation inputAllocation = Allocation.createFromBitmapResource(
 RS, getResources(), R.drawable.image);
Allocation outputAllocation = Allocation.createTyped(
 RS, inputAllocation.getType(),
 Allocation.USAGE_SCRIPT | Allocation.USAGE_IO_OUTPUT);
script.invoke_process(inputAllocation, outputAllocation);
```

This example shows how an algorithm that involves two kernel launches can be implemented completely in the RenderScript language itself. Without Single-Source RenderScript, you would have to launch both kernels from the Java code, separating kernel launches from kernel

definitions and making it harder to understand the whole algorithm. Not only is the Single-Source RenderScript code easier to read, it also eliminates the transitioning between Java and the script across kernel launches. Some iterative algorithms may launch kernels hundreds of times, making the overhead of such transitioning considerable.

## Script Globals

A *script global* is an ordinary non-`static` global variable in a script (`.rs`) file. For a script global named `var` defined in the file `filename.rs`, there will be a method `get_var` reflected in the class `ScriptC_filename`. Unless the global is `const`, there will also be a method `set_var`.

A given script global has two separate values -- a *Java* value and a *script* value. These values behave as follows:

- If `var` has a static initializer in the script, it specifies the initial value of `var` in both Java and the script. Otherwise, that initial value is zero.
- Accesses to `var` within the script read and write its script value.
- The `get_var` method reads the Java value.
- The `set_var` method (if it exists) writes the Java value immediately, and writes the script value *asynchronously*.

**NOTE:** This means that except for any static initializer in the script, values written to a global from within a script are not visible to Java.

## Reduction Kernels in Depth

*Reduction* is the process of combining a collection of data into a single value. This is a useful primitive in parallel programming, with applications such as the following:

- computing the sum or product over all the data
- computing logical operations (`and`, `or`, `xor`) over all the data
- finding the minimum or maximum value within the data
- searching for a specific value or for the coordinate of a specific value within the data

In Android 7.0 (API level 24) and later, RenderScript supports *reduction kernels* to allow efficient user-written reduction algorithms. You may launch reduction kernels on inputs with 1, 2, or 3 dimensions.

An example above shows a simple `addint` reduction kernel. Here is a more complicated `findMinAndMax` reduction kernel that finds the locations of the minimum and maximum `long` values in a 1-dimensional `Allocation`:

```
#define LONG_MAX (long)((1UL << 63) - 1)
#define LONG_MIN (long)(1UL << 63)

#pragma rs reduce(findMinAndMax) \
 initializer(fMMInit) accumulator(fMMAccumulator) \
 combiner(fMMCombiner) outconverter(fMMOutConverter)

// Either a value and the location where it was found, or INITVAL.
typedef struct {
 long val;
 int idx; // -1 indicates INITVAL
} IndexedVal;

typedef struct {
 IndexedVal min, max;
} MinAndMax;

// In discussion below, this initial value { { LONG_MAX, -1 }, { LONG_MIN, -1 } }
// is called INITVAL.
static void fMMInit(MinAndMax *accum) {
 accum->min.val = LONG_MAX;
 accum->min.idx = -1;
 accum->max.val = LONG_MIN;
 accum->max.idx = -1;
}
```

```

// In describing the behavior of the accumulator and combiner functions,

// it is helpful to describe hypothetical functions

// IndexedVal min(IndexedVal a, IndexedVal b)

// IndexVal max(IndexedVal a, IndexedVal b)

// MinAndMax minmax(MinAndMax a, MinAndMax b)

// MinAndMax minmax(MinAndMax accum, IndexedVal val)

//

// The effect of

// IndexVal min(IndexedVal a, IndexedVal b)

// is to return the IndexVal from among the two arguments

// whose val is lesser, except that when an IndexVal

// has a negative index, that IndexVal is never less than

// any other IndexVal; therefore, if exactly one of the

// two arguments has a negative index, the min is the other

// argument. Like ordinary arithmetic min and max, this function

// is commutative and associative; that is,

//

// min(A, B) == min(B, A) // commutative

// min(A, min(B, C)) == min((A, B), C) // associative

//

// The effect of

// IndexVal max(IndexedVal a, IndexedVal b)

// is analogous (greater . . . never greater than).

//

// Then there is

//

// MinAndMax minmax(MinAndMax a, MinAndMax b) {

// return MinAndMax(min(a.min, b.min), max(a.max, b.max));

// }

//

// Like ordinary arithmetic min and max, the above function

// is commutative and associative; that is:

//

// minmax(A, B) == minmax(B, A) // commutative

// minmax(A, minmax(B, C)) == minmax((A, B), C) // associative

//

// Finally define

//

// MinAndMax minmax(MinAndMax accum, IndexedVal val) {

// return minmax(accum, MinAndMax(val, val));

// }

// This function can be explained as doing:

// *accum = minmax(*accum, IndexedVal(in, x))

//

// This function simply computes minimum and maximum values as if

// INITVAL.min were greater than any other minimum value and

// INITVAL.max were less than any other maximum value. Note that if

// *accum is INITVAL, then this function sets

// *accum = IndexedVal(in, x)

//

// After this function is called, both accum->min.idx and accum->max.idx

// will have nonnegative values:

// - x is always nonnegative, so if this function ever sets one of the

// idx fields, it will set it to a nonnegative value

// - if one of the idx fields is negative, then the corresponding

// val field must be LONG_MAX or LONG_MIN, so the function will always

// set both the val and idx fields

static void fMMAccumulator(MinAndMax *accum, long in, int x) {

 IndexVal me;

 me.val = in;

 me.idx = x;

 if (me.val <= accum->min.val)

 accum->min = me;

 if (me.val >= accum->max.val)

 accum->max = me;

}

// This function can be explained as doing:

// *accum = minmax(*accum, *val)

```

```

// This function simply computes minimum and maximum values as if
// INITVAL.min were greater than any other minimum value and
// INITVAL.max were less than any other maximum value. Note that if
// one of the two accumulator data items is INITVAL, then this
// function sets *accum to the other one.
static void fMMCombiner(MinAndMax *accum,
 const MinAndMax *val) {
 if ((accum->min.idx < 0) || (val->min.val < accum->min.val))
 accum->min = val->min;
 if ((accum->max.idx < 0) || (val->max.val > accum->max.val))
 accum->max = val->max;
}

static void fMMOutConverter(int2 *result,
 const MinAndMax *val) {
 result->x = val->min.idx;
 result->y = val->max.idx;
}

```

**NOTE:** There are more example reduction kernels [here](#).

In order to run a reduction kernel, the RenderScript runtime creates *one or more* variables called **accumulator data items** to hold the state of the reduction process. The RenderScript runtime picks the number of accumulator data items in such a way as to maximize performance. The type of the accumulator data items (*accumType*) is determined by the kernel's *accumulator function* -- the first argument to that function is a pointer to an accumulator data item. By default, every accumulator data item is initialized to zero (as if by `memset`); however, you may write an *initializer function* to do something different.

**Example:** In the `addint` kernel, the accumulator data items (of type `int`) are used to add up input values. There is no initializer function, so each accumulator data item is initialized to zero.

**Example:** In the `findMinAndMax` kernel, the accumulator data items (of type `MinAndMax`) are used to keep track of the minimum and maximum values found so far. There is an initializer function to set these to `LONG_MAX` and `LONG_MIN`, respectively; and to set the locations of these values to -1, indicating that the values are not actually present in the (empty) portion of the input that has been processed.

RenderScript calls your accumulator function once for every coordinate in the input(s). Typically, your function should update the accumulator data item in some way according to the input.

**Example:** In the `addint` kernel, the accumulator function adds the value of an input Element to the accumulator data item.

**Example:** In the `findMinAndMax` kernel, the accumulator function checks to see whether the value of an input Element is less than or equal to the minimum value recorded in the accumulator data item and/or greater than or equal to the maximum value recorded in the accumulator data item, and updates the accumulator data item accordingly.

After the accumulator function has been called once for every coordinate in the input(s), RenderScript must **combine** the **accumulator data items** together into a single accumulator data item. You may write a *combiner function* to do this. If the accumulator function has a single input and no *special arguments*, then you do not need to write a combiner function; RenderScript will use the accumulator function to combine the accumulator data items. (You may still write a combiner function if this default behavior is not what you want.)

**Example:** In the `addint` kernel, there is no combiner function, so the accumulator function will be used. This is the correct behavior, because if we split a collection of values into two pieces, and we add up the values in those two pieces separately, adding up those two sums is the same as adding up the entire collection.

**Example:** In the `findMinAndMax` kernel, the combiner function checks to see whether the minimum value recorded in the "source" accumulator data item `*val` is less than the minimum value recorded in the "destination" accumulator data item `*accum`, and updates `*accum` accordingly. It does similar work for the maximum value. This updates `*accum` to the state it would have had if all of the input values had been accumulated into `*accum` rather than some into `*accum` and some into `*val`.

After all of the accumulator data items have been combined, RenderScript determines the result of the reduction to return to Java. You may write an *outconverter function* to do this. You do not need to write an outconverter function if you want the final value of the combined accumulator data items to be the result of the reduction.

**Example:** In the `addint` kernel, there is no outconverter function. The final value of the combined data items is the sum of all Elements of the input, which is the value we want to return.

**Example:** In the `findMinAndMax` kernel, the outconverter function initializes an `int2` result value to hold the locations of the minimum and maximum values resulting from the combination of all of the accumulator data items.

## Writing a reduction kernel

`#pragma rs reduce` defines a reduction kernel by specifying its name and the names and roles of the functions that make up the kernel. All such functions must be `static`. A reduction kernel always requires an `accumulator` function; you can omit some or all of the other functions, depending on what you want the kernel to do.

```
#pragma rs reduce(kernelName) \
 initializer(initializerName) \
 accumulator(accumulatorName) \
 combiner(combinerName) \
 outconverter(outconverterName)
```

The meaning of the items in the `#pragma` is as follows:

- `reduce(kernelName)` (mandatory): Specifies that a reduction kernel is being defined. A reflected Java method `reduce_kernelName` will launch the kernel.
- `initializer(initializerName)` (optional): Specifies the name of the initializer function for this reduction kernel. When you launch the kernel, RenderScript calls this function once for each `accumulator data item`. The function must be defined like this:

```
static void initializerName(accumType *accum) { ... }
```

`accum` is a pointer to an accumulator data item for this function to initialize.

If you do not provide an initializer function, RenderScript initializes every accumulator data item to zero (as if by `memset`), behaving as if there were an initializer function that looks like this:

```
static void initializerName(accumType *accum) {
 memset(accum, 0, sizeof(*accum));
}
```

- `accumulator(accumulatorName)` (mandatory): Specifies the name of the accumulator function for this reduction kernel. When you launch the kernel, RenderScript calls this function once for every coordinate in the input(s), to update an accumulator data item in some way according to the input(s). The function must be defined like this:

```
static void accumulatorName(accumType *accum,
 in1Type in1, ..., inNType inN
 [, specialArguments]) { ... }
```

`accum` is a pointer to an accumulator data item for this function to modify. `in1` through `inN` are one or more arguments that are automatically filled in based on the inputs passed to the kernel launch, one argument per input. The accumulator function may optionally take any of the `special arguments`.

An example kernel with multiple inputs is `dotProduct`.

- `combiner(combinerName)` (optional): Specifies the name of the combiner function for this reduction kernel. After RenderScript calls the accumulator function once for every coordinate in the input(s), it calls this function as many times as necessary to combine all accumulator data items into a single accumulator data item. The function must be defined like this:

```
static void combinerName(accumType *accum, const accumType *other) { ... }
```

`accum` is a pointer to a "destination" accumulator data item for this function to modify. `other` is a pointer to a "source" accumulator data item for this function to "combine" into `*accum`.

**NOTE:** It is possible that `*accum`, `*other`, or both have been initialized but have never been passed to the accumulator function; that

is, one or both have never been updated according to any input data. For example, in the `findMinAndMax` kernel, the combiner function `fMMCombiner` explicitly checks for `idx < 0` because that indicates such an accumulator data item, whose value is `INITVAL`.

If you do not provide a combiner function, RenderScript uses the accumulator function in its place, behaving as if there were a combiner function that looks like this:

```
static void combinerName(accumType *accum, const accumType *other) {
 accumulatorName(accum, *other);
}
```

A combiner function is mandatory if the kernel has more than one input, if the input data type is not the same as the accumulator data type, or if the accumulator function takes one or more [special arguments](#).

- `outconverter(outconverterName)` (optional): Specifies the name of the outconverter function for this reduction kernel. After RenderScript combines all of the accumulator data items, it calls this function to determine the result of the reduction to return to Java. The function must be defined like this:

```
static void outconverterName(resultType *result, const accumType *accum) { ... }
```

`result` is a pointer to a result data item (allocated but not initialized by the RenderScript runtime) for this function to initialize with the result of the reduction. `resultType` is the type of that data item, which need not be the same as `accumType`. `accum` is a pointer to the final accumulator data item computed by the [combiner function](#).

If you do not provide an outconverter function, RenderScript copies the final accumulator data item to the result data item, behaving as if there were an outconverter function that looks like this:

```
static void outconverterName(accumType *result, const accumType *accum) {
 *result = *accum;
}
```

If you want a different result type than the accumulator data type, then the outconverter function is mandatory.

Note that a kernel has input types, an accumulator data item type, and a result type, none of which need to be the same. For example, in the `findMinAndMax` kernel, the input type `long`, accumulator data item type `MinAndMax`, and result type `int2` are all different.

## What can't you assume?

You must not rely on the number of accumulator data items created by RenderScript for a given kernel launch. There is no guarantee that two launches of the same kernel with the same input(s) will create the same number of accumulator data items.

You must not rely on the order in which RenderScript calls the initializer, accumulator, and combiner functions; it may even call some of them in parallel. There is no guarantee that two launches of the same kernel with the same input will follow the same order. The only guarantee is that only the initializer function will ever see an uninitialized accumulator data item. For example:

- There is no guarantee that all accumulator data items will be initialized before the accumulator function is called, although it will only be called on an initialized accumulator data item.
- There is no guarantee on the order in which input Elements are passed to the accumulator function.
- There is no guarantee that the accumulator function has been called for all input Elements before the combiner function is called.

One consequence of this is that the `findMinAndMax` kernel is not deterministic: If the input contains more than one occurrence of the same minimum or maximum value, you have no way of knowing which occurrence the kernel will find.

## What must you guarantee?

Because the RenderScript system can choose to execute a kernel [in many different ways](#), you must follow certain rules to ensure that your kernel behaves the way you want. If you do not follow these rules, you may get incorrect results, nondeterministic behavior, or runtime errors.

The rules below often say that two accumulator data items must have "[the same value](#)". What does this mean? That depends on what you want the kernel to do. For a mathematical reduction such as `addint`, it usually makes sense for "the same" to mean mathematical equality. For a "pick any" search such as `findMinAndMax` ("find the location of minimum and maximum input values") where there might be more than one occurrence of identical input values, all locations of a given input value must be considered "the same". You could write a similar kernel to "find the location of *leftmost* minimum and maximum input values" where (say) a minimum value at location 100 is preferred over an identical

minimum value at location 200; for this kernel, "the same" would mean identical *location*, not merely identical *value*, and the accumulator and combiner functions would have to be different than those for [findMinAndMax](#).

**The initializer function must create an *identity value*.** That is, if *A* and *I* are accumulator data items initialized by the initializer function, and *I* has never been passed to the accumulator function (but *A* may have been), then

- *combinerName(&A, &I)* must leave *A* [the same](#)
- *combinerName(&I, &A)* must leave *I* [the same](#) as *A*

**Example:** In the [addint](#) kernel, an accumulator data item is initialized to zero. The combiner function for this kernel performs addition; zero is the identity value for addition.

**Example:** In the [findMinAndMax](#) kernel, an accumulator data item is initialized to [INITVAL](#).

- *fMMCombiner(&A, &I)* leaves *A* the same, because *I* is [INITVAL](#).
- *fMMCombiner(&I, &A)* sets *I* to *A*, because *I* is [INITVAL](#).

Therefore, [INITVAL](#) is indeed an identity value.

**The combiner function must be *commutative*.** That is, if *A* and *B* are accumulator data items initialized by the initializer function, and that may have been passed to the accumulator function zero or more times, then *combinerName(&A, &B)* must set *A* to [the same value](#) that *combinerName(&B, &A)* sets *B*.

**Example:** In the [addint](#) kernel, the combiner function adds the two accumulator data item values; addition is commutative.

**Example:** In the [findMinAndMax](#) kernel,

```
fMMCombiner(&A, &B)
```

is the same as

```
A = minmax(A, B)
```

and [minmax](#) is commutative, so [fMMCombiner](#) is also.

**The combiner function must be *associative*.** That is, if *A*, *B*, and *C* are accumulator data items initialized by the initializer function, and that may have been passed to the accumulator function zero or more times, then the following two code sequences must set *A* to [the same value](#):

```
combinerName(&A, &B);
combinerName(&A, &C);
```

```
combinerName(&B, &C);
combinerName(&A, &B);
```

**Example:** In the [addint](#) kernel, the combiner function adds the two accumulator data item values:

```
A = A + B
A = A + C
// Same as
// A = (A + B) + C
```

```
B = B + C
A = A + B
// Same as
// A = A + (B + C)
// B = B + C
```

Addition is associative, and so the combiner function is also.

**Example:** In the `findMinAndMax` kernel,

```
fMMCombiner(&A, &B)
```

is the same as

```
A = minmax(A, B)
```

So the two sequences are

```
A = minmax(A, B)
A = minmax(A, C)
// Same as
// A = minmax(minmax(A, B), C)
```

```
B = minmax(B, C)
A = minmax(A, B)
// Same as
// A = minmax(A, minmax(B, C))
// B = minmax(B, C)
```

`minmax` is associative, and so `fMMCombiner` is also.

The accumulator function and combiner function together must obey the **basic folding rule**. That is, if `A` and `B` are accumulator data items, `A` has been initialized by the initializer function and may have been passed to the accumulator function zero or more times, `B` has not been initialized, and `args` is the list of input arguments and special arguments for a particular call to the accumulator function, then the following two code sequences must set `A` to the same value:

```
accumulatorName(&A, args); // statement 1
```

```
initializerName(&B); // statement 2
accumulatorName(&B, args); // statement 3
combinerName(&A, &B); // statement 4
```

**Example:** In the `addint` kernel, for an input value `V`:

- Statement 1 is the same as `A += V`
- Statement 2 is the same as `B = 0`
- Statement 3 is the same as `B += V`, which is the same as `B = V`
- Statement 4 is the same as `A += B`, which is the same as `A += V`

Statements 1 and 4 set `A` to the same value, and so this kernel obeys the basic folding rule.

**Example:** In the `findMinAndMax` kernel, for an input value `V` at coordinate `X`:

- Statement 1 is the same as `A = minmax(A, IndexedVal(V, X))`
- Statement 2 is the same as `B = INITVAL`
- Statement 3 is the same as

```
B = minmax(B, IndexedVal(V, X))
```

which, because `B` is the initial value, is the same as

```
B = IndexedVal(V, X)
```

- Statement 4 is the same as

```
A = minmax(A, B)
```

which is the same as

```
A = minmax(A, IndexedVal(V, X))
```

Statements 1 and 4 set `A` to the same value, and so this kernel obeys the basic folding rule.

## Calling a reduction kernel from Java code

For a reduction kernel named `kernelName` defined in the file `filename.rs`, there are three methods reflected in the class `ScriptC_filename`:

```
// Method 1
public javaFutureType reduce_kernelName(Allocation ain1, ...,
 Allocation ainN);

// Method 2
public javaFutureType reduce_kernelName(Allocation ain1, ...,
 Allocation ainN,
 Script.LaunchOptions sc);

// Method 3
public javaFutureType reduce_kernelName(devecSiIn1Type[] in1, ...,
 devecSiInNType[] inN);
```

Here are some examples of calling the `addint` kernel:

```
ScriptC_example script = new ScriptC_example(mRenderScript);

// 1D array
// and obtain answer immediately
int input1[] = ...;
int sum1 = script.reduce_addint(input1).get(); // Method 3

// 2D allocation
// and do some additional work before obtaining answer
Type.Builder typeBuilder =
 new Type.Builder(RS, Element.I32(RS));
typeBuilder.setX(...);
typeBuilder.setY(...);
Allocation input2 = createTyped(RS, typeBuilder.create());
populateSomehow(input2); // fill in input Allocation with data
script.result_int result2 = script.reduce_addint(input2); // Method 1
doSomeAdditionalWork(); // might run at same time as reduction
int sum2 = result2.get();
```

**Method 1** has one input `Allocation` argument for every input argument in the kernel's [accumulator function](#). The RenderScript runtime checks to ensure that all of the input Allocations have the same dimensions and that the `Element` type of each of the input Allocations matches that of the corresponding input argument of the accumulator function's prototype. If any of these checks fail, RenderScript throws an exception. The kernel executes over every coordinate in those dimensions.

**Method 2** is the same as Method 1 except that Method 2 takes an additional argument `sc` that can be used to limit the kernel execution to a subset of the coordinates.

**Method 3** is the same as Method 1 except that instead of taking `Allocation` inputs it takes Java array inputs. This is a convenience that saves you from having to write code to explicitly create an `Allocation` and copy data to it from a Java array. *However, using Method 3 instead of Method 1 does not increase the performance of the code.* For each input array, Method 3 creates a temporary 1-dimensional `Allocation` with the appropriate `Element` type and `setAutoPadding(boolean)` enabled, and copies the array to the `Allocation` as if by the appropriate `copyFrom()` method of `Allocation`. It then calls Method 1, passing those temporary `Allocations`.

**NOTE:** If your application will make multiple kernel calls with the same array, or with different arrays of the same dimensions and `Element` type, you may improve performance by explicitly creating, populating, and reusing `Allocations` yourself, instead of by using Method 3.

**javaFutureType**, the return type of the reflected reduction methods, is a reflected static nested class within the **ScriptC\_filename** class. It represents the future result of a reduction kernel run. To obtain the actual result of the run, call the **get()** method of that class, which returns a value of type **javaResultType**. **get()** is **synchronous**.

```
public class ScriptC_filename extends Scriptc {
 public static class javaFutureType {
 public javaResultType get() { ... }
 }
}
```

**javaResultType** is determined from the *resultType* of the **outconverter** function. Unless *resultType* is an unsigned type (scalar, vector, or array), **javaResultType** is the directly corresponding Java type. If *resultType* is an unsigned type and there is a larger Java signed type, then **javaResultType** is that larger Java signed type; otherwise, it is the directly corresponding Java type. For example:

- If *resultType* is **int**, **int2**, or **int[15]**, then **javaResultType** is **int**, **Int2**, or **int[]**. All values of *resultType* can be represented by **javaResultType**.
- If *resultType* is **uint**, **uint2**, or **uint[15]**, then **javaResultType** is **long**, **Long2**, or **long[]**. All values of *resultType* can be represented by **javaResultType**.
- If *resultType* is **ulong**, **ulong2**, or **ulong[15]**, then **javaResultType** is **long**, **Long2**, or **long[]**. There are certain values of *resultType* that cannot be represented by **javaResultType**.

**javaFutureType** is the future result type corresponding to the *resultType* of the **outconverter** function.

- If *resultType* is not an array type, then **javaFutureType** is **result\_resultType**.
- If *resultType* is an array of length *Count* with members of type *memberType*, then **javaFutureType** is **resultArrayCount\_memberType**.

For example:

```

public class ScriptC_filename extends Scriptc {
 // for kernels with int result
 public static class result_int {
 public int get() { ... }
 }

 // for kernels with int[10] result
 public static class resultArray10_int {
 public int[] get() { ... }
 }

 // for kernels with int2 result
 // note that the Java type name "Int2" is not the same as the script type name "int2"
 public static class result_int2 {
 public Int2 get() { ... }
 }

 // for kernels with int2[10] result
 // note that the Java type name "Int2" is not the same as the script type name "int2"
 public static class resultArray10_int2 {
 public Int2[] get() { ... }
 }

 // for kernels with uint result
 // note that the Java type "long" is a wider signed type than the unsigned script type "uint"
 public static class result_uint {
 public long get() { ... }
 }

 // for kernels with uint[10] result
 // note that the Java type "long" is a wider signed type than the unsigned script type "uint"
 public static class resultArray10_uint {
 public long[] get() { ... }
 }

 // for kernels with uint2 result
 // note that the Java type "Long2" is a wider signed type than the unsigned script type "uint2"
 public static class result_uint2 {
 public Long2 get() { ... }
 }

 // for kernels with uint2[10] result
 // note that the Java type "Long2" is a wider signed type than the unsigned script type "uint2"
 public static class resultArray10_uint2 {
 public Long2[] get() { ... }
 }
}

```

If `javaResultType` is an object type (including an array type), each call to `javaFutureType.get()` on the same instance will return the same object.

If `javaResultType` cannot represent all values of type `resultType`, and a reduction kernel produces an unrepresentable value, then `javaFutureType.get()` throws an exception.

### Method 3 and `devecSilnXType`

**devecSilnXType** is the Java type corresponding to the `inXType` of the corresponding argument of the [accumulator function](#). Unless `inXType` is an unsigned type or a vector type, `devecSilnXType` is the directly corresponding Java type. If `inXType` is an unsigned scalar type, then `devecSilnXType` is the Java type directly corresponding to the signed scalar type of the same size. If `inXType` is a signed vector type, then `devecSilnXType` is the Java type directly corresponding to the vector component type. If `inXType` is an unsigned vector type, then `devecSilnXType` is the Java type directly corresponding to the signed scalar type of the same size as the vector component type. For example:

- If `inXType` is `int`, then `devecSilnXType` is `int`.
- If `inXType` is `int2`, then `devecSilnXType` is `int`. The array is a *flattened* representation: It has twice as many *scalar* Elements as the Allocation has 2-component *vector* Elements. This is the same way that the `copyFrom()` methods of [Allocation](#) work.

- If *inXType* is `uint`, then *deviceSilnXType* is `int`. A signed value in the Java array is interpreted as an unsigned value of the same bitpattern in the Allocation. This is the same way that the `copyFrom( )` methods of `Allocation` work.
- If *inXType* is `uint2`, then *deviceSilnXType* is `int`. This is a combination of the way `int2` and `uint` are handled: The array is a flattened representation, and Java array signed values are interpreted as RenderScript unsigned Element values.

Note that for [Method 3](#), input types are handled differently than result types:

- A script's vector input is flattened on the Java side, whereas a script's vector result is not.
- A script's unsigned input is represented as a signed input of the same size on the Java side, whereas a script's unsigned result is represented as a widened signed type on the Java side (except in the case of `ulong`).

## More example reduction kernels

```
#pragma rs reduce(dotProduct) \
 accumulator(dotProductAccum) combiner(dotProductSum)

// Note: No initializer function -- therefore,
// each accumulator data item is implicitly initialized to 0.0f.

static void dotProductAccum(float *accum, float in1, float in2) {
 *accum += in1*in2;
}

// combiner function
static void dotProductSum(float *accum, const float *val) {
 *accum += *val;
}
```

```
// Find a zero Element in a 2D allocation; return (-1, -1) if none
#pragma rs reduce(fz2) \
 initializer(fz2Init) \
 accumulator(fz2Accum) combiner(fz2Combine)

static void fz2Init(int2 *accum) { accum->x = accum->y = -1; }

static void fz2Accum(int2 *accum,
 int inVal,
 int x /* special arg */,
 int y /* special arg */) {
 if (inVal==0) {
 accum->x = x;
 accum->y = y;
 }
}

static void fz2Combine(int2 *accum, const int2 *accum2) {
 if (accum2->x >= 0) *accum = *accum2;
}
```

```

// Note that this kernel returns an array to Java
#pragma rs reduce(histogram) \
 accumulator(hsgAccum) combiner(hsgCombine)

#define BUCKETS 256
typedef uint32_t Histogram[BUCKETS];

// Note: No initializer function --
// therefore, each bucket is implicitly initialized to 0.

static void hsgAccum(Histogram *h, uchar in) { ++(*h)[in]; }

static void hsgCombine(Histogram *accum,
 const Histogram *addend) {
 for (int i = 0; i < BUCKETS; ++i)
 (*accum)[i] += (*addend)[i];
}

// Determines the mode (most frequently occurring value), and returns
// the value and the frequency.
//
// If multiple values have the same highest frequency, returns the lowest
// of those values.
//
// Shares functions with the histogram reduction kernel.
#pragma rs reduce(mode) \
 accumulator(hsgAccum) combiner(hsgCombine) \
 outconverter(modeOutConvert)

static void modeOutConvert(int2 *result, const Histogram *h) {
 uint32_t mode = 0;
 for (int i = 1; i < BUCKETS; ++i)
 if ((*h)[i] > (*h)[mode]) mode = i;
 result->x = mode;
 result->y = (*h)[mode];
}

```



# Advanced RenderScript

In this document

- [RenderScript Runtime Layer](#)
- [Reflected Layer](#)
  - [Functions](#)
  - [Variables](#)
  - [Pointers](#)
  - [Structs](#)
- [Memory Allocation APIs](#)
- [Working with Memory](#)
  - [Allocating and binding memory to the RenderScript](#)
  - [Reading and writing to memory](#)

Because applications that utilize RenderScript still run inside of the Android VM, you have access to all of the framework APIs that you are familiar with, but can utilize RenderScript when appropriate. To facilitate this interaction between the framework and the RenderScript runtime, an intermediate layer of code is also present to facilitate communication and memory management between the two levels of code. This document goes into more detail about these different layers of code as well as how memory is shared between the Android VM and RenderScript runtime.

## RenderScript Runtime Layer

Your RenderScript code is compiled and executed in a compact and well-defined runtime layer. The RenderScript runtime APIs offer support for intensive computation that is portable and automatically scalable to the amount of cores available on a processor.

**Note:** The standard C functions in the NDK must be guaranteed to run on a CPU, so RenderScript cannot access these libraries, because RenderScript is designed to run on different types of processors.

You define your RenderScript code in `.rs` and `.rsh` files in the `src/` directory of your Android project. The code is compiled to intermediate bytecode by the `l1vm` compiler that runs as part of an Android build. When your application runs on a device, the bytecode is then compiled (just-in-time) to machine code by another `l1vm` compiler that resides on the device. The machine code is optimized for the device and also cached, so subsequent uses of the RenderScript enabled application do not recompile the bytecode.

Some key features of the RenderScript runtime libraries include:

- Memory allocation request features
- A large collection of math functions with both scalar and vector typed overloaded versions of many common routines. Operations such as adding, multiplying, dot product, and cross product are available as well as atomic arithmetic and comparison functions.
- Conversion routines for primitive data types and vectors, matrix routines, and date and time routines
- Data types and structures to support the RenderScript system such as Vector types for defining two-, three-, or four-vectors.
- Logging functions

See the RenderScript runtime API reference for more information on the available functions.

## Reflected Layer

The reflected layer is a set of classes that the Android build tools generate to allow access to the RenderScript runtime from the Android framework. This layer also provides methods and constructors that allow you to allocate and work with memory for pointers that are defined in your RenderScript code. The following list describes the major components that are reflected:

- Every `.rs` file that you create is generated into a class named `project_root/gen/package/name/ScriptC_renderscript_filename` of type `ScriptC`. This file is the `.java` version of your `.rs` file, which you can call from the Android framework. This class contains the following items reflected from the `.rs` file:
  - Non-static functions
  - Non-static, global RenderScript variables. Accessor methods are generated for each variable, so you can read and write the RenderScript variables from the Android framework. If a global variable is initialized at the RenderScript runtime layer, those values are used to initialize the corresponding values in the Android framework layer. If global variables are marked as `const`, then a `set` method is not generated. Look [here](#) for more details.
  - Global pointers
- A `struct` is reflected into its own class named `project_root/gen/package/name/ScriptField_struct_name`, which extends `Script.FieldBase`. This class represents an array of the `struct`, which allows you to allocate memory for one or more instances of this `struct`.

## Functions

Functions are reflected into the script class itself, located in `project_root/gen/package/name/ScriptC_renderscript_filename`. For example, if you define the following function in your RenderScript code:

```
void touch(float x, float y, float pressure, int id) {
 if (id >= 10) {
 return;
 }

 touchPos[id].x = x;
 touchPos[id].y = y;
 touchPressure[id] = pressure;
}
```

then the following Java code is generated:

```
public void invoke_touch(float x, float y, float pressure, int id) {
 FieldPacker touch_fp = new FieldPacker(16);
 touch_fp.addF32(x);
 touch_fp.addF32(y);
 touch_fp.addF32(pressure);
 touch_fp.addI32(id);
 invoke(mExportFuncIdx_touch, touch_fp);
}
```

Functions cannot have return values, because the RenderScript system is designed to be asynchronous. When your Android framework code calls into RenderScript, the call is queued and is executed when possible. This restriction allows the RenderScript system to function without constant interruption and increases efficiency. If functions were allowed to have return values, the call would block until the value was returned.

If you want the RenderScript code to send a value back to the Android framework, use the `rsSendToClient()` function.

## Variables

Variables of supported types are reflected into the script class itself, located in `project_root/gen/package/name/ScriptC_renderscript_filename`. A set of accessor methods is generated for each variable. For example, if you define the following variable in your RenderScript code:

```
uint32_t unsignedInteger = 1;
```

then the following Java code is generated:

```
private long mExportVar_unsignedInteger;
public void set_unsignedInteger(long v){
 mExportVar_unsignedInteger = v;
 setVar(mExportVarIdx_unsignedInteger, v);
}

public long get_unsignedInteger(){
 return mExportVar_unsignedInteger;
}
```

## Structs

Structs are reflected into their own classes, located in <project\_root>/gen/com/example/renderScript/ScriptField\_struct\_name. This class represents an array of the **struct** and allows you to allocate memory for a specified number of **structs**. For example, if you define the following struct:

```
typedef struct Point {
 float2 position;
 float size;
} Point_t;
```

then the following code is generated in **ScriptField\_Point.java**:

```
package com.example.android.rs.hellocompute;

import android.renderscript.*;
import android.content.res.Resources;

/**
 * @hide
 */
public class ScriptField_Point extends android.renderscript.Script.FieldBase {

 static public class Item {
 public static final int sizeof = 12;

 Float2 position;
 float size;

 Item() {
 position = new Float2();
 }
 }

 private Item mItemArray[];
 private FieldPacker mIOBuffer;
 public static Element createElement(RenderScript rs) {
 Element.Builder eb = new Element.Builder(rs);
 eb.add(Element.F32_2(rs), "position");
 eb.add(Element.F32(rs), "size");
 return eb.create();
 }

 public ScriptField_Point(RenderScript rs, int count) {
 mItemArray = null;
 mIOBuffer = null;
 mElement = createElement(rs);
 init(rs, count);
 }

 public ScriptField_Point(RenderScript rs, int count, int usages) {
 mItemArray = null;
 mIOBuffer = null;
 mElement = createElement(rs);
 init(rs, count, usages);
 }
}
```

```

}

private void copyToArray(Item i, int index) {
 if (mIOBuffer == null) mIOBuffer = new FieldPacker(Item.sizeof * getType().getX()/* count */);
 mIOBuffer.reset(index * Item.sizeof);
 mIOBuffer.addF32(i.position);
 mIOBuffer.addF32(i.size);
}

public void set(Item i, int index, boolean copyNow) {
 if (mItemArray == null) mItemArray = new Item[getType().getX() /* count */];
 mItemArray[index] = i;
 if (copyNow) {
 copyToArray(i, index);
 mAllocation.setFromFieldPacker(index, mIOBuffer);
 }
}

public Item get(int index) {
 if (mItemArray == null) return null;
 return mItemArray[index];
}

public void set_position(int index, Float2 v, boolean copyNow) {
 if (mIOBuffer == null) mIOBuffer = new FieldPacker(Item.sizeof * getType().getX()/* count */);
 if (mItemArray == null) mItemArray = new Item[getType().getX() /* count */];
 if (mItemArray[index] == null) mItemArray[index] = new Item();
 mItemArray[index].position = v;
 if (copyNow) {
 mIOBuffer.reset(index * Item.sizeof);
 mIOBuffer.addF32(v);
 FieldPacker fp = new FieldPacker(8);
 fp.addF32(v);
 mAllocation.setFromFieldPacker(index, 0, fp);
 }
}

public void set_size(int index, float v, boolean copyNow) {
 if (mIOBuffer == null) mIOBuffer = new FieldPacker(Item.sizeof * getType().getX()/* count */);
 if (mItemArray == null) mItemArray = new Item[getType().getX() /* count */];
 if (mItemArray[index] == null) mItemArray[index] = new Item();
 mItemArray[index].size = v;
 if (copyNow) {
 mIOBuffer.reset(index * Item.sizeof + 8);
 mIOBuffer.addF32(v);
 FieldPacker fp = new FieldPacker(4);
 fp.addF32(v);
 mAllocation.setFromFieldPacker(index, 1, fp);
 }
}

public Float2 get_position(int index) {
 if (mItemArray == null) return null;
 return mItemArray[index].position;
}

public float get_size(int index) {
 if (mItemArray == null) return 0;
 return mItemArray[index].size;
}

public void copyAll() {
 for (int ct = 0; ct < mItemArray.length; ct++) copyToArray(mItemArray[ct], ct);
 mAllocation.setFromFieldPacker(0, mIOBuffer);
}

public void resize(int newSize) {
 if (mItemArray != null) {
 int oldSize = mItemArray.length;
 int copySize = Math.min(oldSize, newSize);
 if (newSize == oldSize) return;
 Item newItem = new Item[newSize];
 for (int i = 0; i < copySize; i++) newItem[i] = mItemArray[i];
 mItemArray = newItem;
 }
}

```

```

 itemIndex = new Item[newSize];
 System.arraycopy(mItemArray, 0, ni, 0, copySize);
 mItemArray = ni;
 }
 mAllocation.resize(newSize);
 if (mIOBuffer != null) mIOBuffer = new FieldPacker(Item.sizeof * getType().getX()/* count */);
}
}

```

The generated code is provided to you as a convenience to allocate memory for structs requested by the RenderScript runtime and to interact with `structs` in memory. Each `struct`'s class defines the following methods and constructors:

- Overloaded constructors that allow you to allocate memory. The `ScriptField_struct_name(RenderScript rs, int count)` constructor allows you to define the number of structures that you want to allocate memory for with the `count` parameter. The `ScriptField_struct_name(RenderScript rs, int count, int usages)` constructor defines an extra parameter, `usages`, that lets you specify the memory space of this memory allocation. There are four memory space possibilities:
  - `USAGE_SCRIPT`: Allocates in the script memory space. This is the default memory space if you do not specify a memory space.
  - `USAGE_GRAPHICS_TEXTURE`: Allocates in the texture memory space of the GPU.
  - `USAGE_GRAPHICS_VERTEX`: Allocates in the vertex memory space of the GPU.
  - `USAGE_GRAPHICS_CONSTANTS`: Allocates in the constants memory space of the GPU that is used by the various program objects.

You can specify multiple memory spaces by using the bitwise `OR` operator. Doing so notifies the RenderScript runtime that you intend on accessing the data in the specified memory spaces. The following example allocates memory for a custom data type in both the script and vertex memory spaces:

```

ScriptField_Point touchPoints = new ScriptField_Point(myRenderScript, 2,
Allocation.USAGE_SCRIPT | Allocation.USAGE_GRAPHICS_VERTEX);

```

- A static nested class, `Item`, allows you to create an instance of the `struct`, in the form of an object. This nested class is useful if it makes more sense to work with the `struct` in your Android code. When you are done manipulating the object, you can push the object to the allocated memory by calling `set(Item i, int index, boolean copyNow)` and setting the `Item` to the desired position in the array. The RenderScript runtime automatically has access to the newly written memory.
- Accessor methods to get and set the values of each field in a struct. Each of these accessor methods has an `index` parameter to specify the `struct` in the array that you want to read or write to. Each setter method also has a `copyNow` parameter that specifies whether or not to immediately sync this memory to the RenderScript runtime. To sync any memory that has not been synced, call `copyAll()`.
- The `createElement()` method creates a description of the struct in memory. This description is used to allocate memory consisting of one or many elements.
- `resize()` works much like a `realloc()` in C, allowing you to expand previously allocated memory, maintaining the current values that were previously created.
- `copyAll()` synchronizes memory that was set on the framework level to the RenderScript runtime. When you call a set accessor method on a member, there is an optional `copyNow` boolean parameter that you can specify. Specifying `true` synchronizes the memory when you call the method. If you specify false, you can call `copyAll()` once, and it synchronizes memory for all the properties that are not yet synchronized.

## Pointers

Global pointers are reflected into the script class itself, located in `project_root/gen/package/name/ScriptC_renderscript_filename`. You can declare pointers to a `struct` or any of the supported RenderScript types, but a `struct` cannot contain pointers or nested arrays. For example, if you define the following pointers to a `struct` and `int32_t`

```

typedef struct Point {
 float2 position;
 float size;
} Point_t;

Point_t *touchPoints;
int32_t *intPointer;

```

then the following Java code is generated:

```

private ScriptField_Point mExportVar_touchPoints;
public void bind_touchPoints(ScriptField_Point v) {
 mExportVar_touchPoints = v;
 if (v == null) bindAllocation(null, mExportVarIdx_touchPoints);
 else bindAllocation(v.getAllocation(), mExportVarIdx_touchPoints);
}

public ScriptField_Point get_touchPoints() {
 return mExportVar_touchPoints;
}

private Allocation mExportVar_intPointer;
public void bind_intPointer(Allocation v) {
 mExportVar_intPointer = v;
 if (v == null) bindAllocation(null, mExportVarIdx_intPointer);
 else bindAllocation(v, mExportVarIdx_intPointer);
}

public Allocation get_intPointer() {
 return mExportVar_intPointer;
}

```

A `get` method and a special method named `bind_pointer_name` (instead of a `set()` method) are generated. The `bind_pointer_name` method allows you to bind the memory that is allocated in the Android VM to the RenderScript runtime (you cannot allocate memory in your `.rs` file). For more information, see [Working with Allocated Memory](#).

## Memory Allocation APIs

Applications that use RenderScript still run in the Android VM. The actual RenderScript code, however, runs natively and needs access to the memory allocated in the Android VM. To accomplish this, you must attach the memory that is allocated in the VM to the RenderScript runtime. This process, called binding, allows the RenderScript runtime to seamlessly work with memory that it requests but cannot explicitly allocate. The end result is essentially the same as if you had called `malloc` in C. The added benefit is that the Android VM can carry out garbage collection as well as share memory with the RenderScript runtime layer. Binding is only necessary for dynamically allocated memory. Statically allocated memory is automatically created for your RenderScript code at compile time. See [Figure 1](#) for more information on how memory allocation occurs.

To support this memory allocation system, there are a set of APIs that allow the Android VM to allocate memory and offer similar functionality to a `malloc` call. These classes essentially describe how memory should be allocated and also carry out the allocation. To better understand how these classes work, it is useful to think of them in relation to a simple `malloc` call that can look like this:

```
array = (int *)malloc(sizeof(int)*10);
```

The `malloc` call can be broken up into two parts: the size of the memory being allocated (`sizeof(int)`), along with how many units of that memory should be allocated (10). The Android framework provides classes for these two parts as well as a class to represent `malloc` itself.

The `Element` class represents the `(sizeof(int))` portion of the `malloc` call and encapsulates one cell of a memory allocation, such as a single float value or a struct. The `Type` class encapsulates the `Element` and the amount of elements to allocate (10 in our example). You can think of a `Type` as an array of `Elements`. The `Allocation` class does the actual memory allocation based on a given `Type` and represents the actual allocated memory.

In most situations, you do not need to call these memory allocation APIs directly. The reflected layer classes generate code to use these APIs

automatically and all you need to do to allocate memory is call a constructor that is declared in one of the reflected layer classes and then bind the resulting memory [Allocation](#) to the RenderScript. There are some situations where you would want to use these classes directly to allocate memory on your own, such as loading a bitmap from a resource or when you want to allocate memory for pointers to primitive types. You can see how to do this in the [Allocating and binding memory to the RenderScript](#) section. The following table describes the three memory management classes in more detail:

| Android Object Type        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">Element</a>    | <p>An element describes one cell of a memory allocation and can have two forms: basic or complex.</p> <p>A basic element contains a single component of data of any valid RenderScript data type. Examples of basic element data types include a single <code>float</code> value, a <code>float4</code> vector, or a single RGB-565 color.</p> <p>Complex elements contain a list of basic elements and are created from <code>structs</code> that you declare in your RenderScript code. For instance an allocation can contain multiple <code>structs</code> arranged in order in memory. Each struct is considered as its own element, rather than each data type within that struct.</p>                                                                                                                                                                                                                              |
| <a href="#">Type</a>       | <p>A type is a memory allocation template and consists of an element and one or more dimensions. It describes the layout of the memory (basically an array of <a href="#">Elements</a>) but does not allocate the memory for the data that it describes.</p> <p>A type consists of five dimensions: X, Y, Z, LOD (level of detail), and Faces (of a cube map). You can set the X,Y,Z dimensions to any positive integer value within the constraints of available memory. A single dimension allocation has an X dimension of greater than zero while the Y and Z dimensions are zero to indicate not present. For example, an allocation of x=10, y=1 is considered two dimensional and x=10, y=0 is considered one dimensional. The LOD and Faces dimensions are booleans to indicate present or not present.</p>                                                                                                       |
| <a href="#">Allocation</a> | <p>An allocation provides the memory for applications based on a description of the memory that is represented by a <a href="#">Type</a>. Allocated memory can exist in many memory spaces concurrently. If memory is modified in one space, you must explicitly synchronize the memory, so that it is updated in all the other spaces in which it exists.</p> <p>Allocation data is uploaded in one of two primary ways: type checked and type unchecked. For simple arrays there are <code>copyFrom()</code> functions that take an array from the Android system and copy it to the native layer memory store. The unchecked variants allow the Android system to copy over arrays of structures because it does not support structures. For example, if there is an allocation that is an array of n floats, the data contained in a <code>float[n]</code> array or a <code>byte[n*4]</code> array can be copied.</p> |

## Working with Memory

Non-static, global variables that you declare in your RenderScript are allocated memory at compile time. You can work with these variables directly in your RenderScript code without having to allocate memory for them at the Android framework level. The Android framework layer also has access to these variables with the provided accessor methods that are generated in the reflected layer classes. If these variables are initialized at the RenderScript runtime layer, those values are used to initialize the corresponding values in the Android framework layer. If global variables are marked as const, then a `set` method is not generated. Look [here](#) for more details.

**Note:** If you are using certain RenderScript structures that contain pointers, such as `rs_program_fragment` and `rs_allocation`, you have to obtain an object of the corresponding Android framework class first and then call the `set` method for that structure to bind the memory to the RenderScript runtime. You cannot directly manipulate these structures at the RenderScript runtime layer. This restriction is not applicable to user-defined structures that contain pointers, because they cannot be exported to a reflected layer class in the first place. A compiler error is generated if you try to declare a non-static, global struct that contains a pointer.

RenderScript also has support for pointers, but you must explicitly allocate the memory in your Android framework code. When you declare a global pointer in your `.rs` file, you allocate memory through the appropriate reflected layer class and bind that memory to the native RenderScript layer. You can interact with this memory from the Android framework layer as well as the RenderScript layer, which offers you the flexibility to modify variables in the most appropriate layer.

## Allocating and binding dynamic memory to the RenderScript

To allocate dynamic memory, you need to call the constructor of a `Script.FieldBase` class, which is the most common way. An alternative is to create an `Allocation` manually, which is required for things such as primitive type pointers. You should use a `Script.FieldBase` class constructor whenever available for simplicity. After obtaining a memory allocation, call the reflected `bind` method of the pointer to bind the allocated memory to the RenderScript runtime.

The example below allocates memory for both a primitive type pointer, `intPointer`, and a pointer to a struct, `touchPoints`. It also binds the memory to the RenderScript:

```
private RenderScript myRenderScript;
private ScriptC_example script;
private Resources resources;

public void init(RenderScript rs, Resources res) {
 myRenderScript = rs;
 resources = res;

 //allocate memory for the struct pointer, calling the constructor
 ScriptField_Point touchPoints = new ScriptField_Point(myRenderScript, 2);

 //Create an element manually and allocate memory for the int pointer
 intPointer = Allocation.createSized(myRenderScript, Element.I32(myRenderScript), 2);

 //create an instance of the RenderScript, pointing it to the bytecode resource
 mScript = new ScriptC_example(myRenderScript, resources, R.raw.example);

 //bind the struct and int pointers to the RenderScript
 mScript.bind_touchPoints(touchPoints);
 script.bind_intPointer(intPointer);

 ...
}
```

## Reading and writing to memory

You can read and write to statically and dynamically allocated memory both at the RenderScript runtime and Android framework layer.

Statically allocated memory comes with a one-way communication restriction at the RenderScript runtime level. When RenderScript code changes the value of a variable, it is not communicated back to the Android framework layer for efficiency purposes. The last value that is set from the Android framework is always returned during a call to a `get` method. However, when Android framework code modifies a variable, that change can be communicated to the RenderScript runtime automatically or synchronized at a later time. If you need to send data from the RenderScript runtime to the Android framework layer, you can use the `rsSendToClient()` function to overcome this limitation.

When working with dynamically allocated memory, any changes at the RenderScript runtime layer are propagated back to the Android framework layer if you modified the memory allocation using its associated pointer. Modifying an object at the Android framework layer immediately propagates that change back to the RenderScript runtime layer.

### Reading and writing to global variables

Reading and writing to global variables is a straightforward process. You can use the accessor methods at the Android framework level or set them directly in the RenderScript code. Keep in mind that any changes that you make in your RenderScript code are not propagated back to the Android framework layer (look [here](#) for more details).

For example, given the following struct declared in a file named `rsfile.rs`:

```
typedef struct Point {
 int x;
 int y;
} Point_t;

Point_t point;
```

You can assign values to the struct like this directly in `rsfile.rs`. These values are not propagated back to the Android framework level:

```
point.x = 1;
point.y = 1;
```

You can assign values to the struct at the Android framework layer like this. These values are propagated back to the RenderScript runtime level **asynchronously**:

```
ScriptC_rsfile mScript;

...
Item i = new ScriptField_Point.Item();
i.x = 1;
i.y = 1;
mScript.set_point(i);
```

You can read the values in your RenderScript code like this:

```
rsDebug("Printing out a Point", point.x, point.y);
```

You can read the values in the Android framework layer with the following code. Keep in mind that this code only returns a value if one was set at the Android framework level. You will get a null pointer exception if you only set the value at the RenderScript runtime level:

```
Log.i("TAGNAME", "Printing out a Point: " + mScript.get_point().x + " " + mScript.get_point().y);
System.out.println(point.get_x() + " " + point.get_y());
```

## Reading and writing global pointers

Assuming that memory has been allocated in the Android framework level and bound to the RenderScript runtime, you can read and write memory from the Android framework level by using the **get** and **set** methods for that pointer. In the RenderScript runtime layer, you can read and write to memory with pointers as normal and the changes are propagated back to the Android framework layer, unlike with statically allocated memory.

For example, given the following pointer to a **struct** in a file named **rsfile.rs**:

```
typedef struct Point {
 int x;
 int y;
} Point_t;

Point_t *point;
```

Assuming you already allocated memory at the Android framework layer, you can access values in the **struct** as normal. Any changes you make to the struct via its pointer variable are automatically available to the Android framework layer:

```
point[index].x = 1;
point[index].y = 1;
```

You can read and write values to the pointer at the Android framework layer as well:

```
ScriptField_Point p = new ScriptField_Point(mRS, 1);
Item i = new ScriptField_Point.Item();
i.x=100;
i.y = 100;
p.set(i, 0, true);
mScript.bind_point(p);

points.get_x(0); //read x and y from index 0
points.get_x(0);
```

Once memory is already bound, you do not have to rebind the memory to the RenderScript runtime every time you make a change to a value.



# RenderScript Runtime API Reference

## Overview

RenderScript is a high-performance runtime that provides compute operations at the native level. RenderScript code is compiled on devices at runtime to allow platform-independence as well.

This reference documentation describes the RenderScript runtime APIs, which you can utilize to write RenderScript code in C99. The RenderScript compute header files are automatically included for you.

To use RenderScript, you need to utilize the RenderScript runtime APIs documented here as well as the Android framework APIs for RenderScript. For documentation on the Android framework APIs, see the [android.renderscript](#) package reference.

For more information on how to develop with RenderScript and how the runtime and Android framework APIs interact, see the [RenderScript developer guide](#) and the [RenderScript samples](#).

## Numerical Types

### Scalars:

RenderScript supports the following scalar numerical types:

|                   | 8 bits                         | 16 bits                          | 32 bits                        | 64 bits                                  |
|-------------------|--------------------------------|----------------------------------|--------------------------------|------------------------------------------|
| Integer:          | char, <a href="#">int8_t</a>   | short, <a href="#">int16_t</a>   | <a href="#">int32_t</a>        | long, long long, <a href="#">int64_t</a> |
| Unsigned integer: | uchar, <a href="#">uint8_t</a> | ushort, <a href="#">uint16_t</a> | uint, <a href="#">uint32_t</a> | ulong, <a href="#">uint64_t</a>          |
| Floating point:   |                                | half                             | float                          | double                                   |

### Vectors:

RenderScript supports fixed size vectors of length 2, 3, and 4. Vectors are declared using the common type name followed by a 2, 3, or 4. E.g. [float4](#), [int3](#), [double2](#), [ulong4](#).

To create vector literals, use the vector type followed by the values enclosed between curly braces, e.g. `(float3){1.0f, 2.0f, 3.0f}`.

Entries of a vector can be accessed using different naming styles.

Single entries can be accessed by following the variable name with a dot and:

- The letters x, y, z, and w,
- The letters r, g, b, and a,
- The letter s or S, followed by a zero based index.

For example, with `int4 myVar`; the following are equivalent:

```
myVar.x == myVar.r == myVar.s0 == myVar.S0
myVar.y == myVar.g == myVar.s1 == myVar.S1
myVar.z == myVar.b == myVar.s2 == myVar.S2
myVar.w == myVar.a == myVar.s3 == myVar.S3
```

Multiple entries of a vector can be accessed at once by using an identifier that is the concatenation of multiple letters or indices. The resulting vector has a size equal to the number of entries named.

With the example above, the middle two entries can be accessed using `myVar.yz`, `myVar.gb`, `myVar.s12`, and `myVar.S12`.

The entries don't have to be contiguous or in increasing order. Entries can even be repeated, as long as we're not trying to assign to it. You also can't mix the naming styles.

Here are examples of what can or can't be done:

```
float4 v4;
float3 v3;
float2 v2;
v2 = v4.xx; // Valid
v3 = v4.zwx; // Valid
v3 = v4.bba; // Valid
v3 = v4.s032; // Valid
v3.s120 = v4.S233; // Valid
v4.yz = v3.rg; // Valid
v4.yzx = v3.rg; // Invalid: mismatched sizes
v4.yzz = v3; // Invalid: z appears twice in an assignment
v3 = v3.xas0; // Invalid: can't mix xyzw with rgba nor s0...
v3 = v4.s034; // Invalid: the digit can only be 0, 1, 2, or 3
```

## Matrices and Quaternions:

RenderScript supports fixed size square matrices of floats of size 2x2, 3x3, and 4x4. The types are named `rs_matrix2x2`, `rs_matrix3x3`, and `rs_matrix4x4`. See [Matrix Functions](#) for the list of operations.

Quaternions are also supported via `rs_quaternion`. See [Quaterion Functions](#) for the list of operations.

| Types                |                              |
|----------------------|------------------------------|
| <code>char2</code>   | Two 8 bit signed integers    |
| <code>char3</code>   | Three 8 bit signed integers  |
| <code>char4</code>   | Four 8 bit signed integers   |
| <code>double2</code> | Two 64 bit floats            |
| <code>double3</code> | Three 64 bit floats          |
| <code>double4</code> | Four 64 bit floats           |
| <code>float2</code>  | Two 32 bit floats            |
| <code>float3</code>  | Three 32 bit floats          |
| <code>float4</code>  | Four 32 bit floats           |
| <code>half</code>    | 16 bit floating point value  |
| <code>half2</code>   | Two 16 bit floats            |
| <code>half3</code>   | Three 16 bit floats          |
| <code>half4</code>   | Four 16 bit floats           |
| <code>int16_t</code> | 16 bit signed integer        |
| <code>int2</code>    | Two 32 bit signed integers   |
| <code>int3</code>    | Three 32 bit signed integers |
| <code>int32_t</code> | 32 bit signed integer        |
| <code>int4</code>    | Four 32 bit signed integers  |
| <code>int64_t</code> | 64 bit signed integer        |
| <code>int8_t</code>  | 8 bit signed integer         |
| <code>long2</code>   | Two 64 bit signed integers   |

|                            |                                |
|----------------------------|--------------------------------|
| <code>long3</code>         | Three 64 bit signed integers   |
| <code>long4</code>         | Four 64 bit signed integers    |
| <code>rs_matrix2x2</code>  | 2x2 matrix of 32 bit floats    |
| <code>rs_matrix3x3</code>  | 3x3 matrix of 32 bit floats    |
| <code>rs_matrix4x4</code>  | 4x4 matrix of 32 bit floats    |
| <code>rs_quaternion</code> | Quaternion                     |
| <code>short2</code>        | Two 16 bit signed integers     |
| <code>short3</code>        | Three 16 bit signed integers   |
| <code>short4</code>        | Four 16 bit signed integers    |
| <code>size_t</code>        | Unsigned size type             |
| <code>ssize_t</code>       | Signed size type               |
| <code>uchar</code>         | 8 bit unsigned integer         |
| <code>uchar2</code>        | Two 8 bit unsigned integers    |
| <code>uchar3</code>        | Three 8 bit unsigned integers  |
| <code>uchar4</code>        | Four 8 bit unsigned integers   |
| <code>uint</code>          | 32 bit unsigned integer        |
| <code>uint16_t</code>      | 16 bit unsigned integer        |
| <code>uint2</code>         | Two 32 bit unsigned integers   |
| <code>uint3</code>         | Three 32 bit unsigned integers |
| <code>uint32_t</code>      | 32 bit unsigned integer        |
| <code>uint4</code>         | Four 32 bit unsigned integers  |
| <code>uint64_t</code>      | 64 bit unsigned integer        |
| <code>uint8_t</code>       | 8 bit unsigned integer         |
| <code>ulong</code>         | 64 bit unsigned integer        |
| <code>ulong2</code>        | Two 64 bit unsigned integers   |
| <code>ulong3</code>        | Three 64 bit unsigned integers |
| <code>ulong4</code>        | Four 64 bit unsigned integers  |
| <code>ushort</code>        | 16 bit unsigned integer        |
| <code>ushort2</code>       | Two 16 bit unsigned integers   |
| <code>ushort3</code>       | Three 16 bit unsigned integers |
| <code>ushort4</code>       | Four 16 bit unsigned integers  |

## Object Types

The types below are used to manipulate RenderScript objects like allocations, samplers, elements, and scripts. Most of these object are created using the Java RenderScript APIs.

| Types                                   |                                               |
|-----------------------------------------|-----------------------------------------------|
| <code>rs_allocation</code>              | Handle to an allocation                       |
| <code>rs_allocation_cubemap_face</code> | Enum for selecting cube map faces             |
| <code>rs_allocation_usage_type</code>   | Bitfield to specify how an allocation is used |

|                                  |                         |
|----------------------------------|-------------------------|
| <a href="#">rs_data_kind</a>     | Element data kind       |
| <a href="#">rs_data_type</a>     | Element basic data type |
| <a href="#">rs_element</a>       | Handle to an element    |
| <a href="#">rs_sampler</a>       | Handle to a Sampler     |
| <a href="#">rs_sampler_value</a> | Sampler wrap T value    |
| <a href="#">rs_script</a>        | Handle to a Script      |
| <a href="#">rs_type</a>          | Handle to a Type        |
| <a href="#">rs_yuv_format</a>    | YUV format              |

## Conversion Functions

The functions below convert from a numerical vector type to another, or from one color representation to another.

| Functions                         |                                  |
|-----------------------------------|----------------------------------|
| <a href="#">convert</a>           | Convert numerical vectors        |
| <a href="#">rsPackColorTo8888</a> | Create a uchar4 RGBA from floats |
| <a href="#">rsUnpackColor8888</a> | Create a float4 RGBA from uchar4 |
| <a href="#">rsYuvToRGBA</a>       | Convert a YUV value to RGBA      |

## Mathematical Constants and Functions

The mathematical functions below can be applied to scalars and vectors. When applied to vectors, the returned value is a vector of the function applied to each entry of the input.

For example:

```
float3 a, b;
// The following call sets
// a.x to sin(b.x),
// a.y to sin(b.y), and
// a.z to sin(b.z).
a = sin(b);
```

See [Vector Math Functions](#) for functions like [distance\(\)](#) and [length\(\)](#) that interpret instead the input as a single vector in n-dimensional space.

The precision of the mathematical operations on 32 bit floats is affected by the pragmas `rs_fp_relaxed` and `rs_fp_full`. Under `rs_fp_relaxed`, subnormal values may be flushed to zero and rounding may be done towards zero. In comparison, `rs_fp_full` requires correct handling of subnormal values, i.e. smaller than `1.17549435e-38f`. `rs_fp_rull` also requires round to nearest with ties to even.

Different precision/speed tradeoffs can be achieved by using variants of the common math functions. Functions with a name starting with

- `native_`: May have custom hardware implementations with weaker precision. Additionally, subnormal values may be flushed to zero, rounding towards zero may be used, and NaN and infinity input may not be handled correctly.
- `half_`: May perform internal computations using 16 bit floats. Additionally, subnormal values may be flushed to zero, and rounding towards zero may be used.

| Constants                  |                                 |
|----------------------------|---------------------------------|
| <a href="#">M_1_PI</a>     | 1 / pi, as a 32 bit float       |
| <a href="#">M_2_PI</a>     | 2 / pi, as a 32 bit float       |
| <a href="#">M_2_SQRTPI</a> | 2 / sqrt(pi), as a 32 bit float |
| <a href="#">M_E</a>        | e, as a 32 bit float            |

|           |                                |
|-----------|--------------------------------|
| M_LN10    | log_e(10), as a 32 bit float   |
| M_LN2     | log_e(2), as a 32 bit float    |
| M_LOG10E  | log_10(e), as a 32 bit float   |
| M_LOG2E   | log_2(e), as a 32 bit float    |
| M_PI      | pi, as a 32 bit float          |
| M_PI_2    | pi / 2, as a 32 bit float      |
| M_PI_4    | pi / 4, as a 32 bit float      |
| M_SQRT1_2 | 1 / sqrt(2), as a 32 bit float |
| M_SQRT2   | sqrt(2), as a 32 bit float     |

| Functions |                                           |
|-----------|-------------------------------------------|
| abs       | Absolute value of an integer              |
| acos      | Inverse cosine                            |
| acosh     | Inverse hyperbolic cosine                 |
| acospi    | Inverse cosine divided by pi              |
| asin      | Inverse sine                              |
| asinh     | Inverse hyperbolic sine                   |
| asinpi    | Inverse sine divided by pi                |
| atan      | Inverse tangent                           |
| atan2     | Inverse tangent of a ratio                |
| atan2pi   | Inverse tangent of a ratio, divided by pi |
| atanh     | Inverse hyperbolic tangent                |
| atanpi    | Inverse tangent divided by pi             |
| cbrt      | Cube root                                 |
| ceil      | Smallest integer not less than a value    |
| clamp     | Restrain a value to a range               |
| clz       | Number of leading 0 bits                  |
| copysign  | Copies the sign of a number to another    |
| cos       | Cosine                                    |
| cosh      | Hypebolic cosine                          |
| cospi     | Cosine of a number multiplied by pi       |
| degrees   | Converts radians into degrees             |
| erf       | Mathematical error function               |
| erfc      | Mathematical complementary error function |
| exp       | e raised to a number                      |
| exp10     | 10 raised to a number                     |
| exp2      | 2 raised to a number                      |
| expm1     | e raised to a number minus one            |
| fabs      | Absolute value of a float                 |
| fdim      | Positive difference between two values    |

|                             |                                                          |
|-----------------------------|----------------------------------------------------------|
| <code>floor</code>          | Smallest integer not greater than a value                |
| <code>fma</code>            | Multiply and add                                         |
| <code>fmax</code>           | Maximum of two floats                                    |
| <code>fmin</code>           | Minimum of two floats                                    |
| <code>fmod</code>           | Modulo                                                   |
| <code>fract</code>          | Positive fractional part                                 |
| <code>frexp</code>          | Binary mantissa and exponent                             |
| <code>half_recip</code>     | Reciprocal computed to 16 bit precision                  |
| <code>half_rsqrt</code>     | Reciprocal of a square root computed to 16 bit precision |
| <code>half_sqrt</code>      | Square root computed to 16 bit precision                 |
| <code>hypot</code>          | Hypotenuse                                               |
| <code>ilogb</code>          | Base two exponent                                        |
| <code>ldexp</code>          | Creates a floating point from mantissa and exponent      |
| <code>lgamma</code>         | Natural logarithm of the gamma function                  |
| <code>log</code>            | Natural logarithm                                        |
| <code>log10</code>          | Base 10 logarithm                                        |
| <code>log1p</code>          | Natural logarithm of a value plus 1                      |
| <code>log2</code>           | Base 2 logarithm                                         |
| <code>logb</code>           | Base two exponent                                        |
| <code>mad</code>            | Multiply and add                                         |
| <code>max</code>            | Maximum                                                  |
| <code>min</code>            | Minimum                                                  |
| <code>mix</code>            | Mixes two values                                         |
| <code>modf</code>           | Integral and fractional components                       |
| <code>nan</code>            | Not a Number                                             |
| <code>nan_half</code>       | Not a Number                                             |
| <code>native_acos</code>    | Approximate inverse cosine                               |
| <code>native_acosh</code>   | Approximate inverse hyperbolic cosine                    |
| <code>native_acospi</code>  | Approximate inverse cosine divided by pi                 |
| <code>native_asin</code>    | Approximate inverse sine                                 |
| <code>native_asinh</code>   | Approximate inverse hyperbolic sine                      |
| <code>native_asinpi</code>  | Approximate inverse sine divided by pi                   |
| <code>native_atan</code>    | Approximate inverse tangent                              |
| <code>native_atan2</code>   | Approximate inverse tangent of a ratio                   |
| <code>native_atan2pi</code> | Approximate inverse tangent of a ratio, divided by pi    |
| <code>native_atanh</code>   | Approximate inverse hyperbolic tangent                   |
| <code>native_atanpi</code>  | Approximate inverse tangent divided by pi                |
| <code>native_cbrt</code>    | Approximate cube root                                    |
| <code>native_cos</code>     | Approximate cosine                                       |
| <code>native_cosh</code>    | Approximate hyperbolic cosine                            |

|               |                                                  |
|---------------|--------------------------------------------------|
| native_cospi  | Approximate cosine of a number multiplied by pi  |
| native_divide | Approximate division                             |
| native_exp    | Approximate e raised to a number                 |
| native_exp10  | Approximate 10 raised to a number                |
| native_exp2   | Approximate 2 raised to a number                 |
| native_expm1  | Approximate e raised to a number minus one       |
| native_hypot  | Approximate hypotenuse                           |
| native_log    | Approximate natural logarithm                    |
| native_log10  | Approximate base 10 logarithm                    |
| native_log1p  | Approximate natural logarithm of a value plus 1  |
| native_log2   | Approximate base 2 logarithm                     |
| native_powr   | Approximate positive base raised to an exponent  |
| native_recip  | Approximate reciprocal                           |
| native_rootn  | Approximate nth root                             |
| native_rsqrt  | Approximate reciprocal of a square root          |
| native_sin    | Approximate sine                                 |
| native_sincos | Approximate sine and cosine                      |
| native_sinh   | Approximate hyperbolic sine                      |
| native_sinpi  | Approximate sine of a number multiplied by pi    |
| native_sqrt   | Approximate square root                          |
| native_tan    | Approximate tangent                              |
| native_tanh   | Approximate hyperbolic tangent                   |
| native_tanpi  | Approximate tangent of a number multiplied by pi |
| nextafter     | Next floating point number                       |
| pow           | Base raised to an exponent                       |
| pown          | Base raised to an integer exponent               |
| powr          | Positive base raised to an exponent              |
| radians       | Converts degrees into radians                    |
| remainder     | Remainder of a division                          |
| remquo        | Remainder and quotient of a division             |
| rint          | Round to even                                    |
| rootn         | Nth root                                         |
| round         | Round away from zero                             |
| rsRand        | Pseudo-random number                             |
| rsqrt         | Reciprocal of a square root                      |
| sign          | Sign of a value                                  |
| sin           | Sine                                             |
| sincos        | Sine and cosine                                  |
| sinh          | Hyperbolic sine                                  |

|                     |                                      |
|---------------------|--------------------------------------|
| <code>sinpi</code>  | Sine of a number multiplied by pi    |
| <code>sqrt</code>   | Square root                          |
| <code>step</code>   | 0 if less than a value, 0 otherwise  |
| <code>tan</code>    | Tangent                              |
| <code>tanh</code>   | Hyperbolic tangent                   |
| <code>tanpi</code>  | Tangent of a number multiplied by pi |
| <code>tgamma</code> | Gamma function                       |
| <code>trunc</code>  | Truncates a floating point           |

## Vector Math Functions

These functions interpret the input arguments as representation of vectors in n-dimensional space.

The precision of the mathematical operations on 32 bit floats is affected by the pragmas `rs_fp_relaxed` and `rs_fp_full`. See [Mathematical Constants and Functions](#) for details.

Different precision/speed tradeoffs can be achieved by using variants of the common math functions. Functions with a name starting with

- `native_`: May have custom hardware implementations with weaker precision. Additionally, subnormal values may be flushed to zero, rounding towards zero may be used, and NaN and infinity input may not be handled correctly.
- `fast_`: May perform internal computations using 16 bit floats. Additionally, subnormal values may be flushed to zero, and rounding towards zero may be used.

| Functions                     |                                         |
|-------------------------------|-----------------------------------------|
| <code>cross</code>            | Cross product of two vectors            |
| <code>distance</code>         | Distance between two points             |
| <code>dot</code>              | Dot product of two vectors              |
| <code>fast_distance</code>    | Approximate distance between two points |
| <code>fast_length</code>      | Approximate length of a vector          |
| <code>fast_normalize</code>   | Approximate normalized vector           |
| <code>length</code>           | Length of a vector                      |
| <code>native_distance</code>  | Approximate distance between two points |
| <code>native_length</code>    | Approximate length of a vector          |
| <code>native_normalize</code> | Approximately normalize a vector        |
| <code>normalize</code>        | Normalize a vector                      |

## Matrix Functions

These functions let you manipulate square matrices of rank 2x2, 3x3, and 4x4. They are particularly useful for graphical transformations and are compatible with OpenGL.

We use a zero-based index for rows and columns. E.g. the last element of a `rs_matrix4x4` is found at (3, 3).

RenderScript uses column-major matrices and column-based vectors. Transforming a vector is done by postmultiplying the vector, e.g. `(matrix * vector)`, as provided by `rsMatrixMultiply()`.

To create a transformation matrix that performs two transformations at once, multiply the two source matrices, with the first transformation as the right argument. E.g. to create a transformation matrix that applies the transformation s1 followed by s2, call `rsMatrixLoadMultiply(&combined, &s2, &s1)`. This derives from `s2 * (s1 * v)`, which is `(s2 * s1) * v`.

We have two style of functions to create transformation matrices: `rsMatrixLoadTransformation` and `rsMatrixTransformation`. The former style simply stores the transformation matrix in the first argument. The latter modifies a pre-existing transformation matrix so that the new transformation happens first. E.g. if you call `rsMatrixTranslate()` on a matrix that already does a scaling, the resulting matrix when applied to a vector will first do the translation then the scaling.

| Functions                             |                                                 |
|---------------------------------------|-------------------------------------------------|
| <code>rsExtractFrustumPlanes</code>   | Compute frustum planes                          |
| <code>rsIsSphereInFrustum</code>      | Checks if a sphere is within the frustum planes |
| <code>rsMatrixGet</code>              | Get one element                                 |
| <code>rsMatrixInverse</code>          | Inverts a matrix in place                       |
| <code>rsMatrixInverseTranspose</code> | Inverts and transpose a matrix in place         |
| <code>rsMatrixLoad</code>             | Load or copy a matrix                           |
| <code>rsMatrixLoadFrustum</code>      | Load a frustum projection matrix                |
| <code>rsMatrixLoadIdentity</code>     | Load identity matrix                            |
| <code>rsMatrixLoadMultiply</code>     | Multiply two matrices                           |
| <code>rsMatrixLoadOrtho</code>        | Load an orthographic projection matrix          |
| <code>rsMatrixLoadPerspective</code>  | Load a perspective projection matrix            |
| <code>rsMatrixLoadRotate</code>       | Load a rotation matrix                          |
| <code>rsMatrixLoadScale</code>        | Load a scaling matrix                           |
| <code>rsMatrixLoadTranslate</code>    | Load a translation matrix                       |
| <code>rsMatrixMultiply</code>         | Multiply a matrix by a vector or another matrix |
| <code>rsMatrixRotate</code>           | Apply a rotation to a transformation matrix     |
| <code>rsMatrixScale</code>            | Apply a scaling to a transformation matrix      |
| <code>rsMatrixSet</code>              | Set one element                                 |
| <code>rsMatrixTranslate</code>        | Apply a translation to a transformation matrix  |
| <code>rsMatrixTranspose</code>        | Transpose a matrix place                        |

## Quaternion Functions

The following functions manipulate quaternions.

| Functions                               |                                                                      |
|-----------------------------------------|----------------------------------------------------------------------|
| <code>rsQuaternionAdd</code>            | Add two quaternions                                                  |
| <code>rsQuaternionConjugate</code>      | Conjugate a quaternion                                               |
| <code>rsQuaternionDot</code>            | Dot product of two quaternions                                       |
| <code>rsQuaternionGetMatrixUnit</code>  | Get a rotation matrix from a quaternion                              |
| <code>rsQuaternionLoadRotate</code>     | Create a rotation quaternion                                         |
| <code>rsQuaternionLoadRotateUnit</code> | Quaternion that represents a rotation about an arbitrary unit vector |
| <code>rsQuaternionMultiply</code>       | Multiply a quaternion by a scalar or another quaternion              |
| <code>rsQuaternionNormalize</code>      | Normalize a quaternion                                               |
| <code>rsQuaternionSet</code>            | Create a quaternion                                                  |
| <code>rsQuaternionSlerp</code>          | Spherical linear interpolation between two quaternions               |

# Atomic Update Functions

To update values shared between multiple threads, use the functions below. They ensure that the values are atomically updated, i.e. that the memory reads, the updates, and the memory writes are done in the right order.

These functions are slower than their non-atomic equivalents, so use them only when synchronization is needed.

Note that in RenderScript, your code is likely to be running in separate threads even though you did not explicitly create them. The RenderScript runtime will very often split the execution of one kernel across multiple threads. Updating globals should be done with atomic functions. If possible, modify your algorithm to avoid them altogether.

| Functions                   |                                  |
|-----------------------------|----------------------------------|
| <a href="#">rsAtomicAdd</a> | Thread-safe addition             |
| <a href="#">rsAtomicAnd</a> | Thread-safe bitwise and          |
| <a href="#">rsAtomicCas</a> | Thread-safe compare and set      |
| <a href="#">rsAtomicDec</a> | Thread-safe decrement            |
| <a href="#">rsAtomicInc</a> | Thread-safe increment            |
| <a href="#">rsAtomicMax</a> | Thread-safe maximum              |
| <a href="#">rsAtomicMin</a> | Thread-safe minimum              |
| <a href="#">rsAtomicOr</a>  | Thread-safe bitwise or           |
| <a href="#">rsAtomicSub</a> | Thread-safe subtraction          |
| <a href="#">rsAtomicXor</a> | Thread-safe bitwise exclusive or |

# Time Functions and Types

The functions below can be used to tell the current clock time and the current system up time. It is not recommended to call these functions inside of a kernel.

| Types                     |                               |
|---------------------------|-------------------------------|
| <a href="#">rs_time_t</a> | Seconds since January 1, 1970 |
| <a href="#">rs_tm</a>     | Date and time structure       |

| Functions                      |                               |
|--------------------------------|-------------------------------|
| <a href="#">rsGetDt</a>        | Elapsed time since last call  |
| <a href="#">rsLocaltime</a>    | Convert to local time         |
| <a href="#">rsTime</a>         | Seconds since January 1, 1970 |
| <a href="#">rsUptimeMillis</a> | System uptime in milliseconds |
| <a href="#">rsUptimeNanos</a>  | System uptime in nanoseconds  |

# Allocation Creation Functions

The functions below can be used to create Allocations from a Script.

These functions can be called directly or indirectly from an invokable function. If some control-flow path can result in a call to these functions from a RenderScript kernel function, a compiler error will be generated.

| Functions                          |                                               |
|------------------------------------|-----------------------------------------------|
| <a href="#">rsCreateAllocation</a> | Create an rs_allocation object of given Type. |

|                                       |                                                                           |
|---------------------------------------|---------------------------------------------------------------------------|
| <a href="#">rsCreateElement</a>       | Creates an rs_element object of the specified data type                   |
| <a href="#">rsCreatePixelElement</a>  | Creates an rs_element object of the specified data type and data kind     |
| <a href="#">rsCreateType</a>          | Creates an rs_type object with the specified Element and shape attributes |
| <a href="#">rsCreateVectorElement</a> | Creates an rs_element object of the specified data type and vector width  |

## Allocation Data Access Functions

The functions below can be used to get and set the cells that comprise an allocation.

- Individual cells are accessed using the rsGetElementAt\* and [rsSetElementAt](#) functions.
- Multiple cells can be copied using the rsAllocationCopy\* and rsAllocationV\* functions.
- For getting values through a sampler, use [rsSample](#).

The [rsGetElementAt](#) and [rsSetElementAt](#) functions are somewhat misnamed. They don't get or set elements, which are akin to data types; they get or set cells. Think of them as rsGetCellAt and rsSetCellAt.

| Functions                                 |                                                        |
|-------------------------------------------|--------------------------------------------------------|
| <a href="#">rsAllocationCopy1DRange</a>   | Copy consecutive cells between allocations             |
| <a href="#">rsAllocationCopy2DRange</a>   | Copy a rectangular region of cells between allocations |
| <a href="#">rsAllocationVLoadX</a>        | Get a vector from an allocation of scalars             |
| <a href="#">rsAllocationVStoreX</a>       | Store a vector into an allocation of scalars           |
| <a href="#">rsGetElementAt</a>            | Return a cell from an allocation                       |
| <a href="#">rsGetElementAtYuv_uchar_U</a> | Get the U component of an allocation of YUVs           |
| <a href="#">rsGetElementAtYuv_uchar_V</a> | Get the V component of an allocation of YUVs           |
| <a href="#">rsGetElementAtYuv_uchar_Y</a> | Get the Y component of an allocation of YUVs           |
| <a href="#">rsSample</a>                  | Sample a value from a texture allocation               |
| <a href="#">rsSetElementAt</a>            | Set a cell of an allocation                            |

## Object Characteristics Functions

The functions below can be used to query the characteristics of an Allocation, Element, or Sampler object. These objects are created from Java. You can't create them from a script.

### Allocations:

Allocations are the primary method used to pass data to and from RenderScript kernels.

They are a structured collection of cells that can be used to store bitmaps, textures, arbitrary data points, etc.

This collection of cells may have many dimensions (X, Y, Z, Array0, Array1, Array2, Array3), faces (for cubemaps), and level of details (for mipmapping).

See the [android.renderscript.Allocation](#) for details on to create Allocations.

### Elements:

The term "element" is used a bit ambiguously in RenderScript, as both type information for the cells of an Allocation and the instantiation of that type. For example:

- [rs\\_element](#) is a handle to a type specification, and
- In functions like [rsGetElementAt\(\)](#), "element" means the instantiation of the type, i.e. a cell of an Allocation.

The functions below let you query the characteristics of the type specification.

An Element can specify a simple data types as found in C, e.g. an integer, float, or boolean. It can also specify a handle to a RenderScript object. See [rs\\_data\\_type](#) for a list of basic types.

Elements can specify fixed size vector (of size 2, 3, or 4) versions of the basic types. Elements can be grouped together into complex Elements, creating the equivalent of C structure definitions.

Elements can also have a kind, which is semantic information used to interpret pixel data. See [rs\\_data\\_kind](#).

When creating Allocations of common elements, you can simply use one of the many predefined Elements like [F32\\_2](#).

To create complex Elements, use the [Element.Builder](#) Java class.

## Samplers:

Samplers objects define how Allocations can be read as structure within a kernel. See [android.renderscript.S](#).

| Functions                                         |                                                         |
|---------------------------------------------------|---------------------------------------------------------|
| <a href="#">rsAllocationGetDimFaces</a>           | Presence of more than one face                          |
| <a href="#">rsAllocationGetDimLOD</a>             | Presence of levels of detail                            |
| <a href="#">rsAllocationGetDimX</a>               | Size of the X dimension                                 |
| <a href="#">rsAllocationGetDimY</a>               | Size of the Y dimension                                 |
| <a href="#">rsAllocationGetDimZ</a>               | Size of the Z dimension                                 |
| <a href="#">rsAllocationGetElement</a>            | Get the object that describes the cell of an Allocation |
| <a href="#">rsClearObject</a>                     | Release an object                                       |
| <a href="#">rsElementGetBytesSize</a>             | Size of an Element                                      |
| <a href="#">rsElementGetDataKind</a>              | Kind of an Element                                      |
| <a href="#">rsElementGetType</a>                  | Data type of an Element                                 |
| <a href="#">rsElementGetSubElement</a>            | Sub-element of a complex Element                        |
| <a href="#">rsElementGetSubElementArraySize</a>   | Array size of a sub-element of a complex Element        |
| <a href="#">rsElementGetSubElementCount</a>       | Number of sub-elements                                  |
| <a href="#">rsElementGetSubElementName</a>        | Name of a sub-element                                   |
| <a href="#">rsElementGetSubElementNameLength</a>  | Length of the name of a sub-element                     |
| <a href="#">rsElementGetSubElementOffsetBytes</a> | Offset of the instantiated sub-element                  |
| <a href="#">rsElementGetVectorSize</a>            | Vector size of the Element                              |
| <a href="#">rsIsObject</a>                        | Check for an empty handle                               |
| <a href="#">rsSamplerGetAnisotropy</a>            | Anisotropy of the Sampler                               |
| <a href="#">rsSamplerGetMagnification</a>         | Sampler magnification value                             |
| <a href="#">rsSamplerGetMinification</a>          | Sampler minification value                              |
| <a href="#">rsSamplerGetWrapS</a>                 | Sampler wrap S value                                    |
| <a href="#">rsSamplerGetWrapT</a>                 | Sampler wrap T value                                    |

## Kernel Invocation Functions and Types

The [rsForEach](#)() function can be used to invoke the root kernel of a script.

The other functions are used to get the characteristics of the invocation of an executing kernel, like dimensions and current indices. These functions take a [rs\\_kernel\\_context](#) as argument.

| Types                               |                                                                             |
|-------------------------------------|-----------------------------------------------------------------------------|
| <code>rs_for_each_strategy_t</code> | Suggested cell processing order                                             |
| <code>rs_kernel</code>              | Handle to a kernel function                                                 |
| <code>rs_kernel_context</code>      | Handle to a kernel invocation context                                       |
| <code>rs_script_call_t</code>       | Cell iteration information                                                  |
| Functions                           |                                                                             |
| <code>rsForEach</code>              | Launches a kernel                                                           |
| <code>rsForEachInternal</code>      | (Internal API) Launch a kernel in the current Script (with the slot number) |
| <code>rsForEachWithOptions</code>   | Launches a kernel with options                                              |
| <code>rsGetArray0</code>            | Index in the Array0 dimension for the specified kernel context              |
| <code>rsGetArray1</code>            | Index in the Array1 dimension for the specified kernel context              |
| <code>rsGetArray2</code>            | Index in the Array2 dimension for the specified kernel context              |
| <code>rsGetArray3</code>            | Index in the Array3 dimension for the specified kernel context              |
| <code>rsGetDimArray0</code>         | Size of the Array0 dimension for the specified kernel context               |
| <code>rsGetDimArray1</code>         | Size of the Array1 dimension for the specified kernel context               |
| <code>rsGetDimArray2</code>         | Size of the Array2 dimension for the specified kernel context               |
| <code>rsGetDimArray3</code>         | Size of the Array3 dimension for the specified kernel context               |
| <code>rsGetDimHasFaces</code>       | Presence of more than one face for the specified kernel context             |
| <code>rsGetDimLod</code>            | Number of levels of detail for the specified kernel context                 |
| <code>rsGetDimX</code>              | Size of the X dimension for the specified kernel context                    |
| <code>rsGetDimY</code>              | Size of the Y dimension for the specified kernel context                    |
| <code>rsGetDimZ</code>              | Size of the Z dimension for the specified kernel context                    |
| <code>rsGetFace</code>              | Coordinate of the Face for the specified kernel context                     |
| <code>rsGetLod</code>               | Index in the Levels of Detail dimension for the specified kernel context    |

## Input/Output Functions

These functions are used to:

- Send information to the Java client, and
- Send the processed allocation or receive the next allocation to process.

| Functions                           |                                            |
|-------------------------------------|--------------------------------------------|
| <code>rsAllocationIoReceive</code>  | Receive new content from the queue         |
| <code>rsAllocationIoSend</code>     | Send new content to the queue              |
| <code>rsSendToClient</code>         | Send a message to the client, non-blocking |
| <code>rsSendToClientBlocking</code> | Send a message to the client, blocking     |

## Debugging Functions

The functions below are intended to be used during application development. They should not be used in shipping applications.

Functions

[rsDebug](#)

Log a message and values

## Graphics Functions and Types

The graphics subsystem of RenderScript was removed at API level 23.



# RenderScript Numerical Types

## Overview

### Scalars:

RenderScript supports the following scalar numerical types:

|                   | 8 bits                      | 16 bits                       | 32 bits                     | 64 bits                               |
|-------------------|-----------------------------|-------------------------------|-----------------------------|---------------------------------------|
| Integer:          | char, <code>int8_t</code>   | short, <code>int16_t</code>   | <code>int32_t</code>        | long, long long, <code>int64_t</code> |
| Unsigned integer: | uchar, <code>uint8_t</code> | ushort, <code>uint16_t</code> | uint, <code>uint32_t</code> | ulong, <code>uint64_t</code>          |
| Floating point:   |                             | half                          | float                       | double                                |

### Vectors:

RenderScript supports fixed size vectors of length 2, 3, and 4. Vectors are declared using the common type name followed by a 2, 3, or 4. E.g. `float4`, `int3`, `double2`, `ulong4`.

To create vector literals, use the vector type followed by the values enclosed between curly braces, e.g. `(float3){1.0f, 2.0f, 3.0f}`.

Entries of a vector can be accessed using different naming styles.

Single entries can be accessed by following the variable name with a dot and:

- The letters x, y, z, and w,
- The letters r, g, b, and a,
- The letter s or S, followed by a zero based index.

For example, with `int4 myVar;` the following are equivalent:

```
myVar.x == myVar.r == myVar.s0 == myVar.S0
myVar.y == myVar.g == myVar.s1 == myVar.S1
myVar.z == myVar.b == myVar.s2 == myVar.S2
myVar.w == myVar.a == myVar.s3 == myVar.S3
```

Multiple entries of a vector can be accessed at once by using an identifier that is the concatenation of multiple letters or indices. The resulting vector has a size equal to the number of entries named.

With the example above, the middle two entries can be accessed using `myVar.yz`, `myVar.gb`, `myVar.s12`, and `myVar.S12`.

The entries don't have to be contiguous or in increasing order. Entries can even be repeated, as long as we're not trying to assign to it. You also can't mix the naming styles.

Here are examples of what can or can't be done:

```
float4 v4;
float3 v3;
float2 v2;
v2 = v4.xx; // Valid
v3 = v4.zwx; // Valid
v3 = v4.bba; // Valid
v3 = v4.s032; // Valid
v3.s120 = v4.S233; // Valid
```

```

v4.yz = v3.rg; // Valid
v4.yzx = v3.rg; // Invalid: mismatched sizes
v4.yzz = v3; // Invalid: z appears twice in an assignment
v3 = v3.xas0; // Invalid: can't mix xyzw with rgba nor s0...
v3 = v4.s034; // Invalid: the digit can only be 0, 1, 2, or 3

```

## Matrices and Quaternions:

RenderScript supports fixed size square matrices of floats of size 2x2, 3x3, and 4x4. The types are named [rs\\_matrix2x2](#), [rs\\_matrix3x3](#), and [rs\\_matrix4x4](#). See [Matrix Functions](#) for the list of operations.

Quaternions are also supported via [rs\\_quaternion](#). See [Quaterion Functions](#) for the list of operations.

## Summary

| Types                         |                              |
|-------------------------------|------------------------------|
| <a href="#">char2</a>         | Two 8 bit signed integers    |
| <a href="#">char3</a>         | Three 8 bit signed integers  |
| <a href="#">char4</a>         | Four 8 bit signed integers   |
| <a href="#">double2</a>       | Two 64 bit floats            |
| <a href="#">double3</a>       | Three 64 bit floats          |
| <a href="#">double4</a>       | Four 64 bit floats           |
| <a href="#">float2</a>        | Two 32 bit floats            |
| <a href="#">float3</a>        | Three 32 bit floats          |
| <a href="#">float4</a>        | Four 32 bit floats           |
| <a href="#">half</a>          | 16 bit floating point value  |
| <a href="#">half2</a>         | Two 16 bit floats            |
| <a href="#">half3</a>         | Three 16 bit floats          |
| <a href="#">half4</a>         | Four 16 bit floats           |
| <a href="#">int16_t</a>       | 16 bit signed integer        |
| <a href="#">int2</a>          | Two 32 bit signed integers   |
| <a href="#">int3</a>          | Three 32 bit signed integers |
| <a href="#">int32_t</a>       | 32 bit signed integer        |
| <a href="#">int4</a>          | Four 32 bit signed integers  |
| <a href="#">int64_t</a>       | 64 bit signed integer        |
| <a href="#">int8_t</a>        | 8 bit signed integer         |
| <a href="#">long2</a>         | Two 64 bit signed integers   |
| <a href="#">long3</a>         | Three 64 bit signed integers |
| <a href="#">long4</a>         | Four 64 bit signed integers  |
| <a href="#">rs_matrix2x2</a>  | 2x2 matrix of 32 bit floats  |
| <a href="#">rs_matrix3x3</a>  | 3x3 matrix of 32 bit floats  |
| <a href="#">rs_matrix4x4</a>  | 4x4 matrix of 32 bit floats  |
| <a href="#">rs_quaternion</a> | Quaternion                   |
| <a href="#">short2</a>        | Two 16 bit signed integers   |

|                       |                                |
|-----------------------|--------------------------------|
| <code>short3</code>   | Three 16 bit signed integers   |
| <code>short4</code>   | Four 16 bit signed integers    |
| <code>size_t</code>   | Unsigned size type             |
| <code>ssize_t</code>  | Signed size type               |
| <code>uchar</code>    | 8 bit unsigned integer         |
| <code>uchar2</code>   | Two 8 bit unsigned integers    |
| <code>uchar3</code>   | Three 8 bit unsigned integers  |
| <code>uchar4</code>   | Four 8 bit unsigned integers   |
| <code>uint</code>     | 32 bit unsigned integer        |
| <code>uint16_t</code> | 16 bit unsigned integer        |
| <code>uint2</code>    | Two 32 bit unsigned integers   |
| <code>uint3</code>    | Three 32 bit unsigned integers |
| <code>uint32_t</code> | 32 bit unsigned integer        |
| <code>uint4</code>    | Four 32 bit unsigned integers  |
| <code>uint64_t</code> | 64 bit unsigned integer        |
| <code>uint8_t</code>  | 8 bit unsigned integer         |
| <code>ulong</code>    | 64 bit unsigned integer        |
| <code>ulong2</code>   | Two 64 bit unsigned integers   |
| <code>ulong3</code>   | Three 64 bit unsigned integers |
| <code>ulong4</code>   | Four 64 bit unsigned integers  |
| <code>ushort</code>   | 16 bit unsigned integer        |
| <code>ushort2</code>  | Two 16 bit unsigned integers   |
| <code>ushort3</code>  | Three 16 bit unsigned integers |
| <code>ushort4</code>  | Four 16 bit unsigned integers  |

## Types

---

### char2 : Two 8 bit signed integers

A typedef of: `char __attribute__((ext_vector_type(2)))`

A vector of two chars. These two chars are packed into a single 16 bit field with a 16 bit alignment.

### char3 : Three 8 bit signed integers

A typedef of: `char __attribute__((ext_vector_type(3)))`

A vector of three chars. These three chars are packed into a single 32 bit field with a 32 bit alignment.

### char4 : Four 8 bit signed integers

A typedef of: `char __attribute__((ext_vector_type(4)))`

A vector of four chars. These four chars are packed into a single 32 bit field with a 32 bit alignment.

### double2 : Two 64 bit floats

A typedef of: `double __attribute__((ext_vector_type(2)))`

A vector of two doubles. These two double fields packed into a single 128 bit field with a 128 bit alignment.

#### double3 : Three 64 bit floats

A typedef of: `double __attribute__((ext_vector_type(3)))`

A vector of three doubles. These three double fields packed into a single 256 bit field with a 256 bit alignment.

#### double4 : Four 64 bit floats

A typedef of: `double __attribute__((ext_vector_type(4)))`

A vector of four doubles. These four double fields packed into a single 256 bit field with a 256 bit alignment.

#### float2 : Two 32 bit floats

A typedef of: `float __attribute__((ext_vector_type(2)))`

A vector of two floats. These two floats are packed into a single 64 bit field with a 64 bit alignment.

A vector of two floats. These two floats are packed into a single 64 bit field with a 64 bit alignment.

#### float3 : Three 32 bit floats

A typedef of: `float __attribute__((ext_vector_type(3)))`

A vector of three floats. These three floats are packed into a single 128 bit field with a 128 bit alignment.

#### float4 : Four 32 bit floats

A typedef of: `float __attribute__((ext_vector_type(4)))`

A vector of four floats type. These four floats are packed into a single 128 bit field with a 128 bit alignment.

#### half : 16 bit floating point value

A typedef of: `__fp16` Added in [API level 23](#)

A 16 bit floating point value.

#### half2 : Two 16 bit floats

A typedef of: `half __attribute__((ext_vector_type(2)))` Added in [API level 23](#)

Vector version of the half float type. Provides two half fields packed into a single 32 bit field with 32 bit alignment.

#### half3 : Three 16 bit floats

A typedef of: `half __attribute__((ext_vector_type(3)))` Added in [API level 23](#)

Vector version of the half float type. Provides three half fields packed into a single 64 bit field with 64 bit alignment.

#### half4 : Four 16 bit floats

A typedef of: `half __attribute__((ext_vector_type(4)))` Added in [API level 23](#)

Vector version of the half float type. Provides four half fields packed into a single 64 bit field with 64 bit alignment.

#### int16\_t : 16 bit signed integer

A typedef of: `short`

A 16 bit signed integer type.

## int2 : Two 32 bit signed integers

A typedef of: int \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two ints. These two ints are packed into a single 64 bit field with a 64 bit alignment.

## int3 : Three 32 bit signed integers

A typedef of: int \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three ints. These three ints are packed into a single 128 bit field with a 128 bit alignment.

## int32\_t : 32 bit signed integer

A typedef of: int

A 32 bit signed integer type.

## int4 : Four 32 bit signed integers

A typedef of: int \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four ints. These two fours are packed into a single 128 bit field with a 128 bit alignment.

## int64\_t : 64 bit signed integer

A typedef of: long long    Removed from [API level 21 and higher](#)

A typedef of: long    Added in [API level 21](#)

A 64 bit signed integer type.

## int8\_t : 8 bit signed integer

A typedef of: char

8 bit signed integer type.

## long2 : Two 64 bit signed integers

A typedef of: long \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two longs. These two longs are packed into a single 128 bit field with a 128 bit alignment.

## long3 : Three 64 bit signed integers

A typedef of: long \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three longs. These three longs are packed into a single 256 bit field with a 256 bit alignment.

## long4 : Four 64 bit signed integers

A typedef of: long \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four longs. These four longs are packed into a single 256 bit field with a 256 bit alignment.

## rs\_matrix2x2 : 2x2 matrix of 32 bit floats

A structure with the following fields:

*float m[4]*

A square 2x2 matrix of floats. The entries are stored in the array at the location [row\*2 + col].

See [Matrix Functions](#).

### rs\_matrix3x3 : 3x3 matrix of 32 bit floats

A structure with the following fields:

*float m[9]*

A square 3x3 matrix of floats. The entries are stored in the array at the location [row\*3 + col].

See [Matrix Functions](#).

### rs\_matrix4x4 : 4x4 matrix of 32 bit floats

A structure with the following fields:

*float m[16]*

A square 4x4 matrix of floats. The entries are stored in the array at the location [row\*4 + col].

See [Matrix Functions](#).

### rs\_quaternion : Quaternion

A typedef of: float4

A square 4x4 matrix of floats that represents a quaternion.

See [Quaternion Functions](#).

### short2 : Two 16 bit signed integers

A typedef of: short \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two shorts. These two shorts are packed into a single 32 bit field with a 32 bit alignment.

### short3 : Three 16 bit signed integers

A typedef of: short \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three shorts. These three short fields packed into a single 64 bit field with a 64 bit alignment.

### short4 : Four 16 bit signed integers

A typedef of: short \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four shorts. These four short fields packed into a single 64 bit field with a 64 bit alignment.

### size\_t : Unsigned size type

A typedef of: uint64\_t When compiling for 64 bits.

A typedef of: uint32\_t When compiling for 32 bits.

Unsigned size type. The number of bits depend on the compilation flags.

### ssize\_t : Signed size type

A typedef of: int64\_t When compiling for 64 bits.

A typedef of: int32\_t When compiling for 32 bits.

Signed size type. The number of bits depend on the compilation flags.

## uchar : 8 bit unsigned integer

A typedef of: uint8\_t

8 bit unsigned integer type.

## uchar2 : Two 8 bit unsigned integers

A typedef of: uchar \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two uchar. These two uchar fields packed into a single 16 bit field with a 16 bit alignment.

## uchar3 : Three 8 bit unsigned integers

A typedef of: uchar \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three uchar. These three uchar fields packed into a single 32 bit field with a 32 bit alignment.

## uchar4 : Four 8 bit unsigned integers

A typedef of: uchar \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four uchar. These four uchar fields packed into a single 32 bit field with a 32 bit alignment.

## uint : 32 bit unsigned integer

A typedef of: uint32\_t

A 32 bit unsigned integer type.

## uint16\_t : 16 bit unsigned integer

A typedef of: unsigned short

A 16 bit unsigned integer type.

## uint2 : Two 32 bit unsigned integers

A typedef of: uint \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two uints. These two uints are packed into a single 64 bit field with a 64 bit alignment.

## uint3 : Three 32 bit unsigned integers

A typedef of: uint \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three uints. These three uints are packed into a single 128 bit field with a 128 bit alignment.

## uint32\_t : 32 bit unsigned integer

A typedef of: unsigned int

A 32 bit unsigned integer type.

## uint4 : Four 32 bit unsigned integers

A typedef of: uint \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four uints. These four uints are packed into a single 128 bit field with a 128 bit alignment.

## uint64\_t : 64 bit unsigned integer

A typedef of: unsigned long long    Removed from [API level 21 and higher](#)

A typedef of: unsigned long    Added in [API level 21](#)

A 64 bit unsigned integer type.

#### uint8\_t : 8 bit unsigned integer

A typedef of: unsigned char

8 bit unsigned integer type.

#### ulong : 64 bit unsigned integer

A typedef of: uint64\_t

A 64 bit unsigned integer type.

#### ulong2 : Two 64 bit unsigned integers

A typedef of: ulong \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two ulongs. These two ulongs are packed into a single 128 bit field with a 128 bit alignment.

#### ulong3 : Three 64 bit unsigned integers

A typedef of: ulong \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three ulongs. These three ulong fields packed into a single 256 bit field with a 256 bit alignment.

#### ulong4 : Four 64 bit unsigned integers

A typedef of: ulong \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four ulongs. These four ulong fields packed into a single 256 bit field with a 256 bit alignment.

#### ushort : 16 bit unsigned integer

A typedef of: uint16\_t

A 16 bit unsigned integer type.

#### ushort2 : Two 16 bit unsigned integers

A typedef of: ushort \_\_attribute\_\_((ext\_vector\_type(2)))

A vector of two ushorts. These two ushort fields packed into a single 32 bit field with a 32 bit alignment.

#### ushort3 : Three 16 bit unsigned integers

A typedef of: ushort \_\_attribute\_\_((ext\_vector\_type(3)))

A vector of three ushorts. These three ushort fields packed into a single 64 bit field with a 64 bit alignment.

#### ushort4 : Four 16 bit unsigned integers

A typedef of: ushort \_\_attribute\_\_((ext\_vector\_type(4)))

A vector of four ushorts. These four ushort fields packed into a single 64 bit field with a 64 bit alignment.

# RenderScript Object Types

## Overview

The types below are used to manipulate RenderScript objects like allocations, samplers, elements, and scripts. Most of these object are created using the Java RenderScript APIs.

## Summary

| Types                                      |                                               |
|--------------------------------------------|-----------------------------------------------|
| <a href="#">rs_allocation</a>              | Handle to an allocation                       |
| <a href="#">rs_allocation_cubemap_face</a> | Enum for selecting cube map faces             |
| <a href="#">rs_allocation_usage_type</a>   | Bitfield to specify how an allocation is used |
| <a href="#">rs_data_kind</a>               | Element data kind                             |
| <a href="#">rs_data_type</a>               | Element basic data type                       |
| <a href="#">rs_element</a>                 | Handle to an element                          |
| <a href="#">rs_sampler</a>                 | Handle to a Sampler                           |
| <a href="#">rs_sampler_value</a>           | Sampler wrap T value                          |
| <a href="#">rs_script</a>                  | Handle to a Script                            |
| <a href="#">rs_type</a>                    | Handle to a Type                              |
| <a href="#">rs_yuv_format</a>              | YUV format                                    |

## Types

### [rs\\_allocation](#) : Handle to an allocation

An opaque handle to a RenderScript allocation.

See [android.renderscript.Allocation](#).

### [rs\\_allocation\\_cubemap\\_face](#) : Enum for selecting cube map faces

An enum with the following values:    Added in [API level 14](#)

*RS\_ALLOCATION\_CUBEMAP\_FACE\_POSITIVE\_X = 0  
RS\_ALLOCATION\_CUBEMAP\_FACE\_NEGATIVE\_X = 1  
RS\_ALLOCATION\_CUBEMAP\_FACE\_POSITIVE\_Y = 2  
RS\_ALLOCATION\_CUBEMAP\_FACE\_NEGATIVE\_Y = 3  
RS\_ALLOCATION\_CUBEMAP\_FACE\_POSITIVE\_Z = 4  
RS\_ALLOCATION\_CUBEMAP\_FACE\_NEGATIVE\_Z = 5*

An enum used to specify one the six faces of a cubemap.

### [rs\\_allocation\\_usage\\_type](#) : Bitfield to specify how an allocation is used

An enum with the following values:    Added in [API level 14](#)

|                                                            |                                                                                                                                                                                  |
|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>RS_ALLOCATION_USAGE_SCRIPT</i> = 0x0001                 | Allocation is bound to and accessed by scripts.                                                                                                                                  |
| <i>RS_ALLOCATION_USAGE_GRAPHICS_TEXTURE</i> = 0x0002       | Allocation is used as a texture source.                                                                                                                                          |
| <i>RS_ALLOCATION_USAGE_GRAPHICS_VERTEX</i> = 0x0004        | Deprecated.                                                                                                                                                                      |
| <i>RS_ALLOCATION_USAGE_GRAPHICS_CONSTANTS</i> = 0x0008     | Deprecated.                                                                                                                                                                      |
| <i>RS_ALLOCATION_USAGE_GRAPHICS_RENDER_TARGET</i> = 0x0010 | Deprecated.                                                                                                                                                                      |
| <i>RS_ALLOCATION_USAGE_IO_INPUT</i> = 0x0020               | Allocation is used as a Surface consumer.                                                                                                                                        |
| <i>RS_ALLOCATION_USAGE_IO_OUTPUT</i> = 0x0040              | Allocation is used as a Surface producer.                                                                                                                                        |
| <i>RS_ALLOCATION_USAGE_SHARED</i> = 0x0080                 | Allocation's backing store is shared with another object (usually a Bitmap). Copying to or from the original source Bitmap will cause a synchronization rather than a full copy. |

These values are ORed together to specify which usages or memory spaces are relevant to an allocation or an operation on an allocation.

### rs\_data\_kind : Element data kind

An enum with the following values:    Added in [API level 16](#)

|                                 |                              |
|---------------------------------|------------------------------|
| <i>RS_KIND_USER</i> = 0         | No special interpretation.   |
| <i>RS_KIND_PIXEL_L</i> = 7      | Luminance.                   |
| <i>RS_KIND_PIXEL_A</i> = 8      | Alpha.                       |
| <i>RS_KIND_PIXEL_LA</i> = 9     | Luminance and Alpha.         |
| <i>RS_KIND_PIXEL_RGB</i> = 10   | Red, Green, Blue.            |
| <i>RS_KIND_PIXEL_RGBA</i> = 11  | Red, Green, Blue, and Alpha. |
| <i>RS_KIND_PIXEL_DEPTH</i> = 12 | Depth for a depth texture.   |
| <i>RS_KIND_PIXEL_YUV</i> = 13   | Luminance and chrominance.   |
| <i>RS_KIND_INVALID</i> = 100    |                              |

This enumeration is primarily useful for graphical data. It provides additional information to help interpret the rs\_data\_type.

*RS\_KIND\_USER* indicates no special interpretation is expected.

The RS\_KIND\_PIXEL\_\* values are used in conjunction with the standard data types for representing texture formats.

See the [Element.createPixel\(\)](#) method.

### rs\_data\_type : Element basic data type

An enum with the following values:    Added in [API level 16](#)

|                               |                                           |
|-------------------------------|-------------------------------------------|
| <i>RS_TYPE_NONE</i> = 0       | Element is a complex type, i.e. a struct. |
| <i>RS_TYPE_FLOAT_16</i> = 1   | A 16 bit floating point value.            |
| <i>RS_TYPE_FLOAT_32</i> = 2   | A 32 bit floating point value.            |
| <i>RS_TYPE_FLOAT_64</i> = 3   | A 64 bit floating point value.            |
| <i>RS_TYPE_SIGNED_8</i> = 4   | An 8 bit signed integer.                  |
| <i>RS_TYPE_SIGNED_16</i> = 5  | A 16 bit signed integer.                  |
| <i>RS_TYPE_SIGNED_32</i> = 6  | A 32 bit signed integer.                  |
| <i>RS_TYPE_SIGNED_64</i> = 7  | A 64 bit signed integer.                  |
| <i>RS_TYPE_UNSIGNED_8</i> = 8 | An 8 bit unsigned integer.                |

|                                        |                                                                                  |
|----------------------------------------|----------------------------------------------------------------------------------|
| <i>RS_TYPE_UNSIGNED_16</i> = 9         | A 16 bit unsigned integer.                                                       |
| <i>RS_TYPE_UNSIGNED_32</i> = 10        | A 32 bit unsigned integer.                                                       |
| <i>RS_TYPE_UNSIGNED_64</i> = 11        | A 64 bit unsigned integer.                                                       |
| <i>RS_TYPE_BOOLEAN</i> = 12            | 0 or 1 (false or true) stored in an 8 bit container.                             |
| <i>RS_TYPE_UNSIGNED_5_6_5</i> = 13     | A 16 bit unsigned integer packing graphical data in 5, 6, and 5 bit sections.    |
| <i>RS_TYPE_UNSIGNED_5_5_5_1</i> = 14   | A 16 bit unsigned integer packing graphical data in 5, 5, 5, and 1 bit sections. |
| <i>RS_TYPE_UNSIGNED_4_4_4_4</i> = 15   | A 16 bit unsigned integer packing graphical data in 4, 4, 4, and 4 bit sections. |
| <i>RS_TYPE_MATRIX_4X4</i> = 16         | A 4x4 matrix of 32 bit floats, aligned on a 32 bit boundary.                     |
| <i>RS_TYPE_MATRIX_3X3</i> = 17         | A 3x3 matrix of 32 bit floats, aligned on a 32 bit boundary.                     |
| <i>RS_TYPE_MATRIX_2X2</i> = 18         | A 2x2 matrix of 32 bit floats, aligned on a 32 bit boundary.                     |
| <i>RS_TYPE_ELEMENT</i> = 1000          | A handle to an Element.                                                          |
| <i>RS_TYPE_TYPE</i> = 1001             | A handle to a Type.                                                              |
| <i>RS_TYPE_ALLOCATION</i> = 1002       | A handle to an Allocation.                                                       |
| <i>RS_TYPE_SAMPLER</i> = 1003          | A handle to a Sampler.                                                           |
| <i>RS_TYPE_SCRIPT</i> = 1004           | A handle to a Script.                                                            |
| <i>RS_TYPE_MESH</i> = 1005             | Deprecated.                                                                      |
| <i>RS_TYPE_PROGRAM_FRAGMENT</i> = 1006 | Deprecated.                                                                      |
| <i>RS_TYPE_PROGRAM_VERTEX</i> = 1007   | Deprecated.                                                                      |
| <i>RS_TYPE_PROGRAM_RASTER</i> = 1008   | Deprecated.                                                                      |
| <i>RS_TYPE_PROGRAM_STORE</i> = 1009    | Deprecated.                                                                      |
| <i>RS_TYPE_FONT</i> = 1010             | Deprecated.                                                                      |
| <i>RS_TYPE_INVALID</i> = 10000         |                                                                                  |

*rs\_data\_type* is used to encode the type information of a basic element.

*RS\_TYPE\_UNSIGNED\_5\_6\_5*, *RS\_TYPE\_UNSIGNED\_5\_5\_5\_1*, *RS\_TYPE\_UNSIGNED\_4\_4\_4\_4* are for packed graphical data formats and represent vectors with per vector member sizes which are treated as a single unit for packing and alignment purposes.

#### *rs\_element* : Handle to an element

An opaque handle to a RenderScript element.

See [android.renderscript.Element](#).

#### *rs\_sampler* : Handle to a Sampler

An opaque handle to a RenderScript sampler object.

See [android.renderscript.Sampler](#).

#### *rs\_sampler\_value* : Sampler wrap T value

An enum with the following values:    Added in [API level 16](#)

|                                          |
|------------------------------------------|
| <i>RS_SAMPLER_NEAREST</i> = 0            |
| <i>RS_SAMPLER_LINEAR</i> = 1             |
| <i>RS_SAMPLER_LINEAR_MIP_LINEAR</i> = 2  |
| <i>RS_SAMPLER_WRAP</i> = 3               |
| <i>RS_SAMPLER_CLAMP</i> = 4              |
| <i>RS_SAMPLER_LINEAR_MIP_NEAREST</i> = 5 |
| <i>RS_SAMPLER_MIRRORED_REPEAT</i> = 6    |
| <i>RS_SAMPLER_INVALID</i> = 100          |

## rs\_script : Handle to a Script

An opaque handle to a RenderScript script object.

See [android.renderscript.ScriptC](#).

## rs\_type : Handle to a Type

An opaque handle to a RenderScript type.

See [android.renderscript.Type](#).

## rs\_yuv\_format : YUV format

An enum with the following values:    Added in [API level 24](#)

*RS\_YUV\_NONE = 0*

*RS\_YUV\_YV12 = 0x32315659*

*RS\_YUV\_NV21 = 0x11*

*RS\_YUV\_420\_888 = 0x23*

Android YUV formats that can be associated with a RenderScript Type.

See [android.graphics.ImageFormat](#) for a description of each format.

# RenderScript Conversion Functions

## Overview

The functions below convert from a numerical vector type to another, or from one color representation to another.

## Summary

| Functions                         |                                  |
|-----------------------------------|----------------------------------|
| <a href="#">convert</a>           | Convert numerical vectors        |
| <a href="#">rsPackColorTo8888</a> | Create a uchar4 RGBA from floats |
| <a href="#">rsUnpackColor8888</a> | Create a float4 RGBA from uchar4 |
| <a href="#">rsYuvToRGBA</a>       | Convert a YUV value to RGBA      |

## Functions

### convert : Convert numerical vectors

```
char2 convert_char2(char2 v);
char2 convert_char2(double2 v); Added in API level 21
char2 convert_char2(float2 v);
char2 convert_char2(half2 v); Added in API level 24
char2 convert_char2(int2 v);
char2 convert_char2(long2 v); Added in API level 21
char2 convert_char2(short2 v);
char2 convert_char2(uchar2 v);
char2 convert_char2(uint2 v);
char2 convert_char2(ulong2 v); Added in API level 21
char2 convert_char2(ushort2 v);
char3 convert_char3(char3 v);
char3 convert_char3(double3 v); Added in API level 21
char3 convert_char3(float3 v);
char3 convert_char3(half3 v); Added in API level 24
char3 convert_char3(int3 v);
char3 convert_char3(long3 v); Added in API level 21
char3 convert_char3(short3 v);
char3 convert_char3(uchar3 v);
char3 convert_char3(uint3 v);
char3 convert_char3(ulong3 v); Added in API level 21
char3 convert_char3(ushort3 v);
char4 convert_char4(char4 v);
char4 convert_char4(double4 v); Added in API level 21
```

```
char4 convert_char4(float4 v);

char4 convert_char4(half4 v); Added in API level 24
char4 convert_char4(int4 v);
char4 convert_char4(long4 v); Added in API level 21
char4 convert_char4(short4 v);
char4 convert_char4(uchar4 v);
char4 convert_char4(uint4 v);
char4 convert_char4(ulong4 v); Added in API level 21
char4 convert_char4(ushort4 v);

double2 convert_double2(char2 v); Added in API level 21
double2 convert_double2(double2 v); Added in API level 21
double2 convert_double2(float2 v); Added in API level 21
double2 convert_double2(half2 v); Added in API level 24
double2 convert_double2(int2 v); Added in API level 21
double2 convert_double2(long2 v); Added in API level 21
double2 convert_double2(short2 v); Added in API level 21
double2 convert_double2(uchar2 v); Added in API level 21
double2 convert_double2(uint2 v); Added in API level 21
double2 convert_double2(ulong2 v); Added in API level 21
double2 convert_double2(ushort2 v); Added in API level 21
double3 convert_double3(char3 v); Added in API level 21
double3 convert_double3(double3 v); Added in API level 21
double3 convert_double3(float3 v); Added in API level 21
double3 convert_double3(half3 v); Added in API level 24
double3 convert_double3(int3 v); Added in API level 21
double3 convert_double3(long3 v); Added in API level 21
double3 convert_double3(short3 v); Added in API level 21
double3 convert_double3(uchar3 v); Added in API level 21
double3 convert_double3(uint3 v); Added in API level 21
double3 convert_double3(ulong3 v); Added in API level 21
double3 convert_double3(ushort3 v); Added in API level 21
double4 convert_double4(char4 v); Added in API level 21
double4 convert_double4(double4 v); Added in API level 21
double4 convert_double4(float4 v); Added in API level 21
double4 convert_double4(half4 v); Added in API level 24
double4 convert_double4(int4 v); Added in API level 21
double4 convert_double4(long4 v); Added in API level 21
double4 convert_double4(short4 v); Added in API level 21
double4 convert_double4(uchar4 v); Added in API level 21
double4 convert_double4(uint4 v); Added in API level 21
double4 convert_double4(ulong4 v); Added in API level 21
double4 convert_double4(ushort4 v); Added in API level 21
float2 convert_float2(char2 v);
float2 convert_float2(double2 v); Added in API level 21
float2 convert_float2(float2 v);
float2 convert_float2(half2 v); Added in API level 24
```

```
float2 convert_float2(int2 v);
float2 convert_float2(long2 v);

float2 convert_float2(short2 v);
float2 convert_float2(uchar2 v);
float2 convert_float2(uint2 v);
float2 convert_float2(ulong2 v);
Added in API level 21

float2 convert_float2(ushort2 v);

float3 convert_float3(char3 v);
float3 convert_float3(double3 v);
Added in API level 21

float3 convert_float3(float3 v);
float3 convert_float3(half3 v);
Added in API level 24

float3 convert_float3(int3 v);
float3 convert_float3(long3 v);
Added in API level 21

float3 convert_float3(short3 v);
float3 convert_float3(uchar3 v);
float3 convert_float3(uint3 v);
float3 convert_float3(ulong3 v);
Added in API level 21

float3 convert_float3(ushort3 v);

float4 convert_float4(char4 v);
float4 convert_float4(double4 v);
Added in API level 21

float4 convert_float4(float4 v);
float4 convert_float4(half4 v);
Added in API level 24

float4 convert_float4(int4 v);
float4 convert_float4(long4 v);
Added in API level 21

float4 convert_float4(short4 v);
float4 convert_float4(uchar4 v);
float4 convert_float4(uint4 v);
float4 convert_float4(ulong4 v);
Added in API level 21

float4 convert_float4(ushort4 v);

half2 convert_half2(char2 v);
Added in API level 24
half2 convert_half2(double2 v);
Added in API level 24
half2 convert_half2(float2 v);
Added in API level 24
half2 convert_half2(half2 v);
Added in API level 24
half2 convert_half2(int2 v);
Added in API level 24
half2 convert_half2(long2 v);
Added in API level 24
half2 convert_half2(short2 v);
Added in API level 24
half2 convert_half2(uchar2 v);
Added in API level 24
half2 convert_half2(uint2 v);
Added in API level 24
half2 convert_half2(ulong2 v);
Added in API level 24
half2 convert_half2(ushort2 v);
Added in API level 24
half3 convert_half3(char3 v);
Added in API level 24
half3 convert_half3(double3 v);
Added in API level 24
half3 convert_half3(float3 v);
Added in API level 24
half3 convert_half3(half3 v);
Added in API level 24
half3 convert_half3(int3 v);
Added in API level 24
half3 convert_half3(long3 v);
Added in API level 24
half3 convert_half3(short3 v);
Added in API level 24
```

|                                 |                       |
|---------------------------------|-----------------------|
| half3 convert_half3(uchar3 v);  | Added in API level 24 |
| half3 convert_half3(uint3 v);   | Added in API level 24 |
| half3 convert_half3(ulong3 v);  | Added in API level 24 |
| half3 convert_half3(ushort3 v); | Added in API level 24 |
| half4 convert_half4(char4 v);   | Added in API level 24 |
| half4 convert_half4(double4 v); | Added in API level 24 |
| half4 convert_half4(float4 v);  | Added in API level 24 |
| half4 convert_half4(half4 v);   | Added in API level 24 |
| half4 convert_half4(int4 v);    | Added in API level 24 |
| half4 convert_half4(long4 v);   | Added in API level 24 |
| half4 convert_half4(short4 v);  | Added in API level 24 |
| half4 convert_half4(uchar4 v);  | Added in API level 24 |
| half4 convert_half4(uint4 v);   | Added in API level 24 |
| half4 convert_half4(ulong4 v);  | Added in API level 24 |
| half4 convert_half4(ushort4 v); | Added in API level 24 |
| int2 convert_int2(char2 v);     |                       |
| int2 convert_int2(double2 v);   | Added in API level 21 |
| int2 convert_int2(float2 v);    |                       |
| int2 convert_int2(half2 v);     | Added in API level 24 |
| int2 convert_int2(int2 v);      |                       |
| int2 convert_int2(long2 v);     | Added in API level 21 |
| int2 convert_int2(short2 v);    |                       |
| int2 convert_int2(uchar2 v);    |                       |
| int2 convert_int2(uint2 v);     |                       |
| int2 convert_int2(ulong2 v);    | Added in API level 21 |
| int2 convert_int2(ushort2 v);   |                       |
| int3 convert_int3(char3 v);     |                       |
| int3 convert_int3(double3 v);   | Added in API level 21 |
| int3 convert_int3(float3 v);    |                       |
| int3 convert_int3(half3 v);     | Added in API level 24 |
| int3 convert_int3(int3 v);      |                       |
| int3 convert_int3(long3 v);     | Added in API level 21 |
| int3 convert_int3(short3 v);    |                       |
| int3 convert_int3(uchar3 v);    |                       |
| int3 convert_int3(uint3 v);     |                       |
| int3 convert_int3(ulong3 v);    | Added in API level 21 |
| int3 convert_int3(ushort3 v);   |                       |
| int4 convert_int4(char4 v);     |                       |
| int4 convert_int4(double4 v);   | Added in API level 21 |
| int4 convert_int4(float4 v);    |                       |
| int4 convert_int4(half4 v);     | Added in API level 24 |
| int4 convert_int4(int4 v);      |                       |
| int4 convert_int4(long4 v);     | Added in API level 21 |
| int4 convert_int4(short4 v);    |                       |
| int4 convert_int4(uchar4 v);    |                       |
| int4 convert_int4(uint4 v);     |                       |

|                                                |                       |
|------------------------------------------------|-----------------------|
| <code>int4 convert_int4(ulong4 v);</code>      | Added in API level 21 |
| <code>int4 convert_int4(ushort4 v);</code>     |                       |
| <code>long2 convert_long2(char2 v);</code>     | Added in API level 21 |
| <code>long2 convert_long2(double2 v);</code>   | Added in API level 21 |
| <code>long2 convert_long2(float2 v);</code>    | Added in API level 21 |
| <code>long2 convert_long2(half2 v);</code>     | Added in API level 24 |
| <code>long2 convert_long2(int2 v);</code>      | Added in API level 21 |
| <code>long2 convert_long2(long2 v);</code>     | Added in API level 21 |
| <code>long2 convert_long2(short2 v);</code>    | Added in API level 21 |
| <code>long2 convert_long2(uchar2 v);</code>    | Added in API level 21 |
| <code>long2 convert_long2(uint2 v);</code>     | Added in API level 21 |
| <code>long2 convert_long2(ulong2 v);</code>    | Added in API level 21 |
| <code>long2 convert_long2(ushort2 v);</code>   | Added in API level 21 |
| <code>long3 convert_long3(char3 v);</code>     | Added in API level 21 |
| <code>long3 convert_long3(double3 v);</code>   | Added in API level 21 |
| <code>long3 convert_long3(float3 v);</code>    | Added in API level 21 |
| <code>long3 convert_long3(half3 v);</code>     | Added in API level 24 |
| <code>long3 convert_long3(int3 v);</code>      | Added in API level 21 |
| <code>long3 convert_long3(long3 v);</code>     | Added in API level 21 |
| <code>long3 convert_long3(short3 v);</code>    | Added in API level 21 |
| <code>long3 convert_long3(uchar3 v);</code>    | Added in API level 21 |
| <code>long3 convert_long3(uint3 v);</code>     | Added in API level 21 |
| <code>long3 convert_long3(ulong3 v);</code>    | Added in API level 21 |
| <code>long3 convert_long3(ushort3 v);</code>   | Added in API level 21 |
| <code>long4 convert_long4(char4 v);</code>     | Added in API level 21 |
| <code>long4 convert_long4(double4 v);</code>   | Added in API level 21 |
| <code>long4 convert_long4(float4 v);</code>    | Added in API level 21 |
| <code>long4 convert_long4(half4 v);</code>     | Added in API level 24 |
| <code>long4 convert_long4(int4 v);</code>      | Added in API level 21 |
| <code>long4 convert_long4(long4 v);</code>     | Added in API level 21 |
| <code>long4 convert_long4(short4 v);</code>    | Added in API level 21 |
| <code>long4 convert_long4(uchar4 v);</code>    | Added in API level 21 |
| <code>long4 convert_long4(uint4 v);</code>     | Added in API level 21 |
| <code>long4 convert_long4(ulong4 v);</code>    | Added in API level 21 |
| <code>long4 convert_long4(ushort4 v);</code>   | Added in API level 21 |
| <code>short2 convert_short2(char2 v);</code>   |                       |
| <code>short2 convert_short2(double2 v);</code> | Added in API level 21 |
| <code>short2 convert_short2(float2 v);</code>  |                       |
| <code>short2 convert_short2(half2 v);</code>   | Added in API level 24 |
| <code>short2 convert_short2(int2 v);</code>    |                       |
| <code>short2 convert_short2(long2 v);</code>   | Added in API level 21 |
| <code>short2 convert_short2(short2 v);</code>  |                       |
| <code>short2 convert_short2(uchar2 v);</code>  |                       |
| <code>short2 convert_short2(uint2 v);</code>   |                       |
| <code>short2 convert_short2(ulong2 v);</code>  | Added in API level 21 |
| <code>short2 convert_short2(ushort2 v);</code> |                       |
| <code>short3 convert_short3(char3 v);</code>   |                       |

`short3 convert_short3(double3 v);` Added in API level 21  
`short3 convert_short3(float3 v);`  
`short3 convert_short3(half3 v);` Added in API level 24  
`short3 convert_short3(int3 v);`  
`short3 convert_short3(long3 v);` Added in API level 21  
`short3 convert_short3(short3 v);`  
`short3 convert_short3(uchar3 v);`  
`short3 convert_short3(uint3 v);`  
`short3 convert_short3(ulong3 v);` Added in API level 21  
`short3 convert_short3(ushort3 v);`  
`short4 convert_short4(char4 v);`  
`short4 convert_short4(double4 v);` Added in API level 21  
`short4 convert_short4(float4 v);`  
`short4 convert_short4(half4 v);` Added in API level 24  
`short4 convert_short4(int4 v);`  
`short4 convert_short4(long4 v);` Added in API level 21  
`short4 convert_short4(short4 v);`  
`short4 convert_short4(uchar4 v);`  
`short4 convert_short4(uint4 v);`  
`short4 convert_short4(ulong4 v);` Added in API level 21  
`short4 convert_short4(ushort4 v);`  
`uchar2 convert_uchar2(char2 v);`  
`uchar2 convert_uchar2(double2 v);` Added in API level 21  
`uchar2 convert_uchar2(float2 v);`  
`uchar2 convert_uchar2(half2 v);` Added in API level 24  
`uchar2 convert_uchar2(int2 v);`  
`uchar2 convert_uchar2(long2 v);` Added in API level 21  
`uchar2 convert_uchar2(short2 v);`  
`uchar2 convert_uchar2(uchar2 v);`  
`uchar2 convert_uchar2(uint2 v);`  
`uchar2 convert_uchar2(ulong2 v);` Added in API level 21  
`uchar2 convert_uchar2(ushort2 v);`  
`uchar3 convert_uchar3(char3 v);`  
`uchar3 convert_uchar3(double3 v);` Added in API level 21  
`uchar3 convert_uchar3(float3 v);`  
`uchar3 convert_uchar3(half3 v);` Added in API level 24  
`uchar3 convert_uchar3(int3 v);`  
`uchar3 convert_uchar3(long3 v);` Added in API level 21  
`uchar3 convert_uchar3(short3 v);`  
`uchar3 convert_uchar3(uchar3 v);`  
`uchar3 convert_uchar3(uint3 v);`  
`uchar3 convert_uchar3(ulong3 v);` Added in API level 21  
`uchar3 convert_uchar3(ushort3 v);`  
`uchar4 convert_uchar4(char4 v);`  
`uchar4 convert_uchar4(double4 v);` Added in API level 21  
`uchar4 convert_uchar4(float4 v);`

|                                                |                       |
|------------------------------------------------|-----------------------|
| <code>uchar4 convert_uchar4(half4 v);</code>   | Added in API level 24 |
| <code>uchar4 convert_uchar4(int4 v);</code>    |                       |
| <code>uchar4 convert_uchar4(long4 v);</code>   | Added in API level 21 |
| <code>uchar4 convert_uchar4(short4 v);</code>  |                       |
| <code>uchar4 convert_uchar4(uchar4 v);</code>  |                       |
| <code>uchar4 convert_uchar4(uint4 v);</code>   |                       |
| <code>uchar4 convert_uchar4(ulong4 v);</code>  | Added in API level 21 |
| <code>uchar4 convert_uchar4(ushort4 v);</code> |                       |
| <code>uint2 convert_uint2(char2 v);</code>     |                       |
| <code>uint2 convert_uint2(double2 v);</code>   | Added in API level 21 |
| <code>uint2 convert_uint2(float2 v);</code>    |                       |
| <code>uint2 convert_uint2(half2 v);</code>     | Added in API level 24 |
| <code>uint2 convert_uint2(int2 v);</code>      |                       |
| <code>uint2 convert_uint2(long2 v);</code>     | Added in API level 21 |
| <code>uint2 convert_uint2(short2 v);</code>    |                       |
| <code>uint2 convert_uint2(uchar2 v);</code>    |                       |
| <code>uint2 convert_uint2(uint2 v);</code>     |                       |
| <code>uint2 convert_uint2(ulong2 v);</code>    | Added in API level 21 |
| <code>uint2 convert_uint2(ushort2 v);</code>   |                       |
| <code>uint3 convert_uint3(char3 v);</code>     |                       |
| <code>uint3 convert_uint3(double3 v);</code>   | Added in API level 21 |
| <code>uint3 convert_uint3(float3 v);</code>    |                       |
| <code>uint3 convert_uint3(half3 v);</code>     | Added in API level 24 |
| <code>uint3 convert_uint3(int3 v);</code>      |                       |
| <code>uint3 convert_uint3(long3 v);</code>     | Added in API level 21 |
| <code>uint3 convert_uint3(short3 v);</code>    |                       |
| <code>uint3 convert_uint3(uchar3 v);</code>    |                       |
| <code>uint3 convert_uint3(uint3 v);</code>     |                       |
| <code>uint3 convert_uint3(ulong3 v);</code>    | Added in API level 21 |
| <code>uint3 convert_uint3(ushort3 v);</code>   |                       |
| <code>uint4 convert_uint4(char4 v);</code>     |                       |
| <code>uint4 convert_uint4(double4 v);</code>   | Added in API level 21 |
| <code>uint4 convert_uint4(float4 v);</code>    |                       |
| <code>uint4 convert_uint4(half4 v);</code>     | Added in API level 24 |
| <code>uint4 convert_uint4(int4 v);</code>      |                       |
| <code>uint4 convert_uint4(long4 v);</code>     | Added in API level 21 |
| <code>uint4 convert_uint4(short4 v);</code>    |                       |
| <code>uint4 convert_uint4(uchar4 v);</code>    |                       |
| <code>uint4 convert_uint4(uint4 v);</code>     |                       |
| <code>uint4 convert_uint4(ulong4 v);</code>    | Added in API level 21 |
| <code>uint4 convert_uint4(ushort4 v);</code>   |                       |
| <code>ulong2 convert_ulong2(char2 v);</code>   | Added in API level 21 |
| <code>ulong2 convert_ulong2(double2 v);</code> | Added in API level 21 |
| <code>ulong2 convert_ulong2(float2 v);</code>  | Added in API level 21 |
| <code>ulong2 convert_ulong2(half2 v);</code>   | Added in API level 24 |
| <code>ulong2 convert_ulong2(int2 v);</code>    | Added in API level 21 |
| <code>ulong2 convert_ulong2(long2 v);</code>   | Added in API level 21 |

|                                                  |                                       |
|--------------------------------------------------|---------------------------------------|
| <code>ulong2 convert_ulong2(short2 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong2 convert_ulong2(uchar2 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong2 convert_ulong2(uint2 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong2 convert_ulong2(ulong2 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong2 convert_ulong2(ushort2 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(char3 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(double3 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(float3 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(half3 v);</code>     | Added in <a href="#">API level 24</a> |
| <code>ulong3 convert_ulong3(int3 v);</code>      | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(long3 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(short3 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(uchar3 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(uint3 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(ulong3 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong3 convert_ulong3(ushort3 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(char4 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(double4 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(float4 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(half4 v);</code>     | Added in <a href="#">API level 24</a> |
| <code>ulong4 convert_ulong4(int4 v);</code>      | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(long4 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(short4 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(uchar4 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(uint4 v);</code>     | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(ulong4 v);</code>    | Added in <a href="#">API level 21</a> |
| <code>ulong4 convert_ulong4(ushort4 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ushort2 convert_ushort2(char2 v);</code>   |                                       |
| <code>ushort2 convert_ushort2(double2 v);</code> | Added in <a href="#">API level 21</a> |
| <code>ushort2 convert_ushort2(float2 v);</code>  |                                       |
| <code>ushort2 convert_ushort2(half2 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>ushort2 convert_ushort2(int2 v);</code>    |                                       |
| <code>ushort2 convert_ushort2(long2 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ushort2 convert_ushort2(short2 v);</code>  |                                       |
| <code>ushort2 convert_ushort2(uchar2 v);</code>  |                                       |
| <code>ushort2 convert_ushort2(uint2 v);</code>   |                                       |
| <code>ushort2 convert_ushort2(ulong2 v);</code>  | Added in <a href="#">API level 21</a> |
| <code>ushort2 convert_ushort2(ushort2 v);</code> |                                       |
| <code>ushort3 convert_ushort3(char3 v);</code>   |                                       |
| <code>ushort3 convert_ushort3(double3 v);</code> | Added in <a href="#">API level 21</a> |
| <code>ushort3 convert_ushort3(float3 v);</code>  |                                       |
| <code>ushort3 convert_ushort3(half3 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>ushort3 convert_ushort3(int3 v);</code>    |                                       |
| <code>ushort3 convert_ushort3(long3 v);</code>   | Added in <a href="#">API level 21</a> |
| <code>ushort3 convert_ushort3(short3 v);</code>  |                                       |
| <code>ushort3 convert_ushort3(uchar3 v);</code>  |                                       |

```
ushort3 convert_ushort3(uint3 v); Added in API level 21
ushort3 convert_ushort3(ulong3 v);
ushort3 convert_ushort3(ushort3 v);

ushort4 convert_ushort4(char4 v);
ushort4 convert_ushort4(double4 v); Added in API level 21
ushort4 convert_ushort4(float4 v);
ushort4 convert_ushort4(half4 v); Added in API level 24
ushort4 convert_ushort4(int4 v);
ushort4 convert_ushort4(long4 v); Added in API level 21
ushort4 convert_ushort4(short4 v);
ushort4 convert_ushort4(uchar4 v);
ushort4 convert_ushort4(uint4 v);
ushort4 convert_ushort4(ulong4 v); Added in API level 21
ushort4 convert_ushort4(ushort4 v);
```

Converts a vector from one numerical type to another. The conversion are done entry per entry.

E.g calling `a = convert_short3(b);` is equivalent to doing `a.x = (short)b.x; a.y = (short)b.y; a.z = (short)b.z;`.

Converting floating point values to integer types truncates.

Converting numbers too large to fit the destination type yields undefined results. For example, converting a float that contains 1.0e18 to a short is undefined. Use `clamp()` to avoid this.

#### rsPackColorTo8888 : Create a uchar4 RGBA from floats

```
uchar4 rsPackColorTo8888(float r, float g, float b);
uchar4 rsPackColorTo8888(float r, float g, float b, float a);
uchar4 rsPackColorTo8888(float3 color);
uchar4 rsPackColorTo8888(float4 color);
```

##### Parameters

*r* Red component.  
*g* Green component.  
*b* Blue component.  
*a* Alpha component.  
*color* Vector of 3 or 4 floats containing the R, G, B, and A values.

Packs three or four floating point RGBA values into a uchar4.

The input values are typically between 0.0f and 1.0f inclusive. For input values outside of this range, the resulting outputs will be clamped to be between 0 and 255. As this clamping may be done after the input is multiplied by 255.f and converted to an integer, input numbers greater than INT\_MAX/255.f or less than INT\_MIN/255.f result in undefined behavior.

If the alpha component is not specified, it is assumed to be 1.0, i.e. the result will have an alpha set to 255.

#### rsUnpackColor8888 : Create a float4 RGBA from uchar4

```
float4 rsUnpackColor8888(uchar4 c);
```

Unpacks a uchar4 color to float4. The resulting floats will be between 0.0 and 1.0 inclusive.

#### rsYuvToRGBA : Convert a YUV value to RGBA

```
float4 rsYuvToRGBA_float4(uchar y, uchar u, uchar v);
uchar4 rsYuvToRGBA_uchar4(uchar y, uchar u, uchar v);
```

## Parameters

- $y$  Luminance component.
- $u$  U chrominance component.
- $v$  V chrominance component.

Converts a color from a YUV representation to RGBA.

We currently don't provide a function to do the reverse conversion.

# RenderScript Mathematical Constants and Functions

## Overview

The mathematical functions below can be applied to scalars and vectors. When applied to vectors, the returned value is a vector of the function applied to each entry of the input.

For example:

```
float3 a, b;
// The following call sets
// a.x to sin(b.x),
// a.y to sin(b.y), and
// a.z to sin(b.z).
a = sin(b);
```

See [Vector Math Functions](#) for functions like `distance()` and `length()` that interpret instead the input as a single vector in n-dimensional space.

The precision of the mathematical operations on 32 bit floats is affected by the pragmas `rs_fp_relaxed` and `rs_fp_full`. Under `rs_fp_relaxed`, subnormal values may be flushed to zero and rounding may be done towards zero. In comparison, `rs_fp_full` requires correct handling of subnormal values, i.e. smaller than `1.17549435e-38f`. `rs_fp_rull` also requires round to nearest with ties to even.

Different precision/speed tradeoffs can be achieved by using variants of the common math functions. Functions with a name starting with

- `native_`: May have custom hardware implementations with weaker precision. Additionally, subnormal values may be flushed to zero, rounding towards zero may be used, and NaN and infinity input may not be handled correctly.
- `half_`: May perform internal computations using 16 bit floats. Additionally, subnormal values may be flushed to zero, and rounding towards zero may be used.

## Summary

| Constants               |                                               |
|-------------------------|-----------------------------------------------|
| <code>M_1_PI</code>     | <code>1 / pi</code> , as a 32 bit float       |
| <code>M_2_PI</code>     | <code>2 / pi</code> , as a 32 bit float       |
| <code>M_2_SQRTPI</code> | <code>2 / sqrt(pi)</code> , as a 32 bit float |
| <code>M_E</code>        | <code>e</code> , as a 32 bit float            |
| <code>M_LN10</code>     | <code>log_e(10)</code> , as a 32 bit float    |
| <code>M_LN2</code>      | <code>log_e(2)</code> , as a 32 bit float     |
| <code>M_LOG10E</code>   | <code>log_10(e)</code> , as a 32 bit float    |
| <code>M_LOG2E</code>    | <code>log_2(e)</code> , as a 32 bit float     |
| <code>M_PI</code>       | <code>pi</code> , as a 32 bit float           |
| <code>M_PI_2</code>     | <code>pi / 2</code> , as a 32 bit float       |
| <code>M_PI_4</code>     | <code>pi / 4</code> , as a 32 bit float       |
| <code>M_SQRT1_2</code>  | <code>1 / sqrt(2)</code> , as a 32 bit float  |
| <code>M_SQRT2</code>    | <code>sqrt(2)</code> , as a 32 bit float      |

| Functions               |                                                          |
|-------------------------|----------------------------------------------------------|
| <code>abs</code>        | Absolute value of an integer                             |
| <code>acos</code>       | Inverse cosine                                           |
| <code>acosh</code>      | Inverse hyperbolic cosine                                |
| <code>acospi</code>     | Inverse cosine divided by pi                             |
| <code>asin</code>       | Inverse sine                                             |
| <code>asinh</code>      | Inverse hyperbolic sine                                  |
| <code>asinpi</code>     | Inverse sine divided by pi                               |
| <code>atan</code>       | Inverse tangent                                          |
| <code>atan2</code>      | Inverse tangent of a ratio                               |
| <code>atan2pi</code>    | Inverse tangent of a ratio, divided by pi                |
| <code>atanh</code>      | Inverse hyperbolic tangent                               |
| <code>atanpi</code>     | Inverse tangent divided by pi                            |
| <code>cbrt</code>       | Cube root                                                |
| <code>ceil</code>       | Smallest integer not less than a value                   |
| <code>clamp</code>      | Restrain a value to a range                              |
| <code>clz</code>        | Number of leading 0 bits                                 |
| <code>copysign</code>   | Copies the sign of a number to another                   |
| <code>cos</code>        | Cosine                                                   |
| <code>cosh</code>       | Hypebolic cosine                                         |
| <code>cospi</code>      | Cosine of a number multiplied by pi                      |
| <code>degrees</code>    | Converts radians into degrees                            |
| <code>erf</code>        | Mathematical error function                              |
| <code>erfc</code>       | Mathematical complementary error function                |
| <code>exp</code>        | e raised to a number                                     |
| <code>exp10</code>      | 10 raised to a number                                    |
| <code>exp2</code>       | 2 raised to a number                                     |
| <code>expm1</code>      | e raised to a number minus one                           |
| <code>fabs</code>       | Absolute value of a float                                |
| <code>fdim</code>       | Positive difference between two values                   |
| <code>floor</code>      | Smallest integer not greater than a value                |
| <code>fma</code>        | Multiply and add                                         |
| <code>fmax</code>       | Maximum of two floats                                    |
| <code>fmin</code>       | Minimum of two floats                                    |
| <code>fmod</code>       | Modulo                                                   |
| <code>fract</code>      | Positive fractional part                                 |
| <code>frexp</code>      | Binary mantissa and exponent                             |
| <code>half_recip</code> | Reciprocal computed to 16 bit precision                  |
| <code>half_rsqrt</code> | Reciprocal of a square root computed to 16 bit precision |
| <code>half_sqrt</code>  | Square root computed to 16 bit precision                 |

|                |                                                       |
|----------------|-------------------------------------------------------|
| hypot          | Hypotenuse                                            |
| ilogb          | Base two exponent                                     |
| ldexp          | Creates a floating point from mantissa and exponent   |
| lgamma         | Natural logarithm of the gamma function               |
| log            | Natural logarithm                                     |
| log10          | Base 10 logarithm                                     |
| log1p          | Natural logarithm of a value plus 1                   |
| log2           | Base 2 logarithm                                      |
| logb           | Base two exponent                                     |
| mad            | Multiply and add                                      |
| max            | Maximum                                               |
| min            | Minimum                                               |
| mix            | Mixes two values                                      |
| modf           | Integral and fractional components                    |
| nan            | Not a Number                                          |
| nan_half       | Not a Number                                          |
| native_acos    | Approximate inverse cosine                            |
| native_acosh   | Approximate inverse hyperbolic cosine                 |
| native_acospi  | Approximate inverse cosine divided by pi              |
| native_asin    | Approximate inverse sine                              |
| native_asinh   | Approximate inverse hyperbolic sine                   |
| native_asinpi  | Approximate inverse sine divided by pi                |
| native_atan    | Approximate inverse tangent                           |
| native_atan2   | Approximate inverse tangent of a ratio                |
| native_atan2pi | Approximate inverse tangent of a ratio, divided by pi |
| native_atanh   | Approximate inverse hyperbolic tangent                |
| native_atanpi  | Approximate inverse tangent divided by pi             |
| native_cbrt    | Approximate cube root                                 |
| native_cos     | Approximate cosine                                    |
| native_cosh    | Approximate hyperbolic cosine                         |
| native_cospi   | Approximate cosine of a number multiplied by pi       |
| native_divide  | Approximate division                                  |
| native_exp     | Approximate e raised to a number                      |
| native_exp10   | Approximate 10 raised to a number                     |
| native_exp2    | Approximate 2 raised to a number                      |
| native_expm1   | Approximate e raised to a number minus one            |
| native_hypot   | Approximate hypotenuse                                |
| native_log     | Approximate natural logarithm                         |
| native_log10   | Approximate base 10 logarithm                         |

|               |                                                  |
|---------------|--------------------------------------------------|
| native_log1p  | Approximate natural logarithm of a value plus 1  |
| native_log2   | Approximate base 2 logarithm                     |
| native_powr   | Approximate positive base raised to an exponent  |
| native_recip  | Approximate reciprocal                           |
| native_rootn  | Approximate nth root                             |
| native_rsqrt  | Approximate reciprocal of a square root          |
| native_sin    | Approximate sine                                 |
| native_sincos | Approximate sine and cosine                      |
| native_sinh   | Approximate hyperbolic sine                      |
| native_sinpi  | Approximate sine of a number multiplied by pi    |
| native_sqrt   | Approximate square root                          |
| native_tan    | Approximate tangent                              |
| native_tanh   | Approximate hyperbolic tangent                   |
| native_tanpi  | Approximate tangent of a number multiplied by pi |
| nextafter     | Next floating point number                       |
| pow           | Base raised to an exponent                       |
| pown          | Base raised to an integer exponent               |
| powr          | Positive base raised to an exponent              |
| radians       | Converts degrees into radians                    |
| remainder     | Remainder of a division                          |
| remquo        | Remainder and quotient of a division             |
| rint          | Round to even                                    |
| rootn         | Nth root                                         |
| round         | Round away from zero                             |
| rsRand        | Pseudo-random number                             |
| rsqrt         | Reciprocal of a square root                      |
| sign          | Sign of a value                                  |
| sin           | Sine                                             |
| sincos        | Sine and cosine                                  |
| sinh          | Hyperbolic sine                                  |
| sinpi         | Sine of a number multiplied by pi                |
| sqrt          | Square root                                      |
| step          | 0 if less than a value, 0 otherwise              |
| tan           | Tangent                                          |
| tanh          | Hyperbolic tangent                               |
| tanpi         | Tangent of a number multiplied by pi             |
| tgamma        | Gamma function                                   |
| trunc         | Truncates a floating point                       |

## Deprecated Functions

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <a href="#">rsClamp</a> | <b>Deprecated.</b> Restrain a value to a range            |
| <a href="#">rsFrac</a>  | <b>Deprecated.</b> Returns the fractional part of a float |

## Constants

---

**M\_1\_PI** :  $1 / \pi$ , as a 32 bit float

Value: 0.318309886183790671537767526745028724f

The inverse of pi, as a 32 bit float.

**M\_2\_PI** :  $2 / \pi$ , as a 32 bit float

Value: 0.636619772367581343075535053490057448f

2 divided by pi, as a 32 bit float.

**M\_2\_SQRTPI** :  $2 / \sqrt{\pi}$ , as a 32 bit float

Value: 1.128379167095512573896158903121545172f

2 divided by the square root of pi, as a 32 bit float.

**M\_E** : e, as a 32 bit float

Value: 2.718281828459045235360287471352662498f

The number e, the base of the natural logarithm, as a 32 bit float.

**M\_LN10** :  $\log_e(10)$ , as a 32 bit float

Value: 2.302585092994045684017991454684364208f

The natural logarithm of 10, as a 32 bit float.

**M\_LN2** :  $\log_e(2)$ , as a 32 bit float

Value: 0.693147180559945309417232121458176568f

The natural logarithm of 2, as a 32 bit float.

**M\_LOG10E** :  $\log_{10}(e)$ , as a 32 bit float

Value: 0.434294481903251827651128918916605082f

The logarithm base 10 of e, as a 32 bit float.

**M\_LOG2E** :  $\log_2(e)$ , as a 32 bit float

Value: 1.442695040888963407359924681001892137f

The logarithm base 2 of e, as a 32 bit float.

## M\_PI : pi, as a 32 bit float

Value: 3.141592653589793238462643383279502884f

The constant pi, as a 32 bit float.

## M\_PI\_2 : pi / 2, as a 32 bit float

Value: 1.570796326794896619231321691639751442f

Pi divided by 2, as a 32 bit float.

## M\_PI\_4 : pi / 4, as a 32 bit float

Value: 0.785398163397448309615660845819875721f

Pi divided by 4, as a 32 bit float.

## M\_SQRT1\_2 : 1 / sqrt(2), as a 32 bit float

Value: 0.707106781186547524400844362104849039f

The inverse of the square root of 2, as a 32 bit float.

## M\_SQRT2 : sqrt(2), as a 32 bit float

Value: 1.414213562373095048801688724209698079f

The square root of 2, as a 32 bit float.

# Functions

---

## abs : Absolute value of an integer

```
uchar abs(char v);
uchar2 abs(char2 v);
uchar3 abs(char3 v);
uchar4 abs(char4 v);
uint abs(int v);
uint2 abs(int2 v);
uint3 abs(int3 v);
uint4 abs(int4 v);
ushort abs(short v);
ushort2 abs(short2 v);
ushort3 abs(short3 v);
ushort4 abs(short4 v);
```

Returns the absolute value of an integer.

For floats, use `fabs()`.

## acos : Inverse cosine

```
float acos(float v);

float2 acos(float2 v);
float3 acos(float3 v);
float4 acos(float4 v);

half acos(half v); Added in API level 24
half2 acos(half2 v); Added in API level 24
half3 acos(half3 v); Added in API level 24
half4 acos(half4 v); Added in API level 24
```

Returns the inverse cosine, in radians.

See also [native\\_acos\(\)](#).

## acosh : Inverse hyperbolic cosine

```
float acosh(float v);

float2 acosh(float2 v);
float3 acosh(float3 v);
float4 acosh(float4 v);

half acosh(half v); Added in API level 24
half2 acosh(half2 v); Added in API level 24
half3 acosh(half3 v); Added in API level 24
half4 acosh(half4 v); Added in API level 24
```

Returns the inverse hyperbolic cosine, in radians.

See also [native\\_acosh\(\)](#).

## acospi : Inverse cosine divided by pi

```
float acospi(float v);

float2 acospi(float2 v);
float3 acospi(float3 v);
float4 acospi(float4 v);

half acospi(half v); Added in API level 24
half2 acospi(half2 v); Added in API level 24
half3 acospi(half3 v); Added in API level 24
half4 acospi(half4 v); Added in API level 24
```

Returns the inverse cosine in radians, divided by pi.

To get an inverse cosine measured in degrees, use `acospi(a) * 180.f.`

See also [native\\_acospi\(\)](#).

## asin : Inverse sine

```
float asin(float v);

float2 asin(float2 v);
float3 asin(float3 v);
float4 asin(float4 v);

half asin(half v); Added in API level 24
half2 asin(half2 v); Added in API level 24
half3 asin(half3 v); Added in API level 24
```

**half4** `asin(half4 v);`     Added in API level 24

Returns the inverse sine, in radians.

See also [native\\_asin\(\)](#).

#### asinh : Inverse hyperbolic sine

```
float asinh(float v);
float2 asinh(float2 v);
float3 asinh(float3 v);
float4 asinh(float4 v);

half asinh(half v); Added in API level 24
half2 asinh(half2 v); Added in API level 24
half3 asinh(half3 v); Added in API level 24
half4 asinh(half4 v); Added in API level 24
```

Returns the inverse hyperbolic sine, in radians.

See also [native\\_asinh\(\)](#).

#### asinpi : Inverse sine divided by pi

```
float asinpi(float v);
float2 asinpi(float2 v);
float3 asinpi(float3 v);
float4 asinpi(float4 v);

half asinpi(half v); Added in API level 24
half2 asinpi(half2 v); Added in API level 24
half3 asinpi(half3 v); Added in API level 24
half4 asinpi(half4 v); Added in API level 24
```

Returns the inverse sine in radians, divided by pi.

To get an inverse sine measured in degrees, use `asinpi(a) * 180.f.`

See also [native\\_asinpi\(\)](#).

#### atan : Inverse tangent

```
float atan(float v);
float2 atan(float2 v);
float3 atan(float3 v);
float4 atan(float4 v);

half atan(half v); Added in API level 24
half2 atan(half2 v); Added in API level 24
half3 atan(half3 v); Added in API level 24
half4 atan(half4 v); Added in API level 24
```

Returns the inverse tangent, in radians.

See also [native\\_atan\(\)](#).

#### atan2 : Inverse tangent of a ratio

```
float atan2(float numerator, float denominator);
```

```
float2 atan2(float2 numerator, float2 denominator);
float3 atan2(float3 numerator, float3 denominator);
float4 atan2(float4 numerator, float4 denominator);
half atan2(half numerator, half denominator); Added in API level 24
half2 atan2(half2 numerator, half2 denominator); Added in API level 24
half3 atan2(half3 numerator, half3 denominator); Added in API level 24
half4 atan2(half4 numerator, half4 denominator); Added in API level 24
```

#### Parameters

*numerator*      Numerator.  
*denominator*      Denominator. Can be 0.

Returns the inverse tangent of (*numerator* / *denominator*), in radians.

See also [native\\_atan2\(\)](#).

### atan2pi : Inverse tangent of a ratio, divided by pi

```
float atan2pi(float numerator, float denominator);
float2 atan2pi(float2 numerator, float2 denominator);
float3 atan2pi(float3 numerator, float3 denominator);
float4 atan2pi(float4 numerator, float4 denominator);
half atan2pi(half numerator, half denominator); Added in API level 24
half2 atan2pi(half2 numerator, half2 denominator); Added in API level 24
half3 atan2pi(half3 numerator, half3 denominator); Added in API level 24
half4 atan2pi(half4 numerator, half4 denominator); Added in API level 24
```

#### Parameters

*numerator*      Numerator.  
*denominator*      Denominator. Can be 0.

Returns the inverse tangent of (*numerator* / *denominator*), in radians, divided by pi.

To get an inverse tangent measured in degrees, use `atan2pi(n, d) * 180.f.`

See also [native\\_atan2pi\(\)](#).

### atanh : Inverse hyperbolic tangent

```
float atanh(float v);
float2 atanh(float2 v);
float3 atanh(float3 v);
float4 atanh(float4 v);
half atanh(half v); Added in API level 24
half2 atanh(half2 v); Added in API level 24
half3 atanh(half3 v); Added in API level 24
half4 atanh(half4 v); Added in API level 24
```

Returns the inverse hyperbolic tangent, in radians.

See also [native\\_atanh\(\)](#).

### atanpi : Inverse tangent divided by pi

```
float atanpi(float v);
```

```
float2 atanpi(float2 v);
float3 atanpi(float3 v);
float4 atanpi(float4 v);
half atanpi(half v); Added in API level 24
half2 atanpi(half2 v); Added in API level 24
half3 atanpi(half3 v); Added in API level 24
half4 atanpi(half4 v); Added in API level 24
```

Returns the inverse tangent in radians, divided by pi.

To get an inverse tangent measured in degrees, use `atanpi(a) * 180.f.`

See also [native\\_atanpi\(\)](#).

## cbrt : Cube root

```
float cbrt(float v);
float2 cbrt(float2 v);
float3 cbrt(float3 v);
float4 cbrt(float4 v);
half cbrt(half v); Added in API level 24
half2 cbrt(half2 v); Added in API level 24
half3 cbrt(half3 v); Added in API level 24
half4 cbrt(half4 v); Added in API level 24
```

Returns the cube root.

See also [native\\_cbrt\(\)](#).

## ceil : Smallest integer not less than a value

```
float ceil(float v);
float2 ceil(float2 v);
float3 ceil(float3 v);
float4 ceil(float4 v);
half ceil(half v); Added in API level 24
half2 ceil(half2 v); Added in API level 24
half3 ceil(half3 v); Added in API level 24
half4 ceil(half4 v); Added in API level 24
```

Returns the smallest integer not less than a value.

For example, `ceil(1.2f)` returns 2.f, and `ceil(-1.2f)` returns -1.f.

See also [floor\(\)](#).

## clamp : Restrain a value to a range

|                                                             |                       |
|-------------------------------------------------------------|-----------------------|
| char clamp(char value, char min_value, char max_value);     | Added in API level 19 |
| char2 clamp(char2 value, char min_value, char max_value);   | Added in API level 19 |
| char2 clamp(char2 value, char2 min_value, char2 max_value); | Added in API level 19 |
| char3 clamp(char3 value, char min_value, char max_value);   | Added in API level 19 |
| char3 clamp(char3 value, char3 min_value, char3 max_value); | Added in API level 19 |
| char4 clamp(char4 value, char min_value, char max_value);   | Added in API level 19 |
| char4 clamp(char4 value, char4 min_value, char4 max_value); | Added in API level 19 |

```
float clamp(float value, float min_value, float max_value);
float2 clamp(float2 value, float min_value, float max_value);
float2 clamp(float2 value, float2 min_value, float2 max_value);
float3 clamp(float3 value, float min_value, float max_value);
float3 clamp(float3 value, float3 min_value, float3 max_value);
float4 clamp(float4 value, float min_value, float max_value);
float4 clamp(float4 value, float4 min_value, float4 max_value);

half clamp(half value, half min_value, half max_value); Added in API level 24
half2 clamp(half2 value, half min_value, half max_value); Added in API level 24
half2 clamp(half2 value, half2 min_value, half2 max_value); Added in API level 24
half3 clamp(half3 value, half min_value, half max_value); Added in API level 24
half3 clamp(half3 value, half3 min_value, half3 max_value); Added in API level 24
half4 clamp(half4 value, half min_value, half max_value); Added in API level 24
half4 clamp(half4 value, half4 min_value, half4 max_value); Added in API level 24

int clamp(int value, int min_value, int max_value); Added in API level 19
int2 clamp(int2 value, int min_value, int max_value); Added in API level 19
int2 clamp(int2 value, int2 min_value, int2 max_value); Added in API level 19
int3 clamp(int3 value, int min_value, int max_value); Added in API level 19
int3 clamp(int3 value, int3 min_value, int3 max_value); Added in API level 19
int4 clamp(int4 value, int min_value, int max_value); Added in API level 19
int4 clamp(int4 value, int4 min_value, int4 max_value); Added in API level 19

long clamp(long value, long min_value, long max_value); Added in API level 19
long2 clamp(long2 value, long min_value, long max_value); Added in API level 19
long2 clamp(long2 value, long2 min_value, long2 max_value); Added in API level 19
long3 clamp(long3 value, long min_value, long max_value); Added in API level 19
long3 clamp(long3 value, long3 min_value, long3 max_value); Added in API level 19
long4 clamp(long4 value, long min_value, long max_value); Added in API level 19
long4 clamp(long4 value, long4 min_value, long4 max_value); Added in API level 19

short clamp(short value, short min_value, short max_value); Added in API level 19
short2 clamp(short2 value, short min_value, short max_value); Added in API level 19
short2 clamp(short2 value, short2 min_value, short2 max_value); Added in API level 19
short3 clamp(short3 value, short min_value, short max_value); Added in API level 19
short3 clamp(short3 value, short3 min_value, short3 max_value); Added in API level 19
short4 clamp(short4 value, short min_value, short max_value); Added in API level 19
short4 clamp(short4 value, short4 min_value, short4 max_value); Added in API level 19

uchar clamp(uchar value, uchar min_value, uchar max_value); Added in API level 19
uchar2 clamp(uchar2 value, uchar min_value, uchar max_value); Added in API level 19
uchar2 clamp(uchar2 value, uchar2 min_value, uchar2 max_value); Added in API level 19
uchar3 clamp(uchar3 value, uchar min_value, uchar max_value); Added in API level 19
uchar3 clamp(uchar3 value, uchar3 min_value, uchar3 max_value); Added in API level 19
uchar4 clamp(uchar4 value, uchar min_value, uchar max_value); Added in API level 19
uchar4 clamp(uchar4 value, uchar4 min_value, uchar4 max_value); Added in API level 19

uint clamp(uint value, uint min_value, uint max_value); Added in API level 19
uint2 clamp(uint2 value, uint min_value, uint max_value); Added in API level 19
uint2 clamp(uint2 value, uint2 min_value, uint2 max_value); Added in API level 19
uint3 clamp(uint3 value, uint min_value, uint max_value); Added in API level 19
uint3 clamp(uint3 value, uint3 min_value, uint3 max_value); Added in API level 19
```

|                                                                                  |                       |
|----------------------------------------------------------------------------------|-----------------------|
| <code>uint4 clamp(uint4 value, uint min_value, uint max_value);</code>           | Added in API level 19 |
| <code>ulong clamp(ulong value, ulong min_value, ulong max_value);</code>         | Added in API level 19 |
| <code>ulong2 clamp(ulong2 value, ulong min_value, ulong max_value);</code>       | Added in API level 19 |
| <code>ulong2 clamp(ulong2 value, ulong2 min_value, ulong2 max_value);</code>     | Added in API level 19 |
| <code>ulong3 clamp(ulong3 value, ulong min_value, ulong max_value);</code>       | Added in API level 19 |
| <code>ulong3 clamp(ulong3 value, ulong3 min_value, ulong3 max_value);</code>     | Added in API level 19 |
| <code>ulong4 clamp(ulong4 value, ulong min_value, ulong max_value);</code>       | Added in API level 19 |
| <code>ulong4 clamp(ulong4 value, ulong4 min_value, ulong4 max_value);</code>     | Added in API level 19 |
| <code>ushort clamp(ushort value, ushort min_value, ushort max_value);</code>     | Added in API level 19 |
| <code>ushort2 clamp(ushort2 value, ushort min_value, ushort max_value);</code>   | Added in API level 19 |
| <code>ushort2 clamp(ushort2 value, ushort2 min_value, ushort2 max_value);</code> | Added in API level 19 |
| <code>ushort3 clamp(ushort3 value, ushort min_value, ushort max_value);</code>   | Added in API level 19 |
| <code>ushort3 clamp(ushort3 value, ushort3 min_value, ushort3 max_value);</code> | Added in API level 19 |
| <code>ushort4 clamp(ushort4 value, ushort min_value, ushort max_value);</code>   | Added in API level 19 |
| <code>ushort4 clamp(ushort4 value, ushort4 min_value, ushort4 max_value);</code> | Added in API level 19 |

## Parameters

- `value` Value to be clamped.  
`min_value` Lower bound, a scalar or matching vector.  
`max_value` High bound, must match the type of low.

Clamps a value to a specified high and low bound. `clamp()` returns `min_value` if `value < min_value`, `max_value` if `value > max_value`, otherwise `value`.

There are two variants of `clamp`: one where the min and max are scalars applied to all entries of the value, the other where the min and max are also vectors.

If `min_value` is greater than `max_value`, the results are undefined.

`clz` : Number of leading 0 bits

```
char clz(char value);
char2 clz(char2 value);
char3 clz(char3 value);
char4 clz(char4 value);

int clz(int value);
int2 clz(int2 value);
int3 clz(int3 value);
int4 clz(int4 value);

short clz(short value);
short2 clz(short2 value);
short3 clz(short3 value);
short4 clz(short4 value);

uchar clz(uchar value);
uchar2 clz(uchar2 value);
uchar3 clz(uchar3 value);
uchar4 clz(uchar4 value);

uint clz(uint value);
uint2 clz(uint2 value);
```

```
uint3 clz(uint3 value);
uint4 clz(uint4 value);

ushort clz(ushort value);
ushort2 clz(ushort2 value);
ushort3 clz(ushort3 value);
ushort4 clz(ushort4 value);
```

Returns the number of leading 0-bits in a value.

For example, `clz((char)0x03)` returns 6.

## copysign : Copies the sign of a number to another

```
float copysign(float magnitude_value, float sign_value);
float2 copysign(float2 magnitude_value, float2 sign_value);
float3 copysign(float3 magnitude_value, float3 sign_value);
float4 copysign(float4 magnitude_value, float4 sign_value);

half copysign(half magnitude_value, half sign_value); Added in API level 24
half2 copysign(half2 magnitude_value, half2 sign_value); Added in API level 24
half3 copysign(half3 magnitude_value, half3 sign_value); Added in API level 24
half4 copysign(half4 magnitude_value, half4 sign_value); Added in API level 24
```

Copies the sign from sign\_value to magnitude\_value.

The value returned is either magnitude\_value or -magnitude\_value.

For example, `copysign(4.0f, -2.7f)` returns -4.0f and `copysign(-4.0f, 2.7f)` returns 4.0f.

## cos : Cosine

```
float cos(float v);
float2 cos(float2 v);
float3 cos(float3 v);
float4 cos(float4 v);

half cos(half v); Added in API level 24
half2 cos(half2 v); Added in API level 24
half3 cos(half3 v); Added in API level 24
half4 cos(half4 v); Added in API level 24
```

Returns the cosine of an angle measured in radians.

See also [native\\_cos\(\)](#).

## cosh : Hyperbolic cosine

```
float cosh(float v);
float2 cosh(float2 v);
float3 cosh(float3 v);
float4 cosh(float4 v);

half cosh(half v); Added in API level 24
half2 cosh(half2 v); Added in API level 24
half3 cosh(half3 v); Added in API level 24
half4 cosh(half4 v); Added in API level 24
```

Returns the hyperbolic cosine of v, where v is measured in radians.

See also [native\\_cosh\(\)](#).

## cospi : Cosine of a number multiplied by pi

```
float cospi(float v);
float2 cospi(float2 v);
float3 cospi(float3 v);
float4 cospi(float4 v);

half cospi(half v); Added in API level 24
half2 cospi(half2 v); Added in API level 24
half3 cospi(half3 v); Added in API level 24
half4 cospi(half4 v); Added in API level 24
```

Returns the cosine of  $(v * \pi)$ , where  $(v * \pi)$  is measured in radians.

To get the cosine of a value measured in degrees, call `cospi(v / 180.f)`.

See also [native\\_cospi\(\)](#).

## degrees : Converts radians into degrees

```
float degrees(float v);
float2 degrees(float2 v);
float3 degrees(float3 v);
float4 degrees(float4 v);

half degrees(half v); Added in API level 24
half2 degrees(half2 v); Added in API level 24
half3 degrees(half3 v); Added in API level 24
half4 degrees(half4 v); Added in API level 24
```

Converts from radians to degrees.

## erf : Mathematical error function

```
float erf(float v);
float2 erf(float2 v);
float3 erf(float3 v);
float4 erf(float4 v);

half erf(half v); Added in API level 24
half2 erf(half2 v); Added in API level 24
half3 erf(half3 v); Added in API level 24
half4 erf(half4 v); Added in API level 24
```

Returns the error function.

## erfc : Mathematical complementary error function

```
float erfc(float v);
float2 erfc(float2 v);
float3 erfc(float3 v);
float4 erfc(float4 v);

half erfc(half v); Added in API level 24
half2 erfc(half2 v); Added in API level 24
half3 erfc(half3 v); Added in API level 24
```

`half4 erfc(half4 v);`     Added in API level 24

Returns the complementary error function.

#### exp : e raised to a number

`float exp(float v);`  
`float2 exp(float2 v);`  
`float3 exp(float3 v);`  
`float4 exp(float4 v);`  
`half exp(half v);`     Added in API level 24  
`half2 exp(half2 v);`     Added in API level 24  
`half3 exp(half3 v);`     Added in API level 24  
`half4 exp(half4 v);`     Added in API level 24

Returns  $e$  raised to  $v$ , i.e.  $e^v$ .

See also [native\\_exp\(\)](#).

#### exp10 : 10 raised to a number

`float exp10(float v);`  
`float2 exp10(float2 v);`  
`float3 exp10(float3 v);`  
`float4 exp10(float4 v);`  
`half exp10(half v);`     Added in API level 24  
`half2 exp10(half2 v);`     Added in API level 24  
`half3 exp10(half3 v);`     Added in API level 24  
`half4 exp10(half4 v);`     Added in API level 24

Returns 10 raised to  $v$ , i.e.  $10.f^v$ .

See also [native\\_exp10\(\)](#).

#### exp2 : 2 raised to a number

`float exp2(float v);`  
`float2 exp2(float2 v);`  
`float3 exp2(float3 v);`  
`float4 exp2(float4 v);`  
`half exp2(half v);`     Added in API level 24  
`half2 exp2(half2 v);`     Added in API level 24  
`half3 exp2(half3 v);`     Added in API level 24  
`half4 exp2(half4 v);`     Added in API level 24

Returns 2 raised to  $v$ , i.e.  $2.f^v$ .

See also [native\\_exp2\(\)](#).

#### expm1 : e raised to a number minus one

`float expm1(float v);`  
`float2 expm1(float2 v);`  
`float3 expm1(float3 v);`  
`float4 expm1(float4 v);`

`half expm1(half v);`      Added in API level 24

`half2 expm1(half2 v);`      Added in API level 24

`half3 expm1(half3 v);`      Added in API level 24

`half4 expm1(half4 v);`      Added in API level 24

Returns  $e$  raised to  $v$  minus 1, i.e.  $(e^v) - 1$ .

See also [native\\_expm1\(\)](#).

## fabs : Absolute value of a float

`float fabs(float v);`

`float2 fabs(float2 v);`

`float3 fabs(float3 v);`

`float4 fabs(float4 v);`

`half fabs(half v);`      Added in API level 24

`half2 fabs(half2 v);`      Added in API level 24

`half3 fabs(half3 v);`      Added in API level 24

`half4 fabs(half4 v);`      Added in API level 24

Returns the absolute value of the float  $v$ .

For integers, use [abs\(\)](#).

## fdim : Positive difference between two values

`float fdim(float a, float b);`

`float2 fdim(float2 a, float2 b);`

`float3 fdim(float3 a, float3 b);`

`float4 fdim(float4 a, float4 b);`

`half fdim(half a, half b);`      Added in API level 24

`half2 fdim(half2 a, half2 b);`      Added in API level 24

`half3 fdim(half3 a, half3 b);`      Added in API level 24

`half4 fdim(half4 a, half4 b);`      Added in API level 24

Returns the positive difference between two values.

If  $a > b$ , returns  $(a - b)$  otherwise returns 0f.

## floor : Smallest integer not greater than a value

`float floor(float v);`

`float2 floor(float2 v);`

`float3 floor(float3 v);`

`float4 floor(float4 v);`

`half floor(half v);`      Added in API level 24

`half2 floor(half2 v);`      Added in API level 24

`half3 floor(half3 v);`      Added in API level 24

`half4 floor(half4 v);`      Added in API level 24

Returns the smallest integer not greater than a value.

For example, `floor(1.2f)` returns 1.f, and `floor(-1.2f)` returns -2.f.

See also [ceil\(\)](#).

## fma : Multiply and add

```
float fma(float multiplicand1, float multiplicand2, float offset);
float2 fma(float2 multiplicand1, float2 multiplicand2, float2 offset);
float3 fma(float3 multiplicand1, float3 multiplicand2, float3 offset);
float4 fma(float4 multiplicand1, float4 multiplicand2, float4 offset);
half fma(half multiplicand1, half multiplicand2, half offset); Added in API level 24
half2 fma(half2 multiplicand1, half2 multiplicand2, half2 offset); Added in API level 24
half3 fma(half3 multiplicand1, half3 multiplicand2, half3 offset); Added in API level 24
half4 fma(half4 multiplicand1, half4 multiplicand2, half4 offset); Added in API level 24
```

Multiply and add. Returns `(multiplicand1 * multiplicand2) + offset`.

This function is similar to `mad()`. `fma()` retains full precision of the multiplied result and rounds only after the addition. `mad()` rounds after the multiplication and the addition. This extra precision is not guaranteed in `rs_fp_relaxed` mode.

## fmax : Maximum of two floats

```
float fmax(float a, float b);
float2 fmax(float2 a, float b);
float2 fmax(float2 a, float2 b);
float3 fmax(float3 a, float b);
float3 fmax(float3 a, float3 b);
float4 fmax(float4 a, float b);
float4 fmax(float4 a, float4 b);
half fmax(half a, half b); Added in API level 24
half2 fmax(half2 a, half b); Added in API level 24
half2 fmax(half2 a, half2 b); Added in API level 24
half3 fmax(half3 a, half b); Added in API level 24
half3 fmax(half3 a, half3 b); Added in API level 24
half4 fmax(half4 a, half b); Added in API level 24
half4 fmax(half4 a, half4 b); Added in API level 24
```

Returns the maximum of a and b, i.e. `(a < b ? b : a)`.

The `max()` function returns identical results but can be applied to more data types.

## fmin : Minimum of two floats

```
float fmin(float a, float b);
float2 fmin(float2 a, float b);
float2 fmin(float2 a, float2 b);
float3 fmin(float3 a, float b);
float3 fmin(float3 a, float3 b);
float4 fmin(float4 a, float b);
float4 fmin(float4 a, float4 b);
half fmin(half a, half b); Added in API level 24
half2 fmin(half2 a, half b); Added in API level 24
half2 fmin(half2 a, half2 b); Added in API level 24
half3 fmin(half3 a, half b); Added in API level 24
half3 fmin(half3 a, half3 b); Added in API level 24
half4 fmin(half4 a, half b); Added in API level 24
```

```
half4 fmin(half4 a, half4 b); Added in API level 24
```

Returns the minimum of a and b, i.e. `(a > b ? b : a)`.

The `min()` function returns identical results but can be applied to more data types.

## fmod : Modulo

```
float fmod(float numerator, float denominator);
float2 fmod(float2 numerator, float2 denominator);
float3 fmod(float3 numerator, float3 denominator);
float4 fmod(float4 numerator, float4 denominator);

half fmod(half numerator, half denominator); Added in API level 24
half2 fmod(half2 numerator, half2 denominator); Added in API level 24
half3 fmod(half3 numerator, half3 denominator); Added in API level 24
half4 fmod(half4 numerator, half4 denominator); Added in API level 24
```

Returns the remainder of (numerator / denominator), where the quotient is rounded towards zero.

The function `remainder()` is similar but rounds toward the closest integer. For example, `fmod(-3.8f, 2.f)` returns -1.8f (-3.8f - -1.f \* 2.f) while `remainder(-3.8f, 2.f)` returns 0.2f (-3.8f - -2.f \* 2.f).

## fract : Positive fractional part

```
float fract(float v);
float fract(float v, float* floor);
float2 fract(float2 v);
float2 fract(float2 v, float2* floor);
float3 fract(float3 v);
float3 fract(float3 v, float3* floor);
float4 fract(float4 v);
float4 fract(float4 v, float4* floor);

half fract(half v); Added in API level 24
half fract(half v, half* floor); Added in API level 24
half2 fract(half2 v); Added in API level 24
half2 fract(half2 v, half2* floor); Added in API level 24
half3 fract(half3 v); Added in API level 24
half3 fract(half3 v, half3* floor); Added in API level 24
half4 fract(half4 v); Added in API level 24
half4 fract(half4 v, half4* floor); Added in API level 24
```

### Parameters

*v* Input value.

*floor* If *floor* is not null, *\*floor* will be set to the floor of *v*.

Returns the positive fractional part of *v*, i.e. `v - floor(v)`.

For example, `fract(1.3f, &val)` returns 0.3f and sets *val* to 1.f. `fract(-1.3f, &val)` returns 0.7f and sets *val* to -2.f.

## frexp : Binary mantissa and exponent

```
float frexp(float v, int* exponent);
float2 frexp(float2 v, int2* exponent);
float3 frexp(float3 v, int3* exponent);
```

```
float4 frexp(float4 v, int4* exponent);

half frexp(half v, int* exponent); Added in API level 24
half2 frexp(half2 v, int2* exponent); Added in API level 24
half3 frexp(half3 v, int3* exponent); Added in API level 24
half4 frexp(half4 v, int4* exponent); Added in API level 24
```

#### Parameters

*v* Input value.  
*exponent* If exponent is not null, \*exponent will be set to the exponent of *v*.  
Returns the binary mantissa and exponent of *v*, i.e. *v* == *mantissa* \* 2 ^ *exponent*.  
The mantissa is always between 0.5 (inclusive) and 1.0 (exclusive).  
See [ldexp\(\)](#) for the reverse operation. See also [logb\(\)](#) and [ilogb\(\)](#).

### half\_recip : Reciprocal computed to 16 bit precision

```
float half_recip(float v); Added in API level 17
float2 half_recip(float2 v); Added in API level 17
float3 half_recip(float3 v); Added in API level 17
float4 half_recip(float4 v); Added in API level 17
```

Returns the approximate reciprocal of a value.  
The precision is that of a 16 bit floating point value.  
See also [native\\_recip\(\)](#).

### half\_rsqrt : Reciprocal of a square root computed to 16 bit precision

```
float half_rsqrt(float v); Added in API level 17
float2 half_rsqrt(float2 v); Added in API level 17
float3 half_rsqrt(float3 v); Added in API level 17
float4 half_rsqrt(float4 v); Added in API level 17
```

Returns the approximate value of `(1.f / sqrt(value))`.  
The precision is that of a 16 bit floating point value.  
See also [rsqrt\(\)](#), [native\\_rsqrt\(\)](#).

### half\_sqrt : Square root computed to 16 bit precision

```
float half_sqrt(float v); Added in API level 17
float2 half_sqrt(float2 v); Added in API level 17
float3 half_sqrt(float3 v); Added in API level 17
float4 half_sqrt(float4 v); Added in API level 17
```

Returns the approximate square root of a value.  
The precision is that of a 16 bit floating point value.  
See also [sqrt\(\)](#), [native\\_sqrt\(\)](#).

### hypot : Hypotenuse

```
float hypot(float a, float b);
float2 hypot(float2 a, float2 b);
```

```
float3 hypot(float3 a, float3 b);
float4 hypot(float4 a, float4 b);
half hypot(half a, half b); Added in API level 24
half2 hypot(half2 a, half2 b); Added in API level 24
half3 hypot(half3 a, half3 b); Added in API level 24
half4 hypot(half4 a, half4 b); Added in API level 24
```

Returns the hypotenuse, i.e. `sqrt(a * a + b * b)`.

See also [native\\_hypot\(\)](#).

## ilogb : Base two exponent

```
int ilogb(float v);
int ilogb(half v); Added in API level 24
int2 ilogb(float2 v);
int2 ilogb(half2 v); Added in API level 24
int3 ilogb(float3 v);
int3 ilogb(half3 v); Added in API level 24
int4 ilogb(float4 v);
int4 ilogb(half4 v); Added in API level 24
```

Returns the base two exponent of a value, where the mantissa is between 1.f (inclusive) and 2.f (exclusive).

For example, `ilogb(8.5f)` returns 3.

Because of the difference in mantissa, this number is one less than is returned by [frexp\(\)](#).

[logb\(\)](#) is similar but returns a float.

## ldexp : Creates a floating point from mantissa and exponent

```
float ldexp(float mantissa, int exponent);
float2 ldexp(float2 mantissa, int exponent);
float2 ldexp(float2 mantissa, int2 exponent);
float3 ldexp(float3 mantissa, int exponent);
float3 ldexp(float3 mantissa, int3 exponent);
float4 ldexp(float4 mantissa, int exponent);
float4 ldexp(float4 mantissa, int4 exponent);
half ldexp(half mantissa, int exponent); Added in API level 24
half2 ldexp(half2 mantissa, int exponent); Added in API level 24
half2 ldexp(half2 mantissa, int2 exponent); Added in API level 24
half3 ldexp(half3 mantissa, int exponent); Added in API level 24
half3 ldexp(half3 mantissa, int3 exponent); Added in API level 24
half4 ldexp(half4 mantissa, int exponent); Added in API level 24
half4 ldexp(half4 mantissa, int4 exponent); Added in API level 24
```

### Parameters

*mantissa* Mantissa.

*exponent* Exponent, a single component or matching vector.

Returns the floating point created from the mantissa and exponent, i.e. (*mantissa* \* 2 ^ *exponent*).

See [frexp\(\)](#) for the reverse operation.

## lgamma : Natural logarithm of the gamma function

```
float lgamma(float v);
float lgamma(float v, int* sign_of_gamma);
float2 lgamma(float2 v);
float2 lgamma(float2 v, int2* sign_of_gamma);
float3 lgamma(float3 v);
float3 lgamma(float3 v, int3* sign_of_gamma);
float4 lgamma(float4 v);
float4 lgamma(float4 v, int4* sign_of_gamma);
half lgamma(half v); Added in API level 24
half lgamma(half v, int* sign_of_gamma); Added in API level 24
half2 lgamma(half2 v); Added in API level 24
half2 lgamma(half2 v, int2* sign_of_gamma); Added in API level 24
half3 lgamma(half3 v); Added in API level 24
half3 lgamma(half3 v, int3* sign_of_gamma); Added in API level 24
half4 lgamma(half4 v); Added in API level 24
half4 lgamma(half4 v, int4* sign_of_gamma); Added in API level 24
```

### Parameters

v

sign\_of\_gamma If sign\_of\_gamma is not null, \*sign\_of\_gamma will be set to -1.f if the gamma of v is negative, otherwise to 1.f.

Returns the natural logarithm of the absolute value of the gamma function, i.e. `log(fabs(tgamma(v)))`.

See also [tgamma\(\)](#).

## log : Natural logarithm

```
float log(float v);
float2 log(float2 v);
float3 log(float3 v);
float4 log(float4 v);
half log(half v); Added in API level 24
half2 log(half2 v); Added in API level 24
half3 log(half3 v); Added in API level 24
half4 log(half4 v); Added in API level 24
```

Returns the natural logarithm.

See also [native\\_log\(\)](#).

## log10 : Base 10 logarithm

```
float log10(float v);
float2 log10(float2 v);
float3 log10(float3 v);
float4 log10(float4 v);
half log10(half v); Added in API level 24
half2 log10(half2 v); Added in API level 24
half3 log10(half3 v); Added in API level 24
half4 log10(half4 v); Added in API level 24
```

Returns the base 10 logarithm.

See also [native\\_log10\(\)](#).

### log1p : Natural logarithm of a value plus 1

```
float log1p(float v);
float2 log1p(float2 v);
float3 log1p(float3 v);
float4 log1p(float4 v);

half log1p(half v); Added in API level 24
half2 log1p(half2 v); Added in API level 24
half3 log1p(half3 v); Added in API level 24
half4 log1p(half4 v); Added in API level 24
```

Returns the natural logarithm of ( $v + 1.f$ ).

See also [native\\_log1p\(\)](#).

### log2 : Base 2 logarithm

```
float log2(float v);
float2 log2(float2 v);
float3 log2(float3 v);
float4 log2(float4 v);

half log2(half v); Added in API level 24
half2 log2(half2 v); Added in API level 24
half3 log2(half3 v); Added in API level 24
half4 log2(half4 v); Added in API level 24
```

Returns the base 2 logarithm.

See also [native\\_log2\(\)](#).

### logb : Base two exponent

```
float logb(float v);
float2 logb(float2 v);
float3 logb(float3 v);
float4 logb(float4 v);

half logb(half v); Added in API level 24
half2 logb(half2 v); Added in API level 24
half3 logb(half3 v); Added in API level 24
half4 logb(half4 v); Added in API level 24
```

Returns the base two exponent of a value, where the mantissa is between 1.f (inclusive) and 2.f (exclusive).

For example, `logb(8.5f)` returns 3.f.

Because of the difference in mantissa, this number is one less than is returned by `frexp()`.

`ilogb()` is similar but returns an integer.

### mad : Multiply and add

```
float mad(float multiplicand1, float multiplicand2, float offset);
```

```
float2 mad(float2 multiplicand1, float2 multiplicand2, float2 offset);
float3 mad(float3 multiplicand1, float3 multiplicand2, float3 offset);
float4 mad(float4 multiplicand1, float4 multiplicand2, float4 offset);
half mad(half multiplicand1, half multiplicand2, half offset); Added in API level 24
half2 mad(half2 multiplicand1, half2 multiplicand2, half2 offset); Added in API level 24
half3 mad(half3 multiplicand1, half3 multiplicand2, half3 offset); Added in API level 24
half4 mad(half4 multiplicand1, half4 multiplicand2, half4 offset); Added in API level 24
```

Multiply and add. Returns `(multiplicand1 * multiplicand2) + offset`.

This function is similar to `fma()`. `fma()` retains full precision of the multiplied result and rounds only after the addition. `mad()` rounds after the multiplication and the addition. In `rs_fp_relaxed` mode, `mad()` may not do the rounding after multiplication.

## max : Maximum

```
char max(char a, char b);
char2 max(char2 a, char2 b);
char3 max(char3 a, char3 b);
char4 max(char4 a, char4 b);
float max(float a, float b);
float2 max(float2 a, float b);
float2 max(float2 a, float2 b);
float3 max(float3 a, float b);
float3 max(float3 a, float3 b);
float4 max(float4 a, float b);
float4 max(float4 a, float4 b);
half max(half a, half b); Added in API level 24
half2 max(half2 a, half b); Added in API level 24
half2 max(half2 a, half2 b); Added in API level 24
half3 max(half3 a, half b); Added in API level 24
half3 max(half3 a, half3 b); Added in API level 24
half4 max(half4 a, half b); Added in API level 24
half4 max(half4 a, half4 b); Added in API level 24
int max(int a, int b);
int2 max(int2 a, int2 b);
int3 max(int3 a, int3 b);
int4 max(int4 a, int4 b);
long max(long a, long b); Added in API level 21
long2 max(long2 a, long2 b); Added in API level 21
long3 max(long3 a, long3 b); Added in API level 21
long4 max(long4 a, long4 b); Added in API level 21
short max(short a, short b);
short2 max(short2 a, short2 b);
short3 max(short3 a, short3 b);
short4 max(short4 a, short4 b);
uchar max(uchar a, uchar b);
uchar2 max(uchar2 a, uchar2 b);
uchar3 max(uchar3 a, uchar3 b);
uchar4 max(uchar4 a, uchar4 b);
```

```
uint max(uint a, uint b);
uint2 max(uint2 a, uint2 b);
uint3 max(uint3 a, uint3 b);
uint4 max(uint4 a, uint4 b);
ulong max(ulong a, ulong b); Added in API level 21
ulong2 max(ulong2 a, ulong2 b); Added in API level 21
ulong3 max(ulong3 a, ulong3 b); Added in API level 21
ulong4 max(ulong4 a, ulong4 b); Added in API level 21
ushort max(ushort a, ushort b);
ushort2 max(ushort2 a, ushort2 b);
ushort3 max(ushort3 a, ushort3 b);
ushort4 max(ushort4 a, ushort4 b);
```

Returns the maximum value of two arguments.

#### min : Minimum

```
char min(char a, char b);
char2 min(char2 a, char2 b);
char3 min(char3 a, char3 b);
char4 min(char4 a, char4 b);
float min(float a, float b);
float2 min(float2 a, float b);
float2 min(float2 a, float2 b);
float3 min(float3 a, float b);
float3 min(float3 a, float3 b);
float4 min(float4 a, float b);
float4 min(float4 a, float4 b);
half min(half a, half b); Added in API level 24
half2 min(half2 a, half b); Added in API level 24
half2 min(half2 a, half2 b); Added in API level 24
half3 min(half3 a, half b); Added in API level 24
half3 min(half3 a, half3 b); Added in API level 24
half4 min(half4 a, half b); Added in API level 24
half4 min(half4 a, half4 b); Added in API level 24
int min(int a, int b);
int2 min(int2 a, int2 b);
int3 min(int3 a, int3 b);
int4 min(int4 a, int4 b);
long min(long a, long b); Added in API level 21
long2 min(long2 a, long2 b); Added in API level 21
long3 min(long3 a, long3 b); Added in API level 21
long4 min(long4 a, long4 b); Added in API level 21
short min(short a, short b);
short2 min(short2 a, short2 b);
short3 min(short3 a, short3 b);
short4 min(short4 a, short4 b);
uchar min(uchar a, uchar b);
```

```
uchar2 min(uchar2 a, uchar2 b);
uchar3 min(uchar3 a, uchar3 b);
uchar4 min(uchar4 a, uchar4 b);
uint min(uint a, uint b);
uint2 min(uint2 a, uint2 b);
uint3 min(uint3 a, uint3 b);
uint4 min(uint4 a, uint4 b);
ulong min(ulong a, ulong b); Added in API level 21
ulong2 min(ulong2 a, ulong2 b); Added in API level 21
ulong3 min(ulong3 a, ulong3 b); Added in API level 21
ulong4 min(ulong4 a, ulong4 b); Added in API level 21
ushort min(ushort a, ushort b);
ushort2 min(ushort2 a, ushort2 b);
ushort3 min(ushort3 a, ushort3 b);
ushort4 min(ushort4 a, ushort4 b);
```

Returns the minimum value of two arguments.

#### mix : Mixes two values

```
float mix(float start, float stop, float fraction);
float2 mix(float2 start, float2 stop, float fraction);
float2 mix(float2 start, float2 stop, float2 fraction);
float3 mix(float3 start, float3 stop, float fraction);
float3 mix(float3 start, float3 stop, float3 fraction);
float4 mix(float4 start, float4 stop, float fraction);
float4 mix(float4 start, float4 stop, float4 fraction);
half mix(half start, half stop, half fraction); Added in API level 24
half2 mix(half2 start, half2 stop, half fraction); Added in API level 24
half2 mix(half2 start, half2 stop, half2 fraction); Added in API level 24
half3 mix(half3 start, half3 stop, half fraction); Added in API level 24
half3 mix(half3 start, half3 stop, half3 fraction); Added in API level 24
half4 mix(half4 start, half4 stop, half fraction); Added in API level 24
half4 mix(half4 start, half4 stop, half4 fraction); Added in API level 24
```

Returns start + ((stop - start) \* fraction).

This can be useful for mixing two values. For example, to create a new color that is 40% color1 and 60% color2, use `mix(color1, color2, 0.6f)`.

#### modf : Integral and fractional components

```
float modf(float v, float* integral_part);
float2 modf(float2 v, float2* integral_part);
float3 modf(float3 v, float3* integral_part);
float4 modf(float4 v, float4* integral_part);
half modf(half v, half* integral_part); Added in API level 24
half2 modf(half2 v, half2* integral_part); Added in API level 24
half3 modf(half3 v, half3* integral_part); Added in API level 24
half4 modf(half4 v, half4* integral_part); Added in API level 24
```

## Parameters

*v*              Source value.  
*integral\_part* \**integral\_part* will be set to the integral portion of the number.

## Returns

Floating point portion of the value.

Returns the integral and fractional components of a number.

Both components will have the same sign as x. For example, for an input of -3.72f, \**integral\_part* will be set to -3.f and .72f will be returned.

## nan : Not a Number

```
float nan(uint v);
```

## Parameters

*v*    Not used.

Returns a NaN value (Not a Number).

## nan\_half : Not a Number

```
half nan_half(); Added in API level 24
```

Returns a half-precision floating point NaN value (Not a Number).

## native\_acos : Approximate inverse cosine

|                               |                       |
|-------------------------------|-----------------------|
| float native_acos(float v);   | Added in API level 21 |
| float2 native_acos(float2 v); | Added in API level 21 |
| float3 native_acos(float3 v); | Added in API level 21 |
| float4 native_acos(float4 v); | Added in API level 21 |
| half native_acos(half v);     | Added in API level 24 |
| half2 native_acos(half2 v);   | Added in API level 24 |
| half3 native_acos(half3 v);   | Added in API level 24 |
| half4 native_acos(half4 v);   | Added in API level 24 |

Returns the approximate inverse cosine, in radians.

This function yields undefined results from input values less than -1 or greater than 1.

See also [acos\(\)](#).

## native\_acosh : Approximate inverse hyperbolic cosine

|                                |                       |
|--------------------------------|-----------------------|
| float native_acosh(float v);   | Added in API level 21 |
| float2 native_acosh(float2 v); | Added in API level 21 |
| float3 native_acosh(float3 v); | Added in API level 21 |
| float4 native_acosh(float4 v); | Added in API level 21 |
| half native_acosh(half v);     | Added in API level 24 |
| half2 native_acosh(half2 v);   | Added in API level 24 |
| half3 native_acosh(half3 v);   | Added in API level 24 |
| half4 native_acosh(half4 v);   | Added in API level 24 |

Returns the approximate inverse hyperbolic cosine, in radians.

See also [acosh\(\)](#).

## native\_acospi : Approximate inverse cosine divided by pi

|                                 |                       |
|---------------------------------|-----------------------|
| float native_acospi(float v);   | Added in API level 21 |
| float2 native_acospi(float2 v); | Added in API level 21 |
| float3 native_acospi(float3 v); | Added in API level 21 |
| float4 native_acospi(float4 v); | Added in API level 21 |
| half native_acospi(half v);     | Added in API level 24 |
| half2 native_acospi(half2 v);   | Added in API level 24 |
| half3 native_acospi(half3 v);   | Added in API level 24 |
| half4 native_acospi(half4 v);   | Added in API level 24 |

Returns the approximate inverse cosine in radians, divided by pi.

To get an inverse cosine measured in degrees, use `acospi(a) * 180.f.`

This function yields undefined results from input values less than -1 or greater than 1.

See also [acospi\(\)](#).

## native\_asin : Approximate inverse sine

|                               |                       |
|-------------------------------|-----------------------|
| float native_asin(float v);   | Added in API level 21 |
| float2 native_asin(float2 v); | Added in API level 21 |
| float3 native_asin(float3 v); | Added in API level 21 |
| float4 native_asin(float4 v); | Added in API level 21 |
| half native_asin(half v);     | Added in API level 24 |
| half2 native_asin(half2 v);   | Added in API level 24 |
| half3 native_asin(half3 v);   | Added in API level 24 |
| half4 native_asin(half4 v);   | Added in API level 24 |

Returns the approximate inverse sine, in radians.

This function yields undefined results from input values less than -1 or greater than 1.

See also [asin\(\)](#).

## native\_asinh : Approximate inverse hyperbolic sine

|                                |                       |
|--------------------------------|-----------------------|
| float native_asinh(float v);   | Added in API level 21 |
| float2 native_asinh(float2 v); | Added in API level 21 |
| float3 native_asinh(float3 v); | Added in API level 21 |
| float4 native_asinh(float4 v); | Added in API level 21 |
| half native_asinh(half v);     | Added in API level 24 |
| half2 native_asinh(half2 v);   | Added in API level 24 |
| half3 native_asinh(half3 v);   | Added in API level 24 |
| half4 native_asinh(half4 v);   | Added in API level 24 |

Returns the approximate inverse hyperbolic sine, in radians.

See also [asinh\(\)](#).

## native\_asinpi : Approximate inverse sine divided by pi

|                                 |                       |
|---------------------------------|-----------------------|
| float native_asinpi(float v);   | Added in API level 21 |
| float2 native_asinpi(float2 v); | Added in API level 21 |

`float3 native_asinpi(float3 v);`    Added in [API level 21](#)  
`float4 native_asinpi(float4 v);`    Added in [API level 21](#)  
`half native_asinpi(half v);`    Added in [API level 24](#)  
`half2 native_asinpi(half2 v);`    Added in [API level 24](#)  
`half3 native_asinpi(half3 v);`    Added in [API level 24](#)  
`half4 native_asinpi(half4 v);`    Added in [API level 24](#)

Returns the approximate inverse sine in radians, divided by pi.

To get an inverse sine measured in degrees, use `asinpi(a) * 180.f.`

This function yields undefined results from input values less than -1 or greater than 1.

See also [asinpi\(\)](#).

## native\_atan : Approximate inverse tangent

`float native_atan(float v);`    Added in [API level 21](#)  
`float2 native_atan(float2 v);`    Added in [API level 21](#)  
`float3 native_atan(float3 v);`    Added in [API level 21](#)  
`float4 native_atan(float4 v);`    Added in [API level 21](#)  
`half native_atan(half v);`    Added in [API level 24](#)  
`half2 native_atan(half2 v);`    Added in [API level 24](#)  
`half3 native_atan(half3 v);`    Added in [API level 24](#)  
`half4 native_atan(half4 v);`    Added in [API level 24](#)

Returns the approximate inverse tangent, in radians.

See also [atan\(\)](#).

## native\_atan2 : Approximate inverse tangent of a ratio

`float native_atan2(float numerator, float denominator);`    Added in [API level 21](#)  
`float2 native_atan2(float2 numerator, float2 denominator);`    Added in [API level 21](#)  
`float3 native_atan2(float3 numerator, float3 denominator);`    Added in [API level 21](#)  
`float4 native_atan2(float4 numerator, float4 denominator);`    Added in [API level 21](#)  
`half native_atan2(half numerator, half denominator);`    Added in [API level 24](#)  
`half2 native_atan2(half2 numerator, half2 denominator);`    Added in [API level 24](#)  
`half3 native_atan2(half3 numerator, half3 denominator);`    Added in [API level 24](#)  
`half4 native_atan2(half4 numerator, half4 denominator);`    Added in [API level 24](#)

### Parameters

*numerator*    Numerator.

*denominator*    Denominator. Can be 0.

Returns the approximate inverse tangent of `(numerator / denominator)`, in radians.

See also [atan2\(\)](#).

## native\_atan2pi : Approximate inverse tangent of a ratio, divided by pi

`float native_atan2pi(float numerator, float denominator);`    Added in [API level 21](#)  
`float2 native_atan2pi(float2 numerator, float2 denominator);`    Added in [API level 21](#)  
`float3 native_atan2pi(float3 numerator, float3 denominator);`    Added in [API level 21](#)  
`float4 native_atan2pi(float4 numerator, float4 denominator);`    Added in [API level 21](#)  
`half native_atan2pi(half numerator, half denominator);`    Added in [API level 24](#)

`half2 native_atan2pi(half2 numerator, half2 denominator);` Added in API level 24

`half3 native_atan2pi(half3 numerator, half3 denominator);` Added in API level 24

`half4 native_atan2pi(half4 numerator, half4 denominator);` Added in API level 24

#### Parameters

*numerator* Numerator.

*denominator* Denominator. Can be 0.

Returns the approximate inverse tangent of (*numerator* / *denominator*), in radians, divided by pi.

To get an inverse tangent measured in degrees, use `atan2pi(n, d) * 180.f.`

See also [atan2pi\(\)](#).

### native\_atanh : Approximate inverse hyperbolic tangent

`float native_atanh(float v);` Added in API level 21

`float2 native_atanh(float2 v);` Added in API level 21

`float3 native_atanh(float3 v);` Added in API level 21

`float4 native_atanh(float4 v);` Added in API level 21

`half native_atanh(half v);` Added in API level 24

`half2 native_atanh(half2 v);` Added in API level 24

`half3 native_atanh(half3 v);` Added in API level 24

`half4 native_atanh(half4 v);` Added in API level 24

Returns the approximate inverse hyperbolic tangent, in radians.

See also [atanh\(\)](#).

### native\_atanpi : Approximate inverse tangent divided by pi

`float native_atanpi(float v);` Added in API level 21

`float2 native_atanpi(float2 v);` Added in API level 21

`float3 native_atanpi(float3 v);` Added in API level 21

`float4 native_atanpi(float4 v);` Added in API level 21

`half native_atanpi(half v);` Added in API level 24

`half2 native_atanpi(half2 v);` Added in API level 24

`half3 native_atanpi(half3 v);` Added in API level 24

`half4 native_atanpi(half4 v);` Added in API level 24

Returns the approximate inverse tangent in radians, divided by pi.

To get an inverse tangent measured in degrees, use `atanpi(a) * 180.f.`

See also [atanpi\(\)](#).

### native\_cbrt : Approximate cube root

`float native_cbrt(float v);` Added in API level 21

`float2 native_cbrt(float2 v);` Added in API level 21

`float3 native_cbrt(float3 v);` Added in API level 21

`float4 native_cbrt(float4 v);` Added in API level 21

`half native_cbrt(half v);` Added in API level 24

`half2 native_cbrt(half2 v);` Added in API level 24

`half3 native_cbrt(half3 v);` Added in API level 24

`half4 native_cbrt(half4 v);`     Added in [API level 24](#)

Returns the approximate cubic root.

See also [cbrt\(\)](#).

#### native\_cos : Approximate cosine

`float native_cos(float v);`     Added in [API level 21](#)

`float2 native_cos(float2 v);`     Added in [API level 21](#)

`float3 native_cos(float3 v);`     Added in [API level 21](#)

`float4 native_cos(float4 v);`     Added in [API level 21](#)

`half native_cos(half v);`     Added in [API level 24](#)

`half2 native_cos(half2 v);`     Added in [API level 24](#)

`half3 native_cos(half3 v);`     Added in [API level 24](#)

`half4 native_cos(half4 v);`     Added in [API level 24](#)

Returns the approximate cosine of an angle measured in radians.

See also [cos\(\)](#).

#### native\_cosh : Approximate hyperbolic cosine

`float native_cosh(float v);`     Added in [API level 21](#)

`float2 native_cosh(float2 v);`     Added in [API level 21](#)

`float3 native_cosh(float3 v);`     Added in [API level 21](#)

`float4 native_cosh(float4 v);`     Added in [API level 21](#)

`half native_cosh(half v);`     Added in [API level 24](#)

`half2 native_cosh(half2 v);`     Added in [API level 24](#)

`half3 native_cosh(half3 v);`     Added in [API level 24](#)

`half4 native_cosh(half4 v);`     Added in [API level 24](#)

Returns the approximate hyperbolic cosine.

See also [cosh\(\)](#).

#### native\_cospি : Approximate cosine of a number multiplied by pi

`float native_cospি(float v);`     Added in [API level 21](#)

`float2 native_cospি(float2 v);`     Added in [API level 21](#)

`float3 native_cospি(float3 v);`     Added in [API level 21](#)

`float4 native_cospি(float4 v);`     Added in [API level 21](#)

`half native_cospি(half v);`     Added in [API level 24](#)

`half2 native_cospি(half2 v);`     Added in [API level 24](#)

`half3 native_cospি(half3 v);`     Added in [API level 24](#)

`half4 native_cospি(half4 v);`     Added in [API level 24](#)

Returns the approximate cosine of ( $v * \pi$ ), where ( $v * \pi$ ) is measured in radians.

To get the cosine of a value measured in degrees, call `cospি(v / 180.f)`.

See also [cospি\(\)](#).

#### native\_divide : Approximate division

`float native_divide(float left_vector, float right_vector);`     Added in [API level 21](#)

`float2 native_divide(float2 left_vector, float2 right_vector);`     Added in [API level 21](#)

|                                                                             |                                       |
|-----------------------------------------------------------------------------|---------------------------------------|
| <code>float3 native_divide(float3 left_vector, float3 right_vector);</code> | Added in <a href="#">API level 21</a> |
| <code>float4 native_divide(float4 left_vector, float4 right_vector);</code> | Added in <a href="#">API level 21</a> |
| <code>half native_divide(half left_vector, half right_vector);</code>       | Added in <a href="#">API level 24</a> |
| <code>half2 native_divide(half2 left_vector, half2 right_vector);</code>    | Added in <a href="#">API level 24</a> |
| <code>half3 native_divide(half3 left_vector, half3 right_vector);</code>    | Added in <a href="#">API level 24</a> |
| <code>half4 native_divide(half4 left_vector, half4 right_vector);</code>    | Added in <a href="#">API level 24</a> |

Computes the approximate division of two values.

#### native\_exp : Approximate e raised to a number

|                                           |                                       |
|-------------------------------------------|---------------------------------------|
| <code>float native_exp(float v);</code>   | Added in <a href="#">API level 18</a> |
| <code>float2 native_exp(float2 v);</code> | Added in <a href="#">API level 18</a> |
| <code>float3 native_exp(float3 v);</code> | Added in <a href="#">API level 18</a> |
| <code>float4 native_exp(float4 v);</code> | Added in <a href="#">API level 18</a> |
| <code>half native_exp(half v);</code>     | Added in <a href="#">API level 24</a> |
| <code>half2 native_exp(half2 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>half3 native_exp(half3 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>half4 native_exp(half4 v);</code>   | Added in <a href="#">API level 24</a> |

Fast approximate exp.

It is valid for inputs from -86.f to 86.f. The precision is no worse than what would be expected from using 16 bit floating point values.

See also [exp\(\)](#).

#### native\_exp10 : Approximate 10 raised to a number

|                                             |                                       |
|---------------------------------------------|---------------------------------------|
| <code>float native_exp10(float v);</code>   | Added in <a href="#">API level 18</a> |
| <code>float2 native_exp10(float2 v);</code> | Added in <a href="#">API level 18</a> |
| <code>float3 native_exp10(float3 v);</code> | Added in <a href="#">API level 18</a> |
| <code>float4 native_exp10(float4 v);</code> | Added in <a href="#">API level 18</a> |
| <code>half native_exp10(half v);</code>     | Added in <a href="#">API level 24</a> |
| <code>half2 native_exp10(half2 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>half3 native_exp10(half3 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>half4 native_exp10(half4 v);</code>   | Added in <a href="#">API level 24</a> |

Fast approximate exp10.

It is valid for inputs from -37.f to 37.f. The precision is no worse than what would be expected from using 16 bit floating point values.

See also [exp10\(\)](#).

#### native\_exp2 : Approximate 2 raised to a number

|                                            |                                       |
|--------------------------------------------|---------------------------------------|
| <code>float native_exp2(float v);</code>   | Added in <a href="#">API level 18</a> |
| <code>float2 native_exp2(float2 v);</code> | Added in <a href="#">API level 18</a> |
| <code>float3 native_exp2(float3 v);</code> | Added in <a href="#">API level 18</a> |
| <code>float4 native_exp2(float4 v);</code> | Added in <a href="#">API level 18</a> |
| <code>half native_exp2(half v);</code>     | Added in <a href="#">API level 24</a> |
| <code>half2 native_exp2(half2 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>half3 native_exp2(half3 v);</code>   | Added in <a href="#">API level 24</a> |
| <code>half4 native_exp2(half4 v);</code>   | Added in <a href="#">API level 24</a> |

Fast approximate exp2.

It is valid for inputs from -125.f to 125.f. The precision is no worse than what would be expected from using 16 bit floating point values.

See also [exp2\(\)](#).

#### native\_expm1 : Approximate e raised to a number minus one

|                                |                                       |
|--------------------------------|---------------------------------------|
| float native_expm1(float v);   | Added in <a href="#">API level 21</a> |
| float2 native_expm1(float2 v); | Added in <a href="#">API level 21</a> |
| float3 native_expm1(float3 v); | Added in <a href="#">API level 21</a> |
| float4 native_expm1(float4 v); | Added in <a href="#">API level 21</a> |
| half native_expm1(half v);     | Added in <a href="#">API level 24</a> |
| half2 native_expm1(half2 v);   | Added in <a href="#">API level 24</a> |
| half3 native_expm1(half3 v);   | Added in <a href="#">API level 24</a> |
| half4 native_expm1(half4 v);   | Added in <a href="#">API level 24</a> |

Returns the approximate  $(e^v) - 1$ .

See also [expm1\(\)](#).

#### native\_hypot : Approximate hypotenuse

|                                          |                                       |
|------------------------------------------|---------------------------------------|
| float native_hypot(float a, float b);    | Added in <a href="#">API level 21</a> |
| float2 native_hypot(float2 a, float2 b); | Added in <a href="#">API level 21</a> |
| float3 native_hypot(float3 a, float3 b); | Added in <a href="#">API level 21</a> |
| float4 native_hypot(float4 a, float4 b); | Added in <a href="#">API level 21</a> |
| half native_hypot(half a, half b);       | Added in <a href="#">API level 24</a> |
| half2 native_hypot(half2 a, half2 b);    | Added in <a href="#">API level 24</a> |
| half3 native_hypot(half3 a, half3 b);    | Added in <a href="#">API level 24</a> |
| half4 native_hypot(half4 a, half4 b);    | Added in <a href="#">API level 24</a> |

Returns the approximate `native_sqrt(a * a + b * b)`

See also [hypot\(\)](#).

#### native\_log : Approximate natural logarithm

|                              |                                       |
|------------------------------|---------------------------------------|
| float native_log(float v);   | Added in <a href="#">API level 18</a> |
| float2 native_log(float2 v); | Added in <a href="#">API level 18</a> |
| float3 native_log(float3 v); | Added in <a href="#">API level 18</a> |
| float4 native_log(float4 v); | Added in <a href="#">API level 18</a> |
| half native_log(half v);     | Added in <a href="#">API level 24</a> |
| half2 native_log(half2 v);   | Added in <a href="#">API level 24</a> |
| half3 native_log(half3 v);   | Added in <a href="#">API level 24</a> |
| half4 native_log(half4 v);   | Added in <a href="#">API level 24</a> |

Fast approximate log.

It is not accurate for values very close to zero.

See also [log\(\)](#).

#### native\_log10 : Approximate base 10 logarithm

|                                |                                       |
|--------------------------------|---------------------------------------|
| float native_log10(float v);   | Added in <a href="#">API level 18</a> |
| float2 native_log10(float2 v); | Added in <a href="#">API level 18</a> |

`float3 native_log10(float3 v);` Added in [API level 18](#)

`float4 native_log10(float4 v);` Added in [API level 18](#)

`half native_log10(half v);` Added in [API level 24](#)

`half2 native_log10(half2 v);` Added in [API level 24](#)

`half3 native_log10(half3 v);` Added in [API level 24](#)

`half4 native_log10(half4 v);` Added in [API level 24](#)

Fast approximate log10.

It is not accurate for values very close to zero.

See also [log10\(\)](#).

## native\_log1p : Approximate natural logarithm of a value plus 1

`float native_log1p(float v);` Added in [API level 21](#)

`float2 native_log1p(float2 v);` Added in [API level 21](#)

`float3 native_log1p(float3 v);` Added in [API level 21](#)

`float4 native_log1p(float4 v);` Added in [API level 21](#)

`half native_log1p(half v);` Added in [API level 24](#)

`half2 native_log1p(half2 v);` Added in [API level 24](#)

`half3 native_log1p(half3 v);` Added in [API level 24](#)

`half4 native_log1p(half4 v);` Added in [API level 24](#)

Returns the approximate natural logarithm of ( $v + 1.0f$ )

See also [log1p\(\)](#).

## native\_log2 : Approximate base 2 logarithm

`float native_log2(float v);` Added in [API level 18](#)

`float2 native_log2(float2 v);` Added in [API level 18](#)

`float3 native_log2(float3 v);` Added in [API level 18](#)

`float4 native_log2(float4 v);` Added in [API level 18](#)

`half native_log2(half v);` Added in [API level 24](#)

`half2 native_log2(half2 v);` Added in [API level 24](#)

`half3 native_log2(half3 v);` Added in [API level 24](#)

`half4 native_log2(half4 v);` Added in [API level 24](#)

Fast approximate log2.

It is not accurate for values very close to zero.

See also [log2\(\)](#).

## native\_powr : Approximate positive base raised to an exponent

`float native_powr(float base, float exponent);` Added in [API level 18](#)

`float2 native_powr(float2 base, float2 exponent);` Added in [API level 18](#)

`float3 native_powr(float3 base, float3 exponent);` Added in [API level 18](#)

`float4 native_powr(float4 base, float4 exponent);` Added in [API level 18](#)

`half native_powr(half base, half exponent);` Added in [API level 24](#)

`half2 native_powr(half2 base, half2 exponent);` Added in [API level 24](#)

`half3 native_powr(half3 base, half3 exponent);` Added in [API level 24](#)

`half4 native_powr(half4 base, half4 exponent);`     Added in [API level 24](#)

**Parameters**

`base`     Must be between 0.f and 256.f. The function is not accurate for values very close to zero.

`exponent`     Must be between -15.f and 15.f.

Fast approximate (`base ^ exponent`).

See also [powr\(\)](#).

### native\_recip : Approximate reciprocal

`float native_recip(float v);`     Added in [API level 21](#)

`float2 native_recip(float2 v);`     Added in [API level 21](#)

`float3 native_recip(float3 v);`     Added in [API level 21](#)

`float4 native_recip(float4 v);`     Added in [API level 21](#)

`half native_recip(half v);`     Added in [API level 24](#)

`half2 native_recip(half2 v);`     Added in [API level 24](#)

`half3 native_recip(half3 v);`     Added in [API level 24](#)

`half4 native_recip(half4 v);`     Added in [API level 24](#)

Returns the approximate approximate reciprocal of a value.

See also [half\\_recip\(\)](#).

### native\_rootn : Approximate nth root

`float native_rootn(float v, int n);`     Added in [API level 21](#)

`float2 native_rootn(float2 v, int2 n);`     Added in [API level 21](#)

`float3 native_rootn(float3 v, int3 n);`     Added in [API level 21](#)

`float4 native_rootn(float4 v, int4 n);`     Added in [API level 21](#)

`half native_rootn(half v, int n);`     Added in [API level 24](#)

`half2 native_rootn(half2 v, int2 n);`     Added in [API level 24](#)

`half3 native_rootn(half3 v, int3 n);`     Added in [API level 24](#)

`half4 native_rootn(half4 v, int4 n);`     Added in [API level 24](#)

Compute the approximate Nth root of a value.

See also [rootn\(\)](#).

### native\_rsqrt : Approximate reciprocal of a square root

`float native_rsqrt(float v);`     Added in [API level 21](#)

`float2 native_rsqrt(float2 v);`     Added in [API level 21](#)

`float3 native_rsqrt(float3 v);`     Added in [API level 21](#)

`float4 native_rsqrt(float4 v);`     Added in [API level 21](#)

`half native_rsqrt(half v);`     Added in [API level 24](#)

`half2 native_rsqrt(half2 v);`     Added in [API level 24](#)

`half3 native_rsqrt(half3 v);`     Added in [API level 24](#)

`half4 native_rsqrt(half4 v);`     Added in [API level 24](#)

Returns approximate (1 / sqrt(v)).

See also [rsqrt\(\)](#), [half\\_rsqrt\(\)](#).

### native\_sin : Approximate sine

`float native_sin(float v);` Added in API level 21

`float2 native_sin(float2 v);` Added in API level 21

`float3 native_sin(float3 v);` Added in API level 21

`float4 native_sin(float4 v);` Added in API level 21

`half native_sin(half v);` Added in API level 24

`half2 native_sin(half2 v);` Added in API level 24

`half3 native_sin(half3 v);` Added in API level 24

`half4 native_sin(half4 v);` Added in API level 24

Returns the approximate sine of an angle measured in radians.

See also [sin\(\)](#).

## native\_sincos : Approximate sine and cosine

`float native_sincos(float v, float* cos);` Added in API level 21

`float2 native_sincos(float2 v, float2* cos);` Added in API level 21

`float3 native_sincos(float3 v, float3* cos);` Added in API level 21

`float4 native_sincos(float4 v, float4* cos);` Added in API level 21

`half native_sincos(half v, half* cos);` Added in API level 24

`half2 native_sincos(half2 v, half2* cos);` Added in API level 24

`half3 native_sincos(half3 v, half3* cos);` Added in API level 24

`half4 native_sincos(half4 v, half4* cos);` Added in API level 24

### Parameters

`v` Incoming value in radians.

`cos` `*cos` will be set to the cosine value.

### Returns

Sine.

Returns the approximate sine and cosine of a value.

See also [sincos\(\)](#).

## native\_sinh : Approximate hyperbolic sine

`float native_sinh(float v);` Added in API level 21

`float2 native_sinh(float2 v);` Added in API level 21

`float3 native_sinh(float3 v);` Added in API level 21

`float4 native_sinh(float4 v);` Added in API level 21

`half native_sinh(half v);` Added in API level 24

`half2 native_sinh(half2 v);` Added in API level 24

`half3 native_sinh(half3 v);` Added in API level 24

`half4 native_sinh(half4 v);` Added in API level 24

Returns the approximate hyperbolic sine of a value specified in radians.

See also [sinh\(\)](#).

## native\_sinpi : Approximate sine of a number multiplied by pi

`float native_sinpi(float v);` Added in API level 21

`float2 native_sinpi(float2 v);` Added in API level 21

`float3 native_sinpi(float3 v);` Added in API level 21

`float4 native_sinpi(float4 v);` Added in API level 21  
`half native_sinpi(half v);` Added in API level 24  
`half2 native_sinpi(half2 v);` Added in API level 24  
`half3 native_sinpi(half3 v);` Added in API level 24  
`half4 native_sinpi(half4 v);` Added in API level 24

Returns the approximate sine of ( $v * \pi$ ), where ( $v * \pi$ ) is measured in radians.

To get the sine of a value measured in degrees, call `sinpi(v / 180.f)`.

See also [sinpi\(\)](#).

#### native\_sqrt : Approximate square root

`float native_sqrt(float v);` Added in API level 21  
`float2 native_sqrt(float2 v);` Added in API level 21  
`float3 native_sqrt(float3 v);` Added in API level 21  
`float4 native_sqrt(float4 v);` Added in API level 21  
`half native_sqrt(half v);` Added in API level 24  
`half2 native_sqrt(half2 v);` Added in API level 24  
`half3 native_sqrt(half3 v);` Added in API level 24  
`half4 native_sqrt(half4 v);` Added in API level 24

Returns the approximate  $\sqrt{v}$ .

See also [sqrt\(\)](#), [half\\_sqrt\(\)](#).

#### native\_tan : Approximate tangent

`float native_tan(float v);` Added in API level 21  
`float2 native_tan(float2 v);` Added in API level 21  
`float3 native_tan(float3 v);` Added in API level 21  
`float4 native_tan(float4 v);` Added in API level 21  
`half native_tan(half v);` Added in API level 24  
`half2 native_tan(half2 v);` Added in API level 24  
`half3 native_tan(half3 v);` Added in API level 24  
`half4 native_tan(half4 v);` Added in API level 24

Returns the approximate tangent of an angle measured in radians.

#### native\_tanh : Approximate hyperbolic tangent

`float native_tanh(float v);` Added in API level 21  
`float2 native_tanh(float2 v);` Added in API level 21  
`float3 native_tanh(float3 v);` Added in API level 21  
`float4 native_tanh(float4 v);` Added in API level 21  
`half native_tanh(half v);` Added in API level 24  
`half2 native_tanh(half2 v);` Added in API level 24  
`half3 native_tanh(half3 v);` Added in API level 24  
`half4 native_tanh(half4 v);` Added in API level 24

Returns the approximate hyperbolic tangent of a value.

See also [tanh\(\)](#).

## native\_tanpi : Approximate tangent of a number multiplied by pi

```
float native_tanpi(float v); Added in API level 21
float2 native_tanpi(float2 v); Added in API level 21
float3 native_tanpi(float3 v); Added in API level 21
float4 native_tanpi(float4 v); Added in API level 21
half native_tanpi(half v); Added in API level 24
half2 native_tanpi(half2 v); Added in API level 24
half3 native_tanpi(half3 v); Added in API level 24
half4 native_tanpi(half4 v); Added in API level 24
```

Returns the approximate tangent of ( $v * \pi$ ), where ( $v * \pi$ ) is measured in radians.

To get the tangent of a value measured in degrees, call `tanpi(v / 180.f)`.

See also [tanpi\(\)](#).

## nextafter : Next floating point number

```
float nextafter(float v, float target);
float2 nextafter(float2 v, float2 target);
float3 nextafter(float3 v, float3 target);
float4 nextafter(float4 v, float4 target);
half nextafter(half v, half target); Added in API level 24
half2 nextafter(half2 v, half2 target); Added in API level 24
half3 nextafter(half3 v, half3 target); Added in API level 24
half4 nextafter(half4 v, half4 target); Added in API level 24
```

Returns the next representable floating point number from v towards target.

In rs\_fp\_relaxed mode, a denormalized input value may not yield the next denormalized value, as support of denormalized values is optional in relaxed mode.

## pow : Base raised to an exponent

```
float pow(float base, float exponent);
float2 pow(float2 base, float2 exponent);
float3 pow(float3 base, float3 exponent);
float4 pow(float4 base, float4 exponent);
half pow(half base, half exponent); Added in API level 24
half2 pow(half2 base, half2 exponent); Added in API level 24
half3 pow(half3 base, half3 exponent); Added in API level 24
half4 pow(half4 base, half4 exponent); Added in API level 24
```

Returns base raised to the power exponent, i.e.  $\text{base}^{\text{exponent}}$ .

`pown()` and `powr()` are similar. `pown()` takes an integer exponent. `powr()` assumes the base to be non-negative.

## pown : Base raised to an integer exponent

```
float pown(float base, int exponent);
float2 pown(float2 base, int2 exponent);
float3 pown(float3 base, int3 exponent);
float4 pown(float4 base, int4 exponent);
```

```
half pow(half base, int exponent); Added in API level 24
half2 pow(half2 base, int2 exponent); Added in API level 24
half3 pow(half3 base, int3 exponent); Added in API level 24
half4 pow(half4 base, int4 exponent); Added in API level 24
```

Returns base raised to the power exponent, i.e. base  $\wedge$  exponent.

`pow()` and `powr()` are similar. The both take a float exponent. `powr()` also assumes the base to be non-negative.

### powr : Positive base raised to an exponent

```
float powr(float base, float exponent);
float2 powr(float2 base, float2 exponent);
float3 powr(float3 base, float3 exponent);
float4 powr(float4 base, float4 exponent);

half powr(half base, half exponent); Added in API level 24
half2 powr(half2 base, half2 exponent); Added in API level 24
half3 powr(half3 base, half3 exponent); Added in API level 24
half4 powr(half4 base, half4 exponent); Added in API level 24
```

Returns base raised to the power exponent, i.e. base  $\wedge$  exponent. base must be  $\geq 0$ .

`pow()` and `pown()` are similar. They both make no assumptions about the base. `pow()` takes a float exponent while `pown()` take an integer.

See also [native\\_powr\(\)](#).

### radians : Converts degrees into radians

```
float radians(float v);
float2 radians(float2 v);
float3 radians(float3 v);
float4 radians(float4 v);

half radians(half v); Added in API level 24
half2 radians(half2 v); Added in API level 24
half3 radians(half3 v); Added in API level 24
half4 radians(half4 v); Added in API level 24
```

Converts from degrees to radians.

### remainder : Remainder of a division

```
float remainder(float numerator, float denominator);
float2 remainder(float2 numerator, float2 denominator);
float3 remainder(float3 numerator, float3 denominator);
float4 remainder(float4 numerator, float4 denominator);

half remainder(half numerator, half denominator); Added in API level 24
half2 remainder(half2 numerator, half2 denominator); Added in API level 24
half3 remainder(half3 numerator, half3 denominator); Added in API level 24
half4 remainder(half4 numerator, half4 denominator); Added in API level 24
```

Returns the remainder of (numerator / denominator), where the quotient is rounded towards the nearest integer.

The function `fmod()` is similar but rounds toward the closest integer. For example, `fmod(-3.8f, 2.f)` returns -1.8f (-3.8f - -1.f \* 2.f) while `remainder(-3.8f, 2.f)` returns 0.2f (-3.8f - -2.f \* 2.f).

### remquo : Remainder and quotient of a division

```
float remquo(float numerator, float denominator, int* quotient);
float2 remquo(float2 numerator, float2 denominator, int2* quotient);
float3 remquo(float3 numerator, float3 denominator, int3* quotient);
float4 remquo(float4 numerator, float4 denominator, int4* quotient);
half remquo(half numerator, half denominator, int* quotient); Added in API level 24
half2 remquo(half2 numerator, half2 denominator, int2* quotient); Added in API level 24
half3 remquo(half3 numerator, half3 denominator, int3* quotient); Added in API level 24
half4 remquo(half4 numerator, half4 denominator, int4* quotient); Added in API level 24
```

## Parameters

*numerator*      Numerator.  
*denominator*    Denominator.  
*quotient*       \*quotient will be set to the integer quotient.

## Returns

Remainder, precise only for the low three bits.

Returns the quotient and the remainder of (numerator / denominator).

Only the sign and lowest three bits of the quotient are guaranteed to be accurate.

This function is useful for implementing periodic functions. The low three bits of the quotient gives the quadrant and the remainder the distance within the quadrant. For example, an implementation of `sin(x)` could call `remquo(x, PI / 2.f, &quadrant)` to reduce very large value of `x` to something within a limited range.

Example: `remquo(-23.5f, 8.f, &quot)` sets the lowest three bits of `quot` to 3 and the sign negative. It returns 0.5f.

## rint : Round to even

```
float rint(float v);
float2 rint(float2 v);
float3 rint(float3 v);
float4 rint(float4 v);
half rint(half v); Added in API level 24
half2 rint(half2 v); Added in API level 24
half3 rint(half3 v); Added in API level 24
half4 rint(half4 v); Added in API level 24
```

Rounds to the nearest integral value.

`rint()` rounds half values to even. For example, `rint(0.5f)` returns 0.f and `rint(1.5f)` returns 2.f. Similarly, `rint(-0.5f)` returns -0.f and `rint(-1.5f)` returns -2.f.

`round()` is similar but rounds away from zero. `trunc()` truncates the decimal fraction.

## rootn : Nth root

```
float rootn(float v, int n);
float2 rootn(float2 v, int2 n);
float3 rootn(float3 v, int3 n);
float4 rootn(float4 v, int4 n);
half rootn(half v, int n); Added in API level 24
half2 rootn(half2 v, int2 n); Added in API level 24
half3 rootn(half3 v, int3 n); Added in API level 24
half4 rootn(half4 v, int4 n); Added in API level 24
```

Compute the Nth root of a value.

See also [native\\_rootn\(\)](#).

## round : Round away from zero

```
float round(float v);
float2 round(float2 v);
float3 round(float3 v);
float4 round(float4 v);
half round(half v); Added in API level 24
half2 round(half2 v); Added in API level 24
half3 round(half3 v); Added in API level 24
half4 round(half4 v); Added in API level 24
```

Round to the nearest integral value.

`round()` rounds half values away from zero. For example, `round(0.5f)` returns 1.f and `round(1.5f)` returns 2.f. Similarly, `round(-0.5f)` returns -1.f and `round(-1.5f)` returns -2.f.

`rint()` is similar but rounds half values toward even. `trunc()` truncates the decimal fraction.

## rsClamp : Restrain a value to a range

```
char rsClamp(char amount, char low, char high);
int rsClamp(int amount, int low, int high);
short rsClamp(short amount, short low, short high);
uchar rsClamp(uchar amount, uchar low, uchar high);
uint rsClamp(uint amount, uint low, uint high);
ushort rsClamp(ushort amount, ushort low, ushort high);
```

### Parameters

*amount* Value to clamp.

*low* Lower bound.

*high* Upper bound.

**Deprecated.** Use `clamp()` instead.

Clamp a value between low and high.

## rsFrac : Returns the fractional part of a float

```
float rsFrac(float v);
```

**Deprecated.** Use `fract()` instead.

Returns the fractional part of a float

## rsRand : Pseudo-random number

```
float rsRand(float max_value);
float rsRand(float min_value, float max_value);
int rsRand(int max_value);
int rsRand(int min_value, int max_value);
```

Return a random value between 0 (or `min_value`) and `max_value`.

## rsqrt : Reciprocal of a square root

```
float rsqrt(float v);
float2 rsqrt(float2 v);
float3 rsqrt(float3 v);
float4 rsqrt(float4 v);
half rsqrt(half v); Added in API level 24
half2 rsqrt(half2 v); Added in API level 24
half3 rsqrt(half3 v); Added in API level 24
half4 rsqrt(half4 v); Added in API level 24
```

Returns  $(1 / \sqrt{v})$ .

See also [half\\_rsqrt\(\)](#), [native\\_rsqrt\(\)](#).

## sign : Sign of a value

```
float sign(float v);
float2 sign(float2 v);
float3 sign(float3 v);
float4 sign(float4 v);
half sign(half v); Added in API level 24
half2 sign(half2 v); Added in API level 24
half3 sign(half3 v); Added in API level 24
half4 sign(half4 v); Added in API level 24
```

Returns the sign of a value.

```
if (v < 0) return -1.f; else if (v > 0) return 1.f; else return 0.f;
```

## sin : Sine

```
float sin(float v);
float2 sin(float2 v);
float3 sin(float3 v);
float4 sin(float4 v);
half sin(half v); Added in API level 24
half2 sin(half2 v); Added in API level 24
half3 sin(half3 v); Added in API level 24
half4 sin(half4 v); Added in API level 24
```

Returns the sine of an angle measured in radians.

See also [native\\_sin\(\)](#).

## sincos : Sine and cosine

```
float sincos(float v, float* cos);
float2 sincos(float2 v, float2* cos);
float3 sincos(float3 v, float3* cos);
float4 sincos(float4 v, float4* cos);
half sincos(half v, half* cos); Added in API level 24
half2 sincos(half2 v, half2* cos); Added in API level 24
half3 sincos(half3 v, half3* cos); Added in API level 24
```

`half4 sincos(half4 v, half4* cos);`      Added in API level 24

#### Parameters

`v`      Incoming value in radians.  
`cos`    \*cos will be set to the cosine value.

#### Returns

Sine of v.

Returns the sine and cosine of a value.

See also [native\\_sincos\(\)](#).

## sinh : Hyperbolic sine

```
float sinh(float v);
float2 sinh(float2 v);
float3 sinh(float3 v);
float4 sinh(float4 v);
half sinh(half v); Added in API level 24
half2 sinh(half2 v); Added in API level 24
half3 sinh(half3 v); Added in API level 24
half4 sinh(half4 v); Added in API level 24
```

Returns the hyperbolic sine of v, where v is measured in radians.

See also [native\\_sinh\(\)](#).

## sinpi : Sine of a number multiplied by pi

```
float sinpi(float v);
float2 sinpi(float2 v);
float3 sinpi(float3 v);
float4 sinpi(float4 v);
half sinpi(half v); Added in API level 24
half2 sinpi(half2 v); Added in API level 24
half3 sinpi(half3 v); Added in API level 24
half4 sinpi(half4 v); Added in API level 24
```

Returns the sine of (v \* pi), where (v \* pi) is measured in radians.

To get the sine of a value measured in degrees, call `sinpi(v / 180.f)`.

See also [native\\_sinpi\(\)](#).

## sqrt : Square root

```
float sqrt(float v);
float2 sqrt(float2 v);
float3 sqrt(float3 v);
float4 sqrt(float4 v);
half sqrt(half v); Added in API level 24
half2 sqrt(half2 v); Added in API level 24
half3 sqrt(half3 v); Added in API level 24
half4 sqrt(half4 v); Added in API level 24
```

Returns the square root of a value.

See also [half\\_sqrt\(\)](#), [native\\_sqrt\(\)](#).

### step : 0 if less than a value, 0 otherwise

```
float step(float edge, float v);
float2 step(float edge, float2 v); Added in API level 21
float2 step(float2 edge, float v);
float2 step(float2 edge, float2 v);
float3 step(float edge, float3 v); Added in API level 21
float3 step(float3 edge, float v);
float3 step(float3 edge, float3 v);
float4 step(float edge, float4 v); Added in API level 21
float4 step(float4 edge, float v);
float4 step(float4 edge, float4 v);
half step(half edge, half v); Added in API level 24
half2 step(half edge, half2 v); Added in API level 24
half2 step(half2 edge, half v); Added in API level 24
half2 step(half2 edge, half2 v); Added in API level 24
half3 step(half edge, half3 v); Added in API level 24
half3 step(half3 edge, half v); Added in API level 24
half3 step(half3 edge, half3 v); Added in API level 24
half4 step(half edge, half4 v); Added in API level 24
half4 step(half4 edge, half v); Added in API level 24
half4 step(half4 edge, half4 v); Added in API level 24
```

Returns 0.f if v < edge, 1.f otherwise.

This can be useful to create conditional computations without using loops and branching instructions. For example, instead of computing `(a[i] < b[i]) ? 0.f : atan2(a[i], b[i])` for the corresponding elements of a vector, you could instead use `step(a, b) * atan2(a, b)`.

### tan : Tangent

```
float tan(float v);
float2 tan(float2 v);
float3 tan(float3 v);
float4 tan(float4 v);
half tan(half v); Added in API level 24
half2 tan(half2 v); Added in API level 24
half3 tan(half3 v); Added in API level 24
half4 tan(half4 v); Added in API level 24
```

Returns the tangent of an angle measured in radians.

See also [native\\_tan\(\)](#).

### tanh : Hyperbolic tangent

```
float tanh(float v);
float2 tanh(float2 v);
float3 tanh(float3 v);
```

```
float4 tanh(float4 v);

half tanh(half v); Added in API level 24
half2 tanh(half2 v); Added in API level 24
half3 tanh(half3 v); Added in API level 24
half4 tanh(half4 v); Added in API level 24
```

Returns the hyperbolic tangent of a value.

See also [native\\_tanh\(\)](#).

#### tanpi : Tangent of a number multiplied by pi

```
float tanpi(float v);

float2 tanpi(float2 v);
float3 tanpi(float3 v);
float4 tanpi(float4 v);

half tanpi(half v); Added in API level 24
half2 tanpi(half2 v); Added in API level 24
half3 tanpi(half3 v); Added in API level 24
half4 tanpi(half4 v); Added in API level 24
```

Returns the tangent of ( $v * \pi$ ), where ( $v * \pi$ ) is measured in radians.

To get the tangent of a value measured in degrees, call `tanpi(v / 180.f)`.

See also [native\\_tanpi\(\)](#).

#### tgamma : Gamma function

```
float tgamma(float v);

float2 tgamma(float2 v);
float3 tgamma(float3 v);
float4 tgamma(float4 v);

half tgamma(half v); Added in API level 24
half2 tgamma(half2 v); Added in API level 24
half3 tgamma(half3 v); Added in API level 24
half4 tgamma(half4 v); Added in API level 24
```

Returns the gamma function of a value.

See also [lgamma\(\)](#).

#### trunc : Truncates a floating point

```
float trunc(float v);

float2 trunc(float2 v);
float3 trunc(float3 v);
float4 trunc(float4 v);

half trunc(half v); Added in API level 24
half2 trunc(half2 v); Added in API level 24
half3 trunc(half3 v); Added in API level 24
half4 trunc(half4 v); Added in API level 24
```

Rounds to integral using truncation.

For example, `trunc(1.7f)` returns 1.f and `trunc(-1.7f)` returns -1.f.

See `rint()` and `round()` for other rounding options.

# RenderScript Vector Math Functions

## Overview

These functions interpret the input arguments as representation of vectors in n-dimensional space.

The precision of the mathematical operations on 32 bit floats is affected by the pragmas `rs_fp_relaxed` and `rs_fp_full`. See [Mathematical Constants and Functions](#) for details.

Different precision/speed tradeoffs can be achieved by using variants of the common math functions. Functions with a name starting with

- `native_`: May have custom hardware implementations with weaker precision. Additionally, subnormal values may be flushed to zero, rounding towards zero may be used, and NaN and infinity input may not be handled correctly.
- `fast_`: May perform internal computations using 16 bit floats. Additionally, subnormal values may be flushed to zero, and rounding towards zero may be used.

## Summary

| Functions                     |                                         |
|-------------------------------|-----------------------------------------|
| <code>cross</code>            | Cross product of two vectors            |
| <code>distance</code>         | Distance between two points             |
| <code>dot</code>              | Dot product of two vectors              |
| <code>fast_distance</code>    | Approximate distance between two points |
| <code>fast_length</code>      | Approximate length of a vector          |
| <code>fast_normalize</code>   | Approximate normalized vector           |
| <code>length</code>           | Length of a vector                      |
| <code>native_distance</code>  | Approximate distance between two points |
| <code>native_length</code>    | Approximate length of a vector          |
| <code>native_normalize</code> | Approximately normalize a vector        |
| <code>normalize</code>        | Normalize a vector                      |

## Functions

`cross` : Cross product of two vectors

```
float3 cross(float3 left_vector, float3 right_vector);
float4 cross(float4 left_vector, float4 right_vector);
half3 cross(half3 left_vector, half3 right_vector); Added in API level 24
half4 cross(half4 left_vector, half4 right_vector); Added in API level 24
```

Computes the cross product of two vectors.

`distance` : Distance between two points

```
float distance(float left_vector, float right_vector);
float distance(float2 left_vector, float2 right_vector);
float distance(float3 left_vector, float3 right_vector);
float distance(float4 left_vector, float4 right_vector);

half distance(half left_vector, half right_vector); Added in API level 24
half distance(half2 left_vector, half2 right_vector); Added in API level 24
half distance(half3 left_vector, half3 right_vector); Added in API level 24
half distance(half4 left_vector, half4 right_vector); Added in API level 24
```

Compute the distance between two points.

See also [fast\\_distance\(\)](#), [native\\_distance\(\)](#).

#### dot : Dot product of two vectors

```
float dot(float left_vector, float right_vector);
float dot(float2 left_vector, float2 right_vector);
float dot(float3 left_vector, float3 right_vector);
float dot(float4 left_vector, float4 right_vector);

half dot(half left_vector, half right_vector); Added in API level 24
half dot(half2 left_vector, half2 right_vector); Added in API level 24
half dot(half3 left_vector, half3 right_vector); Added in API level 24
half dot(half4 left_vector, half4 right_vector); Added in API level 24
```

Computes the dot product of two vectors.

#### fast\_distance : Approximate distance between two points

```
float fast_distance(float left_vector, float right_vector); Added in API level 17
float fast_distance(float2 left_vector, float2 right_vector); Added in API level 17
float fast_distance(float3 left_vector, float3 right_vector); Added in API level 17
float fast_distance(float4 left_vector, float4 right_vector); Added in API level 17
```

Computes the approximate distance between two points.

The precision is what would be expected from doing the computation using 16 bit floating point values.

See also [distance\(\)](#), [native\\_distance\(\)](#).

#### fast\_length : Approximate length of a vector

```
float fast_length(float v); Added in API level 17
float fast_length(float2 v); Added in API level 17
float fast_length(float3 v); Added in API level 17
float fast_length(float4 v); Added in API level 17
```

Computes the approximate length of a vector.

The precision is what would be expected from doing the computation using 16 bit floating point values.

See also [length\(\)](#), [native\\_length\(\)](#).

#### fast\_normalize : Approximate normalized vector

```
float fast_normalize(float v); Added in API level 17
float2 fast_normalize(float2 v); Added in API level 17
float3 fast_normalize(float3 v); Added in API level 17
```

```
float4 fast_normalize(float4 v); Added in API level 17
```

Approximately normalizes a vector.

For vectors of size 1, returns -1.f for negative values, 0.f for null values, and 1.f for positive values.

The precision is what would be expected from doing the computation using 16 bit floating point values.

See also [normalize\(\)](#), [native\\_normalize\(\)](#).

#### length : Length of a vector

```
float length(float v);
float length(float2 v);
float length(float3 v);
float length(float4 v);
half length(half v); Added in API level 24
half length(half2 v); Added in API level 24
half length(half3 v); Added in API level 24
half length(half4 v); Added in API level 24
```

Computes the length of a vector.

See also [fast\\_length\(\)](#), [native\\_length\(\)](#).

#### native\_distance : Approximate distance between two points

```
float native_distance(float left_vector, float right_vector); Added in API level 21
float native_distance(float2 left_vector, float2 right_vector); Added in API level 21
float native_distance(float3 left_vector, float3 right_vector); Added in API level 21
float native_distance(float4 left_vector, float4 right_vector); Added in API level 21
half native_distance(half left_vector, half right_vector); Added in API level 24
half native_distance(half2 left_vector, half2 right_vector); Added in API level 24
half native_distance(half3 left_vector, half3 right_vector); Added in API level 24
half native_distance(half4 left_vector, half4 right_vector); Added in API level 24
```

Computes the approximate distance between two points.

See also [distance\(\)](#), [fast\\_distance\(\)](#).

#### native\_length : Approximate length of a vector

```
float native_length(float v); Added in API level 21
float native_length(float2 v); Added in API level 21
float native_length(float3 v); Added in API level 21
float native_length(float4 v); Added in API level 21
half native_length(half v); Added in API level 24
half native_length(half2 v); Added in API level 24
half native_length(half3 v); Added in API level 24
half native_length(half4 v); Added in API level 24
```

Compute the approximate length of a vector.

See also [length\(\)](#), [fast\\_length\(\)](#).

#### native\_normalize : Approximately normalize a vector

|                                                 |                       |
|-------------------------------------------------|-----------------------|
| <code>float native_normalize(float v);</code>   | Added in API level 21 |
| <code>float2 native_normalize(float2 v);</code> | Added in API level 21 |
| <code>float3 native_normalize(float3 v);</code> | Added in API level 21 |
| <code>float4 native_normalize(float4 v);</code> | Added in API level 21 |
| <code>half native_normalize(half v);</code>     | Added in API level 24 |
| <code>half2 native_normalize(half2 v);</code>   | Added in API level 24 |
| <code>half3 native_normalize(half3 v);</code>   | Added in API level 24 |
| <code>half4 native_normalize(half4 v);</code>   | Added in API level 24 |

Approximately normalizes a vector.

See also [normalize\(\)](#), [fast\\_normalize\(\)](#).

#### normalize : Normalize a vector

|                                          |                       |
|------------------------------------------|-----------------------|
| <code>float normalize(float v);</code>   |                       |
| <code>float2 normalize(float2 v);</code> |                       |
| <code>float3 normalize(float3 v);</code> |                       |
| <code>float4 normalize(float4 v);</code> |                       |
| <code>half normalize(half v);</code>     | Added in API level 24 |
| <code>half2 normalize(half2 v);</code>   | Added in API level 24 |
| <code>half3 normalize(half3 v);</code>   | Added in API level 24 |
| <code>half4 normalize(half4 v);</code>   | Added in API level 24 |

Normalize a vector.

For vectors of size 1, returns -1.f for negative values, 0.f for null values, and 1.f for positive values.

See also [fast\\_normalize\(\)](#), [native\\_normalize\(\)](#).

# RenderScript Matrix Functions

## Overview

These functions let you manipulate square matrices of rank 2x2, 3x3, and 4x4. They are particularly useful for graphical transformations and are compatible with OpenGL.

We use a zero-based index for rows and columns. E.g. the last element of a `rs_matrix4x4` is found at (3, 3).

RenderScript uses column-major matrices and column-based vectors. Transforming a vector is done by postmultiplying the vector, e.g. `(matrix * vector)`, as provided by `rsMatrixMultiply()`.

To create a transformation matrix that performs two transformations at once, multiply the two source matrices, with the first transformation as the right argument. E.g. to create a transformation matrix that applies the transformation s1 followed by s2, call `rsMatrixLoadMultiply(&combined, &s2, &s1)`. This derives from `s2 * (s1 * v)`, which is `(s2 * s1) * v`.

We have two style of functions to create transformation matrices: `rsMatrixLoadTransformation` and `rsMatrixTransformation`. The former style simply stores the transformation matrix in the first argument. The latter modifies a pre-existing transformation matrix so that the new transformation happens first. E.g. if you call `rsMatrixTranslate()` on a matrix that already does a scaling, the resulting matrix when applied to a vector will first do the translation then the scaling.

## Summary

| Functions                             |                                                 |
|---------------------------------------|-------------------------------------------------|
| <code>rsExtractFrustumPlanes</code>   | Compute frustum planes                          |
| <code>rsIsSphereInFrustum</code>      | Checks if a sphere is within the frustum planes |
| <code>rsMatrixGet</code>              | Get one element                                 |
| <code>rsMatrixInverse</code>          | Inverts a matrix in place                       |
| <code>rsMatrixInverseTranspose</code> | Inverts and transpose a matrix in place         |
| <code>rsMatrixLoad</code>             | Load or copy a matrix                           |
| <code>rsMatrixLoadFrustum</code>      | Load a frustum projection matrix                |
| <code>rsMatrixLoadIdentity</code>     | Load identity matrix                            |
| <code>rsMatrixLoadMultiply</code>     | Multiply two matrices                           |
| <code>rsMatrixLoadOrtho</code>        | Load an orthographic projection matrix          |
| <code>rsMatrixLoadPerspective</code>  | Load a perspective projection matrix            |
| <code>rsMatrixLoadRotate</code>       | Load a rotation matrix                          |
| <code>rsMatrixLoadScale</code>        | Load a scaling matrix                           |
| <code>rsMatrixLoadTranslate</code>    | Load a translation matrix                       |
| <code>rsMatrixMultiply</code>         | Multiply a matrix by a vector or another matrix |
| <code>rsMatrixRotate</code>           | Apply a rotation to a transformation matrix     |
| <code>rsMatrixScale</code>            | Apply a scaling to a transformation matrix      |
| <code>rsMatrixSet</code>              | Set one element                                 |
| <code>rsMatrixTranslate</code>        | Apply a translation to a transformation matrix  |

## Functions

### rsExtractFrustumPlanes : Compute frustum planes

```
void rsExtractFrustumPlanes(const rs_matrix4x4* viewProj, float4* left, float4* right, float4* top, float4* bottom, float4* near, float4* far);
```

Added in API level 24

```
void rsExtractFrustumPlanes(const rs_matrix4x4* viewProj, float4* left, float4* right, float4* top, float4* bottom, float4* near, float4* far);
```

Removed from API level 24  
and higher

#### Parameters

*viewProj* Matrix to extract planes from.

*left* Left plane.

*right* Right plane.

*top* Top plane.

*bottom* Bottom plane.

*near* Near plane.

*far* Far plane.

*right*

Computes 6 frustum planes from the view projection matrix

### rsIsSphereInFrustum : Checks if a sphere is within the frustum planes

```
bool rsIsSphereInFrustum(float4* sphere, float4* left, float4* right, float4* top, float4* bottom, float4* near, float4* far);
```

#### Parameters

*sphere* float4 representing the sphere.

*left* Left plane.

*right* Right plane.

*top* Top plane.

*bottom* Bottom plane.

*near* Near plane.

*far* Far plane.

Returns true if the sphere is within the 6 frustum planes.

### rsMatrixGet : Get one element

```
float rsMatrixGet(const rs_matrix2x2* m, uint32_t col, uint32_t row);
```

```
float rsMatrixGet(const rs_matrix3x3* m, uint32_t col, uint32_t row);
```

```
float rsMatrixGet(const rs_matrix4x4* m, uint32_t col, uint32_t row);
```

#### Parameters

*m* Matrix to extract the element from.

*col* Zero-based column of the element to be extracted.

*row* Zero-based row of the element to be extracted.

Returns one element of a matrix.

**Warning:** The order of the column and row parameters may be unexpected.

### rsMatrixInverse : Inverts a matrix in place

```
bool rsMatrixInverse(rs_matrix4x4* m);
```

```
DOOR rsMatrixInverse(rs_matrix4x4 m);
```

## Parameters

*m* Matrix to invert.

Returns true if the matrix was successfully inverted.

### rsMatrixInverseTranspose : Inverts and transpose a matrix in place

```
bool rsMatrixInverseTranspose(rs_matrix4x4* m);
```

## Parameters

*m* Matrix to modify.

The matrix is first inverted then transposed. Returns true if the matrix was successfully inverted.

### rsMatrixLoad : Load or copy a matrix

```
void rsMatrixLoad(rs_matrix2x2* destination, const float* array);
void rsMatrixLoad(rs_matrix2x2* destination, const rs_matrix2x2* source);
void rsMatrixLoad(rs_matrix3x3* destination, const float* array);
void rsMatrixLoad(rs_matrix3x3* destination, const rs_matrix3x3* source);
void rsMatrixLoad(rs_matrix4x4* destination, const float* array);
void rsMatrixLoad(rs_matrix4x4* destination, const rs_matrix2x2* source);
void rsMatrixLoad(rs_matrix4x4* destination, const rs_matrix3x3* source);
void rsMatrixLoad(rs_matrix4x4* destination, const rs_matrix4x4* source);
```

## Parameters

*destination* Matrix to set.

*array* Array of values to set the matrix to. These arrays should be 4, 9, or 16 floats long, depending on the matrix size.

*source* Source matrix.

Set the elements of a matrix from an array of floats or from another matrix.

If loading from an array, the floats should be in row-major order, i.e. the element at **row 0, column 0** should be first, followed by the element at **row 0, column 1**, etc.

If loading from a matrix and the source is smaller than the destination, the rest of the destination is filled with elements of the identity matrix.  
E.g. loading a rs\_matrix2x2 into a rs\_matrix4x4 will give:

|     |     |     |     |
|-----|-----|-----|-----|
| m00 | m01 | 0.0 | 0.0 |
| m10 | m11 | 0.0 | 0.0 |
| 0.0 | 0.0 | 1.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 1.0 |

### rsMatrixLoadFrustum : Load a frustum projection matrix

```
void rsMatrixLoadFrustum(rs_matrix4x4* m, float left, float right, float bottom, float top, float near, float far);
```

## Parameters

*m* Matrix to set.

*left*

*right*

*bottom*

*top*

*near*

*far*

...

Constructs a frustum projection matrix, transforming the box identified by the six clipping planes `left`, `right`, `bottom`, `top`, `near`, `far`.

To apply this projection to a vector, multiply the vector by the created matrix using `rsMatrixMultiply()`.

### rsMatrixLoadIdentity : Load identity matrix

```
void rsMatrixLoadIdentity(rs_matrix2x2* m);
void rsMatrixLoadIdentity(rs_matrix3x3* m);
void rsMatrixLoadIdentity(rs_matrix4x4* m);
```

#### Parameters

*m* Matrix to set.

Set the elements of a matrix to the identity matrix.

### rsMatrixLoadMultiply : Multiply two matrices

```
void rsMatrixLoadMultiply(rs_matrix2x2* m, const rs_matrix2x2* lhs, const rs_matrix2x2* rhs);
void rsMatrixLoadMultiply(rs_matrix3x3* m, const rs_matrix3x3* lhs, const rs_matrix3x3* rhs);
void rsMatrixLoadMultiply(rs_matrix4x4* m, const rs_matrix4x4* lhs, const rs_matrix4x4* rhs);
```

#### Parameters

*m* Matrix to set.

*lhs* Left matrix of the product.

*rhs* Right matrix of the product.

Sets *m* to the matrix product of `lhs * rhs`.

To combine two 4x4 transformaton matrices, multiply the second transformation matrix by the first transformation matrix. E.g. to create a transformation matrix that applies the transformation s1 followed by s2, call `rsMatrixLoadMultiply(&combined, &s2, &s1)`.

**Warning:** Prior to version 21, storing the result back into right matrix is not supported and will result in undefined behavior. Use `rsMatrixMulitply` instead. E.g. instead of doing `rsMatrixLoadMultiply (&m2r, &m2r, &m2l)`, use `rsMatrixMultiply (&m2r, &m2l)`. `rsMatrixLoadMultiply (&m2l, &m2r, &m2l)` works as expected.

### rsMatrixLoadOrtho : Load an orthographic projection matrix

```
void rsMatrixLoadOrtho(rs_matrix4x4* m, float left, float right, float bottom, float top, float near, float far);
```

#### Parameters

*m* Matrix to set.

*left*

*right*

*bottom*

*top*

*near*

*far*

Constructs an orthographic projection matrix, transforming the box identified by the six clipping planes `left`, `right`, `bottom`, `top`, `near`, `far` into a unit cube with a corner at `(-1, -1, -1)` and the opposite at `(1, 1, 1)`.

To apply this projection to a vector, multiply the vector by the created matrix using `rsMatrixMultiply()`.

See [https://en.wikipedia.org/wiki/Orthographic\\_projection](https://en.wikipedia.org/wiki/Orthographic_projection) .

### rsMatrixLoadPerspective : Load a perspective projection matrix

```
void rsMatrixLoadPerspective(rs_matrix4x4* m, float fovy, float aspect, float near, float far);
```

#### Parameters

- m* Matrix to set.
- fovy* Field of view, in degrees along the Y axis.
- aspect* Ratio of x / y.
- near* Near clipping plane.
- far* Far clipping plane.

Constructs a perspective projection matrix, assuming a symmetrical field of view.

To apply this projection to a vector, multiply the vector by the created matrix using [rsMatrixMultiply\(\)](#).

### rsMatrixLoadRotate : Load a rotation matrix

```
void rsMatrixLoadRotate(rs_matrix4x4* m, float rot, float x, float y, float z);
```

#### Parameters

- m* Matrix to set.
- rot* How much rotation to do, in degrees.
- x* X component of the vector that is the axis of rotation.
- y* Y component of the vector that is the axis of rotation.
- z* Z component of the vector that is the axis of rotation.

This function creates a rotation matrix. The axis of rotation is the (*x*, *y*, *z*) vector.

To rotate a vector, multiply the vector by the created matrix using [rsMatrixMultiply\(\)](#).

See [http://en.wikipedia.org/wiki/Rotation\\_matrix](http://en.wikipedia.org/wiki/Rotation_matrix) .

### rsMatrixLoadScale : Load a scaling matrix

```
void rsMatrixLoadScale(rs_matrix4x4* m, float x, float y, float z);
```

#### Parameters

- m* Matrix to set.
- x* Multiple to scale the x components by.
- y* Multiple to scale the y components by.
- z* Multiple to scale the z components by.

This function creates a scaling matrix, where each component of a vector is multiplied by a number. This number can be negative.

To scale a vector, multiply the vector by the created matrix using [rsMatrixMultiply\(\)](#).

### rsMatrixLoadTranslate : Load a translation matrix

```
void rsMatrixLoadTranslate(rs_matrix4x4* m, float x, float y, float z);
```

#### Parameters

- m* Matrix to set.
- x* Number to add to each x component.
- y* Number to add to each y component.
- z* Number to add to each z component.

This function creates a translation matrix, where a number is added to each element of a vector.

To translate a vector, multiply the vector by the created matrix using [rsMatrixMultiply\(\)](#).

## rsMatrixMultiply : Multiply a matrix by a vector or another matrix

|                                                                               |                                      |
|-------------------------------------------------------------------------------|--------------------------------------|
| <code>float2 rsMatrixMultiply(const rs_matrix2x2* m, float2 in);</code>       | Added in API level 14                |
| <code>float2 rsMatrixMultiply(rs_matrix2x2* m, float2 in);</code>             | Removed from API level 14 and higher |
| <code>float3 rsMatrixMultiply(const rs_matrix3x3* m, float2 in);</code>       | Added in API level 14                |
| <code>float3 rsMatrixMultiply(const rs_matrix3x3* m, float3 in);</code>       | Added in API level 14                |
| <code>float3 rsMatrixMultiply(rs_matrix3x3* m, float2 in);</code>             | Removed from API level 14 and higher |
| <code>float3 rsMatrixMultiply(rs_matrix3x3* m, float3 in);</code>             | Removed from API level 14 and higher |
| <code>float4 rsMatrixMultiply(const rs_matrix4x4* m, float2 in);</code>       | Added in API level 14                |
| <code>float4 rsMatrixMultiply(const rs_matrix4x4* m, float3 in);</code>       | Added in API level 14                |
| <code>float4 rsMatrixMultiply(const rs_matrix4x4* m, float4 in);</code>       | Added in API level 14                |
| <code>float4 rsMatrixMultiply(rs_matrix4x4* m, float2 in);</code>             | Removed from API level 14 and higher |
| <code>float4 rsMatrixMultiply(rs_matrix4x4* m, float3 in);</code>             | Removed from API level 14 and higher |
| <code>float4 rsMatrixMultiply(rs_matrix4x4* m, float4 in);</code>             | Removed from API level 14 and higher |
| <code>void rsMatrixMultiply(rs_matrix2x2* m, const rs_matrix2x2* rhs);</code> |                                      |
| <code>void rsMatrixMultiply(rs_matrix3x3* m, const rs_matrix3x3* rhs);</code> |                                      |
| <code>void rsMatrixMultiply(rs_matrix4x4* m, const rs_matrix4x4* rhs);</code> |                                      |

### Parameters

- m* Left matrix of the product and the matrix to be set.
- rhs* Right matrix of the product.
- in*

For the matrix by matrix variant, sets *m* to the matrix product *m* \* *rhs*.

When combining two 4x4 transformation matrices using this function, the resulting matrix will correspond to performing the *rhs* transformation first followed by the original *m* transformation.

For the matrix by vector variant, returns the post-multiplication of the vector by the matrix, ie. *m* \* *in*.

When multiplying a `float3` to a `rs_matrix4x4`, the vector is expanded with (1).

When multiplying a `float2` to a `rs_matrix4x4`, the vector is expanded with (0, 1).

When multiplying a `float2` to a `rs_matrix3x3`, the vector is expanded with (0).

Starting with API 14, this function takes a `const` matrix as the first argument.

## rsMatrixRotate : Apply a rotation to a transformation matrix

```
void rsMatrixRotate(rs_matrix4x4* m, float rot, float x, float y, float z);
```

### Parameters

- m* Matrix to modify.
- rot* How much rotation to do, in degrees.
- x* X component of the vector that is the axis of rotation.
- y* Y component of the vector that is the axis of rotation.
- z* Z component of the vector that is the axis of rotation.

Multiply the matrix *m* with a rotation matrix.

This function modifies a transformation matrix to first do a rotation. The axis of rotation is the (*x*, *y*, *z*) vector.

To apply this combined transformation to a vector, multiply the vector by the created matrix using `rsMatrixMultiply()`.

## rsMatrixScale : Apply a scaling to a transformation matrix

```
void rsMatrixScale(rs_matrix4x4* m, float x, float y, float z);
```

#### Parameters

- m* Matrix to modify.
- x* Multiple to scale the x components by.
- y* Multiple to scale the y components by.
- z* Multiple to scale the z components by.

Multiply the matrix *m* with a scaling matrix.

This function modifies a transformation matrix to first do a scaling. When scaling, each component of a vector is multiplied by a number. This number can be negative.

To apply this combined transformation to a vector, multiply the vector by the created matrix using [rsMatrixMultiply\(\)](#).

### rsMatrixSet : Set one element

```
void rsMatrixSet(rs_matrix2x2* m, uint32_t col, uint32_t row, float v);
void rsMatrixSet(rs_matrix3x3* m, uint32_t col, uint32_t row, float v);
void rsMatrixSet(rs_matrix4x4* m, uint32_t col, uint32_t row, float v);
```

#### Parameters

- m* Matrix that will be modified.
- col* Zero-based column of the element to be set.
- row* Zero-based row of the element to be set.
- v* Value to set.

Set an element of a matrix.

**Warning:** The order of the column and row parameters may be unexpected.

### rsMatrixTranslate : Apply a translation to a transformation matrix

```
void rsMatrixTranslate(rs_matrix4x4* m, float x, float y, float z);
```

#### Parameters

- m* Matrix to modify.
- x* Number to add to each x component.
- y* Number to add to each y component.
- z* Number to add to each z component.

Multiply the matrix *m* with a translation matrix.

This function modifies a transformation matrix to first do a translation. When translating, a number is added to each component of a vector.

To apply this combined transformation to a vector, multiply the vector by the created matrix using [rsMatrixMultiply\(\)](#).

### rsMatrixTranspose : Transpose a matrix place

```
void rsMatrixTranspose(rs_matrix2x2* m);
void rsMatrixTranspose(rs_matrix3x3* m);
void rsMatrixTranspose(rs_matrix4x4* m);
```

#### Parameters

- m* Matrix to transpose.

Transpose the matrix *m* in place.

# RenderScript Quaternion Functions

## Overview

The following functions manipulate quaternions.

## Summary

| Functions                                  |                                                                      |
|--------------------------------------------|----------------------------------------------------------------------|
| <a href="#">rsQuaternionAdd</a>            | Add two quaternions                                                  |
| <a href="#">rsQuaternionConjugate</a>      | Conjugate a quaternion                                               |
| <a href="#">rsQuaternionDot</a>            | Dot product of two quaternions                                       |
| <a href="#">rsQuaternionGetMatrixUnit</a>  | Get a rotation matrix from a quaternion                              |
| <a href="#">rsQuaternionLoadRotate</a>     | Create a rotation quaternion                                         |
| <a href="#">rsQuaternionLoadRotateUnit</a> | Quaternion that represents a rotation about an arbitrary unit vector |
| <a href="#">rsQuaternionMultiply</a>       | Multiply a quaternion by a scalar or another quaternion              |
| <a href="#">rsQuaternionNormalize</a>      | Normalize a quaternion                                               |
| <a href="#">rsQuaternionSet</a>            | Create a quaternion                                                  |
| <a href="#">rsQuaternionSlerp</a>          | Spherical linear interpolation between two quaternions               |

## Functions

### rsQuaternionAdd : Add two quaternions

```
void rsQuaternionAdd(rs_quaternion* q, const rs_quaternion* rhs);
```

#### Parameters

*q* Destination quaternion to add to.

*rhs* Quaternion to add.

Adds two quaternions, i.e. `*q += *rhs;`

### rsQuaternionConjugate : Conjugate a quaternion

```
void rsQuaternionConjugate(rs_quaternion* q);
```

#### Parameters

*q* Quaternion to modify.

Conjugates the quaternion.

### rsQuaternionDot : Dot product of two quaternions

```
float rsQuaternionDot(const rs_quaternion* q0, const rs_quaternion* q1);
```

#### Parameters

*q0* First quaternion.  
*q1* Second quaternion.

Returns the dot product of two quaternions.

### rsQuaternionGetMatrixUnit : Get a rotation matrix from a quaternion

```
void rsQuaternionGetMatrixUnit(rs_matrix4x4* m, const rs_quaternion* q);
```

#### Parameters

*m* Resulting matrix.

*q* Normalized quaternion.

Computes a rotation matrix from the normalized quaternion.

### rsQuaternionLoadRotate : Create a rotation quaternion

```
void rsQuaternionLoadRotate(rs_quaternion* q, float rot, float x, float y, float z);
```

#### Parameters

*q* Destination quaternion.

*rot* Angle to rotate by.

*x* X component of a vector.

*y* Y component of a vector.

*z* Z component of a vector.

Loads a quaternion that represents a rotation about an arbitrary vector (doesn't have to be unit)

### rsQuaternionLoadRotateUnit : Quaternion that represents a rotation about an arbitrary unit vector

```
void rsQuaternionLoadRotateUnit(rs_quaternion* q, float rot, float x, float y, float z);
```

#### Parameters

*q* Destination quaternion.

*rot* Angle to rotate by, in radians.

*x* X component of the vector.

*y* Y component of the vector.

*z* Z component of the vector.

Loads a quaternion that represents a rotation about an arbitrary unit vector.

### rsQuaternionMultiply : Multiply a quaternion by a scalar or another quaternion

```
void rsQuaternionMultiply(rs_quaternion* q, const rs_quaternion* rhs);
```

```
void rsQuaternionMultiply(rs_quaternion* q, float scalar);
```

#### Parameters

*q* Destination quaternion.

*scalar* Scalar to multiply the quaternion by.

*rhs* Quaternion to multiply the destination quaternion by.

Multiples a quaternion by a scalar or by another quaternion, e.g `*q = *q * scalar;` or `*q = *q * *rhs;`.

### rsQuaternionNormalize : Normalize a quaternion

```
void rsQuaternionNormalize(rs_quaternion* q);
```

#### Parameters

*q* Quaternion to normalize.

Normalizes the quaternion.

### rsQuaternionSet : Create a quaternion

```
void rsQuaternionSet(rs_quaternion* q, const rs_quaternion* rhs);
void rsQuaternionSet(rs_quaternion* q, float w, float x, float y, float z);
```

#### Parameters

- q* Destination quaternion.
- w* W component.
- x* X component.
- y* Y component.
- z* Z component.
- rhs* Source quaternion.

Creates a quaternion from its four components or from another quaternion.

### rsQuaternionSlerp : Spherical linear interpolation between two quaternions

```
void rsQuaternionSlerp(rs_quaternion* q, const rs_quaternion* q0, const rs_quaternion* q1, float t);
```

#### Parameters

- q* Result quaternion from the interpolation.
- q0* First input quaternion.
- q1* Second input quaternion.
- t* How much to interpolate by.

Performs spherical linear interpolation between two quaternions.

# RenderScript Atomic Update Functions

## Overview

To update values shared between multiple threads, use the functions below. They ensure that the values are atomically updated, i.e. that the memory reads, the updates, and the memory writes are done in the right order.

These functions are slower than their non-atomic equivalents, so use them only when synchronization is needed.

Note that in RenderScript, your code is likely to be running in separate threads even though you did not explicitly create them. The RenderScript runtime will very often split the execution of one kernel across multiple threads. Updating globals should be done with atomic functions. If possible, modify your algorithm to avoid them altogether.

## Summary

| Functions                   |                                  |
|-----------------------------|----------------------------------|
| <a href="#">rsAtomicAdd</a> | Thread-safe addition             |
| <a href="#">rsAtomicAnd</a> | Thread-safe bitwise and          |
| <a href="#">rsAtomicCas</a> | Thread-safe compare and set      |
| <a href="#">rsAtomicDec</a> | Thread-safe decrement            |
| <a href="#">rsAtomicInc</a> | Thread-safe increment            |
| <a href="#">rsAtomicMax</a> | Thread-safe maximum              |
| <a href="#">rsAtomicMin</a> | Thread-safe minimum              |
| <a href="#">rsAtomicOr</a>  | Thread-safe bitwise or           |
| <a href="#">rsAtomicSub</a> | Thread-safe subtraction          |
| <a href="#">rsAtomicXor</a> | Thread-safe bitwise exclusive or |

## Functions

### rsAtomicAdd : Thread-safe addition

`int32_t rsAtomicAdd(volatile int32_t* addr, int32_t value);`      Added in [API level 14](#)

`int32_t rsAtomicAdd(volatile uint32_t* addr, uint32_t value);`      Added in [API level 20](#)

#### Parameters

`addr`    Address of the value to modify.

`value`    Amount to add.

#### Returns

Value of `*addr` prior to the operation.

Atomically adds a value to the value at `addr`, i.e. `*addr += value`.

### rsAtomicAnd : Thread-safe bitwise and

`int32_t rsAtomicAnd(volatile int32_t* addr, int32_t value);`      Added in [API level 14](#)

`int32_t rsAtomicAnd(volatile uint32_t* addr, uint32_t value);`    Added in API level 20

#### Parameters

- `addr`   Address of the value to modify.
- `value`   Value to and with.

#### Returns

Value of \*addr prior to the operation.

Atomically performs a bitwise and of two values, storing the result back at addr, i.e. `*addr &= value`.

### rsAtomicCas : Thread-safe compare and set

`int32_t rsAtomicCas(volatile int32_t* addr, int32_t compareValue, int32_t newValue);`    Added in API level 14

`uint32_t rsAtomicCas(volatile uint32_t* addr, uint32_t compareValue, uint32_t newValue);`    Added in API level 14

#### Parameters

- `addr`   Address of the value to compare and replace if the test passes.
- `compareValue`   Value to test \*addr against.
- `newValue`   Value to write if the test passes.

#### Returns

Value of \*addr prior to the operation.

If the value at addr matches compareValue then the newValue is written at addr, i.e. `if (*addr == compareValue) { *addr = newValue; }`.

You can check that the value was written by checking that the value returned by rsAtomicCas() is compareValue.

### rsAtomicDec : Thread-safe decrement

`int32_t rsAtomicDec(volatile int32_t* addr);`    Added in API level 14

`int32_t rsAtomicDec(volatile uint32_t* addr);`    Added in API level 20

#### Parameters

- `addr`   Address of the value to decrement.

#### Returns

Value of \*addr prior to the operation.

Atomically subtracts one from the value at addr. This is equivalent to `rsAtomicSub(addr, 1)`.

### rsAtomicInc : Thread-safe increment

`int32_t rsAtomicInc(volatile int32_t* addr);`    Added in API level 14

`int32_t rsAtomicInc(volatile uint32_t* addr);`    Added in API level 20

#### Parameters

- `addr`   Address of the value to increment.

#### Returns

Value of \*addr prior to the operation.

Atomically adds one to the value at addr. This is equivalent to `rsAtomicAdd(addr, 1)`.

### rsAtomicMax : Thread-safe maximum

`int32_t rsAtomicMax(volatile int32_t* addr, int32_t value);`    Added in API level 14

`uint32_t rsAtomicMax(volatile uint32_t* addr, uint32_t value);`    Added in API level 14

#### Parameters

*addr* Address of the value to modify.

*value* Comparison value.

## Returns

Value of \*addr prior to the operation.

Atomicly sets the value at addr to the maximum of \*addr and value, i.e. `*addr = max(*addr, value)`.

### rsAtomicMin : Thread-safe minimum

`int32_t rsAtomicMin(volatile int32_t* addr, int32_t value);` Added in API level 14

`uint32_t rsAtomicMin(volatile uint32_t* addr, uint32_t value);` Added in API level 14

## Parameters

*addr* Address of the value to modify.

*value* Comparison value.

## Returns

Value of \*addr prior to the operation.

Atomicly sets the value at addr to the minimum of \*addr and value, i.e. `*addr = min(*addr, value)`.

### rsAtomicOr : Thread-safe bitwise or

`int32_t rsAtomicOr(volatile int32_t* addr, int32_t value);` Added in API level 14

`int32_t rsAtomicOr(volatile uint32_t* addr, uint32_t value);` Added in API level 20

## Parameters

*addr* Address of the value to modify.

*value* Value to or with.

## Returns

Value of \*addr prior to the operation.

Atomicly perform a bitwise or two values, storing the result at addr, i.e. `*addr |= value`.

### rsAtomicSub : Thread-safe subtraction

`int32_t rsAtomicSub(volatile int32_t* addr, int32_t value);` Added in API level 14

`int32_t rsAtomicSub(volatile uint32_t* addr, uint32_t value);` Added in API level 20

## Parameters

*addr* Address of the value to modify.

*value* Amount to subtract.

## Returns

Value of \*addr prior to the operation.

Atomicly subtracts a value from the value at addr, i.e. `*addr -= value`.

### rsAtomicXor : Thread-safe bitwise exclusive or

`int32_t rsAtomicXor(volatile int32_t* addr, int32_t value);` Added in API level 14

`int32_t rsAtomicXor(volatile uint32_t* addr, uint32_t value);` Added in API level 20

## Parameters

*addr* Address of the value to modify.

*value* Value to xor with.

## Returns

Value of `*addr` prior to the operation.

Atomically performs a bitwise xor of two values, storing the result at `addr`, i.e. `*addr ^= value`.

# RenderScript Time Functions and Types

## Overview

The functions below can be used to tell the current clock time and the current system up time. It is not recommended to call these functions inside of a kernel.

## Summary

| Types                     |                               |
|---------------------------|-------------------------------|
| <a href="#">rs_time_t</a> | Seconds since January 1, 1970 |
| <a href="#">rs_tm</a>     | Date and time structure       |

| Functions                      |                               |
|--------------------------------|-------------------------------|
| <a href="#">rsGetDt</a>        | Elapsed time since last call  |
| <a href="#">rsLocaltime</a>    | Convert to local time         |
| <a href="#">rsTime</a>         | Seconds since January 1, 1970 |
| <a href="#">rsUptimeMillis</a> | System uptime in milliseconds |
| <a href="#">rsUptimeNanos</a>  | System uptime in nanoseconds  |

## Types

### [rs\\_time\\_t](#) : Seconds since January 1, 1970

A typedef of: int When compiling for 32 bits.

A typedef of: long When compiling for 64 bits.

Calendar time interpreted as seconds elapsed since the Epoch (00:00:00 on January 1, 1970, Coordinated Universal Time (UTC)).

### [rs\\_tm](#) : Date and time structure

A structure with the following fields:

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| <i>int</i>     | Seconds after the minute. This ranges from 0 to 59, but possibly up to 60 for leap seconds. |
| <i>tm_sec</i>  |                                                                                             |
| <i>int</i>     | Minutes after the hour. This ranges from 0 to 59.                                           |
| <i>tm_min</i>  |                                                                                             |
| <i>int</i>     | Hours past midnight. This ranges from 0 to 23.                                              |
| <i>tm_hour</i> |                                                                                             |
| <i>int</i>     | Day of the month. This ranges from 1 to 31.                                                 |
| <i>tm_mday</i> |                                                                                             |
| <i>int</i>     | Months since January. This ranges from 0 to 11.                                             |
| <i>tm_mon</i>  |                                                                                             |
| <i>int</i>     | Years since 1900.                                                                           |
| <i>tm_year</i> |                                                                                             |

|                 |                                                                                                                                                                            |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>int</i>      | Days since Sunday. This ranges from 0 to 6.                                                                                                                                |
| <i>tm_wday</i>  |                                                                                                                                                                            |
| <i>int</i>      | Days since January 1. This ranges from 0 to 365.                                                                                                                           |
| <i>tm_yday</i>  |                                                                                                                                                                            |
| <i>int</i>      | Flag to indicate whether daylight saving time is in effect. The value is positive if it is in effect, zero if it is not, and negative if the information is not available. |
| <i>tm_isdst</i> |                                                                                                                                                                            |

Data structure for broken-down time components.

## Functions

---

### rsGetDt : Elapsed time since last call

```
float rsGetDt();
```

#### Returns

Time in seconds.

Returns the time in seconds since this function was last called in this script.

### rsLocaltime : Convert to local time

```
rs_tm* rsLocaltime(rs_tm* local, const rs_time_t* timer);
```

#### Parameters

*local* Pointer to time structure where the local time will be stored.

*timer* Input time as a number of seconds since January 1, 1970.

#### Returns

Pointer to the output local time, i.e. the same value as the parameter local.

Converts the time specified by timer into a `rs_tm` structure that provides year, month, hour, etc. This value is stored at \*local.

This functions returns the same pointer that is passed as first argument. If the local parameter is NULL, this function does nothing and returns NULL.

### rsTime : Seconds since January 1, 1970

```
rs_time_t rsTime(rs_time_t* timer);
```

#### Parameters

*timer* Location to also store the returned calendar time.

#### Returns

Seconds since the Epoch, -1 if there's an error.

Returns the number of seconds since the Epoch (00:00:00 UTC, January 1, 1970).

If timer is non-NULL, the result is also stored in the memory pointed to by this variable.

### rsUptimeMillis : System uptime in milliseconds

```
int64_t rsUptimeMillis();
```

#### Returns

Uptime in milliseconds.

Returns the current system clock (uptime) in milliseconds.

### rsUptimeNanos : System uptime in nanoseconds

```
int64_t rsUptimeNanos();
```

## Returns

Uptime in nanoseconds.

Returns the current system clock (uptime) in nanoseconds.

The granularity of the values return by this call may be much larger than a nanosecond.

# RenderScript Allocation Data Access Functions

## Overview

The functions below can be used to get and set the cells that comprise an allocation.

- Individual cells are accessed using the `rsGetElementAt*` and `rsSetElementAt` functions.
- Multiple cells can be copied using the `rsAllocationCopy*` and `rsAllocationV*` functions.
- For getting values through a sampler, use `rsSample`.

The `rsGetElementAt` and `rsSetElementAt` functions are somewhat misnamed. They don't get or set elements, which are akin to data types; they get or set cells. Think of them as `rsGetCellAt` and `rsSetCellAt`.

## Summary

| Functions                                 |                                                        |
|-------------------------------------------|--------------------------------------------------------|
| <a href="#">rsAllocationCopy1DRange</a>   | Copy consecutive cells between allocations             |
| <a href="#">rsAllocationCopy2DRange</a>   | Copy a rectangular region of cells between allocations |
| <a href="#">rsAllocationVLoadX</a>        | Get a vector from an allocation of scalars             |
| <a href="#">rsAllocationVStoreX</a>       | Store a vector into an allocation of scalars           |
| <a href="#">rsGetElementAt</a>            | Return a cell from an allocation                       |
| <a href="#">rsGetElementAtYuv_uchar_U</a> | Get the U component of an allocation of YUVs           |
| <a href="#">rsGetElementAtYuv_uchar_V</a> | Get the V component of an allocation of YUVs           |
| <a href="#">rsGetElementAtYuv_uchar_Y</a> | Get the Y component of an allocation of YUVs           |
| <a href="#">rsSample</a>                  | Sample a value from a texture allocation               |
| <a href="#">rsSetElementAt</a>            | Set a cell of an allocation                            |

## Functions

### rsAllocationCopy1DRange : Copy consecutive cells between allocations

```
void rsAllocationCopy1DRange(rs_allocation dstAlloc, uint32_t dstOff, uint32_t dstMip, uint32_t count, rs_allocation srcAlloc, uint32_t srcOff, uint32_t srcMip);
```

Added in API level 14

#### Parameters

- |                       |                                                                        |
|-----------------------|------------------------------------------------------------------------|
| <code>dstAlloc</code> | Allocation to copy cells into.                                         |
| <code>dstOff</code>   | Offset in the destination of the first cell to be copied into.         |
| <code>dstMip</code>   | Mip level in the destination allocation. 0 if mip mapping is not used. |
| <code>count</code>    | Number of cells to be copied.                                          |
| <code>srcAlloc</code> | Source allocation.                                                     |
| <code>srcOff</code>   | Offset in the source of the first cell to be copied.                   |
| <code>srcMip</code>   | Mip level in the source allocation. 0 if mip mapping is not used.      |

Copies the specified number of cells from one allocation to another.

The two allocations must be different. Using this function to copy whithin the same allocation yields undefined results.

The function does not validate whether the offset plus count exceeds the size of either allocation. Be careful!

This function should only be called between 1D allocations. Calling it on other allocations is undefined.

This function should not be called from inside a kernel, or from any function that may be called directly or indirectly from a kernel. Doing so would cause a runtime error.

### rsAllocationCopy2DRange : Copy a rectangular region of cells between allocations

```
void rsAllocationCopy2DRange(rs_allocation dstAlloc, uint32_t dstXoff, uint32_t dstYoff, uint32_t dstMip,
 rs_allocation_cubemap_face dstFace, uint32_t width, uint32_t height, rs_allocation srcAlloc, uint32_t srcXoff, uint32_t srcYoff,
 uint32_t srcMip, rs_allocation_cubemap_face srcFace);
```

Added in API level 14

#### Parameters

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <i>dstAlloc</i> | Allocation to copy cells into.                                                            |
| <i>dstXoff</i>  | X offset in the destination of the region to be set.                                      |
| <i>dstYoff</i>  | Y offset in the destination of the region to be set.                                      |
| <i>dstMip</i>   | Mip level in the destination allocation. 0 if mip mapping is not used.                    |
| <i>dstFace</i>  | Cubemap face of the destination allocation. Ignored for allocations that aren't cubemaps. |
| <i>width</i>    | Width of the incoming region to update.                                                   |
| <i>height</i>   | Height of the incoming region to update.                                                  |
| <i>srcAlloc</i> | Source allocation.                                                                        |
| <i>srcXoff</i>  | X offset in the source.                                                                   |
| <i>srcYoff</i>  | Y offset in the source.                                                                   |
| <i>srcMip</i>   | Mip level in the source allocation. 0 if mip mapping is not used.                         |
| <i>srcFace</i>  | Cubemap face of the source allocation. Ignored for allocations that aren't cubemaps.      |

Copies a rectangular region of cells from one allocation to another. (width \* height) cells are copied.

The two allocations must be different. Using this function to copy whithin the same allocation yields undefined results.

The function does not validate whether the the source or destination region exceeds the size of its respective allocation. Be careful!

This function should only be called between 2D allocations. Calling it on other allocations is undefined.

This function should not be called from inside a kernel, or from any function that may be called directly or indirectly from a kernel. Doing so would cause a runtime error.

### rsAllocationVLoadX : Get a vector from an allocation of scalars

```
char2 rsAllocationVLoadX_char2(rs_allocation a, uint32_t x);
char2 rsAllocationVLoadX_char2(rs_allocation a, uint32_t x, uint32_t y);
char2 rsAllocationVLoadX_char2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
char3 rsAllocationVLoadX_char3(rs_allocation a, uint32_t x);
char3 rsAllocationVLoadX_char3(rs_allocation a, uint32_t x, uint32_t y);
char3 rsAllocationVLoadX_char3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
char4 rsAllocationVLoadX_char4(rs_allocation a, uint32_t x);
char4 rsAllocationVLoadX_char4(rs_allocation a, uint32_t x, uint32_t y);
char4 rsAllocationVLoadX_char4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
double2 rsAllocationVLoadX_double2(rs_allocation a, uint32_t x);
double2 rsAllocationVLoadX_double2(rs_allocation a, uint32_t x, uint32_t y);
```

Added in API level 22



|                                                                                                       |                       |
|-------------------------------------------------------------------------------------------------------|-----------------------|
| <code>uchar3 rsAllocationVLoadX_uchar3(rs_allocation a, uint32_t x, uint32_t y);</code>               | Added in API level 22 |
| <code>uchar3 rsAllocationVLoadX_uchar3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 22 |
| <code>uchar4 rsAllocationVLoadX_uchar4(rs_allocation a, uint32_t x);</code>                           | Added in API level 22 |
| <code>uchar4 rsAllocationVLoadX_uchar4(rs_allocation a, uint32_t x, uint32_t y);</code>               | Added in API level 22 |
| <code>uchar4 rsAllocationVLoadX_uchar4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 22 |
| <code>uint2 rsAllocationVLoadX_uint2(rs_allocation a, uint32_t x);</code>                             | Added in API level 22 |
| <code>uint2 rsAllocationVLoadX_uint2(rs_allocation a, uint32_t x, uint32_t y);</code>                 | Added in API level 22 |
| <code>uint2 rsAllocationVLoadX_uint2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 22 |
| <code>uint3 rsAllocationVLoadX_uint3(rs_allocation a, uint32_t x);</code>                             | Added in API level 22 |
| <code>uint3 rsAllocationVLoadX_uint3(rs_allocation a, uint32_t x, uint32_t y);</code>                 | Added in API level 22 |
| <code>uint3 rsAllocationVLoadX_uint3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 22 |
| <code>uint4 rsAllocationVLoadX_uint4(rs_allocation a, uint32_t x);</code>                             | Added in API level 22 |
| <code>uint4 rsAllocationVLoadX_uint4(rs_allocation a, uint32_t x, uint32_t y);</code>                 | Added in API level 22 |
| <code>uint4 rsAllocationVLoadX_uint4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 22 |
| <code>ulong2 rsAllocationVLoadX_ulong2(rs_allocation a, uint32_t x);</code>                           | Added in API level 22 |
| <code>ulong2 rsAllocationVLoadX_ulong2(rs_allocation a, uint32_t x, uint32_t y);</code>               | Added in API level 22 |
| <code>ulong2 rsAllocationVLoadX_ulong2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 22 |
| <code>ulong3 rsAllocationVLoadX_ulong3(rs_allocation a, uint32_t x);</code>                           | Added in API level 22 |
| <code>ulong3 rsAllocationVLoadX_ulong3(rs_allocation a, uint32_t x, uint32_t y);</code>               | Added in API level 22 |
| <code>ulong3 rsAllocationVLoadX_ulong3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 22 |
| <code>ulong4 rsAllocationVLoadX_ulong4(rs_allocation a, uint32_t x);</code>                           | Added in API level 22 |
| <code>ulong4 rsAllocationVLoadX_ulong4(rs_allocation a, uint32_t x, uint32_t y);</code>               | Added in API level 22 |
| <code>ulong4 rsAllocationVLoadX_ulong4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 22 |
| <code>ushort2 rsAllocationVLoadX_ushort2(rs_allocation a, uint32_t x);</code>                         | Added in API level 22 |
| <code>ushort2 rsAllocationVLoadX_ushort2(rs_allocation a, uint32_t x, uint32_t y);</code>             | Added in API level 22 |
| <code>ushort2 rsAllocationVLoadX_ushort2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code> | Added in API level 22 |
| <code>ushort3 rsAllocationVLoadX_ushort3(rs_allocation a, uint32_t x);</code>                         | Added in API level 22 |
| <code>ushort3 rsAllocationVLoadX_ushort3(rs_allocation a, uint32_t x, uint32_t y);</code>             | Added in API level 22 |
| <code>ushort3 rsAllocationVLoadX_ushort3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code> | Added in API level 22 |
| <code>ushort4 rsAllocationVLoadX_ushort4(rs_allocation a, uint32_t x);</code>                         | Added in API level 22 |
| <code>ushort4 rsAllocationVLoadX_ushort4(rs_allocation a, uint32_t x, uint32_t y);</code>             | Added in API level 22 |
| <code>ushort4 rsAllocationVLoadX_ushort4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);</code> | Added in API level 22 |

## Parameters

- `a` Allocation to get the data from.
- `x` X offset in the allocation of the first cell to be copied from.
- `y` Y offset in the allocation of the first cell to be copied from.
- `z` Z offset in the allocation of the first cell to be copied from.

This function returns a vector composed of successive cells of the allocation. It assumes that the allocation contains scalars.

The "X" in the name indicates that successive values are extracted by increasing the X index. There are currently no functions to get successive values incrementing other dimensions. Use multiple calls to `rsGetElementAt()` instead.

For example, when calling `rsAllocationVLoadX_int4(a, 20, 30)`, an int4 composed of `a[20, 30], a[21, 30], a[22, 30],` and `a[23, 30]` is returned.

When retrieving from a three dimensional allocations, use the x, y, z variant. Similarly, use the x, y variant for two dimensional allocations and x for the mono dimensional allocations.

For efficiency, this function does not validate the inputs. Trying to wrap the X index, exceeding the size of the allocation, or using indices incompatible with the dimensionality of the allocation yields undefined results.

See also `rsAllocationVStoreX()`.

`rsAllocationVStoreX` : Store a vector into an allocation of scalars



## Parameters

- a Allocation to store the data into.
- val Value to be stored.
- x X offset in the allocation of the first cell to be copied into.
- y Y offset in the allocation of the first cell to be copied into.
- z Z offset in the allocation of the first cell to be copied into.

This function stores the entries of a vector into successive cells of an allocation. It assumes that the allocation contains scalars.

The "X" in the name indicates that successive values are stored by increasing the X index. There are currently no functions to store successive values incrementing other dimensions. Use multiple calls to rsSetElementAt() instead.

For example, when calling rsAllocationVStoreX\_int3(a, v, 20, 30), v.x is stored at a[20, 30], v.y at a[21, 30], and v.z at a[22, 30].

When storing into a three dimensional allocations, use the x, y, z variant. Similarly, use the x, y variant for two dimensional allocations and x for the mono dimensional allocations.

For efficiency, this function does not validate the inputs. Trying to wrap the X index, exceeding the size of the allocation, or using indices incompatible with the dimensionality of the allocation yields undefined results.

See also [rsAllocationVLoadX\(\)](#).

## rsGetElementAt : Return a cell from an allocation

```
char rsGetElementAt_char(rs_allocation a, uint32_t x);
char rsGetElementAt_char(rs_allocation a, uint32_t x, uint32_t y);
char rsGetElementAt_char(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

char2 rsGetElementAt_char2(rs_allocation a, uint32_t x);
char2 rsGetElementAt_char2(rs_allocation a, uint32_t x, uint32_t y);
char2 rsGetElementAt_char2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

char3 rsGetElementAt_char3(rs_allocation a, uint32_t x);
char3 rsGetElementAt_char3(rs_allocation a, uint32_t x, uint32_t y);
char3 rsGetElementAt_char3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

char4 rsGetElementAt_char4(rs_allocation a, uint32_t x);
char4 rsGetElementAt_char4(rs_allocation a, uint32_t x, uint32_t y);
char4 rsGetElementAt_char4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

const void* rsGetElementAt(rs_allocation a, uint32_t x);
const void* rsGetElementAt(rs_allocation a, uint32_t x, uint32_t y);
const void* rsGetElementAt(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

double rsGetElementAt_double(rs_allocation a, uint32_t x);
double rsGetElementAt_double(rs_allocation a, uint32_t x, uint32_t y);
double rsGetElementAt_double(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

double2 rsGetElementAt_double2(rs_allocation a, uint32_t x);
double2 rsGetElementAt_double2(rs_allocation a, uint32_t x, uint32_t y);
double2 rsGetElementAt_double2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

double3 rsGetElementAt_double3(rs_allocation a, uint32_t x);
double3 rsGetElementAt_double3(rs_allocation a, uint32_t x, uint32_t y);
double3 rsGetElementAt_double3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

double4 rsGetElementAt_double4(rs_allocation a, uint32_t x);
double4 rsGetElementAt_double4(rs_allocation a, uint32_t x, uint32_t y);
double4 rsGetElementAt_double4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

float rsGetElementAt_float(rs_allocation a, uint32_t x);
```

```
float rsGetElementAt_float(rs_allocation a, uint32_t x, uint32_t y);

float rsGetElementAt_float(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
float2 rsGetElementAt_float2(rs_allocation a, uint32_t x);
float2 rsGetElementAt_float2(rs_allocation a, uint32_t x, uint32_t y);
float2 rsGetElementAt_float2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
float3 rsGetElementAt_float3(rs_allocation a, uint32_t x);
float3 rsGetElementAt_float3(rs_allocation a, uint32_t x, uint32_t y);
float3 rsGetElementAt_float3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
float4 rsGetElementAt_float4(rs_allocation a, uint32_t x);
float4 rsGetElementAt_float4(rs_allocation a, uint32_t x, uint32_t y);
float4 rsGetElementAt_float4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

half rsGetElementAt_half(rs_allocation a, uint32_t x); Added in API level 23
half rsGetElementAt_half(rs_allocation a, uint32_t x, uint32_t y); Added in API level 23
half rsGetElementAt_half(rs_allocation a, uint32_t x, uint32_t y, uint32_t z); Added in API level 23
half2 rsGetElementAt_half2(rs_allocation a, uint32_t x); Added in API level 23
half2 rsGetElementAt_half2(rs_allocation a, uint32_t x, uint32_t y); Added in API level 23
half2 rsGetElementAt_half2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z); Added in API level 23
half3 rsGetElementAt_half3(rs_allocation a, uint32_t x); Added in API level 23
half3 rsGetElementAt_half3(rs_allocation a, uint32_t x, uint32_t y); Added in API level 23
half3 rsGetElementAt_half3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z); Added in API level 23
half4 rsGetElementAt_half4(rs_allocation a, uint32_t x); Added in API level 23
half4 rsGetElementAt_half4(rs_allocation a, uint32_t x, uint32_t y); Added in API level 23
half4 rsGetElementAt_half4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z); Added in API level 23

int rsGetElementAt_int(rs_allocation a, uint32_t x);
int rsGetElementAt_int(rs_allocation a, uint32_t x, uint32_t y);
int rsGetElementAt_int(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
int2 rsGetElementAt_int2(rs_allocation a, uint32_t x);
int2 rsGetElementAt_int2(rs_allocation a, uint32_t x, uint32_t y);
int2 rsGetElementAt_int2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
int3 rsGetElementAt_int3(rs_allocation a, uint32_t x);
int3 rsGetElementAt_int3(rs_allocation a, uint32_t x, uint32_t y);
int3 rsGetElementAt_int3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
int4 rsGetElementAt_int4(rs_allocation a, uint32_t x);
int4 rsGetElementAt_int4(rs_allocation a, uint32_t x, uint32_t y);
int4 rsGetElementAt_int4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

long rsGetElementAt_long(rs_allocation a, uint32_t x);
long rsGetElementAt_long(rs_allocation a, uint32_t x, uint32_t y);
long rsGetElementAt_long(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
long2 rsGetElementAt_long2(rs_allocation a, uint32_t x);
long2 rsGetElementAt_long2(rs_allocation a, uint32_t x, uint32_t y);
long2 rsGetElementAt_long2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
long3 rsGetElementAt_long3(rs_allocation a, uint32_t x);
long3 rsGetElementAt_long3(rs_allocation a, uint32_t x, uint32_t y);
long3 rsGetElementAt_long3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
long4 rsGetElementAt_long4(rs_allocation a, uint32_t x);
long4 rsGetElementAt_long4(rs_allocation a, uint32_t x, uint32_t y);
```

```
long4 rsGetElementAt_long4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
short rsGetElementAt_short(rs_allocation a, uint32_t x);

short rsGetElementAt_short(rs_allocation a, uint32_t x, uint32_t y);
short rsGetElementAt_short(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

short2 rsGetElementAt_short2(rs_allocation a, uint32_t x);
short2 rsGetElementAt_short2(rs_allocation a, uint32_t x, uint32_t y);
short2 rsGetElementAt_short2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

short3 rsGetElementAt_short3(rs_allocation a, uint32_t x);
short3 rsGetElementAt_short3(rs_allocation a, uint32_t x, uint32_t y);
short3 rsGetElementAt_short3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

short4 rsGetElementAt_short4(rs_allocation a, uint32_t x);
short4 rsGetElementAt_short4(rs_allocation a, uint32_t x, uint32_t y);
short4 rsGetElementAt_short4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uchar rsGetElementAt_uchar(rs_allocation a, uint32_t x);
uchar rsGetElementAt_uchar(rs_allocation a, uint32_t x, uint32_t y);
uchar rsGetElementAt_uchar(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uchar2 rsGetElementAt_uchar2(rs_allocation a, uint32_t x);
uchar2 rsGetElementAt_uchar2(rs_allocation a, uint32_t x, uint32_t y);
uchar2 rsGetElementAt_uchar2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uchar3 rsGetElementAt_uchar3(rs_allocation a, uint32_t x);
uchar3 rsGetElementAt_uchar3(rs_allocation a, uint32_t x, uint32_t y);
uchar3 rsGetElementAt_uchar3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uchar4 rsGetElementAt_uchar4(rs_allocation a, uint32_t x);
uchar4 rsGetElementAt_uchar4(rs_allocation a, uint32_t x, uint32_t y);
uchar4 rsGetElementAt_uchar4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uint rsGetElementAt_uint(rs_allocation a, uint32_t x);
uint rsGetElementAt_uint(rs_allocation a, uint32_t x, uint32_t y);
uint rsGetElementAt_uint(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uint2 rsGetElementAt_uint2(rs_allocation a, uint32_t x);
uint2 rsGetElementAt_uint2(rs_allocation a, uint32_t x, uint32_t y);
uint2 rsGetElementAt_uint2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uint3 rsGetElementAt_uint3(rs_allocation a, uint32_t x);
uint3 rsGetElementAt_uint3(rs_allocation a, uint32_t x, uint32_t y);
uint3 rsGetElementAt_uint3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

uint4 rsGetElementAt_uint4(rs_allocation a, uint32_t x);
uint4 rsGetElementAt_uint4(rs_allocation a, uint32_t x, uint32_t y);
uint4 rsGetElementAt_uint4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

ulong rsGetElementAt_ulong(rs_allocation a, uint32_t x);
ulong rsGetElementAt_ulong(rs_allocation a, uint32_t x, uint32_t y);
ulong rsGetElementAt_ulong(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

ulong2 rsGetElementAt_ulong2(rs_allocation a, uint32_t x);
ulong2 rsGetElementAt_ulong2(rs_allocation a, uint32_t x, uint32_t y);
ulong2 rsGetElementAt_ulong2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

ulong3 rsGetElementAt_ulong3(rs_allocation a, uint32_t x);
ulong3 rsGetElementAt_ulong3(rs_allocation a, uint32_t x, uint32_t y);
ulong3 rsGetElementAt_ulong3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);

ulong4 rsGetElementAt_ulong4(rs_allocation a, uint32_t x);
```

```
ulong4 rsGetElementAt_ulong4(rs_allocation a, uint32_t x, uint32_t y);
ulong4 rsGetElementAt_ulong4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
ushort rsGetElementAt_ushort(rs_allocation a, uint32_t x);
ushort rsGetElementAt_ushort(rs_allocation a, uint32_t x, uint32_t y);
ushort rsGetElementAt_ushort(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
ushort2 rsGetElementAt_ushort2(rs_allocation a, uint32_t x);
ushort2 rsGetElementAt_ushort2(rs_allocation a, uint32_t x, uint32_t y);
ushort2 rsGetElementAt_ushort2(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
ushort3 rsGetElementAt_ushort3(rs_allocation a, uint32_t x);
ushort3 rsGetElementAt_ushort3(rs_allocation a, uint32_t x, uint32_t y);
ushort3 rsGetElementAt_ushort3(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
ushort4 rsGetElementAt_ushort4(rs_allocation a, uint32_t x);
ushort4 rsGetElementAt_ushort4(rs_allocation a, uint32_t x, uint32_t y);
ushort4 rsGetElementAt_ushort4(rs_allocation a, uint32_t x, uint32_t y, uint32_t z);
```

This function extracts a single cell from an allocation.

When retrieving from a three dimensional allocations, use the x, y, z variant. Similarly, use the x, y variant for two dimensional allocations and x for the mono dimensional allocations.

This function has two styles. One returns the address of the value using a void\*, the other returns the actual value, e.g. rsGetElementAt() vs. rsGetElementAt\_int4(). For primitive types, always use the latter as it is more efficient.

### rsGetElementAtYuv\_uchar\_U : Get the U component of an allocation of YUVs

uchar rsGetElementAtYuv\_uchar\_U(rs\_allocation a, uint32\_t x, uint32\_t y);    Added in API level 18

Extracts the U component of a single YUV value from a 2D allocation of YUVs.

Inside an allocation, Y, U, and V components may be stored if different planes and at different resolutions. The x, y coordinates provided here are in the dimensions of the Y plane.

See [rsGetElementAtYuv\\_uchar\\_Y\(\)](#).

### rsGetElementAtYuv\_uchar\_V : Get the V component of an allocation of YUVs

uchar rsGetElementAtYuv\_uchar\_V(rs\_allocation a, uint32\_t x, uint32\_t y);    Added in API level 18

Extracts the V component of a single YUV value from a 2D allocation of YUVs.

Inside an allocation, Y, U, and V components may be stored if different planes and at different resolutions. The x, y coordinates provided here are in the dimensions of the Y plane.

See [rsGetElementAtYuv\\_uchar\\_Y\(\)](#).

### rsGetElementAtYuv\_uchar\_Y : Get the Y component of an allocation of YUVs

uchar rsGetElementAtYuv\_uchar\_Y(rs\_allocation a, uint32\_t x, uint32\_t y);    Added in API level 18

Extracts the Y component of a single YUV value from a 2D allocation of YUVs.

Inside an allocation, Y, U, and V components may be stored if different planes and at different resolutions. The x, y coordinates provided here are in the dimensions of the Y plane.

See [rsGetElementAtYuv\\_uchar\\_U\(\)](#) and [rsGetElementAtYuv\\_uchar\\_V\(\)](#).

### rsSample : Sample a value from a texture allocation

float4 rsSample(rs\_allocation a, rs\_sampler s, float location);    Added in API level 16

`float4 rsSample(rs_allocation a, rs_sampler s, float location, float lod);` Added in API level 16

`float4 rsSample(rs_allocation a, rs_sampler s, float2 location);` Added in API level 16

`float4 rsSample(rs_allocation a, rs_sampler s, float2 location, float lod);` Added in API level 16

## Parameters

*a* Allocation to sample from.

*s* Sampler state.

*location* Location to sample from.

*lod* Mip level to sample from, for fractional values mip levels will be interpolated if RS\_SAMPLER\_LINEAR\_MIP\_LINEAR is used.

Fetches a value from a texture allocation in a way described by the sampler.

If your allocation is 1D, use the variant with float for location. For 2D, use the float2 variant.

See [android.renderscript.Sampler](#) for more details.

## rsSetElementAt : Set a cell of an allocation

|                                                                                                             |                       |
|-------------------------------------------------------------------------------------------------------------|-----------------------|
| <code>void rsSetElementAt(rs_allocation a, void* ptr, uint32_t x);</code>                                   | Added in API level 18 |
| <code>void rsSetElementAt(rs_allocation a, void* ptr, uint32_t x, uint32_t y);</code>                       | Added in API level 18 |
| <code>void rsSetElementAt_char(rs_allocation a, char val, uint32_t x);</code>                               | Added in API level 18 |
| <code>void rsSetElementAt_char(rs_allocation a, char val, uint32_t x, uint32_t y);</code>                   | Added in API level 18 |
| <code>void rsSetElementAt_char(rs_allocation a, char val, uint32_t x, uint32_t y, uint32_t z);</code>       | Added in API level 18 |
| <code>void rsSetElementAt_char2(rs_allocation a, char2 val, uint32_t x);</code>                             | Added in API level 18 |
| <code>void rsSetElementAt_char2(rs_allocation a, char2 val, uint32_t x, uint32_t y);</code>                 | Added in API level 18 |
| <code>void rsSetElementAt_char2(rs_allocation a, char2 val, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 18 |
| <code>void rsSetElementAt_char3(rs_allocation a, char3 val, uint32_t x);</code>                             | Added in API level 18 |
| <code>void rsSetElementAt_char3(rs_allocation a, char3 val, uint32_t x, uint32_t y);</code>                 | Added in API level 18 |
| <code>void rsSetElementAt_char3(rs_allocation a, char3 val, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 18 |
| <code>void rsSetElementAt_char4(rs_allocation a, char4 val, uint32_t x);</code>                             | Added in API level 18 |
| <code>void rsSetElementAt_char4(rs_allocation a, char4 val, uint32_t x, uint32_t y);</code>                 | Added in API level 18 |
| <code>void rsSetElementAt_char4(rs_allocation a, char4 val, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 18 |
| <code>void rsSetElementAt_double(rs_allocation a, double val, uint32_t x);</code>                           | Added in API level 18 |
| <code>void rsSetElementAt_double(rs_allocation a, double val, uint32_t x, uint32_t y);</code>               | Added in API level 18 |
| <code>void rsSetElementAt_double(rs_allocation a, double val, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 18 |
| <code>void rsSetElementAt_double2(rs_allocation a, double2 val, uint32_t x);</code>                         | Added in API level 18 |
| <code>void rsSetElementAt_double2(rs_allocation a, double2 val, uint32_t x, uint32_t y);</code>             | Added in API level 18 |
| <code>void rsSetElementAt_double2(rs_allocation a, double2 val, uint32_t x, uint32_t y, uint32_t z);</code> | Added in API level 18 |
| <code>void rsSetElementAt_double3(rs_allocation a, double3 val, uint32_t x);</code>                         | Added in API level 18 |
| <code>void rsSetElementAt_double3(rs_allocation a, double3 val, uint32_t x, uint32_t y);</code>             | Added in API level 18 |
| <code>void rsSetElementAt_double3(rs_allocation a, double3 val, uint32_t x, uint32_t y, uint32_t z);</code> | Added in API level 18 |
| <code>void rsSetElementAt_double4(rs_allocation a, double4 val, uint32_t x);</code>                         | Added in API level 18 |
| <code>void rsSetElementAt_double4(rs_allocation a, double4 val, uint32_t x, uint32_t y);</code>             | Added in API level 18 |
| <code>void rsSetElementAt_double4(rs_allocation a, double4 val, uint32_t x, uint32_t y, uint32_t z);</code> | Added in API level 18 |
| <code>void rsSetElementAt_float(rs_allocation a, float val, uint32_t x);</code>                             | Added in API level 18 |
| <code>void rsSetElementAt_float(rs_allocation a, float val, uint32_t x, uint32_t y);</code>                 | Added in API level 18 |
| <code>void rsSetElementAt_float(rs_allocation a, float val, uint32_t x, uint32_t y, uint32_t z);</code>     | Added in API level 18 |
| <code>void rsSetElementAt_float2(rs_allocation a, float2 val, uint32_t x);</code>                           | Added in API level 18 |
| <code>void rsSetElementAt_float2(rs_allocation a, float2 val, uint32_t x, uint32_t y);</code>               | Added in API level 18 |
| <code>void rsSetElementAt_float2(rs_allocation a, float2 val, uint32_t x, uint32_t y, uint32_t z);</code>   | Added in API level 18 |





|                                                                                                |                       |
|------------------------------------------------------------------------------------------------|-----------------------|
| void rsSetElementAt_ushort(rs_allocation a, ushort val, uint32_t x, uint32_t y, uint32_t z);   | Added in API level 18 |
| void rsSetElementAt_ushort2(rs_allocation a, ushort2 val, uint32_t x);                         | Added in API level 18 |
| void rsSetElementAt_ushort2(rs_allocation a, ushort2 val, uint32_t x, uint32_t y);             | Added in API level 18 |
| void rsSetElementAt_ushort2(rs_allocation a, ushort2 val, uint32_t x, uint32_t y, uint32_t z); | Added in API level 18 |
| void rsSetElementAt_ushort3(rs_allocation a, ushort3 val, uint32_t x);                         | Added in API level 18 |
| void rsSetElementAt_ushort3(rs_allocation a, ushort3 val, uint32_t x, uint32_t y);             | Added in API level 18 |
| void rsSetElementAt_ushort3(rs_allocation a, ushort3 val, uint32_t x, uint32_t y, uint32_t z); | Added in API level 18 |
| void rsSetElementAt_ushort4(rs_allocation a, ushort4 val, uint32_t x);                         | Added in API level 18 |
| void rsSetElementAt_ushort4(rs_allocation a, ushort4 val, uint32_t x, uint32_t y);             | Added in API level 18 |
| void rsSetElementAt_ushort4(rs_allocation a, ushort4 val, uint32_t x, uint32_t y, uint32_t z); | Added in API level 18 |

This function stores a value into a single cell of an allocation.

When storing into a three dimensional allocations, use the x, y, z variant. Similarly, use the x, y variant for two dimensional allocations and x for the mono dimensional allocations.

This function has two styles. One passes the value to be stored using a void\*, the other has the actual value as an argument, e.g. rsSetElementAt() vs. rsSetElementAt\_int4(). For primitive types, always use the latter as it is more efficient.

See also [rsGetElementAt\(\)](#).

# RenderScript Object Characteristics Functions

## Overview

The functions below can be used to query the characteristics of an Allocation, Element, or Sampler object. These objects are created from Java. You can't create them from a script.

### Allocations:

Allocations are the primary method used to pass data to and from RenderScript kernels.

They are a structured collection of cells that can be used to store bitmaps, textures, arbitrary data points, etc.

This collection of cells may have many dimensions (X, Y, Z, Array0, Array1, Array2, Array3), faces (for cubemaps), and level of details (for mipmapping).

See the [android.renderscript.Allocation](#) for details on how to create Allocations.

### Elements:

The term "element" is used a bit ambiguously in RenderScript, as both type information for the cells of an Allocation and the instantiation of that type. For example:

- [rs\\_element](#) is a handle to a type specification, and
- In functions like [rsGetElementAt\(\)](#), "element" means the instantiation of the type, i.e. a cell of an Allocation.

The functions below let you query the characteristics of the type specification.

An Element can specify a simple data types as found in C, e.g. an integer, float, or boolean. It can also specify a handle to a RenderScript object. See [rs\\_data\\_type](#) for a list of basic types.

Elements can specify fixed size vector (of size 2, 3, or 4) versions of the basic types. Elements can be grouped together into complex Elements, creating the equivalent of C structure definitions.

Elements can also have a kind, which is semantic information used to interpret pixel data. See [rs\\_data\\_kind](#).

When creating Allocations of common elements, you can simply use one of the many predefined Elements like [F32\\_2](#).

To create complex Elements, use the [Element.Builder](#) Java class.

### Samplers:

Samplers objects define how Allocations can be read as structure within a kernel. See [android.renderscript.S](#).

## Summary

| Functions                               |                                |
|-----------------------------------------|--------------------------------|
| <a href="#">rsAllocationGetDimFaces</a> | Presence of more than one face |
| <a href="#">rsAllocationGetDimLOD</a>   | Presence of levels of detail   |
| <a href="#">rsAllocationGetDimX</a>     | Size of the X dimension        |
| <a href="#">rsAllocationGetDimY</a>     | Size of the Y dimension        |
| <a href="#">rsAllocationGetDimZ</a>     | Size of the Z dimension        |

|                                                   |                                                         |
|---------------------------------------------------|---------------------------------------------------------|
| <a href="#">rsAllocationGetElement</a>            | Get the object that describes the cell of an Allocation |
| <a href="#">rsClearObject</a>                     | Release an object                                       |
| <a href="#">rsElementGetBytesSize</a>             | Size of an Element                                      |
| <a href="#">rsElementGetDataKind</a>              | Kind of an Element                                      |
| <a href="#">rsElementGetType</a>                  | Data type of an Element                                 |
| <a href="#">rsElementGetSubElement</a>            | Sub-element of a complex Element                        |
| <a href="#">rsElementGetSubElementArraySize</a>   | Array size of a sub-element of a complex Element        |
| <a href="#">rsElementGetSubElementCount</a>       | Number of sub-elements                                  |
| <a href="#">rsElementGetSubElementName</a>        | Name of a sub-element                                   |
| <a href="#">rsElementGetSubElementNameLength</a>  | Length of the name of a sub-element                     |
| <a href="#">rsElementGetSubElementOffsetBytes</a> | Offset of the instantiated sub-element                  |
| <a href="#">rsElementGetVectorSize</a>            | Vector size of the Element                              |
| <a href="#">rsIsObject</a>                        | Check for an empty handle                               |
| <a href="#">rsSamplerGetAnisotropy</a>            | Anisotropy of the Sampler                               |
| <a href="#">rsSamplerGetMagnification</a>         | Sampler magnification value                             |
| <a href="#">rsSamplerGetMinification</a>          | Sampler minification value                              |
| <a href="#">rsSamplerGetWrapS</a>                 | Sampler wrap S value                                    |
| <a href="#">rsSamplerGetWrapT</a>                 | Sampler wrap T value                                    |

### Deprecated Functions

[rsGetAllocation](#)    **Deprecated.** Return the Allocation for a given pointer

## Functions

### rsAllocationGetDimFaces : Presence of more than one face

`uint32_t rsAllocationGetDimFaces(rs_allocation a);`

#### Returns

Returns 1 if more than one face is present, 0 otherwise.

If the Allocation is a cubemap, this function returns 1 if there's more than one face present. In all other cases, it returns 0.

Use [rsGetDimHasFaces\(\)](#) to get the dimension of a currently running kernel.

### rsAllocationGetDimLOD : Presence of levels of detail

`uint32_t rsAllocationGetDimLOD(rs_allocation a);`

#### Returns

Returns 1 if more than one LOD is present, 0 otherwise.

Query an Allocation for the presence of more than one Level Of Detail. This is useful for mipmaps.

Use [rsGetDimLod\(\)](#) to get the dimension of a currently running kernel.

### rsAllocationGetDimX : Size of the X dimension

`uint32_t rsAllocationGetDimX(rs_allocation a);`

#### Returns

X dimension of the Allocation.

Returns the size of the X dimension of the Allocation.

Use [rsGetDimX\(\)](#) to get the dimension of a currently running kernel.

### rsAllocationGetDimY : Size of the Y dimension

```
uint32_t rsAllocationGetDimY(rs_allocation a);
```

#### Returns

Y dimension of the Allocation.

Returns the size of the Y dimension of the Allocation. If the Allocation has less than two dimensions, returns 0.

Use [rsGetDimY\(\)](#) to get the dimension of a currently running kernel.

### rsAllocationGetDimZ : Size of the Z dimension

```
uint32_t rsAllocationGetDimZ(rs_allocation a);
```

#### Returns

Z dimension of the Allocation.

Returns the size of the Z dimension of the Allocation. If the Allocation has less than three dimensions, returns 0.

Use [rsGetDimZ\(\)](#) to get the dimension of a currently running kernel.

### rsAllocationGetElement : Get the object that describes the cell of an Allocation

```
rs_element rsAllocationGetElement(rs_allocation a);
```

#### Parameters

a Allocation to get data from.

#### Returns

Element describing Allocation layout.

Get the Element object describing the type, kind, and other characteristics of a cell of an Allocation. See the [rsElement\\*](#) functions below.

### rsClearObject : Release an object

```
void rsClearObject(rs_allocation* dst);
```

```
void rsClearObject(rs_element* dst);
```

```
void rsClearObject(rs_font* dst);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsClearObject(rs_mesh* dst);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsClearObject(rs_program_fragment* dst);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsClearObject(rs_program_raster* dst);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsClearObject(rs_program_store* dst);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsClearObject(rs_program_vertex* dst);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsClearObject(rs_sampler* dst);
```

```
void rsClearObject(rs_script* dst);
```

```
void rsClearObject(rs_type* dst);
```

Tells the run time that this handle will no longer be used to access the the related object. If this was the last handle to that object, resource recovery may happen.

After calling this function, \*dst will be set to an empty handle. See [rsIsObject\(\)](#).

### rsElementGetBytesSize : Size of an Element

---

`uint32_t rsElementGetBytesSize(rs_element e);`    Added in [API level 16](#)

Returns the size in bytes that an instantiation of this Element will occupy.

### rsElementGetDataKind : Kind of an Element

`rs_data_kind rsElementGetDataKind(rs_element e);`    Added in [API level 16](#)

Returns the Element's data kind. This is used to interpret pixel data.

See [rs\\_data\\_kind](#).

### rsElementGetType : Data type of an Element

`rs_data_type rsElementGetType(rs_element e);`    Added in [API level 16](#)

Returns the Element's base data type. This can be a type similar to C/C++ (e.g. RS\_TYPE\_UNSIGNED\_8), a handle (e.g. RS\_TYPE\_ALLOCATION and RS\_TYPE\_ELEMENT), or a more complex numerical type (e.g. RS\_TYPE\_UNSIGNED\_5\_6\_5 and RS\_TYPE\_MATRIX\_4X4). See [rs\\_data\\_type](#).

If the Element describes a vector, this function returns the data type of one of its items. Use [rsElementGetVectorSize](#) to get the size of the vector.

If the Element describes a structure, RS\_TYPE\_NONE is returned. Use the `rsElementGetSub*` functions to explore this complex Element.

### rsElementGetSubElement : Sub-element of a complex Element

`rs_element rsElementGetSubElement(rs_element e, uint32_t index);`    Added in [API level 16](#)

#### Parameters

`e`    Element to query.

`index`    Index of the sub-element to return.

#### Returns

Sub-element at the given index.

For Elements that represents a structure, this function returns the sub-element at the specified index.

If the Element is not a structure or the index is greater or equal to the number of sub-elements, an invalid handle is returned.

### rsElementGetSubElementArraySize : Array size of a sub-element of a complex Element

`uint32_t rsElementGetSubElementArraySize(rs_element e, uint32_t index);`    Added in [API level 16](#)

#### Parameters

`e`    Element to query.

`index`    Index of the sub-element.

#### Returns

Array size of the sub-element.

For complex Elements, sub-elements can be statically sized arrays. This function returns the array size of the sub-element at the index. This sub-element repetition is different than fixed size vectors.

### rsElementGetSubElementCount : Number of sub-elements

`uint32_t rsElementGetSubElementCount(rs_element e);`    Added in [API level 16](#)

#### Parameters

`e`    Element to get data from.

#### Returns

Number of sub-elements.

Elements can be simple, such as an int or a float, or a structure with multiple sub-elements. This function returns zero for simple Elements and the number of sub-elements for complex Elements.

### rsElementGetSubElementName : Name of a sub-element

`uint32_t rsElementGetSubElementName(rs_element e, uint32_t index, char* name, uint32_t nameLength);`    Added in API level 16

#### Parameters

- e*      Element to get data from.
- index*    Index of the sub-element.
- name*     Address of the array to store the name into.
- nameLength* Length of the provided name array.

#### Returns

Number of characters copied, excluding the null terminator.

For complex Elements, this function returns the name of the sub-element at the specified index.

### rsElementGetSubElementNameLength : Length of the name of a sub-element

`uint32_t rsElementGetSubElementNameLength(rs_element e, uint32_t index);`    Added in API level 16

#### Parameters

- e*      Element to get data from.
- index*    Index of the sub-element.

#### Returns

Length of the sub-element name including the null terminator.

For complex Elements, this function returns the length of the name of the sub-element at the specified index.

### rsElementGetSubElementOffsetBytes : Offset of the instantiated sub-element

`uint32_t rsElementGetSubElementOffsetBytes(rs_element e, uint32_t index);`    Added in API level 16

#### Parameters

- e*      Element to get data from.
- index*    Index of the sub-element.

#### Returns

Offset in bytes.

This function returns the relative position of the instantiation of the specified sub-element within the instantiation of the Element.

For example, if the Element describes a 32 bit float followed by a 32 bit integer, the offset return for the first will be 0 and the second 4.

### rsElementGetVectorSize : Vector size of the Element

`uint32_t rsElementGetVectorSize(rs_element e);`    Added in API level 16

#### Parameters

- e*      Element to get data from.

#### Returns

Length of the element vector.

Returns the Element's vector size. If the Element does not represent a vector, 1 is returned.

### rsGetAllocation : Return the Allocation for a given pointer

```
rs_allocation rsGetAllocation(const void* p);
```

**Deprecated.** This function is deprecated and will be removed from the SDK in a future release.

Returns the Allocation for a given pointer. The pointer should point within a valid allocation. The results are undefined if the pointer is not from a valid Allocation.

#### rsIsObject : Check for an empty handle

```
bool rsIsObject(rs_allocation v);
```

```
bool rsIsObject(rs_element v);
```

```
bool rsIsObject(rs_font v);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
bool rsIsObject(rs_mesh v);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
bool rsIsObject(rs_program_fragment v);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
bool rsIsObject(rs_program_raster v);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
bool rsIsObject(rs_program_store v);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
bool rsIsObject(rs_program_vertex v);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
bool rsIsObject(rs_sampler v);
```

```
bool rsIsObject(rs_script v);
```

```
bool rsIsObject(rs_type v);
```

Returns true if the handle contains a non-null reference.

This function does not validate that the internal pointer used in the handle points to an actual valid object; it only checks for null.

This function can be used to check the Element returned by [rsElementGetSubElement\(\)](#) or see if [rsClearObject\(\)](#) has been called on a handle.

#### rsSamplerGetAnisotropy : Anisotropy of the Sampler

```
float rsSamplerGetAnisotropy(rs_sampler s); Added in API level 16
```

Get the Sampler's anisotropy.

See [android.renderscript.S](#).

#### rsSamplerGetMagnification : Sampler magnification value

```
rs_sampler_value rsSamplerGetMagnification(rs_sampler s); Added in API level 16
```

Get the Sampler's magnification value.

See [android.renderscript.S](#).

#### rsSamplerGetMinification : Sampler minification value

```
rs_sampler_value rsSamplerGetMinification(rs_sampler s); Added in API level 16
```

Get the Sampler's minification value.

See [android.renderscript.S](#).

#### rsSamplerGetWrapS : Sampler wrap S value

```
rs_sampler_value rsSamplerGetWrapS(rs_sampler s); Added in API level 16
```

Get the Sampler's wrap S value.

See [android.renderscript.S](#).

#### rsSamplerGetWrapT : Sampler wrap T value

`rs_sampler_value` rsSamplerGetWrapT(`rs_sampler` s);    Added in API level 16

Get the sampler's wrap T value.

See [android.renderscript.S](#).



# RenderScript Kernel Invocation Functions and Types

## Overview

The `rsForEach()` function can be used to invoke the root kernel of a script.

The other functions are used to get the characteristics of the invocation of an executing kernel, like dimensions and current indices. These functions take a `rs_kernel_context` as argument.

## Summary

| Types                               |                                       |
|-------------------------------------|---------------------------------------|
| <code>rs_for_each_strategy_t</code> | Suggested cell processing order       |
| <code>rs_kernel</code>              | Handle to a kernel function           |
| <code>rs_kernel_context</code>      | Handle to a kernel invocation context |
| <code>rs_script_call_t</code>       | Cell iteration information            |

| Functions                         |                                                                             |
|-----------------------------------|-----------------------------------------------------------------------------|
| <code>rsForEach</code>            | Launches a kernel                                                           |
| <code>rsForEachInternal</code>    | (Internal API) Launch a kernel in the current Script (with the slot number) |
| <code>rsForEachWithOptions</code> | Launches a kernel with options                                              |
| <code>rsGetArray0</code>          | Index in the Array0 dimension for the specified kernel context              |
| <code>rsGetArray1</code>          | Index in the Array1 dimension for the specified kernel context              |
| <code>rsGetArray2</code>          | Index in the Array2 dimension for the specified kernel context              |
| <code>rsGetArray3</code>          | Index in the Array3 dimension for the specified kernel context              |
| <code>rsGetDimArray0</code>       | Size of the Array0 dimension for the specified kernel context               |
| <code>rsGetDimArray1</code>       | Size of the Array1 dimension for the specified kernel context               |
| <code>rsGetDimArray2</code>       | Size of the Array2 dimension for the specified kernel context               |
| <code>rsGetDimArray3</code>       | Size of the Array3 dimension for the specified kernel context               |
| <code>rsGetDimHasFaces</code>     | Presence of more than one face for the specified kernel context             |
| <code>rsGetDimLod</code>          | Number of levels of detail for the specified kernel context                 |
| <code>rsGetDimX</code>            | Size of the X dimension for the specified kernel context                    |
| <code>rsGetDimY</code>            | Size of the Y dimension for the specified kernel context                    |
| <code>rsGetDimZ</code>            | Size of the Z dimension for the specified kernel context                    |
| <code>rsGetFace</code>            | Coordinate of the Face for the specified kernel context                     |
| <code>rsGetLod</code>             | Index in the Levels of Detail dimension for the specified kernel context    |

## Types

`rs_for_each_strategy_t`: Suggested cell processing order

## `rs_for_each_strategy_t` : Suggested cell processing order

An enum with the following values:

|                                                   |                                               |
|---------------------------------------------------|-----------------------------------------------|
| <code>RS_FOR_EACH_STRATEGY_SERIAL = 0</code>      | Prefer contiguous memory regions.             |
| <code>RS_FOR_EACH_STRATEGY_DONT_CARE = 1</code>   | No preferences.                               |
| <code>RS_FOR_EACH_STRATEGY_DST_LINEAR = 2</code>  | Prefer DST.                                   |
| <code>RS_FOR_EACH_STRATEGY_TILE_SMALL = 3</code>  | Prefer processing small rectangular regions.  |
| <code>RS_FOR_EACH_STRATEGY_TILE_MEDIUM = 4</code> | Prefer processing medium rectangular regions. |
| <code>RS_FOR_EACH_STRATEGY_TILE_LARGE = 5</code>  | Prefer processing large rectangular regions.  |

This type is used to suggest how the invoked kernel should iterate over the cells of the allocations. This is a hint only. Implementations may not follow the suggestion.

This specification can help the caching behavior of the running kernel, e.g. the cache locality when the processing is distributed over multiple cores.

## `rs_kernel` : Handle to a kernel function

A typedef of: `void*`    Added in [API level 24](#)

An opaque type for a function that is defined with the `kernel` attribute. A value of this type can be used in a `rsForEach` call to launch a kernel.

## `rs_kernel_context` : Handle to a kernel invocation context

A typedef of: `const struct rs_kernel_context_t *`    Added in [API level 23](#)

The kernel context contains common characteristics of the allocations being iterated over, like dimensions. It also contains rarely used indices of the currently processed cell, like the `Array0` index or the current level of detail.

You can access the kernel context by adding a special parameter named "context" of type `rs_kernel_context` to your kernel function. See `rsGetDimX()` and `rsGetArray0()` for examples.

## `rs_script_call_t` : Cell iteration information

A structure with the following fields:

|                                              |                                                                              |
|----------------------------------------------|------------------------------------------------------------------------------|
| <code>rs_for_each_strategy_t strategy</code> | Currently ignored. In the future, will be suggested cell iteration strategy. |
| <code>uint32_t xStart</code>                 | Starting index in the X dimension.                                           |
| <code>uint32_t xEnd</code>                   | Ending index (exclusive) in the X dimension.                                 |
| <code>uint32_t yStart</code>                 | Starting index in the Y dimension.                                           |
| <code>uint32_t yEnd</code>                   | Ending index (exclusive) in the Y dimension.                                 |
| <code>uint32_t zStart</code>                 | Starting index in the Z dimension.                                           |
| <code>uint32_t zEnd</code>                   | Ending index (exclusive) in the Z dimension.                                 |
| <code>uint32_t arrayStart</code>             | Starting index in the <code>Array0</code> dimension.                         |
| <code>uint32_t arrayEnd</code>               | Ending index (exclusive) in the <code>Array0</code> dimension.               |
| <code>uint32_t array1Start</code>            | Starting index in the <code>Array1</code> dimension.                         |
| <code>uint32_t array1End</code>              | Ending index (exclusive) in the <code>Array1</code> dimension.               |
| <code>uint32_t array2Start</code>            | Starting index in the <code>Array2</code> dimension.                         |
| <code>uint32_t array2End</code>              | Ending index (exclusive) in the <code>Array2</code> dimension.               |
| <code>uint32_t array3Start</code>            | Starting index in the <code>Array3</code> dimension.                         |
| <code>uint32_t array3End</code>              | Ending index (exclusive) in the <code>Array3</code> dimension.               |

This structure is used to provide iteration information to a `rsForEach` call. It is currently used to restrict processing to a subset of cells. In future versions, it will also be used to provide hint on how to best iterate over the cells.

The Start fields are inclusive and the End fields are exclusive. E.g. to iterate over cells 4, 5, 6, and 7 in the X dimension, set xStart to 4 and xEnd to 8.

## Functions

### rsForEach : Launches a kernel

|                                                                                                                                                                                                                                     |                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| void rsForEach( <a href="#">rs_kernel</a> kernel, ... ...);                                                                                                                                                                         | Added in <a href="#">API level 24</a>                |
| void rsForEach( <a href="#">rs_script</a> script, <a href="#">rs_allocation</a> input, <a href="#">rs_allocation</a> output);                                                                                                       | <a href="#">API level 14 - 23</a>                    |
| void rsForEach( <a href="#">rs_script</a> script, <a href="#">rs_allocation</a> input, <a href="#">rs_allocation</a> output, const void* userData);                                                                                 | Removed from <a href="#">API level 14 and higher</a> |
| void rsForEach( <a href="#">rs_script</a> script, <a href="#">rs_allocation</a> input, <a href="#">rs_allocation</a> output, const void* userData, const <a href="#">rs_script_call_t*</a> sc);                                     | Removed from <a href="#">API level 14 and higher</a> |
| void rsForEach( <a href="#">rs_script</a> script, <a href="#">rs_allocation</a> input, <a href="#">rs_allocation</a> output, const void* userData, <a href="#">size_t</a> userDataLen);                                             | <a href="#">API level 14 - 20</a>                    |
| void rsForEach( <a href="#">rs_script</a> script, <a href="#">rs_allocation</a> input, <a href="#">rs_allocation</a> output, const void* userData, <a href="#">size_t</a> userDataLen, const <a href="#">rs_script_call_t*</a> sc); | <a href="#">API level 14 - 20</a>                    |

#### Parameters

|                    |                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>script</i>      | Script to call.                                                                                                                     |
| <i>input</i>       | Allocation to source data from.                                                                                                     |
| <i>output</i>      | Allocation to write date into.                                                                                                      |
| <i>userData</i>    | User defined data to pass to the script. May be NULL.                                                                               |
| <i>sc</i>          | Extra control information used to select a sub-region of the allocation to be processed or suggest a walking strategy. May be NULL. |
| <i>userDataLen</i> | Size of the userData structure. This will be used to perform a shallow copy of the data if necessary.                               |
| <i>kernel</i>      | Function designator to a function that is defined with the kernel attribute.                                                        |
| ...                | Input and output allocations                                                                                                        |

Runs the kernel over zero or more input allocations. They are passed after the [rs\\_kernel](#) argument. If the specified kernel returns a value, an output allocation must be specified as the last argument. All input allocations, and the output allocation if it exists, must have the same dimensions.

This is a synchronous function. A call to this function only returns after all the work has completed for all cells of the input allocations. If the kernel function returns any value, the call waits until all results have been written to the output allocation.

Up to API level 23, the kernel is implicitly specified as the kernel named "root" in the specified script, and only a single input allocation can be used. Starting in API level 24, an arbitrary kernel function can be used, as specified by the kernel argument. The script argument is removed. The kernel must be defined in the current script. In addition, more than one input can be used.

E.g.

```
float __attribute__((kernel)) square(float a) {
 return a * a;
}

void compute(rs_allocation ain, rs_allocation aout) {
 rsForEach(square, ain, aout);
}
```

### rsForEachInternal : (Internal API) Launch a kernel in the current Script (with the slot number)

void rsForEachInternal(int slot, [rs\\_script\\_call\\_t\\*](#) options, int hasOutput, int numInputs, [rs\\_allocation\\*](#) allocs);    Added in [API level 24](#)

#### Parameters

*slot*

*options*

*hasOutput* Indicates whether the kernel generates output

*numInputs* Number of input allocations

*allocs* Input and output allocations

Internal API to launch a kernel.

## rsForEachWithOptions : Launches a kernel with options

void rsForEachWithOptions(rs\_kernel kernel, rs\_script\_call\_t\* options, ... ...);    Added in [API level 24](#)

### Parameters

*kernel* Function designator to a function that is defined with the kernel attribute.

*options* Launch options

... Input and output allocations

Launches kernel in a way similar to [rsForEach](#). However, instead of processing all cells in the input, this function only processes cells in the subspace of the index space specified in options. With the index space explicitly specified by options, no input or output allocation is required for a kernel launch using this API. If allocations are passed in, they must match the number of arguments and return value expected by the kernel function. The output allocation is present if and only if the kernel has a non-void return value.

E.g.,

```
rs_script_call_t opts = {0};
opts.xStart = 0;
opts.xEnd = dimX;
opts.yStart = 0;
opts.yEnd = dimY / 2;
rsForEachWithOptions(foo, &opts, out, out);
```

## rsGetArray0 : Index in the Array0 dimension for the specified kernel context

uint32\_t rsGetArray0(rs\_kernel\_context context);    Added in [API level 23](#)

Returns the index in the Array0 dimension of the cell being processed, as specified by the supplied kernel context.

The kernel context contains common characteristics of the allocations being iterated over and rarely used indices, like the Array0 index.

You can access the kernel context by adding a special parameter named "context" of type rs\_kernel\_context to your kernel function. E.g.

```
short RS_KERNEL myKernel(short value, uint32_t x, rs_kernel_context context) {
 // The current index in the common x, y, z dimensions are accessed by
 // adding these variables as arguments. For the more rarely used indices
 // to the other dimensions, extract them from the kernel context:
 uint32_t index_a0 = rsGetArray0(context);
 //...
}
```

This function returns 0 if the Array0 dimension is not present.

## rsGetArray1 : Index in the Array1 dimension for the specified kernel context

uint32\_t rsGetArray1(rs\_kernel\_context context);    Added in [API level 23](#)

Returns the index in the Array1 dimension of the cell being processed, as specified by the supplied kernel context. See [rsGetArray0\(\)](#) for an explanation of the context.

Returns 0 if the Array1 dimension is not present.

## rsGetArray2 : Index in the Array2 dimension for the specified kernel context

`uint32_t rsGetArray2(rs_kernel_context context);` Added in API level 23

Returns the index in the Array2 dimension of the cell being processed, as specified by the supplied kernel context. See [rsGetArray0\(\)](#) for an explanation of the context.

Returns 0 if the Array2 dimension is not present.

#### rsGetArray3 : Index in the Array3 dimension for the specified kernel context

`uint32_t rsGetArray3(rs_kernel_context context);` Added in API level 23

Returns the index in the Array3 dimension of the cell being processed, as specified by the supplied kernel context. See [rsGetArray0\(\)](#) for an explanation of the context.

Returns 0 if the Array3 dimension is not present.

#### rsGetDimArray0 : Size of the Array0 dimension for the specified kernel context

`uint32_t rsGetDimArray0(rs_kernel_context context);` Added in API level 23

Returns the size of the Array0 dimension for the specified kernel context. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if the Array0 dimension is not present.

#### rsGetDimArray1 : Size of the Array1 dimension for the specified kernel context

`uint32_t rsGetDimArray1(rs_kernel_context context);` Added in API level 23

Returns the size of the Array1 dimension for the specified kernel context. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if the Array1 dimension is not present.

#### rsGetDimArray2 : Size of the Array2 dimension for the specified kernel context

`uint32_t rsGetDimArray2(rs_kernel_context context);` Added in API level 23

Returns the size of the Array2 dimension for the specified kernel context. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if the Array2 dimension is not present.

#### rsGetDimArray3 : Size of the Array3 dimension for the specified kernel context

`uint32_t rsGetDimArray3(rs_kernel_context context);` Added in API level 23

Returns the size of the Array3 dimension for the specified kernel context. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if the Array3 dimension is not present.

#### rsGetDimHasFaces : Presence of more than one face for the specified kernel context

`bool rsGetDimHasFaces(rs_kernel_context context);` Added in API level 23

##### Returns

Returns true if more than one face is present, false otherwise.

If the kernel is iterating over a cubemap, this function returns true if there's more than one face present. In all other cases, it returns false. See [rsGetDimX\(\)](#) for an explanation of the context.

[rsAllocationGetDimFaces\(\)](#) is similar but returns 0 or 1 instead of a bool.

#### rsGetDimLod : Number of levels of detail for the specified kernel context

`uint32_t rsGetDimLod(rs_kernel_context context);` Added in API level 23

Returns the number of levels of detail for the specified kernel context. This is useful for mipmaps. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if Level of Detail is not used.

[rsAllocationGetDimLOD\(\)](#) is similar but returns 0 or 1 instead the actual number of levels.

#### rsGetDimX : Size of the X dimension for the specified kernel context

`uint32_t rsGetDimX(rs_kernel_context context);` Added in [API level 23](#)

Returns the size of the X dimension for the specified kernel context.

The kernel context contains common characteristics of the allocations being iterated over and rarely used indices, like the Array0 index.

You can access it by adding a special parameter named "context" of type `rs_kernel_context` to your kernel function. E.g.

```
int4 RS_KERNEL myKernel(int4 value, rs_kernel_context context) {
 uint32_t size = rsGetDimX(context); //...
```

To get the dimension of specific allocation, use [rsAllocationGetDimX\(\)](#).

#### rsGetDimY : Size of the Y dimension for the specified kernel context

`uint32_t rsGetDimY(rs_kernel_context context);` Added in [API level 23](#)

Returns the size of the X dimension for the specified kernel context. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if the Y dimension is not present.

To get the dimension of specific allocation, use [rsAllocationGetDimY\(\)](#).

#### rsGetDimZ : Size of the Z dimension for the specified kernel context

`uint32_t rsGetDimZ(rs_kernel_context context);` Added in [API level 23](#)

Returns the size of the Z dimension for the specified kernel context. See [rsGetDimX\(\)](#) for an explanation of the context.

Returns 0 if the Z dimension is not present.

To get the dimension of specific allocation, use [rsAllocationGetDimZ\(\)](#).

#### rsGetFace : Coordinate of the Face for the specified kernel context

`rs_allocation_cubemap_face rsGetFace(rs_kernel_context context);` Added in [API level 23](#)

Returns the face on which the cell being processed is found, as specified by the supplied kernel context. See [rsGetArray0\(\)](#) for an explanation of the context.

Returns `RS_ALLOCATION_CUBEMAP_FACE_POSITIVE_X` if the face dimension is not present.

#### rsGetLod : Index in the Levels of Detail dimension for the specified kernel context

`uint32_t rsGetLod(rs_kernel_context context);` Added in [API level 23](#)

Returns the index in the Levels of Detail dimension of the cell being processed, as specified by the supplied kernel context. See [rsGetArray0\(\)](#) for an explanation of the context.

Returns 0 if the Levels of Detail dimension is not present.

# RenderScript Input/Output Functions

## Overview

These functions are used to:

- Send information to the Java client, and
- Send the processed allocation or receive the next allocation to process.

## Summary

| Functions                              |                                            |
|----------------------------------------|--------------------------------------------|
| <a href="#">rsAllocationIoReceive</a>  | Receive new content from the queue         |
| <a href="#">rsAllocationIoSend</a>     | Send new content to the queue              |
| <a href="#">rsSendToClient</a>         | Send a message to the client, non-blocking |
| <a href="#">rsSendToClientBlocking</a> | Send a message to the client, blocking     |

## Functions

### rsAllocationIoReceive : Receive new content from the queue

`void rsAllocationIoReceive(rs_allocation a);`    Added in API level 16

#### Parameters

`a` Allocation to work on.

Receive a new set of contents from the queue.

This function should not be called from inside a kernel, or from any function that may be called directly or indirectly from a kernel. Doing so would cause a runtime error.

### rsAllocationIoSend : Send new content to the queue

`void rsAllocationIoSend(rs_allocation a);`    Added in API level 16

#### Parameters

`a` Allocation to work on.

Send the contents of the Allocation to the queue.

This function should not be called from inside a kernel, or from any function that may be called directly or indirectly from a kernel. Doing so would cause a runtime error.

### rsSendToClient : Send a message to the client, non-blocking

```
bool rsSendToClient(int cmdID);
bool rsSendToClient(int cmdID, const void* data, uint len);
```

#### Parameters

*cmdID* Application specific data.

*len* Length of the data, in bytes.

Sends a message back to the client. This call does not block. It returns true if the message was sent and false if the message queue is full.

A message ID is required. The data payload is optional.

See [RenderScript.RSMessageHandler](#).

### rsSendToClientBlocking : Send a message to the client, blocking

```
void rsSendToClientBlocking(int cmdID);
```

```
void rsSendToClientBlocking(int cmdID, const void* data, uint len);
```

#### Parameters

*cmdID*

*data* Application specific data.

*len* Length of the data, in bytes.

Sends a message back to the client. This function will block until there is room on the message queue for this message. This function may return before the message was delivered and processed by the client.

A message ID is required. The data payload is optional.

See [RenderScript.RSMessageHandler](#).



# RenderScript Debugging Functions

## Overview

The functions below are intended to be used during application development. They should not be used in shipping applications.

## Summary

### Functions

[rsDebug](#) Log a message and values

## Functions

### rsDebug : Log a message and values

|                                                                            |                       |
|----------------------------------------------------------------------------|-----------------------|
| void rsDebug(const char* message, char a);                                 | Added in API level 17 |
| void rsDebug(const char* message, <a href="#">char2</a> a);                | Added in API level 17 |
| void rsDebug(const char* message, <a href="#">char3</a> a);                | Added in API level 17 |
| void rsDebug(const char* message, <a href="#">char4</a> a);                | Added in API level 17 |
| void rsDebug(const char* message, const <a href="#">rs_matrix2x2</a> * a); |                       |
| void rsDebug(const char* message, const <a href="#">rs_matrix3x3</a> * a); |                       |
| void rsDebug(const char* message, const <a href="#">rs_matrix4x4</a> * a); |                       |
| void rsDebug(const char* message, const void* a);                          |                       |
| void rsDebug(const char* message, double a);                               |                       |
| void rsDebug(const char* message, <a href="#">double2</a> a);              | Added in API level 23 |
| void rsDebug(const char* message, <a href="#">double3</a> a);              | Added in API level 23 |
| void rsDebug(const char* message, <a href="#">double4</a> a);              | Added in API level 23 |
| void rsDebug(const char* message, float a);                                |                       |
| void rsDebug(const char* message, float a, float b);                       |                       |
| void rsDebug(const char* message, float a, float b, float c);              |                       |
| void rsDebug(const char* message, float a, float b, float c, float d);     |                       |
| void rsDebug(const char* message, <a href="#">float2</a> a);               |                       |
| void rsDebug(const char* message, <a href="#">float3</a> a);               |                       |
| void rsDebug(const char* message, <a href="#">float4</a> a);               |                       |
| void rsDebug(const char* message, <a href="#">half</a> a);                 | Added in API level 24 |
| void rsDebug(const char* message, <a href="#">half2</a> a);                | Added in API level 24 |
| void rsDebug(const char* message, <a href="#">half3</a> a);                | Added in API level 24 |
| void rsDebug(const char* message, <a href="#">half4</a> a);                | Added in API level 24 |
| void rsDebug(const char* message, int a);                                  |                       |
| void rsDebug(const char* message, <a href="#">int2</a> a);                 | Added in API level 17 |
| void rsDebug(const char* message, <a href="#">int3</a> a);                 | Added in API level 17 |
| void rsDebug(const char* message, <a href="#">int4</a> a);                 | Added in API level 17 |

|                                                          |                       |
|----------------------------------------------------------|-----------------------|
| void rsDebug(const char* message, long a);               |                       |
| void rsDebug(const char* message, long long a);          |                       |
| void rsDebug(const char* message, long2 a);              | Added in API level 17 |
| void rsDebug(const char* message, long3 a);              | Added in API level 17 |
| void rsDebug(const char* message, long4 a);              | Added in API level 17 |
| void rsDebug(const char* message, short a);              | Added in API level 17 |
| void rsDebug(const char* message, short2 a);             | Added in API level 17 |
| void rsDebug(const char* message, short3 a);             | Added in API level 17 |
| void rsDebug(const char* message, short4 a);             | Added in API level 17 |
| void rsDebug(const char* message, uchar a);              | Added in API level 17 |
| void rsDebug(const char* message, uchar2 a);             | Added in API level 17 |
| void rsDebug(const char* message, uchar3 a);             | Added in API level 17 |
| void rsDebug(const char* message, uchar4 a);             | Added in API level 17 |
| void rsDebug(const char* message, uint a);               |                       |
| void rsDebug(const char* message, uint2 a);              | Added in API level 17 |
| void rsDebug(const char* message, uint3 a);              | Added in API level 17 |
| void rsDebug(const char* message, uint4 a);              | Added in API level 17 |
| void rsDebug(const char* message, ulong a);              |                       |
| void rsDebug(const char* message, ulong2 a);             | Added in API level 17 |
| void rsDebug(const char* message, ulong3 a);             | Added in API level 17 |
| void rsDebug(const char* message, ulong4 a);             | Added in API level 17 |
| void rsDebug(const char* message, unsigned long long a); |                       |
| void rsDebug(const char* message, ushort a);             | Added in API level 17 |
| void rsDebug(const char* message, ushort2 a);            | Added in API level 17 |
| void rsDebug(const char* message, ushort3 a);            | Added in API level 17 |
| void rsDebug(const char* message, ushort4 a);            | Added in API level 17 |

This function prints a message to the standard log, followed by the provided values.

This function is intended for debugging only and should not be used in shipping applications.

# RenderScript Graphics Functions and Types

## Overview

The graphics subsystem of RenderScript was removed at API level 23.

## Summary

| Deprecated Types                    |                                                      |
|-------------------------------------|------------------------------------------------------|
| <a href="#">rs_blend_dst_func</a>   | <b>Deprecated.</b> Blend destination function        |
| <a href="#">rs_blend_src_func</a>   | <b>Deprecated.</b> Blend source function             |
| <a href="#">rs_cull_mode</a>        | <b>Deprecated.</b> Culling mode                      |
| <a href="#">rs_depth_func</a>       | <b>Deprecated.</b> Depth function                    |
| <a href="#">rs_font</a>             | <b>Deprecated.</b> Handle to a Font                  |
| <a href="#">rs_mesh</a>             | <b>Deprecated.</b> Handle to a Mesh                  |
| <a href="#">rs_primitive</a>        | <b>Deprecated.</b> How to interpret mesh vertex data |
| <a href="#">rs_program_fragment</a> | <b>Deprecated.</b> Handle to a ProgramFragment       |
| <a href="#">rs_program_raster</a>   | <b>Deprecated.</b> Handle to a ProgramRaster         |
| <a href="#">rs_program_store</a>    | <b>Deprecated.</b> Handle to a ProgramStore          |
| <a href="#">rs_program_vertex</a>   | <b>Deprecated.</b> Handle to a ProgramVertex         |

| Deprecated Functions                     |                                                               |
|------------------------------------------|---------------------------------------------------------------|
| <a href="#">rsgAllocationSyncAll</a>     | <b>Deprecated.</b> Sync the contents of an allocation         |
| <a href="#">rsgBindColorTarget</a>       | <b>Deprecated.</b> Set the color target                       |
| <a href="#">rsgBindConstant</a>          | <b>Deprecated.</b> Bind a constant allocation                 |
| <a href="#">rsgBindDepthTarget</a>       | <b>Deprecated.</b> Set the depth target                       |
| <a href="#">rsgBindFont</a>              | <b>Deprecated.</b> Bind a font object                         |
| <a href="#">rsgBindProgramFragment</a>   | <b>Deprecated.</b> Bind a ProgramFragment                     |
| <a href="#">rsgBindProgramRaster</a>     | <b>Deprecated.</b> Bind a ProgramRaster                       |
| <a href="#">rsgBindProgramStore</a>      | <b>Deprecated.</b> Bind a ProgramStore                        |
| <a href="#">rsgBindProgramVertex</a>     | <b>Deprecated.</b> Bind a ProgramVertex                       |
| <a href="#">rsgBindSampler</a>           | <b>Deprecated.</b> Bind a sampler                             |
| <a href="#">rsgBindTexture</a>           | <b>Deprecated.</b> Bind a texture allocation                  |
| <a href="#">rsgClearAllRenderTargets</a> | <b>Deprecated.</b> Clear all color and depth targets          |
| <a href="#">rsgClearColor</a>            | <b>Deprecated.</b> Clear the specified color from the surface |
| <a href="#">rsgClearColorTarget</a>      | <b>Deprecated.</b> Clear the color target                     |
| <a href="#">rsgClearDepth</a>            | <b>Deprecated.</b> Clear the depth surface                    |

|                                        |                                                                                         |
|----------------------------------------|-----------------------------------------------------------------------------------------|
| rsgClearDepthTarget                    | <b>Deprecated.</b> Clear the depth target                                               |
| rsgDrawMesh                            | <b>Deprecated.</b> Draw a mesh                                                          |
| rsgDrawQuad                            | <b>Deprecated.</b> Draw a quad                                                          |
| rsgDrawQuadTexCoords                   | <b>Deprecated.</b> Draw a textured quad                                                 |
| rsgDrawRect                            | <b>Deprecated.</b> Draw a rectangle                                                     |
| rsgDrawSpriteScreenspace               | <b>Deprecated.</b> Draw rectangles in screenspace                                       |
| rsgDrawText                            | <b>Deprecated.</b> Draw a text string                                                   |
| rsgFinish                              | <b>Deprecated.</b> End rendering commands                                               |
| rsgFontColor                           | <b>Deprecated.</b> Set the font color                                                   |
| rsgGetHeight                           | <b>Deprecated.</b> Get the surface height                                               |
| rsgGetWidth                            | <b>Deprecated.</b> Get the surface width                                                |
| rsgMeasureText                         | <b>Deprecated.</b> Get the bounding box for a text string                               |
| rsgMeshComputeBoundingBox              | <b>Deprecated.</b> Compute a bounding box                                               |
| rsgMeshGetIndexAllocation              | <b>Deprecated.</b> Return an allocation containing index data                           |
| rsgMeshGetPrimitive                    | <b>Deprecated.</b> Return the primitive                                                 |
| rsgMeshGetPrimitiveCount               | <b>Deprecated.</b> Return the number of index sets                                      |
| rsgMeshGetVertexAllocation             | <b>Deprecated.</b> Return a vertex allocation                                           |
| rsgMeshGetVertexAllocationCount        | <b>Deprecated.</b> Return the number of vertex allocations                              |
| rsgProgramFragmentConstantColor        | <b>Deprecated.</b> Set the constant color for a fixed function emulation program        |
| rsgProgramRasterGetCullMode            | <b>Deprecated.</b> Get program raster cull mode                                         |
| rsgProgramRasterIsPointSpriteEnabled   | <b>Deprecated.</b> Get program raster point sprite state                                |
| rsgProgramStoreGetBlendDstFunc         | <b>Deprecated.</b> Get program store blend destination function                         |
| rsgProgramStoreGetBlendSrcFunc         | <b>Deprecated.</b> Get program store blend source function                              |
| rsgProgramStoreGetDepthFunc            | <b>Deprecated.</b> Get program store depth function                                     |
| rsgProgramStoreIsColorMaskAlphaEnabled | <b>Deprecated.</b> Get program store alpha component color mask                         |
| rsgProgramStoreIsColorMaskBlueEnabled  | <b>Deprecated.</b> Get program store blur component color mask                          |
| rsgProgramStoreIsColorMaskGreenEnabled | <b>Deprecated.</b> Get program store green component color mask                         |
| rsgProgramStoreIsColorMaskRedEnabled   | <b>Deprecated.</b> Get program store red component color mask                           |
| rsgProgramStoreIsDepthMaskEnabled      | <b>Deprecated.</b> Get program store depth mask                                         |
| rsgProgramStoreIsDitherEnabled         | <b>Deprecated.</b> Get program store dither state                                       |
| rsgProgramVertexGetProjectionMatrix    | <b>Deprecated.</b> Get the projection matrix for a fixed function vertex program        |
| rsgProgramVertexLoadModelMatrix        | <b>Deprecated.</b> Load the model matrix for a bound fixed function vertex program      |
| rsgProgramVertexLoadProjectionMatrix   | <b>Deprecated.</b> Load the projection matrix for a bound fixed function vertex program |
| rsgProgramVertexLoadTextureMatrix      | <b>Deprecated.</b> Load the texture matrix for a bound fixed function vertex program    |

## Types

rs\_blend\_dst\_func : Blend destination function

An enum with the following values: When compiling for 32 bits. [API level 16 - 22](#)

```
RS_BLEND_DST_ZERO = 0
RS_BLEND_DST_ONE = 1
RS_BLEND_DST_SRC_COLOR = 2
RS_BLEND_DST_ONE_MINUS_SRC_COLOR = 3
RS_BLEND_DST_SRC_ALPHA = 4
RS_BLEND_DST_ONE_MINUS_SRC_ALPHA = 5
RS_BLEND_DST_DST_ALPHA = 6
RS_BLEND_DST_ONE_MINUS_DST_ALPHA = 7
RS_BLEND_DST_INVALID = 100
```

**Deprecated.** Do not use.

#### rs\_blend\_src\_func : Blend source function

An enum with the following values: When compiling for 32 bits. [API level 16 - 22](#)

```
RS_BLEND_SRC_ZERO = 0
RS_BLEND_SRC_ONE = 1
RS_BLEND_SRC_DST_COLOR = 2
RS_BLEND_SRC_ONE_MINUS_DST_COLOR = 3
RS_BLEND_SRC_SRC_ALPHA = 4
RS_BLEND_SRC_ONE_MINUS_SRC_ALPHA = 5
RS_BLEND_SRC_DST_ALPHA = 6
RS_BLEND_SRC_ONE_MINUS_DST_ALPHA = 7
RS_BLEND_SRC_SRC_ALPHA_SATURATE = 8
RS_BLEND_SRC_INVALID = 100
```

**Deprecated.** Do not use.

#### rs\_cull\_mode : Culling mode

An enum with the following values: When compiling for 32 bits. [API level 16 - 22](#)

```
RS_CULL_BACK = 0
RS_CULL_FRONT = 1
RS_CULL_NONE = 2
RS_CULL_INVALID = 100
```

**Deprecated.** Do not use.

#### rs\_depth\_func : Depth function

An enum with the following values: When compiling for 32 bits. [API level 16 - 22](#)

|                             |                                                                                   |
|-----------------------------|-----------------------------------------------------------------------------------|
| RS_DEPTH_FUNC_ALWAYS = 0    | Always drawn                                                                      |
| RS_DEPTH_FUNC_LESS = 1      | Drawn if the incoming depth value is less than that in the depth buffer           |
| RS_DEPTH_FUNC_EQUAL = 2     | Drawn if the incoming depth value is less or equal to that in the depth buffer    |
| RS_DEPTH_FUNC_GREATER = 3   | Drawn if the incoming depth value is greater than that in the depth buffer        |
| RS_DEPTH_FUNC_GEQUAL = 4    | Drawn if the incoming depth value is greater or equal to that in the depth buffer |
| RS_DEPTH_FUNC_EQUAL = 5     | Drawn if the incoming depth value is equal to that in the depth buffer            |
| RS_DEPTH_FUNC_NOTEQUAL = 6  | Drawn if the incoming depth value is not equal to that in the depth buffer        |
| RS_DEPTH_FUNC_INVALID = 100 | Invalid depth function                                                            |

**Deprecated.** Do not use.

Specifies conditional drawing depending on the comparison of the incoming depth to that found in the depth buffer.

#### rs\_font : Handle to a Font

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Opaque handle to a RenderScript font object. See: android.renderscript.Font

#### rs\_mesh : Handle to a Mesh

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Opaque handle to a RenderScript mesh object. See: android.renderscript.Mesh

#### rs\_primitive : How to interpret mesh vertex data

An enum with the following values: When compiling for 32 bits. [API level 16 - 22](#)

|                                        |                                                                                                                                                   |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>RS_PRIMITIVE_POINT</i> = 0          | Vertex data will be rendered as a series of points                                                                                                |
| <i>RS_PRIMITIVE_LINE</i> = 1           | Vertex pairs will be rendered as lines                                                                                                            |
| <i>RS_PRIMITIVE_LINE_STRIP</i> = 2     | Vertex data will be rendered as a connected line strip                                                                                            |
| <i>RS_PRIMITIVE_TRIANGLE</i> = 3       | Vertices will be rendered as individual triangles                                                                                                 |
| <i>RS_PRIMITIVE_TRIANGLE_STRIP</i> = 4 | Vertices will be rendered as a connected triangle strip defined by the first three vertices with each additional triangle defined by a new vertex |
| <i>RS_PRIMITIVE_TRIANGLE_FAN</i> = 5   | Vertices will be rendered as a sequence of triangles that all share first vertex as the origin                                                    |
| <i>RS_PRIMITIVE_INVALID</i> = 100      | Invalid primitive                                                                                                                                 |

**Deprecated.** Do not use.

Describes the way mesh vertex data is interpreted when rendering

#### rs\_program\_fragment : Handle to a ProgramFragment

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Opaque handle to a RenderScript ProgramFragment object. See: android.renderscript.ProgramFragment

#### rs\_program\_raster : Handle to a ProgramRaster

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Opaque handle to a RenderScript ProgramRaster object. See: android.renderscript.ProgramRaster

#### rs\_program\_store : Handle to a ProgramStore

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Opaque handle to a RenderScript ProgramStore object. See: android.renderscript.ProgramStore

### rs\_program\_vertex : Handle to a ProgramVertex

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Opaque handle to a RenderScript ProgramVertex object. See: android.renderscript.ProgramVertex

## Functions

### rsgAllocationSyncAll : Sync the contents of an allocation

void rsgAllocationSyncAll([rs\\_allocation](#) alloc);

When compiling for 32 bits. Removed from [API level 23 and higher](#)

void rsgAllocationSyncAll([rs\\_allocation](#) alloc, [rs\\_allocation\\_usage\\_type](#) source);

When compiling for 32 bits. [API level 14 - 22](#)

**Deprecated.** Do not use.

Sync the contents of an allocation.

If the source is specified, sync from memory space specified by source.

If the source is not specified, sync from its SCRIPT memory space to its HW memory spaces.

### rsgBindColorTarget : Set the color target

void rsgBindColorTarget([rs\\_allocation](#) colorTarget, [uint](#) slot); When compiling for 32 bits. [API level 14 - 22](#)

**Deprecated.** Do not use.

Set the color target used for all subsequent rendering calls

### rsgBindConstant : Bind a constant allocation

void rsgBindConstant([rs\\_program\\_fragment](#) ps, [uint](#) slot, [rs\\_allocation](#) c);

When compiling for 32 bits. Removed from [API level 23 and higher](#)

void rsgBindConstant([rs\\_program\\_vertex](#) pv, [uint](#) slot, [rs\\_allocation](#) c);

When compiling for 32 bits. Removed from [API level 23 and higher](#)

#### Parameters

*ps* program fragment object

*slot* index of the constant buffer on the program

*c* constants to bind

*pv* program vertex object

**Deprecated.** Do not use.

Bind a new Allocation object to a ProgramFragment or ProgramVertex. The Allocation must be a valid constant input for the Program.

### rsgBindDepthTarget : Set the depth target

void rsgBindDepthTarget([rs\\_allocation](#) depthTarget); When compiling for 32 bits. [API level 14 - 22](#)

**Deprecated.** Do not use.

Set the depth target used for all subsequent rendering calls

### rsgBindFont : Bind a font object

## rsgBindFont : Bind a font object

```
void rsgBindFont(rs_font font); When compiling for 32 bits. Removed from API level 23 and higher
```

### Parameters

*font* object to bind

**Deprecated.** Do not use.

Binds the font object to be used for all subsequent font rendering calls

## rsgBindProgramFragment : Bind a ProgramFragment

```
void rsgBindProgramFragment(rs_program_fragment pf); When compiling for 32 bits. Removed from API level 23 and higher
```

**Deprecated.** Do not use.

Bind a new ProgramFragment to the rendering context.

## rsgBindProgramRaster : Bind a ProgramRaster

```
void rsgBindProgramRaster(rs_program_raster pr); When compiling for 32 bits. Removed from API level 23 and higher
```

**Deprecated.** Do not use.

Bind a new ProgramRaster to the rendering context.

## rsgBindProgramStore : Bind a ProgramStore

```
void rsgBindProgramStore(rs_program_store ps); When compiling for 32 bits. Removed from API level 23 and higher
```

**Deprecated.** Do not use.

Bind a new ProgramStore to the rendering context.

## rsgBindProgramVertex : Bind a ProgramVertex

```
void rsgBindProgramVertex(rs_program_vertex pv); When compiling for 32 bits. Removed from API level 23 and higher
```

**Deprecated.** Do not use.

Bind a new ProgramVertex to the rendering context.

## rsgBindSampler : Bind a sampler

```
void rsgBindSampler(rs_program_fragment fragment, uint slot, rs_sampler sampler); When compiling for 32 bits. Removed from API level 23 and higher
```

**Deprecated.** Do not use.

Bind a new Sampler object to a ProgramFragment. The sampler will operate on the texture bound at the matching slot.

## rsgBindTexture : Bind a texture allocation

```
void rsgBindTexture(rs_program_fragment v, uint slot, rs_allocation alloc); When compiling for 32 bits. Removed from API level 23 and higher
```

**Deprecated.** Do not use.

Bind a new Allocation object to a ProgramFragment. The Allocation must be a valid texture for the Program. The sampling of the texture will be controlled by the Sampler bound at the matching slot.

## rsgClearAllRenderTargets : Clear all color and depth targets

```
void rsgClearAllRenderTargets(); When compiling for 32 bits. API level 14 - 22
```

**Deprecated.** Do not use.

Clear all color and depth targets and resume rendering into the framebuffer

#### rsgClearColor : Clear the specified color from the surface

void rsgClearColor(float r, float g, float b, float a); When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Clears the rendering surface to the specified color.

#### rsgClearColorTarget : Clear the color target

void rsgClearColorTarget(uint slot); When compiling for 32 bits. [API level 14 - 22](#)

**Deprecated.** Do not use.

Clear the previously set color target

#### rsgClearDepth : Clear the depth surface

void rsgClearDepth(float value); When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Clears the depth surface to the specified value.

#### rsgClearDepthTarget : Clear the depth target

void rsgClearDepthTarget(); When compiling for 32 bits. [API level 14 - 22](#)

**Deprecated.** Do not use.

Clear the previously set depth target

#### rsgDrawMesh : Draw a mesh

void rsgDrawMesh(rs\_mesh ism);

When compiling for 32 bits. Removed from [API level 23 and higher](#)

void rsgDrawMesh(rs\_mesh ism, uint primitiveIndex);

When compiling for 32 bits. Removed from [API level 23 and higher](#)

void rsgDrawMesh(rs\_mesh ism, uint primitiveIndex, uint start, uint len);

When compiling for 32 bits. Removed from [API level 23 and higher](#)

##### Parameters

*ism* mesh object to render

*primitiveIndex* for meshes that contain multiple primitive groups this parameter specifies the index of the group to draw.

*start* starting index in the range

*len* number of indices to draw

**Deprecated.** Do not use.

Draw a mesh using the current context state.

If primitiveIndex is specified, draw part of a mesh using the current context state.

If start and len are also specified, draw specified index range of part of a mesh using the current context state.

Otherwise the whole mesh is rendered.

#### rsgDrawQuad : Draw a quad

### rsgDrawQuad : Draw a quad

```
void rsgDrawQuad(float x1, float y1, float z1, float x2, float y2, float z2, float x3, float
y3, float z3, float x4, float y4, float z4);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Low performance utility function for drawing a simple quad. Not intended for drawing large quantities of geometry.

### rsgDrawQuadTexCoords : Draw a textured quad

```
void rsgDrawQuadTexCoords(float x1, float y1, float z1, float u1, float v1, float x2, float y2, float z2,
float u2, float v2, float x3, float y3, float z3, float u3, float v3, float x4, float y4, float z4, float u4, float
v4);
```

When compiling for 32 bits.  
Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Low performance utility function for drawing a textured quad. Not intended for drawing large quantities of geometry.

### rsgDrawRect : Draw a rectangle

```
void rsgDrawRect(float x1, float y1, float x2, float y2, float z);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Low performance utility function for drawing a simple rectangle. Not intended for drawing large quantities of geometry.

### rsgDrawSpriteScreenspace : Draw rectangles in screenspace

```
void rsgDrawSpriteScreenspace(float x, float y, float z, float w, float
h);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Low performance function for drawing rectangles in screenspace. This function uses the default passthrough ProgramVertex. Any bound ProgramVertex is ignored. This function has considerable overhead and should not be used for drawing in shipping applications.

### rsgDrawText : Draw a text string

```
void rsgDrawText(const char* text, int x, int y);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

```
void rsgDrawText(rs_allocation alloc, int x, int y);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Draws text given a string and location

### rsgFinish : End rendering commands

```
uint rsgFinish();
```

When compiling for 32 bits. [API level 14 - 22](#)

**Deprecated.** Do not use.

Force RenderScript to finish all rendering commands

### rsgFontColor : Set the font color

```
void rsgFontColor(float r, float g, float b, float a);
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

#### Parameters

*r* red component

*g* green component

*b* blue component

*a* alpha component

**Deprecated.** Do not use.

Sets the font color for all subsequent rendering calls

### rsgGetHeight : Get the surface height

`uint rsgGetHeight();` When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Get the height of the current rendering surface.

### rsgGetWidth : Get the surface width

`uint rsgGetWidth();` When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Get the width of the current rendering surface.

### rsgMeasureText : Get the bounding box for a text string

`void rsgMeasureText(const char* text, int* left, int* right, int* top, int* bottom);`

When compiling for 32 bits. Removed from [API level 23 and higher](#)

`void rsgMeasureText(rs_allocation alloc, int* left, int* right, int* top, int* bottom);`

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Returns the bounding box of the text relative to (0, 0) Any of left, right, top, bottom could be NULL

### rsgMeshComputeBoundingBox : Compute a bounding box

`void rsgMeshComputeBoundingBox(rs_mesh mesh, float* minX, float* minY, float* min, float* maxX, float* maxY, float* maxZ);`

When compiling for 32 bits. Removed from [API level 23 and higher](#)

`void rsgMeshComputeBoundingBox(rs_mesh mesh, float3* bBoxMin, float3* bBoxMax);`

When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Computes an axis aligned bounding box of a mesh object

### rsgMeshGetIndexAllocation : Return an allocation containing index data

`rs_allocation rsgMeshGetIndexAllocation(rs_mesh m, uint32_t index);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*m* mesh to get data from

*index* index of the index allocation

#### Returns

allocation containing index data

**Deprecated.** Do not use.

Returns an allocation containing index data or a null allocation if only the primitive is specified

### rsgMeshGetPrimitive : Return the primitive

`rs_primitive rsgMeshGetPrimitive(rs_mesh m, uint32_t index);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*m* mesh to get data from

*index* index of the primitive

## Returns

primitive describing how the mesh is rendered

**Deprecated.** Do not use.

Returns the primitive describing how a part of the mesh is rendered

### rsgMeshGetPrimitiveCount : Return the number of index sets

`uint32_t rsgMeshGetPrimitiveCount(rs_mesh m);` When compiling for 32 bits. [API level 16 - 22](#)

## Parameters

*m* mesh to get data from

## Returns

number of primitive groups in the mesh. This would include simple primitives as well as allocations containing index data

**Deprecated.** Do not use.

Meshes could have multiple index sets, this function returns the number.

### rsgMeshGetVertexAllocation : Return a vertex allocation

`rs_allocation rsgMeshGetVertexAllocation(rs_mesh m, uint32_t index);` When compiling for 32 bits. [API level 16 - 22](#)

## Parameters

*m* mesh to get data from

*index* index of the vertex allocation

## Returns

allocation containing vertex data

**Deprecated.** Do not use.

Returns an allocation that is part of the mesh and contains vertex data, e.g. positions, normals, texcoords

### rsgMeshGetVertexAllocationCount : Return the number of vertex allocations

`uint32_t rsgMeshGetVertexAllocationCount(rs_mesh m);` When compiling for 32 bits. [API level 16 - 22](#)

## Parameters

*m* mesh to get data from

## Returns

number of allocations in the mesh that contain vertex data

**Deprecated.** Do not use.

Returns the number of allocations in the mesh that contain vertex data

### rsgProgramFragmentConstantColor : Set the constant color for a fixed function emulation program

`void rsgProgramFragmentConstantColor(rs_program_fragment pf, float r, float g, float b, float a);` When compiling for 32 bits. Removed from [API level 23 and higher](#)

**Deprecated.** Do not use.

Set the constant color for a fixed function emulation program.

### rsgProgramRasterGetCullMode : Get program raster cull mode

`rs_cull_mode` `rsgProgramRasterGetCullMode(rs_program_raster pr);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*pr* program raster to query

**Deprecated.** Do not use.

Get program raster cull mode

### `rsgProgramRasterIsPointSpriteEnabled` : Get program raster point sprite state

`bool rsgProgramRasterIsPointSpriteEnabled(rs_program_raster pr);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*pr* program raster to query

**Deprecated.** Do not use.

Get program raster point sprite state

### `rsgProgramStoreGetBlendDstFunc` : Get program store blend destination function

`rs_blend_dst_func` `rsgProgramStoreGetBlendDstFunc(rs_program_store ps);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store blend destination function

### `rsgProgramStoreGetBlendSrcFunc` : Get program store blend source function

`rs_blend_src_func` `rsgProgramStoreGetBlendSrcFunc(rs_program_store ps);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store blend source function

### `rsgProgramStoreGetDepthFunc` : Get program store depth function

`rs_depth_func` `rsgProgramStoreGetDepthFunc(rs_program_store ps);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store depth function

### `rsgProgramStoreIsColorMaskAlphaEnabled` : Get program store alpha component color mask

`bool rsgProgramStoreIsColorMaskAlphaEnabled(rs_program_store ps);` When compiling for 32 bits. [API level 16 - 22](#)

#### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store alpha component color mask

## rsgProgramStoreIsColorMaskBlueEnabled : Get program store blur component color mask

bool rsgProgramStoreIsColorMaskBlueEnabled([rs\\_program\\_store](#) ps); When compiling for 32 bits. [API level 16 - 22](#)

### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store blur component color mask

## rsgProgramStoreIsColorMaskGreenEnabled : Get program store green component color mask

bool rsgProgramStoreIsColorMaskGreenEnabled([rs\\_program\\_store](#) ps); When compiling for 32 bits. [API level 16 - 22](#)

### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store green component color mask

## rsgProgramStoreIsColorMaskRedEnabled : Get program store red component color mask

bool rsgProgramStoreIsColorMaskRedEnabled([rs\\_program\\_store](#) ps); When compiling for 32 bits. [API level 16 - 22](#)

### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store red component color mask

## rsgProgramStoreIsDepthMaskEnabled : Get program store depth mask

bool rsgProgramStoreIsDepthMaskEnabled([rs\\_program\\_store](#) ps); When compiling for 32 bits. [API level 16 - 22](#)

### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store depth mask

## rsgProgramStoreIsDitherEnabled : Get program store dither state

bool rsgProgramStoreIsDitherEnabled([rs\\_program\\_store](#) ps); When compiling for 32 bits. [API level 16 - 22](#)

### Parameters

*ps* program store to query

**Deprecated.** Do not use.

Get program store dither state

## rsgProgramVertexGetProjectionMatrix : Get the projection matrix for a fixed function vertex program

void rsgProgramVertexGetProjectionMatrix([rs\\_matrix4x4](#)\* proj); When compiling for 32 bits. Removed from [API level 23 and higher](#)

### Parameters

*proj* matrix to store the current projection matrix into

**Deprecated.** Do not use.

Get the projection matrix for a currently bound fixed function vertex program. Calling this function with a custom vertex shader would result in an error.

### rsgProgramVertexLoadModelMatrix : Load the model matrix for a bound fixed function vertex program

```
void rsgProgramVertexLoadModelMatrix(const rs_matrix4x4*
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

#### Parameters

*model* model matrix

**Deprecated.** Do not use.

Load the model matrix for a currently bound fixed function vertex program. Calling this function with a custom vertex shader would result in an error.

### rsgProgramVertexLoadProjectionMatrix : Load the projection matrix for a bound fixed function vertex program

```
void rsgProgramVertexLoadProjectionMatrix(const rs_matrix4x4*
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

#### Parameters

*proj* projection matrix

**Deprecated.** Do not use.

Load the projection matrix for a currently bound fixed function vertex program. Calling this function with a custom vertex shader would result in an error.

### rsgProgramVertexLoadTextureMatrix : Load the texture matrix for a bound fixed function vertex program

```
void rsgProgramVertexLoadTextureMatrix(const rs_matrix4x4*
```

When compiling for 32 bits. Removed from [API level 23 and higher](#)

#### Parameters

*tex* texture matrix

**Deprecated.** Do not use.

Load the texture matrix for a currently bound fixed function vertex program. Calling this function with a custom vertex shader would result in an error.

# RenderScript Index

## Constants

|                            |                                 |
|----------------------------|---------------------------------|
| <a href="#">M_1_PI</a>     | 1 / pi, as a 32 bit float       |
| <a href="#">M_2_PI</a>     | 2 / pi, as a 32 bit float       |
| <a href="#">M_2_SQRTPI</a> | 2 / sqrt(pi), as a 32 bit float |
| <a href="#">M_E</a>        | e, as a 32 bit float            |
| <a href="#">M_LN10</a>     | log_e(10), as a 32 bit float    |
| <a href="#">M_LN2</a>      | log_e(2), as a 32 bit float     |
| <a href="#">M_LOG10E</a>   | log_10(e), as a 32 bit float    |
| <a href="#">M_LOG2E</a>    | log_2(e), as a 32 bit float     |
| <a href="#">M_PI</a>       | pi, as a 32 bit float           |
| <a href="#">M_PI_2</a>     | pi / 2, as a 32 bit float       |
| <a href="#">M_PI_4</a>     | pi / 4, as a 32 bit float       |
| <a href="#">M_SQRT1_2</a>  | 1 / sqrt(2), as a 32 bit float  |
| <a href="#">M_SQRT2</a>    | sqrt(2), as a 32 bit float      |

## Types

|                         |                              |
|-------------------------|------------------------------|
| <a href="#">char2</a>   | Two 8 bit signed integers    |
| <a href="#">char3</a>   | Three 8 bit signed integers  |
| <a href="#">char4</a>   | Four 8 bit signed integers   |
| <a href="#">double2</a> | Two 64 bit floats            |
| <a href="#">double3</a> | Three 64 bit floats          |
| <a href="#">double4</a> | Four 64 bit floats           |
| <a href="#">float2</a>  | Two 32 bit floats            |
| <a href="#">float3</a>  | Three 32 bit floats          |
| <a href="#">float4</a>  | Four 32 bit floats           |
| <a href="#">half</a>    | 16 bit floating point value  |
| <a href="#">half2</a>   | Two 16 bit floats            |
| <a href="#">half3</a>   | Three 16 bit floats          |
| <a href="#">half4</a>   | Four 16 bit floats           |
| <a href="#">int16_t</a> | 16 bit signed integer        |
| <a href="#">int2</a>    | Two 32 bit signed integers   |
| <a href="#">int3</a>    | Three 32 bit signed integers |
| <a href="#">int32_t</a> | 32 bit signed integer        |

|                            |                                               |
|----------------------------|-----------------------------------------------|
| int4                       | Four 32 bit signed integers                   |
| int64_t                    | 64 bit signed integer                         |
| int8_t                     | 8 bit signed integer                          |
| long2                      | Two 64 bit signed integers                    |
| long3                      | Three 64 bit signed integers                  |
| long4                      | Four 64 bit signed integers                   |
| rs_allocation              | Handle to an allocation                       |
| rs_allocation_cubemap_face | Enum for selecting cube map faces             |
| rs_allocation_usage_type   | Bitfield to specify how an allocation is used |
| rs_data_kind               | Element data kind                             |
| rs_data_type               | Element basic data type                       |
| rs_element                 | Handle to an element                          |
| rs_for_each_strategy_t     | Suggested cell processing order               |
| rs_kernel                  | Handle to a kernel function                   |
| rs_kernel_context          | Handle to a kernel invocation context         |
| rs_matrix2x2               | 2x2 matrix of 32 bit floats                   |
| rs_matrix3x3               | 3x3 matrix of 32 bit floats                   |
| rs_matrix4x4               | 4x4 matrix of 32 bit floats                   |
| rs_quaternion              | Quaternion                                    |
| rs_sampler                 | Handle to a Sampler                           |
| rs_sampler_value           | Sampler wrap T value                          |
| rs_script                  | Handle to a Script                            |
| rs_script_call_t           | Cell iteration information                    |
| rs_time_t                  | Seconds since January 1, 1970                 |
| rs_tm                      | Date and time structure                       |
| rs_type                    | Handle to a Type                              |
| rs_yuv_format              | YUV format                                    |
| short2                     | Two 16 bit signed integers                    |
| short3                     | Three 16 bit signed integers                  |
| short4                     | Four 16 bit signed integers                   |
| size_t                     | Unsigned size type                            |
| ssize_t                    | Signed size type                              |
| uchar                      | 8 bit unsigned integer                        |
| uchar2                     | Two 8 bit unsigned integers                   |
| uchar3                     | Three 8 bit unsigned integers                 |
| uchar4                     | Four 8 bit unsigned integers                  |
| uint                       | 32 bit unsigned integer                       |
| uint16_t                   | 16 bit unsigned integer                       |
| uint2                      | Two 32 bit unsigned integers                  |

|          |                                |
|----------|--------------------------------|
| uint3    | Three 32 bit unsigned integers |
| uint32_t | 32 bit unsigned integer        |
| uint4    | Four 32 bit unsigned integers  |
| uint64_t | 64 bit unsigned integer        |
| uint8_t  | 8 bit unsigned integer         |
| ulong    | 64 bit unsigned integer        |
| ulong2   | Two 64 bit unsigned integers   |
| ulong3   | Three 64 bit unsigned integers |
| ulong4   | Four 64 bit unsigned integers  |
| ushort   | 16 bit unsigned integer        |
| ushort2  | Two 16 bit unsigned integers   |
| ushort3  | Three 16 bit unsigned integers |
| ushort4  | Four 16 bit unsigned integers  |

## Functions

|          |                                           |
|----------|-------------------------------------------|
| abs      | Absolute value of an integer              |
| acos     | Inverse cosine                            |
| acosh    | Inverse hyperbolic cosine                 |
| acospi   | Inverse cosine divided by pi              |
| asin     | Inverse sine                              |
| asinh    | Inverse hyperbolic sine                   |
| asinpi   | Inverse sine divided by pi                |
| atan     | Inverse tangent                           |
| atan2    | Inverse tangent of a ratio                |
| atan2pi  | Inverse tangent of a ratio, divided by pi |
| atanh    | Inverse hyperbolic tangent                |
| atanpi   | Inverse tangent divided by pi             |
| cbrt     | Cube root                                 |
| ceil     | Smallest integer not less than a value    |
| clamp    | Restrain a value to a range               |
| clz      | Number of leading 0 bits                  |
| convert  | Convert numerical vectors                 |
| copysign | Copies the sign of a number to another    |
| cos      | Cosine                                    |
| cosh     | Hypebolic cosine                          |
| cospi    | Cosine of a number multiplied by pi       |
| cross    | Cross product of two vectors              |
| degrees  | Converts radians into degrees             |
| distance | Distance between two points               |

|                |                                                          |
|----------------|----------------------------------------------------------|
| dot            | Dot product of two vectors                               |
| erf            | Mathematical error function                              |
| erfc           | Mathematical complementary error function                |
| exp            | e raised to a number                                     |
| exp10          | 10 raised to a number                                    |
| exp2           | 2 raised to a number                                     |
| expm1          | e raised to a number minus one                           |
| fabs           | Absolute value of a float                                |
| fast_distance  | Approximate distance between two points                  |
| fast_length    | Approximate length of a vector                           |
| fast_normalize | Approximate normalized vector                            |
| fdim           | Positive difference between two values                   |
| floor          | Smallest integer not greater than a value                |
| fma            | Multiply and add                                         |
| fmax           | Maximum of two floats                                    |
| fmin           | Minimum of two floats                                    |
| fmod           | Modulo                                                   |
| fract          | Positive fractional part                                 |
| frexp          | Binary mantissa and exponent                             |
| half_recip     | Reciprocal computed to 16 bit precision                  |
| half_rsqrt     | Reciprocal of a square root computed to 16 bit precision |
| half_sqrt      | Square root computed to 16 bit precision                 |
| hypot          | Hypotenuse                                               |
| ilogb          | Base two exponent                                        |
| ldexp          | Creates a floating point from mantissa and exponent      |
| length         | Length of a vector                                       |
| lgamma         | Natural logarithm of the gamma function                  |
| log            | Natural logarithm                                        |
| log10          | Base 10 logarithm                                        |
| log1p          | Natural logarithm of a value plus 1                      |
| log2           | Base 2 logarithm                                         |
| logb           | Base two exponent                                        |
| mad            | Multiply and add                                         |
| max            | Maximum                                                  |
| min            | Minimum                                                  |
| mix            | Mixes two values                                         |
| modf           | Integral and fractional components                       |
| nan            | Not a Number                                             |
| nan_half       | Not a Number                                             |
| native_acos    | Approximate inverse cosine                               |

|                  |                                                       |
|------------------|-------------------------------------------------------|
| native_acosh     | Approximate inverse hyperbolic cosine                 |
| native_acospi    | Approximate inverse cosine divided by pi              |
| native_asin      | Approximate inverse sine                              |
| native_asinh     | Approximate inverse hyperbolic sine                   |
| native_asinpi    | Approximate inverse sine divided by pi                |
| native_atan      | Approximate inverse tangent                           |
| native_atan2     | Approximate inverse tangent of a ratio                |
| native_atan2pi   | Approximate inverse tangent of a ratio, divided by pi |
| native_atanh     | Approximate inverse hyperbolic tangent                |
| native_atanpi    | Approximate inverse tangent divided by pi             |
| native_cbrt      | Approximate cube root                                 |
| native_cos       | Approximate cosine                                    |
| native_cosh      | Approximate hyperbolic cosine                         |
| native_cospi     | Approximate cosine of a number multiplied by pi       |
| native_distance  | Approximate distance between two points               |
| native_divide    | Approximate division                                  |
| native_exp       | Approximate e raised to a number                      |
| native_exp10     | Approximate 10 raised to a number                     |
| native_exp2      | Approximate 2 raised to a number                      |
| native_expm1     | Approximate e raised to a number minus one            |
| native_hypot     | Approximate hypotenuse                                |
| native_length    | Approximate length of a vector                        |
| native_log       | Approximate natural logarithm                         |
| native_log10     | Approximate base 10 logarithm                         |
| native_log1p     | Approximate natural logarithm of a value plus 1       |
| native_log2      | Approximate base 2 logarithm                          |
| native_normalize | Approximately normalize a vector                      |
| native_powr      | Approximate positive base raised to an exponent       |
| native_recip     | Approximate reciprocal                                |
| native_rootn     | Approximate nth root                                  |
| native_rsqrt     | Approximate reciprocal of a square root               |
| native_sin       | Approximate sine                                      |
| native_sincos    | Approximate sine and cosine                           |
| native_sinh      | Approximate hyperbolic sine                           |
| native_sinpi     | Approximate sine of a number multiplied by pi         |
| native_sqrt      | Approximate square root                               |
| native_tan       | Approximate tangent                                   |
| native_tanh      | Approximate hyperbolic tangent                        |
| native_tanpi     | Approximate tangent of a number multiplied by pi      |

|                         |                                                                           |
|-------------------------|---------------------------------------------------------------------------|
| nextafter               | Next floating point number                                                |
| normalize               | Normalize a vector                                                        |
| pow                     | Base raised to an exponent                                                |
| pown                    | Base raised to an integer exponent                                        |
| powr                    | Positive base raised to an exponent                                       |
| radians                 | Converts degrees into radians                                             |
| remainder               | Remainder of a division                                                   |
| remquo                  | Remainder and quotient of a division                                      |
| rint                    | Round to even                                                             |
| rootn                   | Nth root                                                                  |
| round                   | Round away from zero                                                      |
| rsAllocationCopy1DRange | Copy consecutive cells between allocations                                |
| rsAllocationCopy2DRange | Copy a rectangular region of cells between allocations                    |
| rsAllocationGetDimFaces | Presence of more than one face                                            |
| rsAllocationGetDimLOD   | Presence of levels of detail                                              |
| rsAllocationGetDimX     | Size of the X dimension                                                   |
| rsAllocationGetDimY     | Size of the Y dimension                                                   |
| rsAllocationGetDimZ     | Size of the Z dimension                                                   |
| rsAllocationGetElement  | Get the object that describes the cell of an Allocation                   |
| rsAllocationIoReceive   | Receive new content from the queue                                        |
| rsAllocationIoSend      | Send new content to the queue                                             |
| rsAllocationVLoadX      | Get a vector from an allocation of scalars                                |
| rsAllocationVStoreX     | Store a vector into an allocation of scalars                              |
| rsAtomicAdd             | Thread-safe addition                                                      |
| rsAtomicAnd             | Thread-safe bitwise and                                                   |
| rsAtomicCas             | Thread-safe compare and set                                               |
| rsAtomicDec             | Thread-safe decrement                                                     |
| rsAtomicInc             | Thread-safe increment                                                     |
| rsAtomicMax             | Thread-safe maximum                                                       |
| rsAtomicMin             | Thread-safe minimum                                                       |
| rsAtomicOr              | Thread-safe bitwise or                                                    |
| rsAtomicSub             | Thread-safe subtraction                                                   |
| rsAtomicXor             | Thread-safe bitwise exclusive or                                          |
| rsClearObject           | Release an object                                                         |
| rsCreateAllocation      | Create an rs_allocation object of given Type.                             |
| rsCreateElement         | Creates an rs_element object of the specified data type                   |
| rsCreatePixelElement    | Creates an rs_element object of the specified data type and data kind     |
| rsCreateType            | Creates an rs_type object with the specified Element and shape attributes |
| rsCreateVectorElement   | Creates an rs_element object of the specified data type and vector width  |

|                                   |                                                                             |
|-----------------------------------|-----------------------------------------------------------------------------|
| rsDebug                           | Log a message and values                                                    |
| rsElementGetBytesSize             | Size of an Element                                                          |
| rsElementGetDataKind              | Kind of an Element                                                          |
| rsElementGetDataType              | Data type of an Element                                                     |
| rsElementGetSubElement            | Sub-element of a complex Element                                            |
| rsElementGetSubElementArraySize   | Array size of a sub-element of a complex Element                            |
| rsElementGetSubElementCount       | Number of sub-elements                                                      |
| rsElementGetSubElementName        | Name of a sub-element                                                       |
| rsElementGetSubElementNameLength  | Length of the name of a sub-element                                         |
| rsElementGetSubElementOffsetBytes | Offset of the instantiated sub-element                                      |
| rsElementGetVectorSize            | Vector size of the Element                                                  |
| rsExtractFrustumPlanes            | Compute frustum planes                                                      |
| rsForEach                         | Launches a kernel                                                           |
| rsForEachInternal                 | (Internal API) Launch a kernel in the current Script (with the slot number) |
| rsForEachWithOptions              | Launches a kernel with options                                              |
| rsGetArray0                       | Index in the Array0 dimension for the specified kernel context              |
| rsGetArray1                       | Index in the Array1 dimension for the specified kernel context              |
| rsGetArray2                       | Index in the Array2 dimension for the specified kernel context              |
| rsGetArray3                       | Index in the Array3 dimension for the specified kernel context              |
| rsGetDimArray0                    | Size of the Array0 dimension for the specified kernel context               |
| rsGetDimArray1                    | Size of the Array1 dimension for the specified kernel context               |
| rsGetDimArray2                    | Size of the Array2 dimension for the specified kernel context               |
| rsGetDimArray3                    | Size of the Array3 dimension for the specified kernel context               |
| rsGetDimHasFaces                  | Presence of more than one face for the specified kernel context             |
| rsGetDimLod                       | Number of levels of detail for the specified kernel context                 |
| rsGetDimX                         | Size of the X dimension for the specified kernel context                    |
| rsGetDimY                         | Size of the Y dimension for the specified kernel context                    |
| rsGetDimZ                         | Size of the Z dimension for the specified kernel context                    |
| rsGetDt                           | Elapsed time since last call                                                |
| rsGetElementAt                    | Return a cell from an allocation                                            |
| rsGetElementAtYuv_uchar_U         | Get the U component of an allocation of YUVs                                |
| rsGetElementAtYuv_uchar_V         | Get the V component of an allocation of YUVs                                |
| rsGetElementAtYuv_uchar_Y         | Get the Y component of an allocation of YUVs                                |
| rsGetFace                         | Coordinate of the Face for the specified kernel context                     |
| rsGetLod                          | Index in the Levels of Detail dimension for the specified kernel context    |
| rsIsObject                        | Check for an empty handle                                                   |
| rsIsSphereInFrustum               | Checks if a sphere is within the frustum planes                             |
| rsLocaltime                       | Convert to local time                                                       |
| rsMatrixGet                       | Get one element                                                             |
| rsMatrixInverse                   | Inverts a matrix in place                                                   |

|                            |                                                                      |
|----------------------------|----------------------------------------------------------------------|
| rsMatrixInverseTranspose   | Inverts and transpose a matrix in place                              |
| rsMatrixLoad               | Load or copy a matrix                                                |
| rsMatrixLoadFrustum        | Load a frustum projection matrix                                     |
| rsMatrixLoadIdentity       | Load identity matrix                                                 |
| rsMatrixLoadMultiply       | Multiply two matrices                                                |
| rsMatrixLoadOrtho          | Load an orthographic projection matrix                               |
| rsMatrixLoadPerspective    | Load a perspective projection matrix                                 |
| rsMatrixLoadRotate         | Load a rotation matrix                                               |
| rsMatrixLoadScale          | Load a scaling matrix                                                |
| rsMatrixLoadTranslate      | Load a translation matrix                                            |
| rsMatrixMultiply           | Multiply a matrix by a vector or another matrix                      |
| rsMatrixRotate             | Apply a rotation to a transformation matrix                          |
| rsMatrixScale              | Apply a scaling to a transformation matrix                           |
| rsMatrixSet                | Set one element                                                      |
| rsMatrixTranslate          | Apply a translation to a transformation matrix                       |
| rsMatrixTranspose          | Transpose a matrix place                                             |
| rsPackColorTo8888          | Create a uchar4 RGBA from floats                                     |
| rsQuaternionAdd            | Add two quaternions                                                  |
| rsQuaternionConjugate      | Conjugate a quaternion                                               |
| rsQuaternionDot            | Dot product of two quaternions                                       |
| rsQuaternionGetMatrixUnit  | Get a rotation matrix from a quaternion                              |
| rsQuaternionLoadRotate     | Create a rotation quaternion                                         |
| rsQuaternionLoadRotateUnit | Quaternion that represents a rotation about an arbitrary unit vector |
| rsQuaternionMultiply       | Multiply a quaternion by a scalar or another quaternion              |
| rsQuaternionNormalize      | Normalize a quaternion                                               |
| rsQuaternionSet            | Create a quaternion                                                  |
| rsQuaternionSlerp          | Spherical linear interpolation between two quaternions               |
| rsRand                     | Pseudo-random number                                                 |
| rsSample                   | Sample a value from a texture allocation                             |
| rsSamplerGetAnisotropy     | Anisotropy of the Sampler                                            |
| rsSamplerGetMagnification  | Sampler magnification value                                          |
| rsSamplerGetMinification   | Sampler minification value                                           |
| rsSamplerGetWrapS          | Sampler wrap S value                                                 |
| rsSamplerGetWrapT          | Sampler wrap T value                                                 |
| rsSendToClient             | Send a message to the client, non-blocking                           |
| rsSendToClientBlocking     | Send a message to the client, blocking                               |
| rsSetElementAt             | Set a cell of an allocation                                          |
| rsTime                     | Seconds since January 1, 1970                                        |
| rsUnpackColor8888          | Create a float4 RGBA from uchar4                                     |

|                |                                      |
|----------------|--------------------------------------|
| rsUptimeMillis | System uptime in milliseconds        |
| rsUptimeNanos  | System uptime in nanoseconds         |
| rsYuvToRGBA    | Convert a YUV value to RGBA          |
| rsqrt          | Reciprocal of a square root          |
| sign           | Sign of a value                      |
| sin            | Sine                                 |
| sincos         | Sine and cosine                      |
| sinh           | Hyperbolic sine                      |
| sinpi          | Sine of a number multiplied by pi    |
| sqrt           | Square root                          |
| step           | 0 if less than a value, 0 otherwise  |
| tan            | Tangent                              |
| tanh           | Hyperbolic tangent                   |
| tanpi          | Tangent of a number multiplied by pi |
| tgamma         | Gamma function                       |
| trunc          | Truncates a floating point           |

## Deprecated Types

|                     |                                                     |
|---------------------|-----------------------------------------------------|
| rs_blend_dst_func   | <b>Deprecated.</b> Blend destination function       |
| rs_blend_src_func   | <b>Deprecated.</b> Blend source function            |
| rs_cull_mode        | <b>Deprecated.</b> Culling mode                     |
| rs_depth_func       | <b>Deprecated.</b> Depth function                   |
| rs_font             | <b>Deprecated.</b> Handle to a Font                 |
| rs_mesh             | <b>Deprecated.</b> Handle to a Mesh                 |
| rs_primitive        | <b>Deprecated.</b> How to intepret mesh vertex data |
| rs_program_fragment | <b>Deprecated.</b> Handle to a ProgramFragment      |
| rs_program_raster   | <b>Deprecated.</b> Handle to a ProgramRaster        |
| rs_program_store    | <b>Deprecated.</b> Handle to a ProgramStore         |
| rs_program_vertex   | <b>Deprecated.</b> Handle to a ProgramVertex        |

## Deprecated Functions

|                      |                                                              |
|----------------------|--------------------------------------------------------------|
| rsClamp              | <b>Deprecated.</b> Restrain a value to a range               |
| rsFrac               | <b>Deprecated.</b> Returns the fractional part of a float    |
| rsGetAllocation      | <b>Deprecated.</b> Return the Allocation for a given pointer |
| rsgAllocationSyncAll | <b>Deprecated.</b> Sync the contents of an allocation        |
| rsgBindColorTarget   | <b>Deprecated.</b> Set the color target                      |
| rsgBindConstant      | <b>Deprecated.</b> Bind a constant allocation                |
| rsgBindDepthTarget   | <b>Deprecated.</b> Set the depth target                      |

|                                                        |                                                                                  |
|--------------------------------------------------------|----------------------------------------------------------------------------------|
| <a href="#">rsgBindFont</a>                            | <b>Deprecated.</b> Bind a font object                                            |
| <a href="#">rsgBindProgramFragment</a>                 | <b>Deprecated.</b> Bind a ProgramFragment                                        |
| <a href="#">rsgBindProgramRaster</a>                   | <b>Deprecated.</b> Bind a ProgramRaster                                          |
| <a href="#">rsgBindProgramStore</a>                    | <b>Deprecated.</b> Bind a ProgramStore                                           |
| <a href="#">rsgBindProgramVertex</a>                   | <b>Deprecated.</b> Bind a ProgramVertex                                          |
| <a href="#">rsgBindSampler</a>                         | <b>Deprecated.</b> Bind a sampler                                                |
| <a href="#">rsgBindTexture</a>                         | <b>Deprecated.</b> Bind a texture allocation                                     |
| <a href="#">rsgClearAllRenderTargets</a>               | <b>Deprecated.</b> Clear all color and depth targets                             |
| <a href="#">rsgClearColor</a>                          | <b>Deprecated.</b> Clear the specified color from the surface                    |
| <a href="#">rsgClearColorTarget</a>                    | <b>Deprecated.</b> Clear the color target                                        |
| <a href="#">rsgClearDepth</a>                          | <b>Deprecated.</b> Clear the depth surface                                       |
| <a href="#">rsgClearDepthTarget</a>                    | <b>Deprecated.</b> Clear the depth target                                        |
| <a href="#">rsgDrawMesh</a>                            | <b>Deprecated.</b> Draw a mesh                                                   |
| <a href="#">rsgDrawQuad</a>                            | <b>Deprecated.</b> Draw a quad                                                   |
| <a href="#">rsgDrawQuadTexCoords</a>                   | <b>Deprecated.</b> Draw a textured quad                                          |
| <a href="#">rsgDrawRect</a>                            | <b>Deprecated.</b> Draw a rectangle                                              |
| <a href="#">rsgDrawSpriteScreenspace</a>               | <b>Deprecated.</b> Draw rectangles in screenspace                                |
| <a href="#">rsgDrawText</a>                            | <b>Deprecated.</b> Draw a text string                                            |
| <a href="#">rsgFinish</a>                              | <b>Deprecated.</b> End rendering commands                                        |
| <a href="#">rsgFontColor</a>                           | <b>Deprecated.</b> Set the font color                                            |
| <a href="#">rsgGetHeight</a>                           | <b>Deprecated.</b> Get the surface height                                        |
| <a href="#">rsgGetWidth</a>                            | <b>Deprecated.</b> Get the surface width                                         |
| <a href="#">rsgMeasureText</a>                         | <b>Deprecated.</b> Get the bounding box for a text string                        |
| <a href="#">rsgMeshComputeBoundingBox</a>              | <b>Deprecated.</b> Compute a bounding box                                        |
| <a href="#">rsgMeshGetIndexAllocation</a>              | <b>Deprecated.</b> Return an allocation containing index data                    |
| <a href="#">rsgMeshGetPrimitive</a>                    | <b>Deprecated.</b> Return the primitive                                          |
| <a href="#">rsgMeshGetPrimitiveCount</a>               | <b>Deprecated.</b> Return the number of index sets                               |
| <a href="#">rsgMeshGetVertexAllocation</a>             | <b>Deprecated.</b> Return a vertex allocation                                    |
| <a href="#">rsgMeshGetVertexAllocationCount</a>        | <b>Deprecated.</b> Return the number of vertex allocations                       |
| <a href="#">rsgProgramFragmentConstantColor</a>        | <b>Deprecated.</b> Set the constant color for a fixed function emulation program |
| <a href="#">rsgProgramRasterGetCullMode</a>            | <b>Deprecated.</b> Get program raster cull mode                                  |
| <a href="#">rsgProgramRasterIsPointSpriteEnabled</a>   | <b>Deprecated.</b> Get program raster point sprite state                         |
| <a href="#">rsgProgramStoreGetBlendDstFunc</a>         | <b>Deprecated.</b> Get program store blend destination function                  |
| <a href="#">rsgProgramStoreGetBlendSrcFunc</a>         | <b>Deprecated.</b> Get program store blend source function                       |
| <a href="#">rsgProgramStoreGetDepthFunc</a>            | <b>Deprecated.</b> Get program store depth function                              |
| <a href="#">rsgProgramStoreIsColorMaskAlphaEnabled</a> | <b>Deprecated.</b> Get program store alpha component color mask                  |
| <a href="#">rsgProgramStoreIsColorMaskBlueEnabled</a>  | <b>Deprecated.</b> Get program store blur component color mask                   |
| <a href="#">rsgProgramStoreIsColorMaskGreenEnabled</a> | <b>Deprecated.</b> Get program store green component color mask                  |

|                                                      |                                                                                         |
|------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <a href="#">rsgProgramStoreIsColorMaskRedEnabled</a> | <b>Deprecated.</b> Get program store red component color mask                           |
| <a href="#">rsgProgramStoreIsDepthMaskEnabled</a>    | <b>Deprecated.</b> Get program store depth mask                                         |
| <a href="#">rsgProgramStoreIsDitherEnabled</a>       | <b>Deprecated.</b> Get program store dither state                                       |
| <a href="#">rsgProgramVertexGetProjectionMatrix</a>  | <b>Deprecated.</b> Get the projection matrix for a fixed function vertex program        |
| <a href="#">rsgProgramVertexLoadModelMatrix</a>      | <b>Deprecated.</b> Load the model matrix for a bound fixed function vertex program      |
| <a href="#">rsgProgramVertexLoadProjectionMatrix</a> | <b>Deprecated.</b> Load the projection matrix for a bound fixed function vertex program |
| <a href="#">rsgProgramVertexLoadTextureMatrix</a>    | <b>Deprecated.</b> Load the texture matrix for a bound fixed function vertex program    |



# Media and Camera

The Android system provides APIs for recording and playing audio and video, and for using the built-in camera as an input device. This section describes the APIs that handle the raw media data.

## Supported Media Formats

The media codec, container, and network protocols supported by the Android platform.

## MediaPlayer

The `MediaPlayer` class provides basic audio and video playback capabilities.

## MediaRecorder

The `MediaRecorder` class provides basic audio recording capabilities using the device's microphone.

## ExoPlayer

ExoPlayer is an open source project that uses code based on low-level Android APIs to provide basic audio and video playback capabilities.

## Media Routing

Apps use the Android routing APIs to play audio and video on remote devices.

## Camera API

The `Camera` API is used to specify image capture settings, start/stop imaging preview, take pictures, and retrieve raw video.

# MediaPlayer

## In this document

- › [The basics](#)
- › [Manifest declarations](#)
- › [Using MediaPlayer](#)
  - › [Asynchronous preparation](#)
  - › [Managing state](#)
  - › [Releasing the MediaPlayer](#)
- › [Using MediaPlayer in a service](#)
  - › [Running asynchronously](#)
  - › [Handling asynchronous errors](#)
  - › [Using wake locks](#)
  - › [Performing cleanup](#)
- › [Retrieving media from a ContentResolver](#)

## Key classes

- › [MediaPlayer](#)
- › [AudioManager](#)
- › [SoundPool](#)

## See also

- › [MediaRecorder](#)
- › [Supported Media Formats](#)
- › [Data Storage](#)

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using [MediaPlayer APIs](#).

This document shows you how to write a media-playing application that interacts with the user and the system in order to obtain good performance and a pleasant user experience.

**Note:** You can play back the audio data only to the standard output device. Currently, that is the mobile device speaker or a Bluetooth headset. You cannot play sound files in the conversation audio during a call.

## The basics

The following classes are used to play sound and video in the Android framework:

### [MediaPlayer](#)

This class is the primary API for playing sound and video.

### [AudioManager](#)

This class manages audio sources and audio output on a device.

# Manifest declarations

Before starting development on your application using MediaPlayer, make sure your manifest has the appropriate declarations to allow use of related features.

- **Internet Permission** - If you are using MediaPlayer to stream network-based content, your application must request network access.

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

- **Wake Lock Permission** - If your player application needs to keep the screen from dimming or the processor from sleeping, or uses the `MediaPlayer.setScreenOnWhilePlaying()` or `MediaPlayer.setWakeMode()` methods, you must request this permission.

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

## Using MediaPlayer

One of the most important components of the media framework is the `MediaPlayer` class. An object of this class can fetch, decode, and play both audio and video with minimal setup. It supports several different media sources such as:

- Local resources
- Internal URLs, such as one you might obtain from a Content Resolver
- External URLs (streaming)

For a list of media formats that Android supports, see the [Supported Media Formats](#) page.

Here is an example of how to play audio that's available as a local raw resource (saved in your application's `res/raw/` directory):

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

In this case, a "raw" resource is a file that the system does not try to parse in any particular way. However, the content of this resource should not be raw audio. It should be a properly encoded and formatted media file in one of the supported formats.

And here is how you might play from a URI available locally in the system (that you obtained through a Content Resolver, for instance):

```
Uri myUri =; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

Playing from a remote URL via HTTP streaming looks like this:

```
String url = "http://....."; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

**Note:** If you're passing a URL to stream an online media file, the file must be capable of progressive download.

**Caution:** You must either catch or pass `IllegalArgumentException` and `IOException` when using `setDataSource()`, because the file you are referencing might not exist.

## Asynchronous preparation

Using `MediaPlayer` can be straightforward in principle. However, it's important to keep in mind that a few more things are necessary to integrate it correctly with a typical Android application. For example, the call to `prepare()` can take a long time to execute, because it might involve fetching and decoding media data. So, as is the case with any method that may take long to execute, you should **never call it from your application's UI thread**. Doing that will cause the UI to hang until the method returns, which is a very bad user experience and can cause an ANR (Application Not Responding) error. Even if you expect your resource to load quickly, remember that anything that takes more than a tenth of a second to respond in the UI will cause a noticeable pause and will give the user the impression that your application is slow.

To avoid hanging your UI thread, spawn another thread to prepare the `MediaPlayer` and notify the main thread when done. However, while you could write the threading logic yourself, this pattern is so common when using `MediaPlayer` that the framework supplies a convenient way to accomplish this task by using the `prepareAsync()` method. This method starts preparing the media in the background and returns immediately. When the media is done preparing, the `onPrepared()` method of the `MediaPlayer.OnPreparedListener`, configured through `setOnPreparedListener()` is called.

## Managing state

Another aspect of a `MediaPlayer` that you should keep in mind is that it's state-based. That is, the `MediaPlayer` has an internal state that you must always be aware of when writing your code, because certain operations are only valid when the player is in specific states. If you perform an operation while in the wrong state, the system may throw an exception or cause other undesirable behaviors.

The documentation in the `MediaPlayer` class shows a complete state diagram, that clarifies which methods move the `MediaPlayer` from one state to another. For example, when you create a new `MediaPlayer`, it is in the *Idle* state. At that point, you should initialize it by calling `setDataSource()`, bringing it to the *Initialized* state. After that, you have to prepare it using either the `prepare()` or `prepareAsync()` method. When the `MediaPlayer` is done preparing, it will then enter the *Prepared* state, which means you can call `start()` to make it play the media. At that point, as the diagram illustrates, you can move between the *Started*, *Paused* and *PlaybackCompleted* states by calling such methods as `start()`, `pause()`, and `seekTo()`, amongst others. When you call `stop()`, however, notice that you cannot call `start()` again until you prepare the `MediaPlayer` again.

Always keep the state diagram in mind when writing code that interacts with a `MediaPlayer` object, because calling its methods from the wrong state is a common cause of bugs.

## Releasing the MediaPlayer

A `MediaPlayer` can consume valuable system resources. Therefore, you should always take extra precautions to make sure you are not hanging on to a `MediaPlayer` instance longer than necessary. When you are done with it, you should always call `release()` to make sure any system resources allocated to it are properly released. For example, if you are using a `MediaPlayer` and your activity receives a call to `onStop()`, you must release the `MediaPlayer`, because it makes little sense to hold on to it while your activity is not interacting with the user (unless you are playing media in the background, which is discussed in the next section). When your activity is resumed or restarted, of course, you need to create a new `MediaPlayer` and prepare it again before resuming playback.

Here's how you should release and then nullify your `MediaPlayer`:

```
MediaPlayer.release();
MediaPlayer = null;
```

As an example, consider the problems that could happen if you forgot to release the `MediaPlayer` when your activity is stopped, but create a new one when the activity starts again. As you may know, when the user changes the screen orientation (or changes the device configuration in another way), the system handles that by restarting the activity (by default), so you might quickly consume all of the system resources as the user rotates the device back and forth between portrait and landscape, because at each orientation change, you create a new `MediaPlayer` that you never release. (For more information about runtime restarts, see [Handling Runtime Changes](#).)

You may be wondering what happens if you want to continue playing "background media" even when the user leaves your activity, much in the same way that the built-in Music application behaves. In this case, what you need is a `MediaPlayer` controlled by a Service, as discussed in the next section

# Using MediaPlayer in a service

If you want your media to play in the background even when your application is not onscreen—that is, you want it to continue playing while the user is interacting with other applications—then you must start a Service and control the [MediaPlayer](#) instance from there. You need to embed the MediaPlayer in a [MediaBrowserServiceCompat](#) service and have it interact with a [MediaBrowserCompat](#) in another activity.

You should be careful about this client/server setup. There are expectations about how a player running in a background service interacts with the rest of the system. If your application does not fulfill those expectations, the user may have a bad experience. Read [Building an Audio App](#) for the full details.

This section describes special instructions for managing a MediaPlayer when it is implemented inside a service.

## Running asynchronously

First of all, like an [Activity](#), all work in a [Service](#) is done in a single thread by default—in fact, if you're running an activity and a service from the same application, they use the same thread (the "main thread") by default. Therefore, services need to process incoming intents quickly and never perform lengthy computations when responding to them. If any heavy work or blocking calls are expected, you must do those tasks asynchronously: either from another thread you implement yourself, or using the framework's many facilities for asynchronous processing.

For instance, when using a [MediaPlayer](#) from your main thread, you should call [prepareAsync\(\)](#) rather than [prepare\(\)](#), and implement a [MediaPlayer.OnPreparedListener](#) in order to be notified when the preparation is complete and you can start playing. For example:

```
public class MyService extends Service implements MediaPlayer.OnPreparedListener {
 private static final String ACTION_PLAY = "com.example.action.PLAY";
 MediaPlayer mMediaPlayer = null;

 public int onStartCommand(Intent intent, int flags, int startId) {
 ...
 if (intent.getAction().equals(ACTION_PLAY)) {
 mMediaPlayer = ... // initialize it here
 mMediaPlayer.setOnPreparedListener(this);
 mMediaPlayer.prepareAsync(); // prepare async to not block main thread
 }
 }

 /** Called when MediaPlayer is ready */
 public void onPrepared(MediaPlayer player) {
 player.start();
 }
}
```

## Handling asynchronous errors

On synchronous operations, errors would normally be signaled with an exception or an error code, but whenever you use asynchronous resources, you should make sure your application is notified of errors appropriately. In the case of a [MediaPlayer](#), you can accomplish this by implementing a [MediaPlayer.OnErrorListener](#) and setting it in your [MediaPlayer](#) instance:

```
public class MyService extends Service implements MediaPlayer.OnErrorListener {
 MediaPlayer mMediaPlayer;

 public void initMediaPlayer() {
 // ...initialize the MediaPlayer here...

 mMediaPlayer.setOnErrorListener(this);
 }

 @Override
 public boolean onError(MediaPlayer mp, int what, int extra) {
 // ... react appropriately ...
 // The MediaPlayer has moved to the Error state, must be reset!
 }
}
```

It's important to remember that when an error occurs, the `MediaPlayer` moves to the *Error* state (see the documentation for the `MediaPlayer` class for the full state diagram) and you must reset it before you can use it again.

## Using wake locks

When designing applications that play media in the background, the device may go to sleep while your service is running. Because the Android system tries to conserve battery while the device is sleeping, the system tries to shut off any of the phone's features that are not necessary, including the CPU and the WiFi hardware. However, if your service is playing or streaming music, you want to prevent the system from interfering with your playback.

In order to ensure that your service continues to run under those conditions, you have to use "wake locks." A wake lock is a way to signal to the system that your application is using some feature that should stay available even if the phone is idle.

**Notice:** You should always use wake locks sparingly and hold them only for as long as truly necessary, because they significantly reduce the battery life of the device.

To ensure that the CPU continues running while your `MediaPlayer` is playing, call the `setWakeMode()` method when initializing your `MediaPlayer`. Once you do, the `MediaPlayer` holds the specified lock while playing and releases the lock when paused or stopped:

```
mMediaPlayer = new MediaPlayer();
// ... other initialization here ...
mMediaPlayer.setWakeMode(getApplicationContext(), PowerManager.PARTIAL_WAKE_LOCK);
```

However, the wake lock acquired in this example guarantees only that the CPU remains awake. If you are streaming media over the network and you are using Wi-Fi, you probably want to hold a `WifiLock` as well, which you must acquire and release manually. So, when you start preparing the `MediaPlayer` with the remote URL, you should create and acquire the Wi-Fi lock. For example:

```
WifiLock wifiLock = ((WifiManager) getSystemService(Context.WIFI_SERVICE))
 .createWifiLock(WifiManager.WIFI_MODE_FULL, "mylock");

wifiLock.acquire();
```

When you pause or stop your media, or when you no longer need the network, you should release the lock:

```
wifiLock.release();
```

## Performing cleanup

As mentioned earlier, a `MediaPlayer` object can consume a significant amount of system resources, so you should keep it only for as long as you need and call `release()` when you are done with it. It's important to call this cleanup method explicitly rather than rely on system garbage collection because it might take some time before the garbage collector reclaims the `MediaPlayer`, as it's only sensitive to memory needs and not to shortage of other media-related resources. So, in the case when you're using a service, you should always override the `onDestroy()` method to make sure you are releasing the `MediaPlayer`:

```

public class MyService extends Service {
 MediaPlayer mMediaPlayer;
 // ...

 @Override
 public void onDestroy() {
 if (mMediaPlayer != null) mMediaPlayer.release();
 }
}

```

You should always look for other opportunities to release your `MediaPlayer` as well, apart from releasing it when being shut down. For example, if you expect not to be able to play media for an extended period of time (after losing audio focus, for example), you should definitely release your existing `MediaPlayer` and create it again later. On the other hand, if you only expect to stop playback for a very short time, you should probably hold on to your `MediaPlayer` to avoid the overhead of creating and preparing it again.

## Retrieving media from a ContentResolver

Another feature that may be useful in a media player application is the ability to retrieve music that the user has on the device. You can do that by querying the `ContentResolver` for external media:

```

ContentResolver contentResolver = getContentResolver();
Uri uri = android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
Cursor cursor = contentResolver.query(uri, null, null, null, null);
if (cursor == null) {
 // query failed, handle error.
} else if (!cursor.moveToFirst()) {
 // no media on the device
} else {
 int titleColumn = cursor.getColumnIndex(android.provider.MediaStore.Audio.Media.TITLE);
 int idColumn = cursor.getColumnIndex(android.provider.MediaStore.Audio.Media._ID);
 do {
 long thisId = cursor.getLong(idColumn);
 String thisTitle = cursor.getString(titleColumn);
 // ...process entry...
 } while (cursor.moveToNext());
}

```

To use this with the `MediaPlayer`, you can do this:

```

long id = /* retrieve it from somewhere */;
Uri contentUri = ContentUris.withAppendedId(
 android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, id);

mMediaPlayer = new MediaPlayer();
mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mMediaPlayer.setDataSource(getApplicationContext(), contentUri);

// ...prepare and start...

```

# MediaRouter API

## In this document

- [The media route button](#)
- [Use an AppCompatActivity](#)
- [Define the media route button menu item](#)
- [Create a MediaRouteSelector](#)
- [Add the media route button to the action bar](#)
- [MediaRouter callbacks](#)
- [Controlling a remote playback route](#)

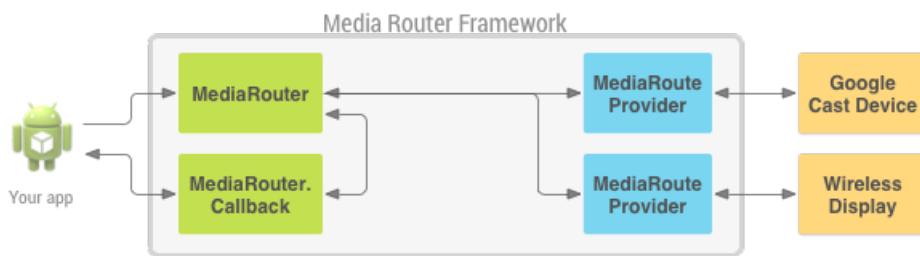
## Key classes

- [MediaRouter](#)
- [MediaRouter.Callback](#)
- [MediaRouteProvider](#)

## Related samples

- [MediaRouter](#)

In order to use the MediaRouter framework within your app, you must get an instance of the [MediaRouter](#) object and attach a [MediaRouter.Callback](#) object to listen for routing events. Content sent over a media route passes through the route's associated [MediaRouteProvider](#) (except in a few special cases, such as a Bluetooth output device). Figure 1 provides a high-level view of the classes used to route content between devices.



**Figure 1.** Overview of key media router classes used by apps.

**Note:** If you want your app to support [Google Cast](#) devices, you should use the [Cast SDK](#) and build your app as a Cast sender. Follow the directions in the [Cast documentation](#) instead of using the MediaRouter framework directly.

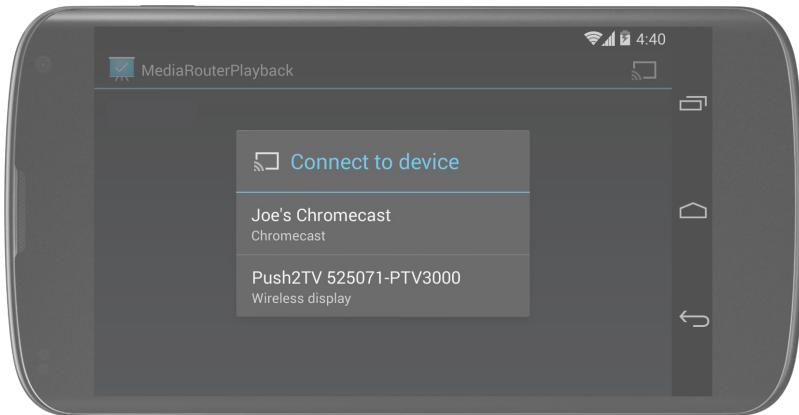
## The media route button

Android apps should use a media route button to control media routing. The MediaRouter framework provides a standard interface for the button, which helps users recognize and use routing when it's available. The media route button is usually placed on the right side of your app's action bar, as shown in Figure 2.



**Figure 2.** Media route button in the action bar.

When the user presses the media route button, the available media routes appear in a list as shown in figure 3.



**Figure 3.** A list of available media routes, shown after pressing the media route button.

Follow these steps to create a media route button:

1. Use an `AppCompatActivity`
2. Define the media route button menu item
3. Create a `MediaRouteSelector`
4. Add the media route button to the action bar
5. Create and manage the `MediaRouter.Callback` methods in your activity's lifecycle

This section describes the first four steps. The next section describes Callback methods.

## Use an `AppCompatActivity`

When you use the media router framework in an activity you should extend the activity from `AppCompatActivity` and import the package `android.support.v7.media`. You must add the `v7-appcompat` and `v7-mediarouter` support libraries to your app development project. For more information on adding support libraries to your project, see [Support Library Setup](#).

**Caution:** Be sure to use the `android.support.v7.media` implementation of the media router framework. Do not use the older `android.media` package.

## Define the media route button menu item

Create an xml file that defines a menu item for the media route button. The item's action should be the `MediaRouteActionProvider` class. Here is an example file:

```
// myMediaRouteButtonItem.xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 >

 <item android:id="@+id/media_route_menu_item"
 android:title="@string/media_route_menu_title"
 app:actionProviderClass="android.support.v7.app.MediaRouteActionProvider"
 app:showAsAction="always"
 />
</menu>
```

## Create a MediaRouteSelector

The routes that appear in the media route button menu are determined by a [MediaRouteSelector](#). Extend your activity from [AppCompatActivity](#) and build the selector when the activity is created calling [MediaRouteSelector.Builder](#) from the `onCreate()` method as shown in the following code sample. Note that the selector is saved in a class variable, and the allowable route types are specified by adding [MediaControlIntent](#) objects:

```
public class MediaRouterPlaybackActivity extends AppCompatActivity {
 private MediaRouteSelector mSelector;

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

 // Create a route selector for the type of routes your app supports.
 mSelector = new MediaRouteSelector.Builder()
 // These are the framework-supported intents
 .addControlCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)
 .build();
 }
}
```

For most applications, the only route type needed is [CATEGORY\\_REMOTE\\_PLAYBACK](#). This route type treats the device running your app as a remote control. The connected receiver device handles all content data retrieval, decoding, and playback. This is how apps that support [Google Cast](#), like [Chromecast](#), work.

A few manufacturers support a special routing option called "secondary output." With this routing, your media app retrieves, renders, and streams video or music directly to the screen and/or speakers on the selected remote receiver device. Use secondary output to send content to wireless-enabled music systems or video displays. To enable the discovery and selection of these devices, you need to add the [CATEGORY\\_LIVE\\_AUDIO](#) or [CATEGORY\\_LIVE\\_VIDEO](#) control categories to the [MediaRouteSelector](#). You also need to create and handle your own [Presentation](#) dialog.

## Add the media route button to the action bar

With the media route menu and [MediaRouteSelector](#) defined, you can now add the media route button to an activity. Override the `onCreateOptionsMenu()` method for each of your activities to add an options menu.

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
 super.onCreateOptionsMenu(menu);

 // Inflate the menu and configure the media router action provider.
 getMenuInflater().inflate(R.menu.sample_media_router_menu, menu);

 // Attach the MediaRouteSelector to the menu item
 MenuItem mediaRouteMenuItem = menu.findItem(R.id.media_route_menu_item);
 MediaRouteActionProvider mediaRouteActionProvider =
 (MediaRouteActionProvider)MenuItemCompat.getActionProvider(
 mediaRouteMenuItem);
 // Attach the MediaRouteSelector that you built in onCreate()
 mediaRouteActionProvider.setRouteSelector(mSelector);

 // Return true to show the menu.
 return true;
}

```

For more information about implementing the action bar in your app, see the [Action Bar](#) developer guide.

You can also add a media route button as a [MediaRouteButton](#) in any view. You must attach a MediaRouteSelector to the button using the [setRouteSelector\(\)](#) method. See the [Google Cast Design Checklist](#) for guidelines on incorporating the media route button into your application.

## MediaRouter callbacks

All the apps running on the same device share a single [MediaRouter](#) instance and its routes (filtered per app by the app's MediaRouteSelector). Each activity communicates with the MediaRouter using its own implementation of [MediaRouter.Callback](#) methods. The MediaRouter calls the callback methods whenever the user selects, changes, or disconnects a route.

There are several methods in the callback that you can override to receive information about routing events. At a minimum, your implementation of the [MediaRouter.Callback](#) class should override [onRouteSelected\(\)](#) and [onRouteUnselected\(\)](#).

Since the MediaRouter is a shared resource, your app needs to manage its MediaRouter callbacks in response to the usual activity lifecycle callbacks:

- When the activity is created ([onCreate\(Bundle\)](#)) grab a pointer to the [MediaRouter](#) and hold onto it for the lifetime of the app.
- Attach callbacks to MediaRouter when the activity becomes visible ([onStart\(\)](#)), and detach them when it is hidden ([onStop\(\)](#)).

The following code sample demonstrates how to create and save the callback object, how to obtain an instance of [MediaRouter](#), and how to manage callbacks. Note the use of the [CALLBACK\\_FLAG\\_REQUEST\\_DISCOVERY](#) flag when attaching the callbacks in [onStart\(\)](#). This allows your MediaRouteSelector to refresh the media route button's list of available routes.

```

public class MediaRouterPlaybackActivity extends AppCompatActivity {
 private MediaRouter mMediaRouter;
 private MediaRouteSelector mSelector;

 // Variables to hold the currently selected route and its playback client
 private MediaRoute mRoute;
 private RemotePlaybackClient mRemotePlaybackClient;

 // Define the Callback object and its methods, save the object in a class variable
 private final MediaRouter.Callback mMediaRouterCallback =
 new MediaRouter.Callback() {

 @Override
 public void onRouteSelected(MediaRouter router, RouteInfo route) {
 Log.d(TAG, "onRouteSelected: route=" + route);

 if (route.supportsControlCategory(
 MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)){
 // Stop local playback (if necessary)
 // ...
 }
 }
}

```

```

 // Save the new route
 mRoute = route;

 // Attach a new playback client
 mRemotePlaybackClient = new RemotePlaybackClient(this, mRoute);

 // Start remote playback (if necessary)
 // ...
 }

}

@Override
public void onRouteUnselected(MediaRouter router, RouteInfo route, int reason) {
 Log.d(TAG, "onRouteUnselected: route=" + route);

 if (route.supportsControlCategory(
 MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)) {

 // Changed route: tear down previous client
 if (mRoute != null && mRemotePlaybackClient != null) {
 mRemotePlaybackClient.release();
 mRemotePlaybackClient = null;
 }

 // Save the new route
 mRoute = route;

 if (reason != MediaRouter.UNSELECT_REASON_ROUTE_CHANGED) {
 // Resume local playback (if necessary)
 // ...
 }
 }
}

// Retain a pointer to the MediaRouter
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 // Get the media router service.
 mMediaRouter = MediaRouter.getInstance(this);
 ...
}

// Use this callback to run your MediaRouteSelector to generate the list of available media routes
@Override
public void onStart() {
 mMediaRouter.addCallback(mSelector, mMediaRouterCallback,
 MediaRouter.FLAG_REQUEST_DISCOVERY);
 super.onStart();
}

// Remove the selector on stop to tell the media router that it no longer
// needs to discover routes for your app.
@Override
public void onStop() {
 mMediaRouter.removeCallback(mMediaRouterCallback);
 super.onStop();
}
...
}

```

The media router framework also provides a [MediaRouteDiscoveryFragment](#) class, which takes care of adding and removing the callback for an activity.

**Note:** If you are writing a music playback app and want the app to play music while it is in the background, you must build a [Service](#) for playback and call the media router framework from the Service's lifecycle callbacks.

# Controlling a remote playback route

---

When you select a remote playback route your app acts as a remote control. The device at the other end of the route handles all content data retrieval, decoding, and playback functions. The controls in your app's UI communicate with the receiver device using a [RemotePlaybackClient](#) object.

The [RemotePlaybackClient](#) class provides additional methods for managing content playback. Here are a few of the key playback methods from the [RemotePlaybackClient](#) class:

- [play\(\)](#) — Play a specific media file, specified by a [Uri](#).
- [pause\(\)](#) — Pause the currently playing media track.
- [resume\(\)](#) — Continue playing the current track after a pause command.
- [seek\(\)](#) — Move to a specific position in the current track.
- [release\(\)](#) — Tear down the connection from your app to the remote playback device.

You can use these methods to attach actions to the playback controls that you provide in your app. Most of these methods also allow you to include a callback object so you can monitor the progress of the playback task or control request.

The [RemotePlaybackClient](#) class also supports queueing of multiple media items for playback and management of the media queue.

# Media Route Provider API

## In this document

- » [Overview](#)
  - » [Distribution of route providers](#)
  - » [Media router library](#)
  - » [Creating a provider service](#)
  - » [Specifying route capabilities](#)
    - » [Route categories](#)
    - » [Media types and protocols](#)
    - » [Playback controls](#)
    - » [MediaRouteProviderDescriptor](#)
  - » [Controlling routes](#)

## Key classes

- » [MediaRouteProvider](#)
- » [MediaRouteProviderDescriptor](#)
- » [RouteController](#)

## Related samples

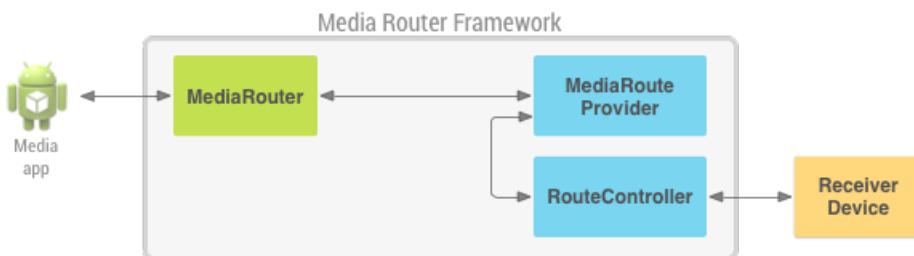
- » [MediaRouter](#)

The Android media router framework allows manufacturers to enable playback on their devices through a standardized interface called a [MediaRouteProvider](#). A route provider defines a common interface for playing media on a receiver device, making it possible to play media on your equipment from any Android application that supports media routes.

This guide discusses how to create a media route provider for a receiver device and make it available to other media playback applications that run on Android.

## Overview

The Android media router framework enables media app developers and media playback device manufacturers to connect through a common API and common user interface. App developers that implement a [MediaRouter](#) interface can then connect to the framework and play content to devices that participate in the media router framework. Media playback device manufacturers can participate in the framework by publishing a [MediaRouteProvider](#) that allows other applications to connect to and play media on the receiver devices. Figure 1 illustrates how an app connects to a receiving device through the media router framework.



**Figure 1.** Overview of how media route provider classes provide communication from a media app to a receiver device.

When you build a media route provider for your receiver device, the provider serves the following purposes:

- Describe and publish the capabilities of the receiver device so other apps can discover it and use its playback features.
- Wrap the programming interface of the receiver device and its communication transport mechanisms to make the device compatible with the media router framework.

## Distribution of route providers

A media route provider is distributed as part of an Android app. Your route provider can be made available to other apps by extending [MediaRouteProviderService](#) or wrapping your implementation of [MediaRouteProvider](#) with your own service and declaring an intent filter for the media route provider. These steps allow other apps to discover and make use of your media route.

**Note:** The app containing the media route provider can also include a [MediaRouter](#) interface to the route provider, but this is not required.

## Media router library

The media router APIs are defined in the [v7-mediarouter](#) support library. You must add this library to your app development project. For more information on adding support libraries to your project, see [Support Library Setup](#).

**Caution:** Be sure to use the `android.support.v7.media` implementation of the media router framework. Do not use the older `android.media` package.

## Creating a Provider Service

The media router framework must be able to discover and connect to your media route provider to allow other applications to use your route. In order to do this, the media router framework looks for apps that declare a media route provider intent action. When another app wants to connect to your provider, the framework must be able to invoke and connect to it, so your provider must be encapsulated in a [Service](#).

The following example code shows the declaration of a media route provider service and the intent filter in a manifest, which allows it to be discovered and used by the media router framework:

```
<service android:name=".provider.SampleMediaRouteProviderService"
 android:label="@string/sample_media_route_provider_service"
 android:process=":mrp">
 <intent-filter>
 <action android:name="android.media.MediaRouteProviderService" />
 </intent-filter>
</service>
```

This manifest example declares a service that wraps the actual media route provider classes. The Android media router framework provides the [MediaRouteProviderService](#) class for use as a service wrapper for media route providers. The following example code demonstrates how to use this wrapper class:

```
public class SampleMediaRouteProviderService extends MediaRouteProviderService {

 @Override
 public MediaRouteProvider onCreateMediaRouteProvider() {
 return new SampleMediaRouteProvider(this);
 }
}
```

## Specifying Route Capabilities

Apps connecting to the media router framework can discover your media route through your app's manifest declarations, but they also need to know the capabilities of the media routes you are providing. Media routes can be of different types and have different features, and other apps need to be able to discover these details to determine if they are compatible with your route.

The media router framework allows you to define and publish the capabilities of your media route through [IntentFilter](#) objects, [MediaRouteDescriptor](#) objects and a [MediaRouteProviderDescriptor](#). This section explains how to use these classes to publish the details of your media route for other apps.

## Route categories

As part of the programmatic description of your media route provider, you must specify whether your provider supports remote playback, secondary output, or both. These are the route categories provided by the media router framework:

- **CATEGORY\_LIVE\_AUDIO** — Output of audio to a secondary output device, such as a wireless-enabled music system.
- **CATEGORY\_LIVE\_VIDEO** — Output of video to a secondary output device, such as Wireless Display devices.
- **CATEGORY\_REMOTE\_PLAYBACK** — Play video or audio on a separate device which handles media retrieval, decoding, and playback, such as Chromecast devices.

In order to include these settings in a description of your media route, you insert them into an `IntentFilter` object, which you later add to a `MediaRouteDescriptor` object:

```
public final class SampleMediaRouteProvider extends MediaRouteProvider {
 private static final ArrayList<IntentFilter> CONTROL_FILTERS_BASIC;
 static {
 IntentFilter videoPlayback = new IntentFilter();
 videoPlayback.addCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK);
 CONTROL_FILTERS_BASIC = new ArrayList<IntentFilter>();
 CONTROL_FILTERS_BASIC.add(videoPlayback);
 }
}
```

If you specify the `CATEGORY_REMOTE_PLAYBACK` intent, you must also define what media types and playback controls are supported by your media route provider. The next section describes how to specify these settings for your device.

## Media types and protocols

A media route provider for a remote playback device must specify the media types and transfer protocols it supports. You specify these settings using the `IntentFilter` class and the `addDataScheme()` and `addDataType()` methods of that object. The following code snippet demonstrates how to define an intent filter for supporting remote video playback using http, https, and Real Time Streaming Protocol (RTSP):

```
public final class SampleMediaRouteProvider extends MediaRouteProvider {

 private static final ArrayList<IntentFilter> CONTROL_FILTERS_BASIC;

 static {
 IntentFilter videoPlayback = new IntentFilter();
 videoPlayback.addCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK);
 videoPlayback.addAction(MediaControlIntent.ACTION_PLAY);
 videoPlayback.addDataScheme("http");
 videoPlayback.addDataScheme("https");
 videoPlayback.addDataScheme("rtsp");
 addDataTypeUnchecked(videoPlayback, "video/*");
 CONTROL_FILTERS_BASIC = new ArrayList<IntentFilter>();
 CONTROL_FILTERS_BASIC.add(videoPlayback);
 }
 ...

 private static void addDataTypeUnchecked(IntentFilter filter, String type) {
 try {
 filter.addDataType(type);
 } catch (MalformedMimeTypeException ex) {
 throw new RuntimeException(ex);
 }
 }
}
```

## Playback controls

A media route provider that offers remote playback must specify the types of media controls it supports. These are the general types of control that media routes can provide:

- **Playback controls**, such as play, pause, rewind, and fast-forward.
- **Queuing features**, which allow the sending app to add and remove items from a playlist which is maintained by the receiver device.
- **Session features**, which prevent sending apps from interfering with each other by having the receiver device provide a session id to the requesting app and then checking that id with each subsequent playback control request.

The following code example demonstrates how to construct an intent filter for supporting basic media route playback controls:

```
public final class SampleMediaRouteProvider extends MediaRouteProvider {
 private static final ArrayList<IntentFilter> CONTROL_FILTERS_BASIC;
 static {
 ...
 IntentFilter playControls = new IntentFilter();
 playControls.addAction(MediaControlIntent.ACTION_SEEK);
 playControls.addAction(MediaControlIntent.ACTION_GET_STATUS);
 playControls.addAction(MediaControlIntent.ACTION_PAUSE);
 playControls.addAction(MediaControlIntent.ACTION_RESUME);
 playControls.addAction(MediaControlIntent.ACTION_STOP);
 CONTROL_FILTERS_BASIC = new ArrayList<IntentFilter>();
 CONTROL_FILTERS_BASIC.add(videoPlayback);
 CONTROL_FILTERS_BASIC.add(playControls);
 }
 ...
}
```

For more information about the available playback control intents, see the [MediaControlIntent](#) class.

## MediaRouteProviderDescriptor

After defining the capabilities of your media route using [IntentFilter](#) objects, you can then create a descriptor object for publishing to the Android media router framework. This descriptor object contains the specifics of your media route's capabilities so that other applications can determine how to interact with your media route.

The following example code demonstrates how to add the previously created intent filters to a [MediaRouteProviderDescriptor](#) and set the descriptor for use by the media router framework:

```
public SampleMediaRouteProvider(Context context) {
 super(context);
 publishRoutes();
}

private void publishRoutes() {
 Resources r = getContext().getResources();
 // Create a route descriptor using previously created IntentFilters
 MediaRouteDescriptor routeDescriptor = new MediaRouteDescriptor.Builder(
 VARIABLE_VOLUME_BASIC_ROUTE_ID,
 r.getString(R.string.variable_volume_basic_route_name))
 .setDescription(r.getString(R.string.sample_route_description))
 .addControlFilters(CONTROL_FILTERS_BASIC)
 .setPlaybackStream(AudioManager.STREAM_MUSIC)
 .setPlaybackType(MediaRouter.RouteInfo.PLAYBACK_TYPE_REMOTE)
 .setVolumeHandling(MediaRouter.RouteInfo.PLAYBACK_VOLUME_VARIABLE)
 .setVolumeMax(VOLUME_MAX)
 .setVolume(mVolume)
 .build();
 // Add the route descriptor to the provider descriptor
 MediaRouteProviderDescriptor providerDescriptor =
 new MediaRouteProviderDescriptor.Builder()
 .addRoute(routeDescriptor)
 .build();

 // Publish the descriptor to the framework
 setDescriptor(providerDescriptor);
}
```

For more information on the available descriptor settings, see the reference documentation for [MediaRouteDescriptor](#) and [MediaRouteProviderDescriptor](#).

## Controlling Routes

---

When an application connects to your media route provider, the provider receives playback commands through the media router framework sent to your route by other apps. To handle these requests, you must provide an implementation of a [MediaRouteProvider.RouteController](#) class, which processes the commands and handles the actual communication to your receiver device.

The media router framework calls the [onCreateRouteController\(\)](#) method of your route provider to obtain an instance of this class and then routes requests to it. These are the key methods of the [MediaRouteProvider.RouteController](#) class, which you must implement for your media route provider:

- [onSelect\(\)](#) — Called when an application selects your route for playback. You use this method to do any preparation work that may be required before media playback begins.
- [onControlRequest\(\)](#) — Sends specific playback commands to the receiving device.
- [onSetVolume\(\)](#) — Sends a request to the receiving device to set the playback volume to a specific value.
- [onUpdateVolume\(\)](#) — Sends a request to the receiving device to modify the playback volume by a specified amount.
- [onUnselect\(\)](#) — Called when an application unselects a route.
- [onRelease\(\)](#) — Called when the route is no longer needed by the framework, allowing it to free its resources.

All playback control requests, except for volume changes, are directed to the [onControlRequest\(\)](#) method. Your implementation of this method must parse the control requests and respond to them appropriately. Here is an example implementation of this method which processes commands for a remote playback media route:

```

private final class SampleRouteController extends
 MediaRouteProvider.RouteController {
 ...

 @Override
 public boolean onControlRequest(Intent intent, ControlRequestCallback callback) {

 String action = intent.getAction();

 if (intent.hasCategory(MediaControlIntent.CATEGORY_REMOTE_PLAYBACK)) {
 boolean success = false;
 if (action.equals(MediaControlIntent.ACTION_PLAY)) {
 success = handlePlay(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_ENQUEUE)) {
 success = handleEnqueue(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_REMOVE)) {
 success = handleRemove(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_SEEK)) {
 success = handleSeek(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_GET_STATUS)) {
 success = handleGetStatus(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_PAUSE)) {
 success = handlePause(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_RESUME)) {
 success = handleResume(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_STOP)) {
 success = handleStop(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_START_SESSION)) {
 success = handleStartSession(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_GET_SESSION_STATUS)) {
 success = handleGetSessionStatus(intent, callback);
 } else if (action.equals(MediaControlIntent.ACTION_END_SESSION)) {
 success = handleEndSession(intent, callback);
 }
 }

 Log.d(TAG, mSessionManager.toString());
 return success;
 }
 return false;
}
...
}

```

It is important to understand that the `MediaRouteProvider.RouteController` class is intended to act as a wrapper for the API to your media playback equipment. The implementation of the methods in this class is entirely dependent on the programmatic interface provided by your receiving device.



# ExoPlayer

[ExoPlayer](#) is an open source project that is not part of the Android framework and is distributed separately from the Android SDK. ExoPlayer's standard audio and video components are built on Android's MediaCodec API, which was released in Android 4.1 (API level 16). Because ExoPlayer is a library, you can easily take advantage of new features as they become available by updating your app.

ExoPlayer supports features like Dynamic adaptive streaming over HTTP (DASH), SmoothStreaming and Common Encryption, which are not supported by [MediaPlayer](#). It's designed to be easy to customize and extend.

To learn more, read the [developer guide](#) and watch the [I/O 2017 ExoPlayer session](#).

## Sample code

---

There are a number of sample apps that demonstrate how to use ExoPlayer for audio and video playback:

- [UAMP](#)—The Universal Music Player uses ExoPlayer for local audio playback.
- [Leanback sample app](#)—Android TV Leanback Support Library Sample App uses ExoPlayer for video playback on Android TV.
- [ExoPlayer demo app](#)—The official repository contains a demo app that showcases many advanced capabilities of the library.
- [Codelab](#)—Demonstrates how to build an Activity that plays media using ExoPlayer.

# Supported Media Formats

In this document

- [Audio support](#)
- [Video support](#)
- [Image support](#)
- [Network protocols](#)

This document describes the media codec, container, and network protocol support provided by the Android platform.

As an application developer, you can use any media codec that is available on any Android-powered device, including those provided by the Android platform and those that are device-specific. **However, it is a best practice to use media encoding profiles that are device-agnostic.**

The tables below describe the media format support built into the Android platform. Codecs that are not guaranteed to be available on all Android platform versions are noted in parentheses, for example: (Android 3.0+). Note that any given mobile device might support other formats or file types that are not listed in the table.

As stated in section 5 of the [Android Compatibility Definition](#) document, **all device implementations must support the core media formats specified on this page, except where explicitly noted.**

## Audio support

### Audio formats and codecs

Format / Codec	Encoder	Decoder	Details	Supported File Type(s) / Container Formats
AAC LC	•	•	Support for mono/stereo/5.0/5.1 content with standard sampling rates from 8 to 48 kHz.	• 3GPP (.3gp) • MPEG-4 (.mp4, .m4a) • ADTS raw AAC (.aac, decode in Android 3.1+, encode in Android 4.0+, ADIF not supported)
HE-AACv1 (AAC+) (Android 4.1+)	•	•	Support for stereo/5.0/5.1 content with standard sampling rates from 8 to 48 kHz.	• MPEG-TS (.ts, not seekable, Android 3.0+)
HE-AACv2 (enhanced AAC+)		•	Support for mono/stereo content with standard sampling rates from 16 to 48 kHz	
AAC ELD (enhanced low delay AAC)	• (Android 4.1+)	• (Android 4.1+)	Support for mono/stereo content with standard sampling rates from 16 to 48 kHz	
AMR-NB	•	•	4.75 to 12.2 kbps sampled @ 8kHz	3GPP (.3gp)
AMR-WB	•	•	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz	3GPP (.3gp)
FLAC		• (Android 3.1+)	Mono/Stereo (no multichannel). Sample rates up to 48 kHz (but up to 44.1 kHz is recommended on devices with 44.1 kHz output, as the 48 to 44.1 kHz downampler does not include a low-pass filter). 16-bit recommended; no dither applied for 24-bit.	FLAC (.flac) only

MIDI		•	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	<ul style="list-style-type: none"> <li>• Type 0 and 1 (.mid, .xmf, .mxmf)</li> <li>• RTTTL/RTX (.rtttl, .rtx)</li> <li>• OTA (.ota)</li> <li>• iMelody (.imy)</li> </ul>
MP3		•	Mono/Stereo 8-320Kbps constant (CBR) or variable bit-rate (VBR)	MP3 (.mp3)
Opus		• (Android 5.0+)		Matroska (.mkv)
PCM/WAVE	• (Android 4.1+)	•	8- and 16-bit linear PCM (rates up to limit of hardware). Sampling rates for raw PCM recordings at 8000, 16000 and 44100 Hz.	WAVE (.wav)
Vorbis		•		<ul style="list-style-type: none"> <li>• Ogg (.ogg)</li> <li>• Matroska (.mkv, Android 4.0+)</li> </ul>

## Video support

### Video formats and codecs

Format / Codec	Encoder	Decoder	Details	Supported File Type(s) / Container Formats
H.263	•	•	Support for H.263 is optional in Android 7.0+	<ul style="list-style-type: none"> <li>• 3GPP (.3gp)</li> <li>• MPEG-4 (.mp4)</li> </ul>
H.264 AVC Baseline Profile (BP)	• (Android 3.0+)	•		<ul style="list-style-type: none"> <li>• 3GPP (.3gp)</li> <li>• MPEG-4 (.mp4)</li> <li>• MPEG-TS (.ts, AAC audio only, not seekable, Android 3.0+)</li> </ul>
H.264 AVC Main Profile (MP)	• (Android 6.0+)	•	The decoder is required, the encoder is recommended.	
H.265 HEVC		• (Android 5.0+)	Main Profile Level 3 for mobile devices and Main Profile Level 4.1 for Android TV	<ul style="list-style-type: none"> <li>• MPEG-4 (.mp4)</li> </ul>
MPEG-4 SP		•		3GPP (.3gp)
VP8	• (Android 4.3+)	• (Android 2.3.3+)	Streamable only in Android 4.0 and above	<ul style="list-style-type: none"> <li>• <a href="#">WebM</a> (.webm)</li> <li>• Matroska (.mkv, Android 4.0+)</li> </ul>
VP9		• (Android 4.4+)		<ul style="list-style-type: none"> <li>• <a href="#">WebM</a> (.webm)</li> <li>• Matroska (.mkv, Android 4.0+)</li> </ul>

### Video encoding recommendations

The table below lists the Android media framework video encoding profiles and parameters recommended for playback using the H.264 Baseline Profile codec. The same recommendations apply to the Main Profile codec, which is only available in Android 6.0 and later.

	SD (Low quality)	SD (High quality)	HD 720p (N/A on all devices)
Video resolution	176 x 144 px	480 x 360 px	1280 x 720 px
Video frame rate	12 fps	30 fps	30 fps
Video bitrate	56 Kbps	500 Kbps	2 Mbps
Audio codec	AAC-LC	AAC-LC	AAC-LC

Audio channels	<sup>1 (mono)</sup> SD (Low quality)	<sup>2 (stereo)</sup> SD (High quality)	<sup>2 (stereo)</sup> HD 720p (N/A on all devices)
Audio bitrate	24 Kbps	128 Kbps	192 Kbps

The table below lists the Android media framework video encoding profiles and parameters recommended for playback using the VP8 media codec.

	SD (Low quality)	SD (High quality)	HD 720p (N/A on all devices)	HD 1080p (N/A on all devices)
Video resolution	320 x 180 px	640 x 360 px	1280 x 720 px	1920 x 1080 px
Video frame rate	30 fps	30 fps	30 fps	30 fps
Video bitrate	800 Kbps	2 Mbps	4 Mbps	10 Mbps

## Video decoding recommendations

Device implementations must support dynamic video resolution and frame rate switching through the standard Android APIs within the same stream for all VP8, VP9, H.264, and H.265 codecs in real time and up to the maximum resolution supported by each codec on the device.

Implementations that support the Dolby Vision decoder must follow these guidelines:

- Provide a Dolby Vision-capable extractor.
- Properly display Dolby Vision content on the device screen or on a standard video output port (e.g., HDMI).
- Set the track index of backward-compatible base-layer(s) (if present) to be the same as the combined Dolby Vision layer's track index.

## Video streaming requirements

For video content that is streamed over HTTP or RTSP, there are additional requirements:

- For 3GPP and MPEG-4 containers, the `moov` atom must precede any `mdat` atoms, but must succeed the `ftyp` atom.
- For 3GPP, MPEG-4, and WebM containers, audio and video samples corresponding to the same time offset may be no more than 500 KB apart. To minimize this audio/video drift, consider interleaving audio and video in smaller chunk sizes.

## Image support

Format / Codec	Encoder	Decoder	Details	Supported File Type(s) / Container Formats
BMP		•		BMP (.bmp)
GIF		•		GIF (.gif)
JPEG	•	•	Base+progressive	JPEG (.jpg)
PNG	•	•		PNG (.png)
WebP	•  (Android 4.0+)  (Lossless, Transparency, Android 4.2.1+)	•  (Android 4.0+)  (Lossless, Transparency, Android 4.2.1+)		WebP (.webp)

## Network protocols

The following network protocols are supported for audio and video playback:

- RTSP (RTP, SDP)
- HTTP/HTTPS progressive streaming
- HTTP/HTTPS live streaming [draft protocol](#):
  - MPEG-2 TS media files only

- Protocol version 3 (Android 4.0 and above)
- Protocol version 2 (Android 3.x)
- Not supported before Android 3.0

**Note:** HTTPS is not supported before Android 3.1.



# MediaRecorder

In this document

- [Requesting permission to record audio](#)
- [Using MediaRecorder](#)
- [Sample code](#)

Key classes

- [MediaRecorder](#)

See also

- [Requesting Permissions](#)
- [Supported Media Formats](#)
- [Data storage](#)
- [MediaPlayer](#)

The Android multimedia framework includes support for capturing and encoding a variety of common audio and video formats. You can use the [MediaRecorder](#) APIs if supported by the device hardware.

This document shows you how to use [MediaRecorder](#) to write an application that captures audio from a device microphone, save the audio, and play it back (with [MediaPlayer](#)). To record video you'll need to use the device's camera along with [MediaRecorder](#). This is described in the [Camera](#) guide.

**Note:** The Android Emulator cannot record audio. Be sure to test your code on a real device that can record.

## Requesting permission to record audio

To be able to record, your app must tell the user that it will access the device's audio input. You must include this permission tag in the app's manifest file:

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

`RECORD_AUDIO` is considered a ["dangerous" permission](#) because it may pose a risk to the user's privacy. Starting with Android 6.0 (API level 23) an app that uses a dangerous permission must ask the user for approval at run time. After the user has granted permission, the app should remember and not ask again. The sample code below shows how to implement this behavior using [ActivityCompat.requestPermissions\(\)](#).

## Using MediaRecorder

Initialize a new instance of [MediaRecorder](#) with the following calls:

- Set the audio source using [set AudioSource\(\)](#). You'll probably use [MIC](#).

**Note:** Most of the audio sources (including [DEFAULT](#)) apply processing to the audio signal. To record raw audio select [UNPROCESSED](#).

Some devices do not support unprocessed input. Call

[AudioManager.getProperty\("PROPERTY\\_SUPPORT\\_AUDIO\\_SOURCE\\_UNPROCESSED"\)](#) first to verify it's available. If it is not, try using [VOICE\\_RECOGNITION](#) instead, which does not employ AGC or noise suppression. You can use [UNPROCESSED](#) as an audio source even

when the property is not supported, but there is no guarantee whether the signal will be unprocessed or not in that case.

- Set the output file format using `setOutputFormat()`.
- Set the output file name using `setOutputFile()`.
- Set the audio encoder using `setAudioEncoder()`.
- Complete the initialization by calling `prepare()`.

Start and stop the recorder by calling `start()` and `stop()` respectively.

When you are done with the `MediaRecorder` instance free its resources as soon as possible by calling `release()`.

## Sample code

The example activity below shows how to use `MediaRecorder` to record an audio file. It Also uses `MediaPlayer` to play the audio back.

```
package com.android.audiorecordtest;

import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.media.MediaPlayer;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.LinearLayout;

import java.io.IOException;

public class AudioRecordTest extends AppCompatActivity {

 private static final String LOG_TAG = "AudioRecordTest";
 private static final int REQUEST_RECORD_AUDIO_PERMISSION = 200;
 private static String mFileName = null;

 private RecordButton mRecordButton = null;
 private MediaRecorder mRecorder = null;

 private PlayButton mPlayButton = null;
 private MediaPlayer mPlayer = null;

 // Requesting permission to RECORD_AUDIO
 private boolean permissionToRecordAccepted = false;
 private String [] permissions = {Manifest.permission.RECORD_AUDIO};

 @Override
 public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
 super.onRequestPermissionsResult(requestCode, permissions, grantResults);
 switch (requestCode){
 case REQUEST_RECORD_AUDIO_PERMISSION:
 permissionToRecordAccepted = grantResults[0] == PackageManager.PERMISSION_GRANTED;
 break;
 }
 if (!permissionToRecordAccepted) finish();
 }

 private void onRecord(boolean start) {
 if (start) {
 startRecording();
 } else {
 stopRecording();
 }
 }
}
```

```
 }

 private void onPlay(boolean start) {
 if (start) {
 startPlaying();
 } else {
 stopPlaying();
 }
 }

 private void startPlaying() {
 mPlayer = new MediaPlayer();
 try {
 mPlayer.setDataSource(mFileName);
 mPlayer.prepare();
 mPlayer.start();
 } catch (IOException e) {
 Log.e(LOG_TAG, "prepare() failed");
 }
 }

 private void stopPlaying() {
 mPlayer.release();
 mPlayer = null;
 }

 private void startRecording() {
 mRecorder = new MediaRecorder();
 mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
 mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
 mRecorder.setOutputFile(mFileName);
 mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

 try {
 mRecorder.prepare();
 } catch (IOException e) {
 Log.e(LOG_TAG, "prepare() failed");
 }

 mRecorder.start();
 }

 private void stopRecording() {
 mRecorder.stop();
 mRecorder.release();
 mRecorder = null;
 }

 class RecordButton extends Button {
 boolean mStartRecording = true;

 OnClickListener clicker = new OnClickListener() {
 public void onClick(View v) {
 onRecord(mStartRecording);
 if (mStartRecording) {
 setText("Stop recording");
 } else {
 setText("Start recording");
 }
 mStartRecording = !mStartRecording;
 }
 };

 public RecordButton(Context ctx) {
 super(ctx);
 setText("Start recording");
 setOnClickListener(clicker);
 }
 }

 class PlayButton extends Button {
 boolean mStartPlaying = true;
```

```
OnClickListener clicker = new OnClickListener() {
 public void onClick(View v) {
 onPlay(mStartPlaying);
 if (mStartPlaying) {
 setText("Stop playing");
 } else {
 setText("Start playing");
 }
 mStartPlaying = !mStartPlaying;
 }
};

public PlayButton(Context ctx) {
 super(ctx);
 setText("Start playing");
 setOnClickListener(clicker);
}
}

@Override
public void onCreate(Bundle icicle) {
 super.onCreate(icicle);

 // Record to the external cache directory for visibility
 mFileName = getExternalCacheDir().getAbsolutePath();
 mFileName += "/audiorecordtest.3gp";

 ActivityCompat.requestPermissions(this, permissions, REQUEST_RECORD_AUDIO_PERMISSION);

 LinearLayout ll = new LinearLayout(this);
 mRecordButton = new RecordButton(this);
 ll.addView(mRecordButton,
 new LinearLayout.LayoutParams(
 ViewGroup.LayoutParams.WRAP_CONTENT,
 ViewGroup.LayoutParams.WRAP_CONTENT,
 0));
 mPlayButton = new PlayButton(this);
 ll.addView(mPlayButton,
 new LinearLayout.LayoutParams(
 ViewGroup.LayoutParams.WRAP_CONTENT,
 ViewGroup.LayoutParams.WRAP_CONTENT,
 0));
 setContentView(ll);
}

@Override
public void onStop() {
 super.onStop();
 if (mRecorder != null) {
 mRecorder.release();
 mRecorder = null;
 }
 if (mPlayer != null) {
 mPlayer.release();
 mPlayer = null;
 }
}
```

# Camera API

## In this document

- › [Considerations](#)
- › [The basics](#)
- › [Manifest declarations](#)
- › [Using existing camera apps](#)
- › [Building a camera app](#)
  - › [Detecting camera hardware](#)
  - › [Accessing cameras](#)
  - › [Checking camera features](#)
  - › [Creating a preview class](#)
  - › [Placing preview in a layout](#)
  - › [Capturing pictures](#)
  - › [Capturing videos](#)
  - › [Releasing the camera](#)
- › [Saving media files](#)
- › [Camera features](#)
  - › [Checking feature availability](#)
  - › [Using camera features](#)
  - › [Metering and focus areas](#)
  - › [Face detection](#)
  - › [Time lapse video](#)

## Key classes

- › [Camera](#)
- › [SurfaceView](#)
- › [MediaRecorder](#)
- › [Intent](#)

## See also

- › [MediaPlayer](#)
- › [Data Storage](#)

The Android framework includes support for various cameras and camera features available on devices, allowing you to capture pictures and videos in your applications. This document discusses a quick, simple approach to image and video capture and outlines an advanced approach for creating custom camera experiences for your users.

**Note:** This page describes the [Camera](#) class, which has been deprecated. We recommend using the newer class [camera2](#), which works on Android 5.0 (API level 21) or greater. Read more about camera2 on [our blog](#).

## Considerations

Before enabling your application to use cameras on Android devices, you should consider a few questions about how your app intends to use this hardware feature.

- **Camera Requirement** - Is the use of a camera so important to your application that you do not want your application installed on a device

that does not have a camera? If so, you should declare the [camera requirement in your manifest](#).

- **Quick Picture or Customized Camera** - How will your application use the camera? Are you just interested in snapping a quick picture or video clip, or will your application provide a new way to use cameras? For a getting a quick snap or clip, consider [Using Existing Camera Apps](#). For developing a customized camera feature, check out the [Building a Camera App](#) section.
- **Storage** - Are the images or videos your application generates intended to be only visible to your application or shared so that other applications such as Gallery or other media and social apps can use them? Do you want the pictures and videos to be available even if your application is uninstalled? Check out the [Saving Media Files](#) section to see how to implement these options.

## The basics

The Android framework supports capturing images and video through the `android.hardware.camera2` API or camera [Intent](#). Here are the relevant classes:

### `android.hardware.camera2`

This package is the primary API for controlling device cameras. It can be used to take pictures or videos when you are building a camera application.

### `Camera`

This class is the older deprecated API for controlling device cameras.

### `SurfaceView`

This class is used to present a live camera preview to the user.

### `MediaRecorder`

This class is used to record video from the camera.

### `Intent`

An intent action type of `MediaStore.ACTION_IMAGE_CAPTURE` or `MediaStore.ACTION_VIDEO_CAPTURE` can be used to capture images or videos without directly using the `Camera` object.

## Manifest declarations

Before starting development on your application with the Camera API, you should make sure your manifest has the appropriate declarations to allow use of camera hardware and other related features.

- **Camera Permission** - Your application must request permission to use a device camera.

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

**Note:** If you are using the camera [by invoking an existing camera app](#), your application does not need to request this permission.

- **Camera Features** - Your application must also declare use of camera features, for example:

```
<uses-feature android:name="android.hardware.camera" />
```

For a list of camera features, see the manifest [Features Reference](#).

Adding camera features to your manifest causes Google Play to prevent your application from being installed to devices that do not include a camera or do not support the camera features you specify. For more information about using feature-based filtering with Google Play, see [Google Play and Feature-Based Filtering](#).

If your application *can use* a camera or camera feature for proper operation, but does not *require* it, you should specify this in the manifest by including the `android:required` attribute, and setting it to `false`:

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

- **Storage Permission** - If your application saves images or videos to the device's external storage (SD Card), you must also specify this in the manifest.

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

- **Audio Recording Permission** - For recording audio with video capture, your application must request the audio capture permission.

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

- **Location Permission** - If your application tags images with GPS location information, you must request the `ACCESS_FINE_LOCATION` permission. Note that, if your app targets Android 5.0 (API level 21) or higher, you also need to declare that your app uses the device's GPS:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
...
<!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
<uses-feature android:name="android.hardware.location.gps" />
```

For more information about getting user location, see [Location Strategies](#).

## Using existing camera apps

A quick way to enable taking pictures or videos in your application without a lot of extra code is to use an [Intent](#) to invoke an existing Android camera application. The details are described in the training lessons [Taking Photos Simply](#) and [Recording Videos Simply](#).

## Building a camera app

Some developers may require a camera user interface that is customized to the look of their application or provides special features. Writing your own picture-taking code can provide a more compelling experience for your users.

**Note:** The following guide is for the older, deprecated [Camera API](#). For new or advanced camera applications, the newer [android.hardware.camera2 API](#) is recommended.

The general steps for creating a custom camera interface for your application are as follows:

- **Detect and Access Camera** - Create code to check for the existence of cameras and request access.
- **Create a Preview Class** - Create a camera preview class that extends [SurfaceView](#) and implements the [SurfaceHolder](#) interface. This class previews the live images from the camera.
- **Build a Preview Layout** - Once you have the camera preview class, create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture** - Connect listeners for your interface controls to start image or video capture in response to user actions, such as pressing a button.
- **Capture and Save Files** - Setup the code for capturing pictures or videos and saving the output.
- **Release the Camera** - After using the camera, your application must properly release it for use by other applications.

Camera hardware is a shared resource that must be carefully managed so your application does not collide with other applications that may also want to use it. The following sections discusses how to detect camera hardware, how to request access to a camera, how to capture pictures or video and how to release the camera when your application is done using it.

**Caution:** Remember to release the [Camera](#) object by calling the [Camera.release\(\)](#) when your application is done using it! If your application does not properly release the camera, all subsequent attempts to access the camera, including those by your own application, will fail and may cause your or other applications to be shut down.

## Detecting camera hardware

If your application does not specifically require a camera using a manifest declaration, you should check to see if a camera is available at runtime. To perform this check, use the `PackageManager.hasSystemFeature()` method, as shown in the example code below:

```
/** Check if this device has a camera */
private boolean checkCameraHardware(Context context) {
 if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
 // this device has a camera
 return true;
 } else {
 // no camera on this device
 return false;
 }
}
```

Android devices can have multiple cameras, for example a back-facing camera for photography and a front-facing camera for video calls. Android 2.3 (API Level 9) and later allows you to check the number of cameras available on a device using the `Camera.getNumberOfCameras()` method.

## Accessing cameras

If you have determined that the device on which your application is running has a camera, you must request to access it by getting an instance of `Camera` (unless you are using an [intent to access the camera](#)).

To access the primary camera, use the `Camera.open()` method and be sure to catch any exceptions, as shown in the code below:

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance(){
 Camera c = null;
 try {
 c = Camera.open(); // attempt to get a Camera instance
 }
 catch (Exception e){
 // Camera is not available (in use or does not exist)
 }
 return c; // returns null if camera is unavailable
}
```

**Caution:** Always check for exceptions when using `Camera.open()`. Failing to check for exceptions if the camera is in use or does not exist will cause your application to be shut down by the system.

On devices running Android 2.3 (API Level 9) or higher, you can access specific cameras using `Camera.open(int)`. The example code above will access the first, back-facing camera on a device with more than one camera.

## Checking camera features

Once you obtain access to a camera, you can get further information about its capabilities using the `Camera.getParameters()` method and checking the returned `Camera.Parameters` object for supported capabilities. When using API Level 9 or higher, use the `Camera.getCameraInfo()` to determine if a camera is on the front or back of the device, and the orientation of the image.

## Creating a preview class

For users to effectively take pictures or video, they must be able to see what the device camera sees. A camera preview class is a `SurfaceView` that can display the live image data coming from a camera, so users can frame and capture a picture or video.

The following example code demonstrates how to create a basic camera preview class that can be included in a `View` layout. This class implements `SurfaceHolder.Callback` in order to capture the callback events for creating and destroying the view, which are needed for assigning the camera preview input.

```

/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
 private SurfaceHolder mHolder;
 private Camera mCamera;

 public CameraPreview(Context context, Camera camera) {
 super(context);
 mCamera = camera;

 // Install a SurfaceHolder.Callback so we get notified when the
 // underlying surface is created and destroyed.
 mHolder = getHolder();
 mHolder.addCallback(this);
 // deprecated setting, but required on Android versions prior to 3.0
 mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
 }

 public void surfaceCreated(SurfaceHolder holder) {
 // The Surface has been created, now tell the camera where to draw the preview.
 try {
 mCamera.setPreviewDisplay(holder);
 mCamera.startPreview();
 } catch (IOException e) {
 Log.d(TAG, "Error setting camera preview: " + e.getMessage());
 }
 }

 public void surfaceDestroyed(SurfaceHolder holder) {
 // empty. Take care of releasing the Camera preview in your activity.
 }

 public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
 // If your preview can change or rotate, take care of those events here.
 // Make sure to stop the preview before resizing or reformatting it.

 if (mHolder.getSurface() == null){
 // preview surface does not exist
 return;
 }

 // stop preview before making changes
 try {
 mCamera.stopPreview();
 } catch (Exception e){
 // ignore: tried to stop a non-existent preview
 }

 // set preview size and make any resize, rotate or
 // reformatting changes here

 // start preview with new settings
 try {
 mCamera.setPreviewDisplay(mHolder);
 mCamera.startPreview();

 } catch (Exception e){
 Log.d(TAG, "Error starting camera preview: " + e.getMessage());
 }
 }
}

```

If you want to set a specific size for your camera preview, set this in the `surfaceChanged()` method as noted in the comments above. When setting preview size, you *must use* values from `getSupportedPreviewSizes()`. Do not set arbitrary values in the `setPreviewSize()` method.

**Note:** With the introduction of the [Multi-Window](#) feature in Android 7.0 (API level 24) and higher, you can no longer assume the aspect ratio of the preview is the same as your activity even after calling `setDisplayOrientation()`. Depending on the window size and aspect ratio, you may have to fit a wide camera preview into a portrait-orientated layout, or vice versa, using a letterbox layout.

## Placing preview in a layout

A camera preview class, such as the example shown in the previous section, must be placed in the layout of an activity along with other user interface controls for taking a picture or video. This section shows you how to build a basic layout and activity for the preview.

The following layout code provides a very basic view that can be used to display a camera preview. In this example, the [FrameLayout](#) element is meant to be the container for the camera preview class. This layout type is used so that additional picture information or controls can be overlaid on the live camera preview images.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 >
 <FrameLayout
 android:id="@+id/camera_preview"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:layout_weight="1"
 />
 <Button
 android:id="@+id/button_capture"
 android:text="Capture"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 />
</LinearLayout>
```

On most devices, the default orientation of the camera preview is landscape. This example layout specifies a horizontal (landscape) layout and the code below fixes the orientation of the application to landscape. For simplicity in rendering a camera preview, you should change your application's preview activity orientation to landscape by adding the following to your manifest.

```
<activity android:name=".CameraActivity"
 android:label="@string/app_name"

 android:screenOrientation="landscape">
 <!-- configure this activity to use landscape orientation -->

 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
</activity>
```

**Note:** A camera preview does not have to be in landscape mode. Starting in Android 2.2 (API Level 8), you can use the [setDisplayOrientation\(\)](#) method to set the rotation of the preview image. In order to change preview orientation as the user re-orientates the phone, within the [surfaceChanged\(\)](#) method of your preview class, first stop the preview with [Camera.stopPreview\(\)](#) change the orientation and then start the preview again with [Camera.startPreview\(\)](#).

In the activity for your camera view, add your preview class to the [FrameLayout](#) element shown in the example above. Your camera activity must also ensure that it releases the camera when it is paused or shut down. The following example shows how to modify a camera activity to attach the preview class shown in [Creating a preview class](#).

```

public class CameraActivity extends Activity {

 private Camera mCamera;
 private CameraPreview mPreview;

 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 // Create an instance of Camera
 mCamera = getCameraInstance();

 // Create our Preview view and set it as the content of our activity.
 mPreview = new CameraPreview(this, mCamera);
 FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
 preview.addView(mPreview);
 }
}

```

**Note:** The `getCameraInstance()` method in the example above refers to the example method shown in [Accessing cameras](#).

## Capturing pictures

Once you have built a preview class and a view layout in which to display it, you are ready to start capturing images with your application. In your application code, you must set up listeners for your user interface controls to respond to a user action by taking a picture.

In order to retrieve a picture, use the `Camera.takePicture()` method. This method takes three parameters which receive data from the camera. In order to receive data in a JPEG format, you must implement an `Camera.PictureCallback` interface to receive the image data and write it to a file. The following code shows a basic implementation of the `Camera.PictureCallback` interface to save an image received from the camera.

```

private PictureCallback mPicture = new PictureCallback() {

 @Override
 public void onPictureTaken(byte[] data, Camera camera) {

 File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
 if (pictureFile == null){
 Log.d(TAG, "Error creating media file, check storage permissions: " +
 e.getMessage());
 return;
 }

 try {
 FileOutputStream fos = new FileOutputStream(pictureFile);
 fos.write(data);
 fos.close();
 } catch (FileNotFoundException e) {
 Log.d(TAG, "File not found: " + e.getMessage());
 } catch (IOException e) {
 Log.d(TAG, "Error accessing file: " + e.getMessage());
 }
 }
};

```

Trigger capturing an image by calling the `Camera.takePicture()` method. The following example code shows how to call this method from a button `View.OnClickListener`.

```
// Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
 new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 // get an image from the camera
 mCamera.takePicture(null, null, mPicture);
 }
 }
);
```

**Note:** The `mPicture` member in the following example refers to the example code above.

**Caution:** Remember to release the `Camera` object by calling the `Camera.release()` when your application is done using it! For information about how to release the camera, see [Releasing the camera](#).

## Capturing videos

Video capture using the Android framework requires careful management of the `Camera` object and coordination with the `MediaRecorder` class. When recording video with `Camera`, you must manage the `Camera.lock()` and `Camera.unlock()` calls to allow `MediaRecorder` access to the camera hardware, in addition to the `Camera.open()` and `Camera.release()` calls.

**Note:** Starting with Android 4.0 (API level 14), the `Camera.lock()` and `Camera.unlock()` calls are managed for you automatically.

Unlike taking pictures with a device camera, capturing video requires a very particular call order. You must follow a specific order of execution to successfully prepare for and capture video with your application, as detailed below.

1. **Open Camera** - Use the `Camera.open()` to get an instance of the camera object.
2. **Connect Preview** - Prepare a live camera image preview by connecting a `SurfaceView` to the camera using `Camera.setPreviewDisplay()`.
3. **Start Preview** - Call `Camera.startPreview()` to begin displaying the live camera images.
4. **Start Recording Video** - The following steps must be completed *in order* to successfully record video:
  - a. **Unlock the Camera** - Unlock the camera for use by `MediaRecorder` by calling `Camera.unlock()`.
  - b. **Configure MediaRecorder** - Call in the following `MediaRecorder` methods *in this order*. For more information, see the `MediaRecorder` reference documentation.
    1. `setCamera()` - Set the camera to be used for video capture, use your application's current instance of `Camera`.
    2. `set AudioSource()` - Set the audio source, use `MediaRecorder.AudioSource.CAMCORDER`.
    3. `set Video Source()` - Set the video source, use `MediaRecorder.VideoSource.CAMERA`.
  4. Set the video output format and encoding. For Android 2.2 (API Level 8) and higher, use the `MediaRecorder.setProfile` method, and get a profile instance using `CamcorderProfile.get()`. For versions of Android prior to 2.2, you must set the video output format and encoding parameters:
    - i. `setOutputFormat()` - Set the output format, specify the default setting or `MediaRecorder.OutputFormat.MPEG_4`.
    - ii. `setAudioEncoder()` - Set the sound encoding type, specify the default setting or `MediaRecorder.AudioEncoder.AMR_NB`.
    - iii. `setVideoEncoder()` - Set the video encoding type, specify the default setting or `MediaRecorder.VideoEncoder.MPEG_4_SP`.
  5. `setOutputFile()` - Set the output file, use `getOutputMediaFile(MEDIA_TYPE_VIDEO).toString()` from the example method in the [Saving Media Files](#) section.
  6. `setPreviewDisplay()` - Specify the `SurfaceView` preview layout element for your application. Use the same object you specified for **Connect Preview**.

**Caution:** You must call these `MediaRecorder` configuration methods *in this order*, otherwise your application will encounter errors

and the recording will fail.

- c. **Prepare MediaRecorder** - Prepare the `MediaRecorder` with provided configuration settings by calling `MediaRecorder.prepare()`.
  - d. **Start MediaRecorder** - Start recording video by calling `MediaRecorder.start()`.
5. **Stop Recording Video** - Call the following methods *in order*, to successfully complete a video recording:
- a. **Stop MediaRecorder** - Stop recording video by calling `MediaRecorder.stop()`.
  - b. **Reset MediaRecorder** - Optionally, remove the configuration settings from the recorder by calling `MediaRecorder.reset()`.
  - c. **Release MediaRecorder** - Release the `MediaRecorder` by calling `MediaRecorder.release()`.
- d. **Lock the Camera** - Lock the camera so that future `MediaRecorder` sessions can use it by calling `Camera.lock()`. Starting with Android 4.0 (API level 14), this call is not required unless the `MediaRecorder.prepare()` call fails.
6. **Stop the Preview** - When your activity has finished using the camera, stop the preview using `Camera.stopPreview()`.
7. **Release Camera** - Release the camera so that other applications can use it by calling `Camera.release()`.

**Note:** It is possible to use `MediaRecorder` without creating a camera preview first and skip the first few steps of this process. However, since users typically prefer to see a preview before starting a recording, that process is not discussed here.

**Tip:** If your application is typically used for recording video, set `setRecordingHint(boolean)` to `true` prior to starting your preview. This setting can help reduce the time it takes to start recording.

## Configuring MediaRecorder

When using the `MediaRecorder` class to record video, you must perform configuration steps in a *specific order* and then call the `MediaRecorder.prepare()` method to check and implement the configuration. The following example code demonstrates how to properly configure and prepare the `MediaRecorder` class for video recording.

```

private boolean prepareVideoRecorder(){

 mCamera = getCameraInstance();
 mMediaRecorder = new MediaRecorder();

 // Step 1: Unlock and set camera to MediaRecorder
 mCamera.unlock();
 mMediaRecorder.setCamera(mCamera);

 // Step 2: Set sources
 mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
 mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

 // Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
 mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));

 // Step 4: Set output file
 mMediaRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toString());

 // Step 5: Set the preview output
 mMediaRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());

 // Step 6: Prepare configured MediaRecorder
 try {
 mMediaRecorder.prepare();
 } catch (IllegalStateException e) {
 Log.d(TAG, "IllegalStateException preparing MediaRecorder: " + e.getMessage());
 releaseMediaRecorder();
 return false;
 } catch (IOException e) {
 Log.d(TAG, "IOException preparing MediaRecorder: " + e.getMessage());
 releaseMediaRecorder();
 return false;
 }
 return true;
}

```

Prior to Android 2.2 (API Level 8), you must set the output format and encoding formats parameters directly, instead of using [CamcorderProfile](#). This approach is demonstrated in the following code:

```

// Step 3: Set output format and encoding (for versions prior to API Level 8)
mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);

```

The following video recording parameters for [MediaRecorder](#) are given default settings, however, you may want to adjust these settings for your application:

- [setVideoEncodingBitRate\(\)](#)
- [setVideoSize\(\)](#)
- [setVideoFrameRate\(\)](#)
- [setAudioEncodingBitRate\(\)](#)
- [setAudioChannels\(\)](#)
- [setAudioSamplingRate\(\)](#)

## Starting and stopping MediaRecorder

When starting and stopping video recording using the [MediaRecorder](#) class, you must follow a specific order, as listed below.

1. Unlock the camera with [Camera.unlock\(\)](#)
2. Configure [MediaRecorder](#) as shown in the code example above
3. Start recording using [MediaRecorder.start\(\)](#)

4. Record the video
5. Stop recording using `MediaRecorder.stop()`
6. Release the media recorder with `MediaRecorder.release()`
7. Lock the camera using `Camera.lock()`

The following example code demonstrates how to wire up a button to properly start and stop video recording using the camera and the `MediaRecorder` class.

**Note:** When completing a video recording, do not release the camera or else your preview will be stopped.

```
private boolean isRecording = false;

// Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
 new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 if (isRecording) {
 // stop recording and release camera
 mMediaRecorder.stop(); // stop the recording
 releaseMediaRecorder(); // release the MediaRecorder object
 mCamera.lock(); // take camera access back from MediaRecorder

 // inform the user that recording has stopped
 setCaptureButtonText("Capture");
 isRecording = false;
 } else {
 // initialize video camera
 if (prepareVideoRecorder()) {
 // Camera is available and unlocked, MediaRecorder is prepared,
 // now you can start recording
 mMediaRecorder.start();

 // inform the user that recording has started
 setCaptureButtonText("Stop");
 isRecording = true;
 } else {
 // prepare didn't work, release the camera
 releaseMediaRecorder();
 // inform user
 }
 }
 }
 });

```

**Note:** In the above example, the `prepareVideoRecorder()` method refers to the example code shown in [Configuring MediaRecorder](#). This method takes care of locking the camera, configuring and preparing the `MediaRecorder` instance.

## Releasing the camera

Cameras are a resource that is shared by applications on a device. Your application can make use of the camera after getting an instance of `Camera`, and you must be particularly careful to release the camera object when your application stops using it, and as soon as your application is paused (`Activity.onPause()`). If your application does not properly release the camera, all subsequent attempts to access the camera, including those by your own application, will fail and may cause your or other applications to be shut down.

To release an instance of the `Camera` object, use the `Camera.release()` method, as shown in the example code below.

```

public class CameraActivity extends Activity {
 private Camera mCamera;
 private SurfaceView mPreview;
 private MediaRecorder mMediaRecorder;

 ...

 @Override
 protected void onPause() {
 super.onPause();
 releaseMediaRecorder(); // if you are using MediaRecorder, release it first
 releaseCamera(); // release the camera immediately on pause event
 }

 private void releaseMediaRecorder(){
 if (mMediaRecorder != null) {
 mMediaRecorder.reset(); // clear recorder configuration
 mMediaRecorder.release(); // release the recorder object
 mMediaRecorder = null;
 mCamera.lock(); // lock camera for later use
 }
 }

 private void releaseCamera(){
 if (mCamera != null){
 mCamera.release(); // release the camera for other applications
 mCamera = null;
 }
 }
}

```

**Caution:** If your application does not properly release the camera, all subsequent attempts to access the camera, including those by your own application, will fail and may cause your or other applications to be shut down.

## Saving media files

Media files created by users such as pictures and videos should be saved to a device's external storage directory (SD Card) to conserve system space and to allow users to access these files without their device. There are many possible directory locations to save media files on a device, however there are only two standard locations you should consider as a developer:

- [Environment.getExternalStoragePublicDirectory\(Environment.DIRECTORY\\_PICTURES\)](#) - This method returns the standard, shared and recommended location for saving pictures and videos. This directory is shared (public), so other applications can easily discover, read, change and delete files saved in this location. If your application is uninstalled by the user, media files saved to this location will not be removed. To avoid interfering with users existing pictures and videos, you should create a sub-directory for your application's media files within this directory, as shown in the code sample below. This method is available in Android 2.2 (API Level 8), for equivalent calls in earlier API versions, see [Saving Shared Files](#).
- [Context.getExternalFilesDir\(Environment.DIRECTORY\\_PICTURES\)](#) - This method returns a standard location for saving pictures and videos which are associated with your application. If your application is uninstalled, any files saved in this location are removed. Security is not enforced for files in this location and other applications may read, change and delete them.

The following example code demonstrates how to create a [File](#) or [Uri](#) location for a media file that can be used when invoking a device's camera with an [Intent](#) or as part of a [Building a Camera App](#).

```

public static final int MEDIA_TYPE_IMAGE = 1;
public static final int MEDIA_TYPE_VIDEO = 2;

/** Create a file Uri for saving an image or video */
private static Uri getOutputMediaFileUri(int type){
 return Uri.fromFile(getOutputMediaFile(type));
}

/** Create a File for saving an image or video */
private static File getOutputMediaFile(int type){
 // To be safe, you should check that the SDCard is mounted
 // using Environment.getExternalStorageState() before doing this.

 File mediaStorageDir = new File(Environment.getExternalStoragePublicDirectory(
 Environment.DIRECTORY_PICTURES), "MyCameraApp");
 // This location works best if you want the created images to be shared
 // between applications and persist after your app has been uninstalled.

 // Create the storage directory if it does not exist
 if (!mediaStorageDir.exists()){
 if (!mediaStorageDir.mkdirs()){
 Log.d("MyCameraApp", "failed to create directory");
 return null;
 }
 }

 // Create a media file name
 String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
 File mediaFile;
 if (type == MEDIA_TYPE_IMAGE){
 mediaFile = new File(mediaStorageDir.getPath() + File.separator +
 "IMG_" + timeStamp + ".jpg");
 } else if(type == MEDIA_TYPE_VIDEO) {
 mediaFile = new File(mediaStorageDir.getPath() + File.separator +
 "VID_" + timeStamp + ".mp4");
 } else {
 return null;
 }

 return mediaFile;
}

```

**Note:** `Environment.getExternalStoragePublicDirectory()` is available in Android 2.2 (API Level 8) or higher. If you are targeting devices with earlier versions of Android, use `Environment.getExternalStorageDirectory()` instead. For more information, see [Saving Shared Files](#).

To make the URI support work profiles, first [convert the file URI to a content URI](#). Then, add the content URI to `EXTRA_OUTPUT` of an [Intent](#).

For more information about saving files on an Android device, see [Data Storage](#).

## Camera features

Android supports a wide array of camera features you can control with your camera application, such as picture format, flash mode, focus settings, and many more. This section lists the common camera features, and briefly discusses how to use them. Most camera features can be accessed and set using the through `Camera.Parameters` object. However, there are several important features that require more than simple settings in `Camera.Parameters`. These features are covered in the following sections:

- [Metering and focus areas](#)
- [Face detection](#)
- [Time lapse video](#)

For general information about how to use features that are controlled through `Camera.Parameters`, review the [Using camera features](#) section. For more detailed information about how to use features controlled through the camera parameters object, follow the links in the feature list below to the API reference documentation.

**Table 1.** Common camera features sorted by the Android API Level in which they were introduced.

Feature	API Level	Description
Face Detection	14	Identify human faces within a picture and use them for focus, metering and white balance
Metering Areas	14	Specify one or more areas within an image for calculating white balance
Focus Areas	14	Set one or more areas within an image to use for focus
White Balance Lock	14	Stop or start automatic white balance adjustments
Exposure Lock	14	Stop or start automatic exposure adjustments
Video Snapshot	14	Take a picture while shooting video (frame grab)
Time Lapse Video	11	Record frames with set delays to record a time lapse video
Multiple Cameras	9	Support for more than one camera on a device, including front-facing and back-facing cameras
Focus Distance	9	Reports distances between the camera and objects that appear to be in focus
Zoom	8	Set image magnification
Exposure Compensation	8	Increase or decrease the light exposure level
GPS Data	5	Include or omit geographic location data with the image
White Balance	5	Set the white balance mode, which affects color values in the captured image
Focus Mode	5	Set how the camera focuses on a subject such as automatic, fixed, macro or infinity
Scene Mode	5	Apply a preset mode for specific types of photography situations such as night, beach, snow or candlelight scenes
JPEG Quality	5	Set the compression level for a JPEG image, which increases or decreases image output file quality and size
Flash Mode	5	Turn flash on, off, or use automatic setting
Color Effects	5	Apply a color effect to the captured image such as black and white, sepia tone or negative.
Anti-Banding	5	Reduces the effect of banding in color gradients due to JPEG compression
Picture Format	1	Specify the file format for the picture
Picture Size	1	Specify the pixel dimensions of the saved picture

**Note:** These features are not supported on all devices due to hardware differences and software implementation. For information on checking the availability of features on the device where your application is running, see [Checking feature availability](#).

## Checking feature availability

The first thing to understand when setting out to use camera features on Android devices is that not all camera features are supported on all devices. In addition, devices that support a particular feature may support them to different levels or with different options. Therefore, part of your decision process as you develop a camera application is to decide what camera features you want to support and to what level. After making that decision, you should plan on including code in your camera application that checks to see if device hardware supports those features and fails gracefully if a feature is not available.

You can check the availability of camera features by getting an instance of a camera's parameters object, and checking the relevant methods. The following code sample shows you how to obtain a `Camera.Parameters` object and check if the camera supports the autofocus feature:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();

List<String> focusModes = params.getSupportedFocusModes();
if (focusModes.contains(Camera.Parameters.FOCUS_MODE_AUTO)) {
 // Autofocus mode is supported
}
```

You can use the technique shown above for most camera features. The `Camera.Parameters` object provides a `getSupported...()`, `is...Supported()` or `getMax...()` method to determine if (and to what extent) a feature is supported.

If your application requires certain camera features in order to function properly, you can require them through additions to your application manifest. When you declare the use of specific camera features, such as flash and auto-focus, Google Play restricts your application from being installed on devices which do not support these features. For a list of camera features that can be declared in your app manifest, see the manifest [Features Reference](#).

## Using camera features

Most camera features are activated and controlled using a `Camera.Parameters` object. You obtain this object by first getting an instance of the `Camera` object, calling the `getParameters()` method, changing the returned parameter object and then setting it back into the camera object, as demonstrated in the following example code:

```
// get Camera parameters
Camera.Parameters params = mCamera.getParameters();
// set the focus mode
params.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
// set Camera parameters
mCamera.setParameters(params);
```

This technique works for nearly all camera features, and most parameters can be changed at any time after you have obtained an instance of the `Camera` object. Changes to parameters are typically visible to the user immediately in the application's camera preview. On the software side, parameter changes may take several frames to actually take effect as the camera hardware processes the new instructions and then sends updated image data.

**Important:** Some camera features cannot be changed at will. In particular, changing the size or orientation of the camera preview requires that you first stop the preview, change the preview size, and then restart the preview. Starting with Android 4.0 (API Level 14) preview orientation can be changed without restarting the preview.

Other camera features require more code in order to implement, including:

- Metering and focus areas
- Face detection
- Time lapse video

A quick outline of how to implement these features is provided in the following sections.

## Metering and focus areas

In some photographic scenarios, automatic focusing and light metering may not produce the desired results. Starting with Android 4.0 (API Level 14), your camera application can provide additional controls to allow your app or users to specify areas in an image to use for determining focus or light level settings and pass these values to the camera hardware for use in capturing images or video.

Areas for metering and focus work very similarly to other camera features, in that you control them through methods in the `Camera.Parameters` object. The following code demonstrates setting two light metering areas for an instance of `Camera`:

```

// Create an instance of Camera
mCamera = getCameraInstance();

// set Camera parameters
Camera.Parameters params = mCamera.getParameters();

if (params.getMaxNumMeteringAreas() > 0){ // check that metering areas are supported
 List<Camera.Area> meteringAreas = new ArrayList<Camera.Area>();

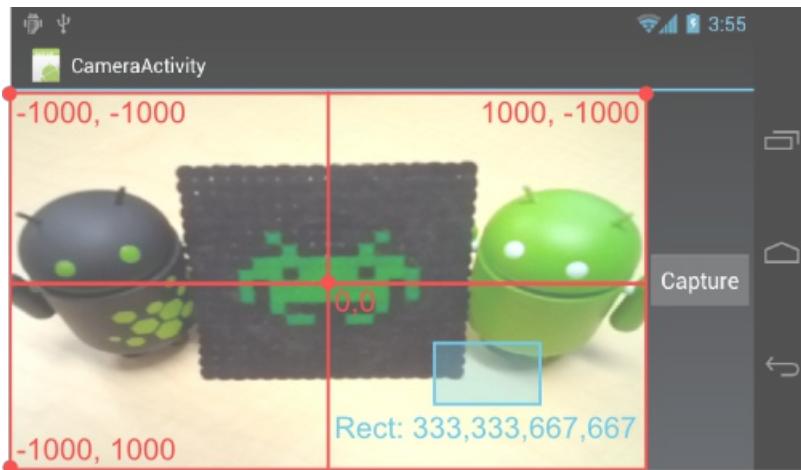
 Rect areaRect1 = new Rect(-100, -100, 100, 100); // specify an area in center of image
 meteringAreas.add(new Camera.Area(areaRect1, 600)); // set weight to 60%
 Rect areaRect2 = new Rect(800, -1000, 1000, -800); // specify an area in upper right of image
 meteringAreas.add(new Camera.Area(areaRect2, 400)); // set weight to 40%
 params.setMeteringAreas(meteringAreas);
}

mCamera.setParameters(params);

```

The `Camera.Area` object contains two data parameters: A `Rect` object for specifying an area within the camera's field of view and a weight value, which tells the camera what level of importance this area should be given in light metering or focus calculations.

The `Rect` field in a `Camera.Area` object describes a rectangular shape mapped on a 2000 x 2000 unit grid. The coordinates -1000, -1000 represent the top, left corner of the camera image, and coordinates 1000, 1000 represent the bottom, right corner of the camera image, as shown in the illustration below.



**Figure 1.** The red lines illustrate the coordinate system for specifying a `Camera.Area` within a camera preview. The blue box shows the location and shape of an camera area with the `Rect` values 333,333,667,667.

The bounds of this coordinate system always correspond to the outer edge of the image visible in the camera preview and do not shrink or expand with the zoom level. Similarly, rotation of the image preview using `Camera.setDisplayOrientation()` does not remap the coordinate system.

## Face detection

For pictures that include people, faces are usually the most important part of the picture, and should be used for determining both focus and white balance when capturing an image. The Android 4.0 (API Level 14) framework provides APIs for identifying faces and calculating picture settings using face recognition technology.

**Note:** While the face detection feature is running, `setWhiteBalance(String)`, `setFocusAreas(List<Camera.Area>)` and `setMeteringAreas(List<Camera.Area>)` have no effect.

Using the face detection feature in your camera application requires a few general steps:

- Check that face detection is supported on the device
- Create a face detection listener
- Add the face detection listener to your camera object
- Start face detection after preview (and after every preview restart)

The face detection feature is not supported on all devices. You can check that this feature is supported by calling `getMaxNumDetectedFaces()`. An example of this check is shown in the `startFaceDetection()` sample method below.

In order to be notified and respond to the detection of a face, your camera application must set a listener for face detection events. In order to do this, you must create a listener class that implements the `Camera.FaceDetectionListener` interface as shown in the example code below.

```
class MyFaceDetectionListener implements Camera.FaceDetectionListener {

 @Override
 public void onFaceDetection(Face[] faces, Camera camera) {
 if (faces.length > 0){
 Log.d("FaceDetection", "face detected: "+ faces.length +
 " Face 1 Location X: " + faces[0].rect.centerX() +
 "Y: " + faces[0].rect.centerY());
 }
 }
}
```

After creating this class, you then set it into your application's `Camera` object, as shown in the example code below:

```
mCamera.setFaceDetectionListener(new MyFaceDetectionListener());
```

Your application must start the face detection function each time you start (or restart) the camera preview. Create a method for starting face detection so you can call it as needed, as shown in the example code below.

```
public void startFaceDetection(){
 // Try starting Face Detection
 Camera.Parameters params = mCamera.getParameters();

 // start face detection only *after* preview has started
 if (params.getMaxNumDetectedFaces() > 0){
 // camera supports face detection, so can start it:
 mCamera.startFaceDetection();
 }
}
```

You must start face detection *each time* you start (or restart) the camera preview. If you use the preview class shown in [Creating a preview class](#), add your `startFaceDetection()` method to both the `surfaceCreated()` and `surfaceChanged()` methods in your preview class, as shown in the sample code below.

```

public void surfaceCreated(SurfaceHolder holder) {
 try {
 mCamera.setPreviewDisplay(holder);
 mCamera.startPreview();

 startFaceDetection(); // start face detection feature

 } catch (IOException e) {
 Log.d(TAG, "Error setting camera preview: " + e.getMessage());
 }
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {

 if (mHolder.getSurface() == null){
 // preview surface does not exist
 Log.d(TAG, "mHolder.getSurface() == null");
 return;
 }

 try {
 mCamera.stopPreview();

 } catch (Exception e){
 // ignore: tried to stop a non-existent preview
 Log.d(TAG, "Error stopping camera preview: " + e.getMessage());
 }

 try {
 mCamera.setPreviewDisplay(mHolder);
 mCamera.startPreview();

 startFaceDetection(); // re-start face detection feature

 } catch (Exception e){
 // ignore: tried to stop a non-existent preview
 Log.d(TAG, "Error starting camera preview: " + e.getMessage());
 }
}

```

**Note:** Remember to call this method *after* calling `startPreview()`. Do not attempt to start face detection in the `onCreate()` method of your camera app's main activity, as the preview is not available by this point in your application's the execution.

## Time lapse video

Time lapse video allows users to create video clips that combine pictures taken a few seconds or minutes apart. This feature uses `MediaRecorder` to record the images for a time lapse sequence.

To record a time lapse video with `MediaRecorder`, you must configure the recorder object as if you are recording a normal video, setting the captured frames per second to a low number and using one of the time lapse quality settings, as shown in the code example below.

```

// Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_TIME_LAPSE_HIGH));
...
// Step 5.5: Set the video capture rate to a low number
mMediaRecorder.setCaptureRate(0.1); // capture a frame every 10 seconds

```

These settings must be done as part of a larger configuration procedure for `MediaRecorder`. For a full configuration code example, see [Configuring MediaRecorder](#). Once the configuration is complete, you start the video recording as if you were recording a normal video clip. For more information about configuring and running `MediaRecorder`, see [Capturing videos](#).



# Location and Sensors APIs

Use sensors on the device to add rich location and motion capabilities to your app, from GPS or network location to accelerometer, gyroscope, temperature, barometer, and more.

TRAINING

---

## Making Your App Location Aware

This class teaches you how to incorporate location based services in your Android application. You'll learn a number of methods to receive location updates and related best practices.



# Location and Maps

In this document

- [Location Services](#)
- [Google Maps Android API](#)

**Note:** This is a guide to the *Android framework* location APIs in the package `android.location`. The Google Location Services API, part of Google Play Services, provides a more powerful, high-level framework that automates tasks such as location provider choice and power management. Location Services also provides new features such as activity detection that aren't available in the framework API. Developers who are using the framework API, as well as developers who are just now adding location-awareness to their apps, should strongly consider using the Location Services API.

To learn more about the Location Services API, see [Google Location Services for Android](#).

Location and maps-based apps offer a compelling experience on mobile devices. You can build these capabilities into your app using the classes of the `android.location` package and the Google Maps Android API. The sections below provide an introduction to how you can add the features.

## Location Services

Android gives your applications access to the location services supported by the device through classes in the `android.location` package. The central component of the location framework is the `LocationManager` system service, which provides APIs to determine location and bearing of the underlying device (if available).

As with other system services, you do not instantiate a `LocationManager` directly. Rather, you request an instance from the system by calling `getSystemService(Context.LOCATION_SERVICE)`. The method returns a handle to a new `LocationManager` instance.

Once your application has a `LocationManager`, your application is able to do three things:

- Query for the list of all `LocationProvider`s for the last known user location.
- Register/unregister for periodic updates of the user's current location from a location provider (specified either by criteria or name).
- Register/unregister for a given `Intent` to be fired if the device comes within a given proximity (specified by radius in meters) of a given lat/long.

For more information about acquiring the user location, read the [Location Strategies](#) guide.

## Google Maps Android API

With the [Google Maps Android API](#), you can add maps to your app that are based on Google Maps data. The API automatically handles access to Google Maps servers, data downloading, map display, and touch gestures on the map. You can also use API calls to add markers, polygons and overlays, and to change the user's view of a particular map area.

The key class in the Google Maps Android API is `MapView`. A `MapView` displays a map with data obtained from the Google Maps service. When the `MapView` has focus, it will capture keypresses and touch gestures to pan and zoom the map automatically, including handling network requests for additional maps tiles. It also provides all of the UI elements necessary for users to control the map. Your application can also use `MapView` class methods to control the map programmatically and draw a number of overlays on top of the map.

The Google Maps Android APIs are not included in the Android platform, but are available on any device with the Google Play Store running Android 2.2 or higher, through [Google Play services](#).

To integrate Google Maps into your app, you need to install the Google Play services libraries for your Android SDK. For more details, read about [Google Play services](#).

# Location Strategies

## In this document

- › [Challenges in Determining User Location](#)
- › [Requesting Location Updates](#)
  - › [Requesting User Permissions](#)
- › [Defining a Model for the Best Performance](#)
  - › [Flow for obtaining user location](#)
  - › [Deciding when to start listening for updates](#)
  - › [Getting a fast fix with the last known location](#)
  - › [Deciding when to stop listening for updates](#)
  - › [Maintaining a current best estimate](#)
  - › [Adjusting the model to save battery and data exchange](#)
- › [Providing Mock Location Data](#)

## Key classes

- › [LocationManager](#)
- › [LocationListener](#)

**Note:** The strategies described in this guide apply to the platform location API in `android.location`. The Google Location Services API, part of Google Play Services, provides a more powerful, high-level framework that automatically handles location providers, user movement, and location accuracy. It also handles location update scheduling based on power consumption parameters you provide. In most cases, you'll get better battery performance, as well as more appropriate accuracy, by using the Location Services API.

To learn more about the Location Services API, see [Google Location Services for Android](#).

Knowing where the user is allows your application to be smarter and deliver better information to the user. When developing a location-aware application for Android, you can utilize GPS and Android's Network Location Provider to acquire the user location. Although GPS is most accurate, it only works outdoors, it quickly consumes battery power, and doesn't return the location as quickly as users want. Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information in a way that works indoors and outdoors, responds faster, and uses less battery power. To obtain the user location in your application, you can use both GPS and the Network Location Provider, or just one.

## Challenges in Determining User Location

Obtaining user location from a mobile device can be complicated. There are several reasons why a location reading (regardless of the source) can contain errors and be inaccurate. Some sources of error in the user location include:

- **Multitude of location sources**

GPS, Cell-ID, and Wi-Fi can each provide a clue to users location. Determining which to use and trust is a matter of trade-offs in accuracy, speed, and battery-efficiency.

- **User movement**

Because the user location changes, you must account for movement by re-estimating user location every so often.

- **Varying accuracy**

Location estimates coming from each location source are not consistent in their accuracy. A location obtained 10 seconds ago from one source might be more accurate than the newest location from another or same source.

These problems can make it difficult to obtain a reliable user location reading. This document provides information to help you meet these challenges to obtain a reliable location reading. It also provides ideas that you can use in your application to provide the user with an accurate and responsive geo-location experience.

## Requesting Location Updates

Before addressing some of the location errors described above, here is an introduction to how you can obtain user location on Android.

Getting user location in Android works by means of callback. You indicate that you'd like to receive location updates from the [LocationManager](#) ("Location Manager") by calling `requestLocationUpdates()`, passing it a [LocationListener](#). Your [LocationListener](#) must implement several callback methods that the Location Manager calls when the user location changes or when the status of the service changes.

For example, the following code shows how to define a [LocationListener](#) and request location updates:

```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
 public void onLocationChanged(Location location) {
 // Called when a new location is found by the network location provider.
 makeUseOfNewLocation(location);
 }

 public void onStatusChanged(String provider, int status, Bundle extras) {}

 public void onProviderEnabled(String provider) {}

 public void onProviderDisabled(String provider) {}
};

// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
```

The first parameter in `requestLocationUpdates()` is the type of location provider to use (in this case, the Network Location Provider for cell tower and Wi-Fi based location). You can control the frequency at which your listener receives updates with the second and third parameter—the second is the minimum time interval between notifications and the third is the minimum change in distance between notifications—setting both to zero requests location notifications as frequently as possible. The last parameter is your [LocationListener](#), which receives callbacks for location updates.

To request location updates from the GPS provider, use `GPS_PROVIDER` instead of `NETWORK_PROVIDER`. You can also request location updates from both the GPS and the Network Location Provider by calling `requestLocationUpdates()` twice—once for `NETWORK_PROVIDER` and once for `GPS_PROVIDER`.

## Requesting User Permissions

In order to receive location updates from `NETWORK_PROVIDER` or `GPS_PROVIDER`, you must request the user's permission by declaring either the `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION` permission, respectively, in your Android manifest file. Without these permissions, your application will fail at runtime when requesting location updates.

If you are using both `NETWORK_PROVIDER` and `GPS_PROVIDER`, then you need to request only the `ACCESS_FINE_LOCATION` permission, because it includes permission for both providers. Permission for `ACCESS_COARSE_LOCATION` allows access only to `NETWORK_PROVIDER`.

**Caution:** If your app targets Android 5.0 (API level 21) or higher, you *must* declare that your app uses the `android.hardware.location.network` or `android.hardware.location.gps` hardware feature in the manifest file, depending on whether your app receives location updates from `NETWORK_PROVIDER` or from `GPS_PROVIDER`. If your app receives location information from either of these location provider sources, you need to declare that the app uses these hardware features in your app manifest. On devices running versions prior to Android 5.0 (API 21), requesting the `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION` permission

includes an implied request for location hardware features. However, requesting those permissions *does not* automatically request location hardware features on Android 5.0 (API level 21) and higher.

The following code sample demonstrates how to declare the permission and hardware feature in the manifest file of an app that reads data from the device's GPS:

```
<manifest ... >
 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
 ...
 <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
 <uses-feature android:name="android.hardware.location.gps" />
 ...
</manifest>
```

## Defining a Model for the Best Performance

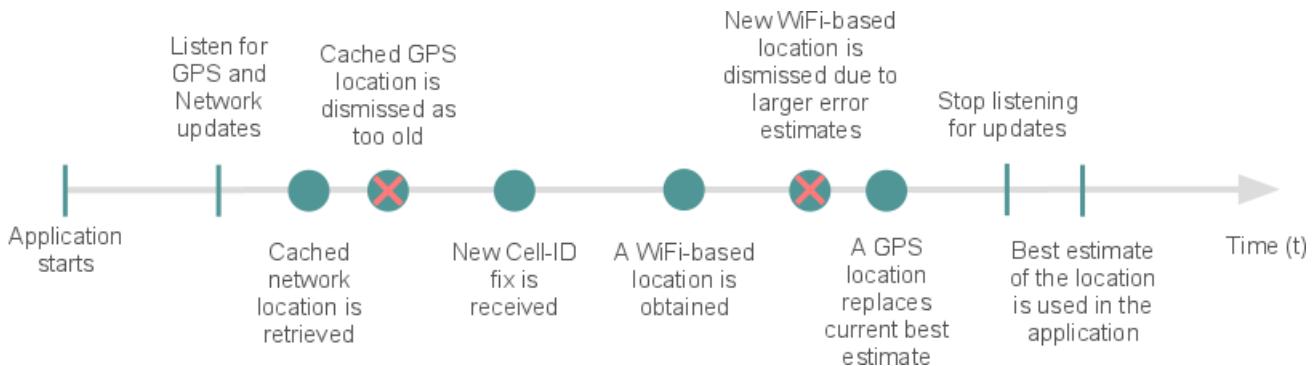
Location-based applications are now commonplace, but due to the less than optimal accuracy, user movement, the multitude of methods to obtain the location, and the desire to conserve battery, getting user location is complicated. To overcome the obstacles of obtaining a good user location while preserving battery power, you must define a consistent model that specifies how your application obtains the user location. This model includes when you start and stop listening for updates and when to use cached location data.

### Flow for obtaining user location

Here's the typical flow of procedures for obtaining the user location:

1. Start application.
2. Sometime later, start listening for updates from desired location providers.
3. Maintain a "current best estimate" of location by filtering out new, but less accurate fixes.
4. Stop listening for location updates.
5. Take advantage of the last best location estimate.

Figure 1 demonstrates this model in a timeline that visualizes the period in which an application is listening for location updates and the events that occur during that time.



**Figure 1.** A timeline representing the window in which an application listens for location updates.

This model of a window—during which location updates are received—frames many of the decisions you need to make when adding location-based services to your application.

### Deciding when to start listening for updates

You might want to start listening for location updates as soon as your application starts, or only after users activate a certain feature. Be aware that long windows of listening for location fixes can consume a lot of battery power, but short periods might not allow for sufficient accuracy.

As demonstrated above, you can begin listening for updates by calling `requestLocationUpdates()`:

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or, use GPS location data:
// String locationProvider = LocationManager.GPS_PROVIDER;

locationManager.requestLocationUpdates(locationProvider, 0, 0, locationListener);
```

## Getting a fast fix with the last known location

The time it takes for your location listener to receive the first location fix is often too long for users wait. Until a more accurate location is provided to your location listener, you should utilize a cached location by calling `getLastKnownLocation(String)`:

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or use LocationManager.GPS_PROVIDER

Location lastKnownLocation = locationManager.getLastKnownLocation(locationProvider);
```

## Deciding when to stop listening for updates

The logic of deciding when new fixes are no longer necessary might range from very simple to very complex depending on your application. A short gap between when the location is acquired and when the location is used, improves the accuracy of the estimate. Always beware that listening for a long time consumes a lot of battery power, so as soon as you have the information you need, you should stop listening for updates by calling `removeUpdates(PendingIntent)`:

```
// Remove the listener you previously added
locationManager.removeUpdates(locationListener);
```

## Maintaining a current best estimate

You might expect that the most recent location fix is the most accurate. However, because the accuracy of a location fix varies, the most recent fix is not always the best. You should include logic for choosing location fixes based on several criteria. The criteria also varies depending on the use-cases of the application and field testing.

Here are a few steps you can take to validate the accuracy of a location fix:

- Check if the location retrieved is significantly newer than the previous estimate.
- Check if the accuracy claimed by the location is better or worse than the previous estimate.
- Check which provider the new location is from and determine if you trust it more.

An elaborate example of this logic can look something like this:

```

private static final int TWO_MINUTES = 1000 * 60 * 2;

/** Determines whether one Location reading is better than the current Location fix
 * @param location The new Location that you want to evaluate
 * @param currentBestLocation The current Location fix, to which you want to compare the new one
 */
protected boolean isBetterLocation(Location location, Location currentBestLocation) {
 if (currentBestLocation == null) {
 // A new location is always better than no location
 return true;
 }

 // Check whether the new location fix is newer or older
 long timeDelta = location.getTime() - currentBestLocation.getTime();
 boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
 boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
 boolean isNewer = timeDelta > 0;

 // If it's been more than two minutes since the current location, use the new location
 // because the user has likely moved
 if (isSignificantlyNewer) {
 return true;
 } else if (isSignificantlyOlder) {
 return false;
 }

 // Check whether the new location fix is more or less accurate
 int accuracyDelta = (int) (location.getAccuracy() - currentBestLocation.getAccuracy());
 boolean isLessAccurate = accuracyDelta > 0;
 boolean isMoreAccurate = accuracyDelta < 0;
 boolean isSignificantlyLessAccurate = accuracyDelta > 200;

 // Check if the old and new location are from the same provider
 boolean isFromSameProvider = isSameProvider(location.getProvider(),
 currentBestLocation.getProvider());

 // Determine location quality using a combination of timeliness and accuracy
 if (isMoreAccurate) {
 return true;
 } else if (isNewer && !isLessAccurate) {
 return true;
 } else if (isNewer && !isSignificantlyLessAccurate && isFromSameProvider) {
 return true;
 }
 return false;
}

/** Checks whether two providers are the same */
private boolean isSameProvider(String provider1, String provider2) {
 if (provider1 == null) {
 return provider2 == null;
 }
 return provider1.equals(provider2);
}

```

## Adjusting the model to save battery and data exchange

As you test your application, you might find that your model for providing good location and good performance needs some adjustment. Here are some things you might change to find a good balance between the two.

### Reduce the size of the window

A smaller window in which you listen for location updates means less interaction with GPS and network location services, thus, preserving battery life. But it also allows for fewer locations from which to choose a best estimate.

### Set the location providers to return updates less frequently

Reducing the rate at which new updates appear during the window can also improve battery efficiency, but at the cost of accuracy. The value

of the trade-off depends on how your application is used. You can reduce the rate of updates by increasing the parameters in `requestLocationUpdates()` that specify the interval time and minimum distance change.

## Restrict a set of providers

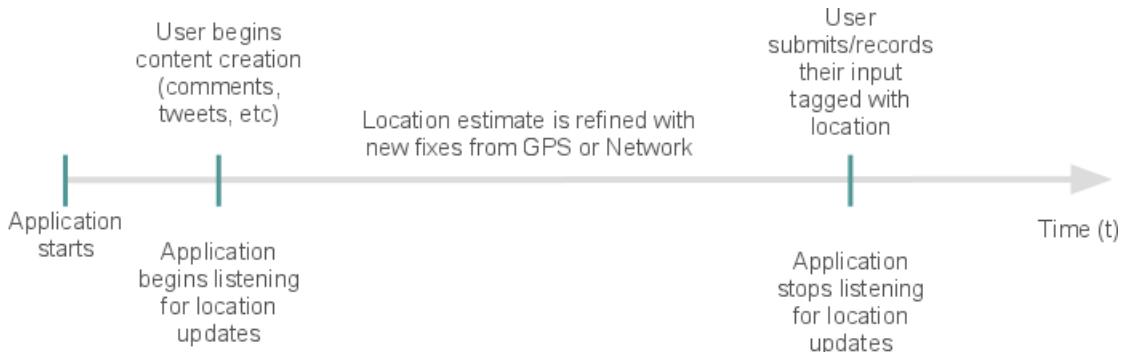
Depending on the environment where your application is used or the desired level of accuracy, you might choose to use only the Network Location Provider or only GPS, instead of both. Interacting with only one of the services reduces battery usage at a potential cost of accuracy.

# Common application cases

There are many reasons you might want to obtain the user location in your application. Below are a couple scenarios in which you can use the user location to enrich your application. Each scenario also describes good practices for when you should start and stop listening for the location, in order to get a good reading and help preserve battery life.

## Tagging user-created content with a location

You might be creating an application where user-created content is tagged with a location. Think of users sharing their local experiences, posting a review for a restaurant, or recording some content that can be augmented with their current location. A model of how this interaction might happen, with respect to the location services, is visualized in figure 2.



**Figure 2.** A timeline representing the window in which the user location is obtained and listening stops when the user consumes the current location.

This lines up with the previous model of how user location is obtained in code (figure 1). For best location accuracy, you might choose to start listening for location updates when users begin creating the content or even when the application starts, then stop listening for updates when content is ready to be posted or recorded. You might need to consider how long a typical task of creating the content takes and judge if this duration allows for efficient collection of a location estimate.

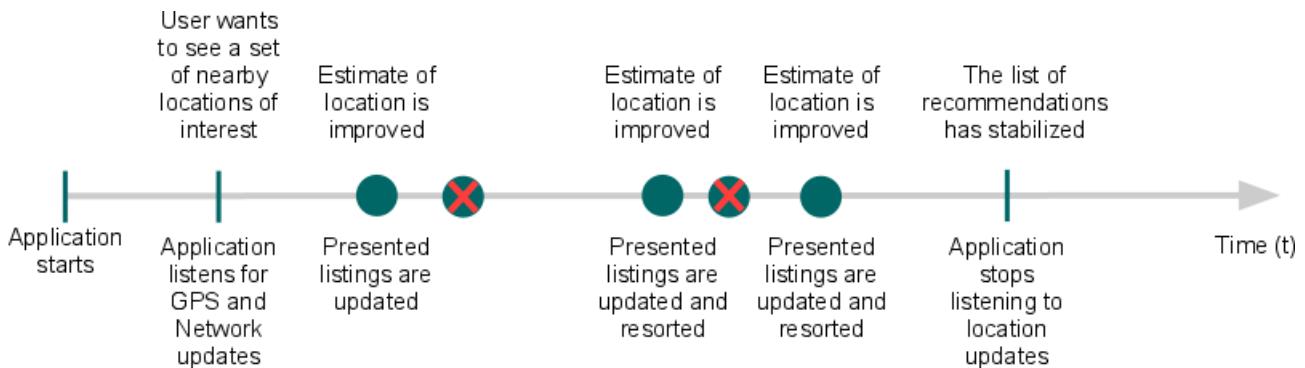
## Helping the user decide on where to go

You might be creating an application that attempts to provide users with a set of options about where to go. For example, you're looking to provide a list of nearby restaurants, stores, and entertainment and the order of recommendations changes depending on the user location.

To accommodate such a flow, you might choose to:

- Rearrange recommendations when a new best estimate is obtained
- Stop listening for updates if the order of recommendations has stabilized

This kind of model is visualized in figure 3.



**Figure 3.** A timeline representing the window in which a dynamic set of data is updated each time the user location updates.

## Providing Mock Location Data

As you develop your application, you'll certainly need to test how well your model for obtaining user location works. This is most easily done using a real Android-powered device. If, however, you don't have a device, you can still test your location-based features by mocking location data in the Android emulator. There are three different ways to send your application mock location data: using Android Studio, DDMS, or the "geo" command in the emulator console.

**Note:** Providing mock location data is injected as GPS location data, so you must request location updates from [GPS\\_PROVIDER](#) in order for mock location data to work.

## Using Android Studio

Select **Tools > Android > AVD Manager**. In the Android Virtual Device Manager window, choose your AVD and launch it in the emulator by selecting the green play arrow in the Actions column.

Then, select **Tools > Android > Android Device Monitor**. Select the Emulator Control tab in the Android Device Monitor window, and enter GPS coordinates under Location Controls as individual lat/long coordinates, with a GPX file for route playback, or a KML file for multiple place marks.

## Using DDMS

With the DDMS tool, you can simulate location data a few different ways:

- Manually send individual longitude/latitude coordinates to the device.
- Use a GPX file describing a route for playback to the device.
- Use a KML file describing individual place marks for sequenced playback to the device.

For more information on using DDMS to spoof location data, see [Using DDMS](#).

## Using the "geo" command in the emulator console

To send mock location data from the command line:

1. Launch your application in the Android emulator and open a terminal/console in your SDK's `/tools` directory.
2. Connect to the emulator console:

```
telnet localhost <console-port>
```

3. Send the location data:

- `geo fix` to send a fixed geo-location.

This command accepts a longitude and latitude in decimal degrees, and an optional altitude in meters. For example:

```
geo fix -121.45356 46.51119 4392
```

- `geo nmea` to send an NMEA 0183 sentence.

This command accepts a single NMEA sentence of type '\$GPGGA' (fix data) or '\$GPRMC' (transit data). For example:

```
geo nmea $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
```

For information about how to connect to the emulator console, see [Using the Emulator Console](#).

# Sensors Overview

## Quickview

- › Learn about the sensors that Android supports and the Android sensor framework.
- › Find out how to list sensors, determine sensor capabilities, and monitor sensor data.
- › Learn about best practices for accessing and using sensors.

## In this document

- › [Introduction to Sensors](#)
- › [Identifying Sensors and Sensor Capabilities](#)
- › [Monitoring Sensor Events](#)
- › [Handling Different Sensor Configurations](#)
- › [Sensor Coordinate System](#)
- › [Best Practices for Accessing and Using Sensors](#)

## Key classes and interfaces

- › [Sensor](#)
- › [SensorEvent](#)
- › [SensorManager](#)
- › [SensorEventListener](#)

## Related samples

- › [Accelerometer Play](#)
- › [API Demos \(OS - RotationVectorDemo\)](#)
- › [API Demos \(OS - Sensors\)](#)

## See also

- › [Sensors](#)
- › [Motion Sensors](#)
- › [Position Sensors](#)
- › [Environment Sensors](#)

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

The Android platform supports three broad categories of sensors:

- Motion sensors

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

- Environmental sensors

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

- Position sensors

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

You can access sensors available on the device and acquire raw sensor data by using the Android sensor framework. The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks. For example, you can use the sensor framework to do the following:

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

This topic provides an overview of the sensors that are available on the Android platform. It also provides an introduction to the sensor framework.

## Introduction to Sensors

The Android sensor framework lets you access many types of sensors. Some of these sensors are hardware-based and some are software-based. Hardware-based sensors are physical components built into a handset or tablet device. They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change. Software-based sensors are not physical devices, although they mimic hardware-based sensors. Software-based sensors derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors. The linear acceleration sensor and the gravity sensor are examples of software-based sensors. Table 1 summarizes the sensors that are supported by the Android platform.

Few Android-powered devices have every type of sensor. For example, most handset devices and tablets have an accelerometer and a magnetometer, but fewer devices have barometers or thermometers. Also, a device can have more than one sensor of a given type. For example, a device can have two gravity sensors, each one having a different range.

**Table 1.** Sensor types supported by the Android platform.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s <sup>2</sup> that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s <sup>2</sup> that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or	Measures the acceleration force in m/s <sup>2</sup> that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration

	Hardware		along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in $\mu\text{T}$ .	Creating a compass.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <code>getRotationMatrix()</code> method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius ( $^{\circ}\text{C}$ ). This sensor implementation varies across devices and this sensor was replaced with the <code>TYPE_AMBIENT_TEMPERATURE</code> sensor in API Level 14	Monitoring temperatures.

## Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the `android.hardware` package and includes the following classes and interfaces:

### [SensorManager](#)

You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

### [Sensor](#)

You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

### [SensorEvent](#)

The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

### [SensorEventListener](#)

You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or

when sensor accuracy changes.

In a typical application you use these sensor-related APIs to perform two basic tasks:

- **Identifying sensors and sensor capabilities**

Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. For example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.

- **Monitor sensor events**

Monitoring sensor events is how you acquire raw sensor data. A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information: the name of the sensor that triggered the event, the timestamp for the event, the accuracy of the event, and the raw sensor data that triggered the event.

## Sensor Availability

While sensor availability varies from device to device, it can also vary between Android versions. This is because the Android sensors have been introduced over the course of several platform releases. For example, many sensors were introduced in Android 1.5 (API Level 3), but some were not implemented and were not available for use until Android 2.3 (API Level 9). Likewise, several sensors were introduced in Android 2.3 (API Level 9) and Android 4.0 (API Level 14). Two sensors have been deprecated and replaced by newer, better sensors.

Table 2 summarizes the availability of each sensor on a platform-by-platform basis. Only four platforms are listed because those are the platforms that involved sensor changes. Sensors that are listed as deprecated are still available on subsequent platforms (provided the sensor is present on a device), which is in line with Android's forward compatibility policy.

**Table 2.** Sensor availability by platform.

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
TYPE_AMBIENT_TEMPERATURE	<b>Yes</b>	n/a	n/a	n/a
TYPE_GRAVITY	<b>Yes</b>	<b>Yes</b>	n/a	n/a
TYPE_GYROSCOPE	<b>Yes</b>	<b>Yes</b>	n/a <sup>1</sup>	n/a <sup>1</sup>
TYPE_LIGHT	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
TYPE_LINEAR_ACCELERATION	<b>Yes</b>	<b>Yes</b>	n/a	n/a
TYPE_MAGNETIC_FIELD	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
TYPE_ORIENTATION	<b>Yes</b> <sup>2</sup>	<b>Yes</b> <sup>2</sup>	<b>Yes</b> <sup>2</sup>	<b>Yes</b>
TYPE_PRESSURE	<b>Yes</b>	<b>Yes</b>	n/a <sup>1</sup>	n/a <sup>1</sup>
TYPE_PROXIMITY	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
TYPE_RELATIVE_HUMIDITY	<b>Yes</b>	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	<b>Yes</b>	<b>Yes</b>	n/a	n/a
TYPE_TEMPERATURE	<b>Yes</b> <sup>2</sup>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>

<sup>1</sup> This sensor type was added in Android 1.5 (API Level 3), but it was not available for use until Android 2.3 (API Level 9).

<sup>2</sup> This sensor is available, but it has been deprecated.

# Identifying Sensors and Sensor Capabilities

The Android sensor framework provides several methods that make it easy for you to determine at runtime which sensors are on a device. The API also provides methods that let you determine the capabilities of each sensor, such as its maximum range, its resolution, and its power requirements.

To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the [SensorManager](#) class by calling the `getSystemService()` method and passing in the `SENSOR_SERVICE` argument. For example:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Next, you can get a listing of every sensor on a device by calling the `getSensorList()` method and using the `TYPE_ALL` constant. For example:

```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

If you want to list all of the sensors of a given type, you could use another constant instead of `TYPE_ALL` such as `TYPE_GYROSCOPE`, `TYPE_LINEAR_ACCELERATION`, or `TYPE_GRAVITY`.

You can also determine whether a specific type of sensor exists on a device by using the `getDefaultSensor()` method and passing in the type constant for a specific sensor. If a device has more than one sensor of a given type, one of the sensors must be designated as the default sensor. If a default sensor does not exist for a given type of sensor, the method call returns null, which means the device does not have that type of sensor. For example, the following code checks whether there's a magnetometer on a device:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
 // Success! There's a magnetometer.
}
else {
 // Failure! No magnetometer.
}
```

**Note:** Android does not require device manufacturers to build any particular types of sensors into their Android-powered devices, so devices can have a wide range of sensor configurations.

In addition to listing the sensors that are on a device, you can use the public methods of the `Sensor` class to determine the capabilities and attributes of individual sensors. This is useful if you want your application to behave differently based on which sensors or sensor capabilities are available on a device. For example, you can use the `getResolution()` and `getMaximumRange()` methods to obtain a sensor's resolution and maximum range of measurement. You can also use the `getPower()` method to obtain a sensor's power requirements.

Two of the public methods are particularly useful if you want to optimize your application for different manufacturer's sensors or different versions of a sensor. For example, if your application needs to monitor user gestures such as tilt and shake, you could create one set of data filtering rules and optimizations for newer devices that have a specific vendor's gravity sensor, and another set of data filtering rules and optimizations for devices that do not have a gravity sensor and have only an accelerometer. The following code sample shows you how you can use the `getVendor()` and `getVersion()` methods to do this. In this sample, we're looking for a gravity sensor that lists Google Inc. as the vendor and has a version number of 3. If that particular sensor is not present on the device, we try to use the accelerometer.

```

private SensorManager mSensorManager;
private Sensor mSensor;

...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = null;

if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
 List<Sensor> gravSensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
 for(int i=0; i<gravSensors.size(); i++) {
 if ((gravSensors.get(i).getVendor().contains("Google Inc.")) &&
 (gravSensors.get(i).getVersion() == 3)){
 // Use the version 3 gravity sensor.
 mSensor = gravSensors.get(i);
 }
 }
}
if (mSensor == null){
 // Use the accelerometer.
 if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
 mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
 }
 else{
 // Sorry, there are no accelerometers on your device.
 // You can't play this game.
 }
}

```

Another useful method is the `getMinDelay()` method, which returns the minimum time interval (in microseconds) a sensor can use to sense data. Any sensor that returns a non-zero value for the `getMinDelay()` method is a streaming sensor. Streaming sensors sense data at regular intervals and were introduced in Android 2.3 (API Level 9). If a sensor returns zero when you call the `getMinDelay()` method, it means the sensor is not a streaming sensor because it reports data only when there is a change in the parameters it is sensing.

The `getMinDelay()` method is useful because it lets you determine the maximum rate at which a sensor can acquire data. If certain features in your application require high data acquisition rates or a streaming sensor, you can use this method to determine whether a sensor meets those requirements and then enable or disable the relevant features in your application accordingly.

**Caution:** A sensor's maximum data acquisition rate is not necessarily the rate at which the sensor framework delivers sensor data to your application. The sensor framework reports data through sensor events, and several factors influence the rate at which your application receives sensor events. For more information, see [Monitoring Sensor Events](#).

## Monitoring Sensor Events

To monitor raw sensor data you need to implement two callback methods that are exposed through the `SensorEventListener` interface: `onAccuracyChanged()` and `onSensorChanged()`. The Android system calls these methods whenever the following occurs:

- **A sensor's accuracy changes.**

In this case the system invokes the `onAccuracyChanged()` method, providing you with a reference to the `Sensor` object that changed and the new accuracy of the sensor. Accuracy is represented by one of four status constants: `SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`, `SENSOR_STATUS_ACCURACY_HIGH`, or `SENSOR_STATUS_UNRELIABLE`.

- **A sensor reports a new value.**

In this case the system invokes the `onSensorChanged()` method, providing you with a `SensorEvent` object. A `SensorEvent` object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

The following code shows how to use the `onSensorChanged()` method to monitor data from the light sensor. This example displays the raw sensor data in a `TextView` that is defined in the main.xml file as `sensor_data`.

```

public class SensorActivity extends Activity implements SensorEventListener {
 private SensorManager mSensorManager;
 private Sensor mLight;

 @Override
 public final void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
 mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
 }

 @Override
 public final void onAccuracyChanged(Sensor sensor, int accuracy) {
 // Do something here if sensor accuracy changes.
 }

 @Override
 public final void onSensorChanged(SensorEvent event) {
 // The light sensor returns a single value.
 // Many sensors return 3 values, one for each axis.
 float lux = event.values[0];
 // Do something with this sensor value.
 }

 @Override
 protected void onResume() {
 super.onResume();
 mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
 }

 @Override
 protected void onPause() {
 super.onPause();
 mSensorManager.unregisterListener(this);
 }
}

```

In this example, the default data delay (`SENSOR_DELAY_NORMAL`) is specified when the `registerListener()` method is invoked. The data delay (or sampling rate) controls the interval at which sensor events are sent to your application via the `onSensorChanged()` callback method. The default data delay is suitable for monitoring typical screen orientation changes and uses a delay of 200,000 microseconds. You can specify other data delays, such as `SENSOR_DELAY_GAME` (20,000 microsecond delay), `SENSOR_DELAY_UI` (60,000 microsecond delay), or `SENSOR_DELAY_FASTEST` (0 microsecond delay). As of Android 3.0 (API Level 11) you can also specify the delay as an absolute value (in microseconds).

The delay that you specify is only a suggested delay. The Android system and other applications can alter this delay. As a best practice, you should specify the largest delay that you can because the system typically uses a smaller delay than the one you specify (that is, you should choose the slowest sampling rate that still meets the needs of your application). Using a larger delay imposes a lower load on the processor and therefore uses less power.

There is no public method for determining the rate at which the sensor framework is sending sensor events to your application; however, you can use the timestamps that are associated with each sensor event to calculate the sampling rate over several events. You should not have to change the sampling rate (delay) once you set it. If for some reason you do need to change the delay, you will have to unregister and reregister the sensor listener.

It's also important to note that this example uses the `onResume()` and `onPause()` callback methods to register and unregister the sensor event listener. As a best practice you should always disable sensors you don't need, especially when your activity is paused. Failing to do so can drain the battery in just a few hours because some sensors have substantial power requirements and can use up battery power quickly. The system will not disable sensors automatically when the screen turns off.

# Handling Different Sensor Configurations

Android does not specify a standard sensor configuration for devices, which means device manufacturers can incorporate any sensor configuration that they want into their Android-powered devices. As a result, devices can include a variety of sensors in a wide range of configurations. If your application relies on a specific type of sensor, you have to ensure that the sensor is present on a device so your app can run successfully.

You have two options for ensuring that a given sensor is present on a device:

- Detect sensors at runtime and enable or disable application features as appropriate.
- Use Google Play filters to target devices with specific sensor configurations.

Each option is discussed in the following sections.

## Detecting sensors at runtime

If your application uses a specific type of sensor, but doesn't rely on it, you can use the sensor framework to detect the sensor at runtime and then disable or enable application features as appropriate. For example, a navigation application might use the temperature sensor, pressure sensor, GPS sensor, and geomagnetic field sensor to display the temperature, barometric pressure, location, and compass bearing. If a device doesn't have a pressure sensor, you can use the sensor framework to detect the absence of the pressure sensor at runtime and then disable the portion of your application's UI that displays pressure. For example, the following code checks whether there's a pressure sensor on a device:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){
 // Success! There's a pressure sensor.
}
else {
 // Failure! No pressure sensor.
}
```

## Using Google Play filters to target specific sensor configurations

If you are publishing your application on Google Play you can use the `<uses-feature>` element in your manifest file to filter your application from devices that do not have the appropriate sensor configuration for your application. The `<uses-feature>` element has several hardware descriptors that let you filter applications based on the presence of specific sensors. The sensors you can list include: accelerometer, barometer, compass (geomagnetic field), gyroscope, light, and proximity. The following is an example manifest entry that filters apps that do not have an accelerometer:

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
 android:required="true" />
```

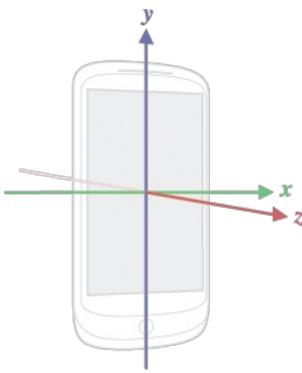
If you add this element and descriptor to your application's manifest, users will see your application on Google Play only if their device has an accelerometer.

You should set the descriptor to `android:required="true"` only if your application relies entirely on a specific sensor. If your application uses a sensor for some functionality, but still runs without the sensor, you should list the sensor in the `<uses-feature>` element, but set the descriptor to `android:required="false"`. This helps ensure that devices can install your app even if they do not have that particular sensor. This is also a project management best practice that helps you keep track of the features your application uses. Keep in mind, if your application uses a particular sensor, but still runs without the sensor, then you should detect the sensor at runtime and disable or enable application features as appropriate.

# Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the

screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:



**Figure 1.** Coordinate system (relative to a device) that's used by the Sensor API.

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Finally, if your application matches sensor data to the on-screen display, you need to use the [getRotation\(\)](#) method to determine screen rotation, and then use the [remapCoordinateSystem\(\)](#) method to map sensor coordinates to screen coordinates. You need to do this even if your manifest specifies portrait-only display.

**Note:** Some sensors and methods use a coordinate system that is relative to the world's frame of reference (as opposed to the device's frame of reference). These sensors and methods return data that represent device motion or device position relative to the earth. For more information, see the [getOrientation\(\)](#) method, the [getRotationMatrix\(\)](#) method, [Orientation Sensor](#), and [Rotation Vector Sensor](#).

## Best Practices for Accessing and Using Sensors

As you design your sensor implementation, be sure to follow the guidelines that are discussed in this section. These guidelines are recommended best practices for anyone who is using the sensor framework to access sensors and acquire sensor data.

### Unregister sensor listeners

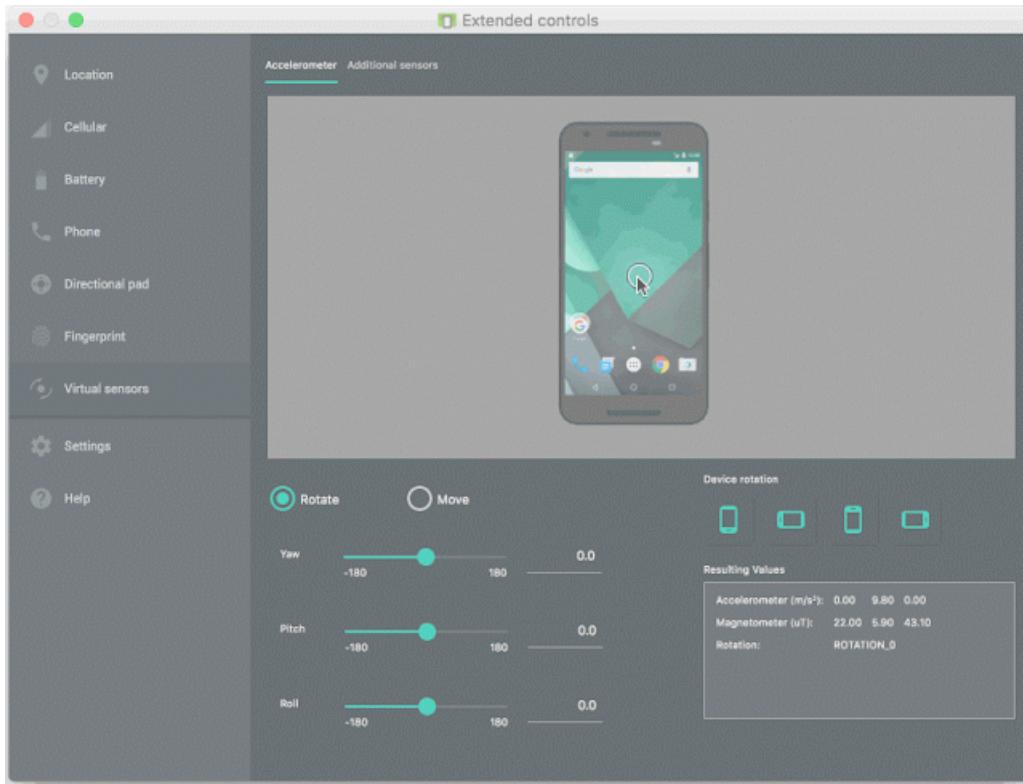
Be sure to unregister a sensor's listener when you are done using the sensor or when the sensor activity pauses. If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor. The following code shows how to use the [onPause\(\)](#) method to unregister a listener:

```
private SensorManager mSensorManager;
...
@Override
protected void onPause() {
 super.onPause();
 mSensorManager.unregisterListener(this);
}
```

For more information, see [unregisterListener\(SensorEventListener\)](#).

## Test with the Android Emulator

The Android Emulator includes a set of virtual sensor controls that allow you to test sensors such as accelerometer, ambient temperature, magnetometer, proximity, light, and more.



The emulator uses a connection with an Android device that is running the [SdkControllerSensor](#) app. Note that this app is available only on devices running Android 4.0 (API level 14) or higher. (If the device is running Android 4.0, it must have Revision 2 installed.) The *SdkControllerSensor* app monitors changes in the sensors on the device and transmits them to the emulator. The emulator is then transformed based on the new values that it receives from the sensors on your device.

You can view the source code for the *SdkControllerSensor* app in the following location:

```
$ your-android-sdk-directory/tools/apps/SdkController
```

To transfer data between your device and the emulator, follow these steps:

1. Check that [USB debugging is enabled](#) on your device.
2. Connect your device to your development machine using a USB cable.
3. Start the *SdkControllerSensor* app on your device.
4. In the app, select the sensors that you want to emulate.
5. Run the following `adb` command:

```
$ adb forward tcp:1968 tcp:1968
```

6. Start the emulator. You should now be able to apply transformations to the emulator by moving your device.

**Note:** If the movements that you make to your physical device aren't transforming the emulator, try running the `adb` command from step 5 again.

For more information, see the [Android Emulator guide](#).

## Don't block the `onSensorChanged()` method

Sensor data can change at a high rate, which means the system may call the [onSensorChanged\(SensorEvent\)](#) method quite often. As a

best practice, you should do as little as possible within the `onSensorChanged(SensorEvent)` method so you don't block it. If your application requires you to do any data filtering or reduction of sensor data, you should perform that work outside of the `onSensorChanged(SensorEvent)` method.

## Avoid using deprecated methods or sensor types

Several methods and constants have been deprecated. In particular, the `TYPE_ORIENTATION` sensor type has been deprecated. To get orientation data you should use the `getOrientation()` method instead. Likewise, the `TYPE_TEMPERATURE` sensor type has been deprecated. You should use the `TYPE_AMBIENT_TEMPERATURE` sensor type instead on devices that are running Android 4.0.

## Verify sensors before you use them

Always verify that a sensor exists on a device before you attempt to acquire data from it. Don't assume that a sensor exists simply because it's a frequently-used sensor. Device manufacturers are not required to provide any particular sensors in their devices.

## Choose sensor delays carefully

When you register a sensor with the `registerListener()` method, be sure you choose a delivery rate that is suitable for your application or use-case. Sensors can provide data at very high rates. Allowing the system to send extra data that you don't need wastes system resources and uses battery power.

# Motion Sensors

## In this document

- [Using the Gravity Sensor](#)
- [Using the Linear Accelerometer](#)
- [Using the Rotation Vector Sensor](#)
- [Using the Significant Motion Sensor](#)
- [Using the Step Counter Sensor](#)
- [Using the Step Detector Sensor](#)
- [Working with Raw Data](#)
  - [Using the Accelerometer](#)
  - [Using the Gyroscope](#)

## Key classes and interfaces

- [Sensor](#)
- [SensorEvent](#)
- [SensorManager](#)
- [SensorEventListener](#)

## Related samples

- [Accelerometer Play](#)
- [API Demos \(OS - RotationVectorDemo\)](#)
- [API Demos \(OS - Sensors\)](#)

## See also

- [Sensors](#)
- [Sensors Overview](#)
- [Position Sensors](#)
- [Environment Sensors](#)

The Android platform provides several sensors that let you monitor the motion of a device.

The sensors' possible architectures vary by sensor type:

- The gravity, linear acceleration, rotation vector, significant motion, step counter, and step detector sensors are either hardware-based or software-based.
- The accelerometer and gyroscope sensors are always hardware-based.

Most Android-powered devices have an accelerometer, and many now include a gyroscope. The availability of the software-based sensors is more variable because they often rely on one or more hardware sensors to derive their data. Depending on the device, these software-based sensors can derive their data either from the accelerometer and magnetometer or from the gyroscope.

Motion sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a ball in a game), but it can also be a reflection of the physical environment in which the device is sitting (for example, moving with you while you drive your car). In the first case, you are monitoring motion relative to the device's frame of reference or your application's frame of reference; in the second case you are monitoring motion relative to the world's frame of reference. Motion sensors by themselves are not typically used to monitor device position, but they can

be used with other sensors, such as the geomagnetic field sensor, to determine a device's position relative to the world's frame of reference (see [Position Sensors](#) for more information).

All of the motion sensors return multi-dimensional arrays of sensor values for each `SensorEvent`. For example, during a single sensor event the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation data for the three coordinate axes. These data values are returned in a `float` array (`values`) along with other `SensorEvent` parameters. Table 1 summarizes the motion sensors that are available on the Android platform.

**Table 1.** Motion sensors that are supported on the Android platform.

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (including gravity).	$\text{m/s}^2$
	<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (including gravity).	
	<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	<code>SensorEvent.values[0]</code>	Force of gravity along the x axis.	$\text{m/s}^2$
	<code>SensorEvent.values[1]</code>	Force of gravity along the y axis.	
	<code>SensorEvent.values[2]</code>	Force of gravity along the z axis.	
TYPE_GYROSCOPE	<code>SensorEvent.values[0]</code>	Rate of rotation around the x axis.	$\text{rad/s}$
	<code>SensorEvent.values[1]</code>	Rate of rotation around the y axis.	
	<code>SensorEvent.values[2]</code>	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	<code>SensorEvent.values[0]</code>	Rate of rotation (without drift compensation) around the x axis.	$\text{rad/s}$
	<code>SensorEvent.values[1]</code>	Rate of rotation (without drift compensation) around the y axis.	
	<code>SensorEvent.values[2]</code>	Rate of rotation (without drift compensation) around the z axis.	
	<code>SensorEvent.values[3]</code>	Estimated drift around the x axis.	
	<code>SensorEvent.values[4]</code>	Estimated drift around the y axis.	
	<code>SensorEvent.values[5]</code>	Estimated drift around the z axis.	
TYPE_LINEAR_ACCELERATION	<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (excluding gravity).	$\text{m/s}^2$
	<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (excluding gravity).	
	<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis ( $x * \sin(\theta/2)$ ).	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis ( $y * \sin(\theta/2)$ ).	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis ( $z * \sin(\theta/2)$ ).	
	<code>SensorEvent.values[3]</code>	Scalar component of the rotation vector ( $(\cos(\theta/2))$ ). <sup>1</sup>	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A

TYPE_STEP_COUNTER	SensorEvent.values[0]	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A

<sup>1</sup> The scalar component is an optional value.

The rotation vector sensor and the gravity sensor are the most frequently used sensors for motion detection and monitoring. The rotational vector sensor is particularly versatile and can be used for a wide range of motion-related tasks, such as detecting gestures, monitoring angular change, and monitoring relative orientation changes. For example, the rotational vector sensor is ideal if you are developing a game, an augmented reality application, a 2-dimensional or 3-dimensional compass, or a camera stabilization app. In most cases, using these sensors is a better choice than using the accelerometer and geomagnetic field sensor or the orientation sensor.

## Android Open Source Project Sensors

The Android Open Source Project (AOSP) provides three software-based motion sensors: a gravity sensor, a linear acceleration sensor, and a rotation vector sensor. These sensors were updated in Android 4.0 and now use a device's gyroscope (in addition to other sensors) to improve stability and performance. If you want to try these sensors, you can identify them by using the `getVendor()` method and the `getVersion()` method (the vendor is Google Inc.; the version number is 3). Identifying these sensors by vendor and version number is necessary because the Android system considers these three sensors to be secondary sensors. For example, if a device manufacturer provides their own gravity sensor, then the AOSP gravity sensor shows up as a secondary gravity sensor. All three of these sensors rely on a gyroscope: if a device does not have a gyroscope, these sensors do not show up and are not available for use.

## Using the Gravity Sensor

The gravity sensor provides a three dimensional vector indicating the direction and magnitude of gravity. Typically, this sensor is used to determine the device's relative orientation in space. The following code shows you how to get an instance of the default gravity sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
```

The units are the same as those used by the acceleration sensor ( $\text{m/s}^2$ ), and the coordinate system is the same as the one used by the acceleration sensor.

**Note:** When a device is at rest, the output of the gravity sensor should be identical to that of the accelerometer.

## Using the Linear Accelerometer

The linear acceleration sensor provides you with a three-dimensional vector representing acceleration along each device axis, excluding gravity. You can use this value to perform gesture detection. The value can also serve as input to an inertial navigation system, which uses dead reckoning. The following code shows you how to get an instance of the default linear acceleration sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
```

Conceptually, this sensor provides you with acceleration data according to the following relationship:

`linear acceleration = acceleration - acceleration due to gravity`

You typically use this sensor when you want to obtain acceleration data without the influence of gravity. For example, you could use this sensor to see how fast your car is going. The linear acceleration sensor always has an offset, which you need to remove. The simplest way to do this is to build a calibration step into your application. During calibration you can ask the user to set the device on a table, and then read the offsets for all three axes. You can then subtract that offset from the acceleration sensor's direct readings to get the actual linear

acceleration.

The sensor [coordinate system](#) is the same as the one used by the acceleration sensor, as are the units of measure ( $\text{m/s}^2$ ).

## Using the Rotation Vector Sensor

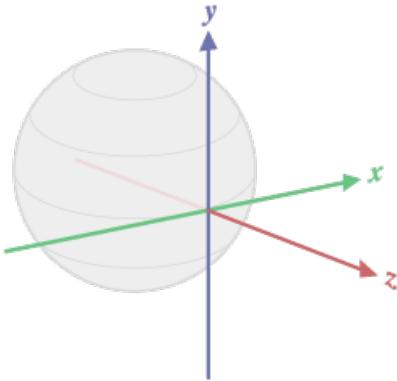
The rotation vector represents the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle  $\theta$  around an axis (x, y, or z). The following code shows you how to get an instance of the default rotation vector sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
```

The three elements of the rotation vector are expressed as follows:

$$x \cdot \sin\left(\frac{\theta}{2}\right)$$
$$y \cdot \sin\left(\frac{\theta}{2}\right)$$
$$z \cdot \sin\left(\frac{\theta}{2}\right)$$

Where the magnitude of the rotation vector is equal to  $\sin(\theta/2)$ , and the direction of the rotation vector is equal to the direction of the axis of rotation.



**Figure 1.** Coordinate system used by the rotation vector sensor.

The three elements of the rotation vector are equal to the last three components of a unit quaternion ( $\cos(\theta/2)$ ,  $x \cdot \sin(\theta/2)$ ,  $y \cdot \sin(\theta/2)$ ,  $z \cdot \sin(\theta/2)$ ). Elements of the rotation vector are unitless. The x, y, and z axes are defined in the same way as the acceleration sensor. The reference coordinate system is defined as a direct orthonormal basis (see figure 1). This coordinate system has the following characteristics:

- X is defined as the vector product  $Y \times Z$ . It is tangential to the ground at the device's current location and points approximately East.
- Y is tangential to the ground at the device's current location and points toward the geomagnetic North Pole.
- Z points toward the sky and is perpendicular to the ground plane.

The Android SDK provides a sample application that shows how to use the rotation vector sensor. The sample application is located in the API Demos code ([OS - RotationVectorDemo](#)).

## Using the Significant Motion Sensor

The significant motion sensor triggers an event each time significant motion is detected and then it disables itself. A significant motion is a motion that might lead to a change in the user's location; for example walking, biking, or sitting in a moving car. The following code shows you

how to get an instance of the default significant motion sensor and how to register an event listener:

```
private SensorManager mSensorManager;
private Sensor mSensor;
private TriggerEventListener mTriggerEventListener;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_SIGNIFICANT_MOTION);

mTriggerEventListener = new TriggerEventListener() {
 @Override
 public void onTrigger(TriggerEvent event) {
 // Do work
 }
};

mSensorManager.requestTriggerSensor(mTriggerEventListener, mSensor);
```

For more information, see [TriggerEventListener](#).

## Using the Step Counter Sensor

The step counter sensor provides the number of steps taken by the user since the last reboot while the sensor was activated. The step counter has more latency (up to 10 seconds) but more accuracy than the step detector sensor. The following code shows you how to get an instance of the default step counter sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
```

To preserve the battery on devices running your app, you should use the [JobScheduler](#) class to retrieve the current value from the step counter sensor at a specific interval. Although different types of apps require different sensor-reading intervals, you should make this interval as long as possible unless your app requires real-time data from the sensor.

## Using the Step Detector Sensor

The step detector sensor triggers an event each time the user takes a step. The latency is expected to be below 2 seconds. The following code shows you how to get an instance of the default step detector sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
```

## Working with Raw Data

The following sensors provide your app with raw data about the linear and rotational forces being applied to the device. In order to use the values from these sensors effectively, you need to filter out factors from the environment, such as gravity. You might also need to apply a smoothing algorithm to the trend of values to reduce noise.

## Using the Accelerometer

An acceleration sensor measures the acceleration applied to the device, including the force of gravity. The following code shows you how to get an instance of the default acceleration sensor:

```

private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

```

Conceptually, an acceleration sensor determines the acceleration that is applied to a device ( $A_d$ ) by measuring the forces that are applied to the sensor itself ( $F_s$ ) using the following relationship:

$$A_d = -\left(\frac{1}{mass}\right) \sum F_s$$

However, the force of gravity is always influencing the measured acceleration according to the following relationship:

$$A_d = -g - \left(\frac{1}{mass}\right) \sum F_s$$

For this reason, when the device is sitting on a table (and not accelerating), the accelerometer reads a magnitude of  $g = 9.81 \text{ m/s}^2$ . Similarly, when the device is in free fall and therefore rapidly accelerating toward the ground at  $9.81 \text{ m/s}^2$ , its accelerometer reads a magnitude of  $g = 0 \text{ m/s}^2$ . Therefore, to measure the real acceleration of the device, the contribution of the force of gravity must be removed from the accelerometer data. This can be achieved by applying a high-pass filter. Conversely, a low-pass filter can be used to isolate the force of gravity. The following example shows how you can do this:

```

public void onSensorChanged(SensorEvent event){
 // In this example, alpha is calculated as t / (t + dT),
 // where t is the low-pass filter's time-constant and
 // dT is the event delivery rate.

 final float alpha = 0.8;

 // Isolate the force of gravity with the low-pass filter.
 gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
 gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
 gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

 // Remove the gravity contribution with the high-pass filter.
 linear_acceleration[0] = event.values[0] - gravity[0];
 linear_acceleration[1] = event.values[1] - gravity[1];
 linear_acceleration[2] = event.values[2] - gravity[2];
}

```

**Note:** You can use many different techniques to filter sensor data. The code sample above uses a simple filter constant (alpha) to create a low-pass filter. This filter constant is derived from a time constant (t), which is a rough representation of the latency that the filter adds to the sensor events, and the sensor's event delivery rate (dt). The code sample uses an alpha value of 0.8 for demonstration purposes. If you use this filtering method you may need to choose a different alpha value.

Accelerometers use the standard sensor [coordinate system](#). In practice, this means that the following conditions apply when a device is laying flat on a table in its natural orientation:

- If you push the device on the left side (so it moves to the right), the x acceleration value is positive.
- If you push the device on the bottom (so it moves away from you), the y acceleration value is positive.
- If you push the device toward the sky with an acceleration of  $A \text{ m/s}^2$ , the z acceleration value is equal to  $A + 9.81$ , which corresponds to the acceleration of the device ( $+A \text{ m/s}^2$ ) minus the force of gravity ( $-9.81 \text{ m/s}^2$ ).
- The stationary device will have an acceleration value of  $+9.81$ , which corresponds to the acceleration of the device ( $0 \text{ m/s}^2$  minus the force of gravity, which is  $-9.81 \text{ m/s}^2$ ).

In general, the accelerometer is a good sensor to use if you are monitoring device motion. Almost every Android-powered handset and tablet has an accelerometer, and it uses about 10 times less power than the other motion sensors. One drawback is that you might have to implement low-pass and high-pass filters to eliminate gravitational forces and reduce noise.

The Android SDK provides a sample application that shows how to use the acceleration sensor ([Accelerometer Play](#)).

## Using the Gyroscope

The gyroscope measures the rate of rotation in rad/s around a device's x, y, and z axis. The following code shows you how to get an instance of the default gyroscope:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

The sensor's [coordinate system](#) is the same as the one used for the acceleration sensor. Rotation is positive in the counter-clockwise direction; that is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. This is the standard mathematical definition of positive rotation and is not the same as the definition for roll that is used by the orientation sensor.

Usually, the output of the gyroscope is integrated over time to calculate a rotation describing the change of angles over the timestep. For example:

```
// Create a constant to convert nanoseconds to seconds.
private static final float NS2S = 1.0f / 1000000000.0f;
private final float[] deltaRotationVector = new float[4]();
private float timestamp;

public void onSensorChanged(SensorEvent event) {
 // This timestep's delta rotation to be multiplied by the current rotation
 // after computing it from the gyro sample data.
 if (timestamp != 0) {
 final float dT = (event.timestamp - timestamp) * NS2S;
 // Axis of the rotation sample, not normalized yet.
 float axisX = event.values[0];
 float axisY = event.values[1];
 float axisZ = event.values[2];

 // Calculate the angular speed of the sample
 float omegaMagnitude = sqrt(axisX*axisX + axisY*axisY + axisZ*axisZ);

 // Normalize the rotation vector if it's big enough to get the axis
 // (that is, EPSILON should represent your maximum allowable margin of error)
 if (omegaMagnitude > EPSILON) {
 axisX /= omegaMagnitude;
 axisY /= omegaMagnitude;
 axisZ /= omegaMagnitude;
 }

 // Integrate around this axis with the angular speed by the timestep
 // in order to get a delta rotation from this sample over the timestep
 // We will convert this axis-angle representation of the delta rotation
 // into a quaternion before turning it into the rotation matrix.
 float thetaOverTwo = omegaMagnitude * dT / 2.0f;
 float sinThetaOverTwo = sin(thetaOverTwo);
 float cosThetaOverTwo = cos(thetaOverTwo);
 deltaRotationVector[0] = sinThetaOverTwo * axisX;
 deltaRotationVector[1] = sinThetaOverTwo * axisY;
 deltaRotationVector[2] = sinThetaOverTwo * axisZ;
 deltaRotationVector[3] = cosThetaOverTwo;
 }
 timestamp = event.timestamp;
 float[] deltaRotationMatrix = new float[9];
 SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector);
 // User code should concatenate the delta rotation we computed with the current rotation
 // in order to get the updated rotation.
 // rotationCurrent = rotationCurrent * deltaRotationMatrix;
}
}
```

Standard gyroscopes provide raw rotational data without any filtering or correction for noise and drift (bias). In practice, gyroscope noise and drift will introduce errors that need to be compensated for. You usually determine the drift (bias) and noise by monitoring other sensors, such as the gravity sensor or accelerometer.

## Using the Uncalibrated Gyroscope

The uncalibrated gyroscope is similar to the [gyroscope](#), except that no gyro-drift compensation is applied to the rate of rotation. Factory calibration and temperature compensation are still applied to the rate of rotation. The uncalibrated gyroscope is useful for post-processing and melding orientation data. In general, `gyroscope_event.values[0]` will be close to `uncalibrated_gyroscope_event.values[0]` - `uncalibrated_gyroscope_event.values[3]`. That is,

```
calibrated_x ~= uncalibrated_x - bias_estimate_x
```

**Note:** Uncalibrated sensors provide more raw results and may include some bias, but their measurements contain fewer jumps from corrections applied through calibration. Some applications may prefer these uncalibrated results as smoother and more reliable. For instance, if an application is attempting to conduct its own sensor fusion, introducing calibrations can actually distort results.

In addition to the rates of rotation, the uncalibrated gyroscope also provides the estimated drift around each axis. The following code shows you how to get an instance of the default uncalibrated gyroscope:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE_UNCALIBRATED);
```

# Position Sensors

## In this document

- › [Using the Game Rotation Vector Sensor](#)
- › [Using the Geomagnetic Rotation Vector Sensor](#)
- › [Computing the Device's Orientation](#)
- › [Using the Geomagnetic Field Sensor](#)
- › [Using the Proximity Sensor](#)

## Key classes and interfaces

- › [Sensor](#)
- › [SensorEvent](#)
- › [SensorManager](#)
- › [SensorEventListener](#)

## Related samples

- › [Accelerometer Play](#)
- › [API Demos \(OS - RotationVectorDemo\)](#)
- › [API Demos \(OS - Sensors\)](#)

## See also

- › [Sensors](#)
- › [Sensors Overview](#)
- › [Motion Sensors](#)
- › [Environment Sensors](#)

The Android platform provides two sensors that let you determine the position of a device: the geomagnetic field sensor and the accelerometer. The Android platform also provides a sensor that lets you determine how close the face of a device is to an object (known as the *proximity sensor*). The geomagnetic field sensor and the proximity sensor are hardware-based. Most handset and tablet manufacturers include a geomagnetic field sensor. Likewise, handset manufacturers usually include a proximity sensor to determine when a handset is being held close to a user's face (for example, during a phone call). For determining a device's orientation, you can use the readings from the device's accelerometer and the geomagnetic field sensor.

**Note:** The orientation sensor was deprecated in Android 2.2 (API level 8), and the orientation sensor type was deprecated in Android 4.4W (API level 20).

Position sensors are useful for determining a device's physical position in the world's frame of reference. For example, you can use the geomagnetic field sensor in combination with the accelerometer to determine a device's position relative to the magnetic north pole. You can also use these sensors to determine a device's orientation in your application's frame of reference. Position sensors are not typically used to monitor device movement or motion, such as shake, tilt, or thrust (for more information, see [Motion Sensors](#)).

The geomagnetic field sensor and accelerometer return multi-dimensional arrays of sensor values for each [SensorEvent](#). For example, the geomagnetic field sensor provides geomagnetic field strength values for each of the three coordinate axes during a single sensor event. Likewise, the accelerometer sensor measures the acceleration applied to the device during a sensor event. For more information about the coordinate systems that are used by sensors, see [Sensor Coordinate Systems](#). The proximity sensor provides a single value for each sensor event. Table 1 summarizes the position sensors that are supported on the Android platform.

**Table 1.** Position sensors that are supported on the Android platform.

Sensor	Sensor event data	Description	Units of measure
TYPE_GAME_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis ( $x * \sin(\theta/2)$ ).	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis ( $y * \sin(\theta/2)$ ).	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis ( $z * \sin(\theta/2)$ ).	
TYPE_GEO MAGNETIC_ROTATION_VECTOR	<code>SensorEvent.values[0]</code>	Rotation vector component along the x axis ( $x * \sin(\theta/2)$ ).	Unitless
	<code>SensorEvent.values[1]</code>	Rotation vector component along the y axis ( $y * \sin(\theta/2)$ ).	
	<code>SensorEvent.values[2]</code>	Rotation vector component along the z axis ( $z * \sin(\theta/2)$ ).	
TYPE_MAGNETIC_FIELD	<code>SensorEvent.values[0]</code>	Geomagnetic field strength along the x axis.	$\mu\text{T}$
	<code>SensorEvent.values[1]</code>	Geomagnetic field strength along the y axis.	
	<code>SensorEvent.values[2]</code>	Geomagnetic field strength along the z axis.	
TYPE_MAGNETIC_FIELD_UNCALIBRATED	<code>SensorEvent.values[0]</code>	Geomagnetic field strength (without hard iron calibration) along the x axis.	$\mu\text{T}$
	<code>SensorEvent.values[1]</code>	Geomagnetic field strength (without hard iron calibration) along the y axis.	
	<code>SensorEvent.values[2]</code>	Geomagnetic field strength (without hard iron calibration) along the z axis.	
	<code>SensorEvent.values[3]</code>	Iron bias estimation along the x axis.	
	<code>SensorEvent.values[4]</code>	Iron bias estimation along the y axis.	
	<code>SensorEvent.values[5]</code>	Iron bias estimation along the z axis.	
TYPE_ORIENTATION <sup>1</sup>	<code>SensorEvent.values[0]</code>	Azimuth (angle around the z-axis).	Degrees
	<code>SensorEvent.values[1]</code>	Pitch (angle around the x-axis).	
	<code>SensorEvent.values[2]</code>	Roll (angle around the y-axis).	
TYPE_PROXIMITY	<code>SensorEvent.values[0]</code>	Distance from object. <sup>2</sup>	cm

<sup>1</sup>This sensor was deprecated in Android 2.2 (API level 8), and this sensor type was deprecated in Android 4.4W (API level 20). The sensor framework provides alternate methods for acquiring device orientation, which are discussed in [Computing the Device's Orientation](#).

<sup>2</sup> Some proximity sensors provide only binary values representing near and far.

## Using the Game Rotation Vector Sensor

The game rotation vector sensor is identical to the [Rotation Vector Sensor](#), except it does not use the geomagnetic field. Therefore the Y axis does not point north but instead to some other reference. That reference is allowed to drift by the same order of magnitude as the gyroscope drifts around the Z axis.

Because the game rotation vector sensor does not use the magnetic field, relative rotations are more accurate, and not impacted by magnetic

field changes. Use this sensor in a game if you do not care about where north is, and the normal rotation vector does not fit your needs because of its reliance on the magnetic field.

The following code shows you how to get an instance of the default game rotation vector sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GAME_ROTATION_VECTOR);
```

## Using the Geomagnetic Rotation Vector Sensor

The geomagnetic rotation vector sensor is similar to the [Rotation Vector Sensor](#), but it uses a magnetometer instead of a gyroscope. The accuracy of this sensor is lower than the normal rotation vector sensor, but the power consumption is reduced. Only use this sensor if you want to collect some rotation information in the background without draining too much battery. This sensor is most useful when used in conjunction with batching.

The following code shows you how to get an instance of the default geomagnetic rotation vector sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GEOGRAPHIC_ROTATION_VECTOR);
```

## Computing the Device's Orientation

By computing a device's orientation, you can monitor the position of the device relative to the earth's frame of reference (specifically, the magnetic north pole). The following code shows you how to compute a device's orientation:

```
private SensorManager mSensorManager;
...
// Rotation matrix based on current readings from accelerometer and magnetometer.
final float[] rotationMatrix = new float[9];
mSensorManager.getRotationMatrix(rotationMatrix, null,
 accelerometerReading, magnetometerReading);

// Express the updated rotation matrix as three orientation angles.
final float[] orientationAngles = new float[3];
mSensorManager.getOrientation(rotationMatrix, orientationAngles);
```

The system computes the orientation angles by using a device's geomagnetic field sensor in combination with the device's accelerometer. Using these two hardware sensors, the system provides data for the following three orientation angles:

- **Azimuth (degrees of rotation about the -z axis).** This is the angle between the device's current compass direction and magnetic north. If the top edge of the device faces magnetic north, the azimuth is 0 degrees; if the top edge faces south, the azimuth is 180 degrees. Similarly, if the top edge faces east, the azimuth is 90 degrees, and if the top edge faces west, the azimuth is 270 degrees.
- **Pitch (degrees of rotation about the x axis).** This is the angle between a plane parallel to the device's screen and a plane parallel to the ground. If you hold the device parallel to the ground with the bottom edge closest to you and tilt the top edge of the device toward the ground, the pitch angle becomes positive. Tilting in the opposite direction—moving the top edge of the device away from the ground—causes the pitch angle to become negative. The range of values is -180 degrees to 180 degrees.
- **Roll (degrees of rotation about the y axis).** This is the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. If you hold the device parallel to the ground with the bottom edge closest to you and tilt the left edge of the device toward the ground, the roll angle becomes positive. Tilting in the opposite direction—moving the right edge of the device toward the ground—causes the roll angle to become negative. The range of values is -90 degrees to 90 degrees.

**Note:** The sensor's roll definition has changed to reflect the vast majority of implementations in the geosensor ecosystem.

Note that these angles work off of a different coordinate system than the one used in aviation (for yaw, pitch, and roll). In the aviation system, the x axis is along the long side of the plane, from tail to nose.

The orientation sensor derives its data by processing the raw sensor data from the accelerometer and the geomagnetic field sensor. Because of the heavy processing that is involved, the accuracy and precision of the orientation sensor is diminished. Specifically, this sensor is reliable only when the roll angle is 0. As a result, the orientation sensor was deprecated in Android 2.2 (API level 8), and the orientation sensor type was deprecated in Android 4.4W (API level 20). Instead of using raw data from the orientation sensor, we recommend that you use the `getRotationMatrix()` method in conjunction with the `getOrientation()` method to compute orientation values, as shown in the following code sample. As part of this process, you can use the `remapCoordinateSystem()` method to translate the orientation values to your application's frame of reference.

```
public class SensorActivity extends Activity implements SensorEventListener {

 private SensorManager mSensorManager;
 private final float[] mAccelerometerReading = new float[3];
 private final float[] mMagnetometerReading = new float[3];

 private final float[] mRotationMatrix = new float[9];
 private final float[] mOrientationAngles = new float[3];

 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
 }

 @Override
 public void onAccuracyChanged(Sensor sensor, int accuracy) {
 // Do something here if sensor accuracy changes.
 // You must implement this callback in your code.
 }

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

 // Get updates from the accelerometer and magnetometer at a constant rate.
 // To make batch operations more efficient and reduce power consumption,
 // provide support for delaying updates to the application.
 //
 // In this example, the sensor reporting delay is small enough such that
 // the application receives an update before the system checks the sensor
 // readings again.
 mSensorManager.registerListener(this, Sensor.TYPE_ACCELEROMETER,
 SensorManagerSENSOR_DELAY_NORMAL, SensorManagerSENSOR_DELAY_UI);
 mSensorManager.registerListener(this, Sensor.TYPE_MAGNETIC_FIELD,
 SensorManagerSENSOR_DELAY_NORMAL, SensorManagerSENSOR_DELAY_UI);
 }

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

 // Don't receive any more updates from either sensor.
 mSensorManager.unregisterListener(this);
 }

 // Get readings from accelerometer and magnetometer. To simplify calculations,
 // consider storing these readings as unit vectors.
 @Override
 public void onSensorChanged(SensorEvent event) {
 if (event.sensor == Sensor.TYPE_ACCELEROMETER) {
 System.arraycopy(event.values, 0, mAccelerometerReading,
 0, mAccelerometerReading.length);
 }
 else if (event.sensor == Sensor.TYPE_MAGNETIC_FIELD) {
 System.arraycopy(event.values, 0, mMagnetometerReading,
 0, mMagnetometerReading.length);
 }
 }
}
```

```

 }

 // Compute the three orientation angles based on the most recent readings from
 // the device's accelerometer and magnetometer.
 public void updateOrientationAngles() {
 // Update rotation matrix, which is needed to update orientation angles.
 mSensorManager.getRotationMatrix(mRotationMatrix, null,
 mAccelerometerReading, mMagnetometerReading);

 // "mRotationMatrix" now has up-to-date information.

 mSensorManager.getOrientation(mRotationMatrix, mOrientationAngles);

 // "mOrientationAngles" now has up-to-date information.
 }
}

```

You don't usually need to perform any data processing or filtering of the device's raw orientation angles other than translating the sensor's coordinate system to your application's frame of reference.

## Using the Geomagnetic Field Sensor

The geomagnetic field sensor lets you monitor changes in the earth's magnetic field. The following code shows you how to get an instance of the default geomagnetic field sensor:

```

private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

```

This sensor provides raw field strength data (in  $\mu\text{T}$ ) for each of the three coordinate axes. Usually, you do not need to use this sensor directly. Instead, you can use the rotation vector sensor to determine raw rotational movement or you can use the accelerometer and geomagnetic field sensor in conjunction with the `getRotationMatrix()` method to obtain the rotation matrix and the inclination matrix. You can then use these matrices with the `getOrientation()` and `getInclination()` methods to obtain azimuth and geomagnetic inclination data.

**Note:** When testing your app, you can improve the sensor's accuracy by waving the device in a figure-8 pattern.

## Using the Uncalibrated Magnetometer

The uncalibrated magnetometer is similar to the [geomagnetic field sensor](#), except that no hard iron calibration is applied to the magnetic field. Factory calibration and temperature compensation are still applied to the magnetic field. The uncalibrated magnetometer is useful to handle bad hard iron estimations. In general, `geomagneticSensor_event.values[0]` will be close to `uncalibrated_magnetometer_event.values[0] - uncalibrated_magnetometer_event.values[3]`. That is,

```

calibrated_x ~= uncalibrated_x - bias_estimate_x

```

**Note:** Uncalibrated sensors provide more raw results and may include some bias, but their measurements contain fewer jumps from corrections applied through calibration. Some applications may prefer these uncalibrated results as smoother and more reliable. For instance, if an application is attempting to conduct its own sensor fusion, introducing calibrations can actually distort results.

In addition to the magnetic field, the uncalibrated magnetometer also provides the estimated hard iron bias in each axis. The following code shows you how to get an instance of the default uncalibrated magnetometer:

```

private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED);

```

## Using the Proximity Sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

```
public class SensorActivity extends Activity implements SensorEventListener {
 private SensorManager mSensorManager;
 private Sensor mProximity;

 @Override
 public final void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 // Get an instance of the sensor service, and use that to get an instance of
 // a particular sensor.
 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
 mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
 }

 @Override
 public final void onAccuracyChanged(Sensor sensor, int accuracy) {
 // Do something here if sensor accuracy changes.
 }

 @Override
 public final void onSensorChanged(SensorEvent event) {
 float distance = event.values[0];
 // Do something with this sensor data.
 }

 @Override
 protected void onResume() {
 // Register a listener for the sensor.
 super.onResume();
 mSensorManager.registerListener(this, mProximity, SensorManager.SENSOR_DELAY_NORMAL);
 }

 @Override
 protected void onPause() {
 // Be sure to unregister the sensor when the activity pauses.
 super.onPause();
 mSensorManager.unregisterListener(this);
 }
}
```

**Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.



# Environment Sensors

## In this document

- › [Using the Light, Pressure, and Temperature Sensors](#)
- › [Using the Humidity Sensor](#)

## Related samples

- › [Accelerometer Play](#)
- › [API Demos \(OS - RotationVectorDemo\)](#)
- › [API Demos \(OS - Sensors\)](#)

## See also

- › [Sensors](#)
- › [Sensors Overview](#)
- › [Position Sensors](#)
- › [Motion Sensors](#)

The Android platform provides four sensors that let you monitor various environmental properties. You can use these sensors to monitor relative ambient humidity, illuminance, ambient pressure, and ambient temperature near an Android-powered device. All four environment sensors are hardware-based and are available only if a device manufacturer has built them into a device. With the exception of the light sensor, which most device manufacturers use to control screen brightness, environment sensors are not always available on devices. Because of this, it's particularly important that you verify at runtime whether an environment sensor exists before you attempt to acquire data from it.

Unlike most motion sensors and position sensors, which return a multi-dimensional array of sensor values for each [SensorEvent](#), environment sensors return a single sensor value for each data event. For example, the temperature in °C or the pressure in hPa. Also, unlike motion sensors and position sensors, which often require high-pass or low-pass filtering, environment sensors do not typically require any data filtering or data processing. Table 1 provides a summary of the environment sensors that are supported on the Android platform.

**Table 1.** Environment sensors that are supported on the Android platform.

Sensor	Sensor event data	Units of measure	Data description
<a href="#">TYPE_AMBIENT_TEMPERATURE</a>	<code>event.values[0]</code>	°C	Ambient air temperature.
<a href="#">TYPE_LIGHT</a>	<code>event.values[0]</code>	lx	Illuminance.
<a href="#">TYPE_PRESSURE</a>	<code>event.values[0]</code>	hPa or mbar	Ambient air pressure.
<a href="#">TYPE_RELATIVE_HUMIDITY</a>	<code>event.values[0]</code>	%	Ambient relative humidity.
<a href="#">TYPE_TEMPERATURE</a>	<code>event.values[0]</code>	°C	Device temperature. <sup>1</sup>

<sup>1</sup> Implementations vary from device to device. This sensor was deprecated in Android 4.0 (API Level 14).

## Using the Light, Pressure, and Temperature Sensors

The raw data you acquire from the light, pressure, and temperature sensors usually requires no calibration, filtering, or modification, which makes them some of the easiest sensors to use. To acquire data from these sensors you first create an instance of the [SensorManager](#) class, which you can use to get an instance of a physical sensor. Then you register a sensor listener in the [onResume\(\)](#) method, and start handling incoming sensor data in the [onSensorChanged\(\)](#) callback method. The following code shows you how to do this:

```

public class SensorActivity extends Activity implements SensorEventListener {
 private SensorManager mSensorManager;
 private Sensor mPressure;

 @Override
 public final void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 // Get an instance of the sensor service, and use that to get an instance of
 // a particular sensor.
 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
 mPressure = mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
 }

 @Override
 public final void onAccuracyChanged(Sensor sensor, int accuracy) {
 // Do something here if sensor accuracy changes.
 }

 @Override
 public final void onSensorChanged(SensorEvent event) {
 float millibars_of_pressure = event.values[0];
 // Do something with this sensor data.
 }

 @Override
 protected void onResume() {
 // Register a listener for the sensor.
 super.onResume();
 mSensorManager.registerListener(this, mPressure, SensorManager.SENSOR_DELAY_NORMAL);
 }

 @Override
 protected void onPause() {
 // Be sure to unregister the sensor when the activity pauses.
 super.onPause();
 mSensorManager.unregisterListener(this);
 }
}

```

You must always include implementations of both the `onAccuracyChanged()` and `onSensorChanged()` callback methods. Also, be sure that you always unregister a sensor when an activity pauses. This prevents a sensor from continually sensing data and draining the battery.

## Using the Humidity Sensor

You can acquire raw relative humidity data by using the humidity sensor the same way that you use the light, pressure, and temperature sensors. However, if a device has both a humidity sensor (`TYPE_RELATIVE_HUMIDITY`) and a temperature sensor (`TYPE_AMBIENT_TEMPERATURE`) you can use these two data streams to calculate the dew point and the absolute humidity.

### Dew point

The dew point is the temperature at which a given volume of air must be cooled, at constant barometric pressure, for water vapor to condense into water. The following equation shows how you can calculate the dew point:

$$t_d(t, RH) = T_n \cdot \frac{\ln\left(\frac{RH}{100}\right) + \frac{m \cdot t}{T_n + t}}{m - \left[\ln\left(\frac{RH}{100}\right) + \frac{m \cdot t}{T_n + t}\right]}$$

Where,

$t_d$  = dew point temperature in degrees C

$t$  = actual temperature in degrees C

RH = actual relative humidity in percent (%)

$m = 17.62$

$$T_n = 243.12$$

## Absolute humidity

The absolute humidity is the mass of water vapor in a given volume of dry air. Absolute humidity is measured in grams/meter<sup>3</sup>. The following equation shows how you can calculate the absolute humidity:

$$d_v(t, RH) = 216.7 \cdot \frac{\frac{RH}{100} \cdot A \cdot \exp\left(\frac{mt}{T_n+t}\right)}{273.15 + t}$$

Where,

$d_v$  = absolute humidity in grams/meter<sup>3</sup>

$t$  = actual temperature in degrees C

RH = actual relative humidity in percent (%)

$m = 17.62$

$T_n = 243.12$  degrees C

$A = 6.112$  hPa



# Connectivity

Android provides rich APIs to let your app connect and interact with other devices over Bluetooth, NFC, Wi-Fi P2P, USB, and SIP, in addition to standard network connections.

## TRAINING

---

### Transferring Data Without Draining the Battery

This class demonstrates the best practices for scheduling and executing downloads using techniques such as caching, polling, and prefetching. You will learn how the power-use profile of the wireless radio can affect your choices on when, what, and how to transfer data in order to minimize impact on battery life.

### Backing up App Data to the Cloud

This class covers techniques for backing up data to the cloud so that users can restore their data when recovering from a data loss (such as a factory reset) or installing your application on a new device.



# Wi-Fi Peer-to-Peer

In this document

- [API Overview](#)
- [Creating a Broadcast Receiver for Wi-Fi P2P Intents](#)
- [Creating a Wi-Fi P2P Application](#)
  - [Initial setup](#)
  - [Discovering peers](#)
  - [Connecting to peers](#)
  - [Transferring data](#)

See also

- [Creating P2P Connections with Wi-Fi](#)

Wi-Fi peer-to-peer (P2P) allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point (Android's Wi-Fi P2P framework complies with the Wi-Fi Alliance's Wi-Fi Direct™ certification program). Using these APIs, you can discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection. This is useful for applications that share data among users, such as a multiplayer game or a photo sharing application.

The Wi-Fi P2P APIs consist of the following main parts:

- Methods that allow you to discover, request, and connect to peers are defined in the [WifiP2pManager](#) class.
- Listeners that allow you to be notified of the success or failure of [WifiP2pManager](#) method calls. When calling [WifiP2pManager](#) methods, each method can receive a specific listener passed in as a parameter.
- Intents that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.

You often use these three main components of the APIs together. For example, you can provide a [WifiP2pManager.ActionListener](#) to a call to [discoverPeers\(\)](#), so that you can be notified with the [ActionListener.onSuccess\(\)](#) and [ActionListener.onFailure\(\)](#) methods. A [WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#) intent is also broadcast if the [discoverPeers\(\)](#) method discovers that the peers list has changed.

## API Overview

The [WifiP2pManager](#) class provides methods to allow you to interact with the Wi-Fi hardware on your device to do things like discover and connect to peers. The following actions are available:

**Table 1.** Wi-Fi P2P Methods

Method	Description
<a href="#">initialize()</a>	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.
<a href="#">connect()</a>	Starts a peer-to-peer connection with a device with the specified configuration.
<a href="#">cancelConnect()</a>	Cancels any ongoing peer-to-peer group negotiation.
<a href="#">requestConnectInfo()</a>	Requests a device's connection information.

<code>createGroup()</code>	Creates a peer-to-peer group with the current device as the group owner.
<code>removeGroup()</code>	Removes the current peer-to-peer group.
<code>requestGroupInfo()</code>	Requests peer-to-peer group information.
<code>discoverPeers()</code>	Initiates peer discovery
<code>requestPeers()</code>	Requests the current list of discovered peers.

`WifiP2pManager` methods let you pass in a listener, so that the Wi-Fi P2P framework can notify your activity of the status of a call. The available listener interfaces and the corresponding `WifiP2pManager` method calls that use the listeners are described in the following table:

**Table 2.** Wi-Fi P2P Listeners

Listener interface	Associated actions
<code>WifiP2pManager.ActionListener</code>	<code>connect()</code> , <code>cancelConnect()</code> , <code>createGroup()</code> , <code>removeGroup()</code> , and <code>discoverPeers()</code>
<code>WifiP2pManager.ChannelListener</code>	<code>initialize()</code>
<code>WifiP2pManager.ConnectionInfoListener</code>	<code>requestConnectInfo()</code>
<code>WifiP2pManager.GroupInfoListener</code>	<code>requestGroupInfo()</code>
<code>WifiP2pManager.PeerListListener</code>	<code>requestPeers()</code>

The Wi-Fi P2P APIs define intents that are broadcast when certain Wi-Fi P2P events happen, such as when a new peer is discovered or when a device's Wi-Fi state changes. You can register to receive these intents in your application by [creating a broadcast receiver](#) that handles these intents:

**Table 3.** Wi-Fi P2P Intents

Intent	Description
<code>WIFI_P2P_CONNECTION_CHANGED_ACTION</code>	Broadcast when the state of the device's Wi-Fi connection changes.
<code>WIFI_P2P_PEERS_CHANGED_ACTION</code>	Broadcast when you call <code>discoverPeers()</code> . You usually want to call <code>requestPeers()</code> to get an updated list of peers if you handle this intent in your application.
<code>WIFI_P2P_STATE_CHANGED_ACTION</code>	Broadcast when Wi-Fi P2P is enabled or disabled on the device.
<code>WIFI_P2P_THIS_DEVICE_CHANGED_ACTION</code>	Broadcast when a device's details have changed, such as the device's name.

## Creating a Broadcast Receiver for Wi-Fi P2P Intents

A broadcast receiver allows you to receive intents broadcast by the Android system, so that your application can respond to events that you are interested in. The basic steps for creating a broadcast receiver to handle Wi-Fi P2P intents are as follows:

1. Create a class that extends the `BroadcastReceiver` class. For the class' constructor, you most likely want to have parameters for the `WifiP2pManager`, `WifiP2pManager.Channel`, and the activity that this broadcast receiver will be registered in. This allows the broadcast receiver to send updates to the activity as well as have access to the Wi-Fi hardware and a communication channel if needed.
2. In the broadcast receiver, check for the intents that you are interested in `onReceive()`. Carry out any necessary actions depending on the intent that is received. For example, if the broadcast receiver receives a `WIFI_P2P_PEERS_CHANGED_ACTION` intent, you can call the `requestPeers()` method to get a list of the currently discovered peers.

The following code shows you how to create a typical broadcast receiver. The broadcast receiver takes a `WifiP2pManager` object and an activity as arguments and uses these two classes to appropriately carry out the needed actions when the broadcast receiver receives an intent:

```

/**
 * A BroadcastReceiver that notifies of important Wi-Fi p2p events.
 */
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

 private WifiP2pManager mManager;
 private Channel mChannel;
 private MyWiFiActivity mActivity;

 public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
 MyWiFiActivity activity) {
 super();
 this.mManager = manager;
 this.mChannel = channel;
 this.mActivity = activity;
 }

 @Override
 public void onReceive(Context context, Intent intent) {
 String action = intent.getAction();

 if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
 // Check to see if Wi-Fi is enabled and notify appropriate activity
 } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
 // Call WifiP2pManager.requestPeers() to get a list of current peers
 } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
 // Respond to new connection or disconnections
 } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
 // Respond to this device's wifi state changing
 }
 }
}

```

## Creating a Wi-Fi P2P Application

Creating a Wi-Fi P2P application involves creating and registering a broadcast receiver for your application, discovering peers, connecting to a peer, and transferring data to a peer. The following sections describe how to do this.

### Initial setup

Before using the Wi-Fi P2P APIs, you must ensure that your application can access the hardware and that the device supports the Wi-Fi P2P protocol. If Wi-Fi P2P is supported, you can obtain an instance of [WifiP2pManager](#), create and register your broadcast receiver, and begin using the Wi-Fi P2P APIs.

1. Request permission to use the Wi-Fi hardware on the device and also declare your application to have the correct minimum SDK version in the Android manifest:

```

<uses-sdk android:minSdkVersion="14" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

```

2. Check to see if Wi-Fi P2P is on and supported. A good place to check this is in your broadcast receiver when it receives the [WIFI\\_P2P\\_STATE\\_CHANGED\\_ACTION](#) intent. Notify your activity of the Wi-Fi P2P state and react accordingly:

```

@Override
public void onReceive(Context context, Intent intent) {
 ...
 String action = intent.getAction();
 if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
 int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
 if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
 // WiFi P2P is enabled
 } else {
 // Wi-Fi P2P is not enabled
 }
 }
 ...
}

```

3. In your activity's `onCreate()` method, obtain an instance of `WifiP2pManager` and register your application with the Wi-Fi P2P framework by calling `initialize()`. This method returns a `WifiP2pManager.Channel`, which is used to connect your application to the Wi-Fi P2P framework. You should also create an instance of your broadcast receiver with the `WifiP2pManager` and `WifiP2pManager.Channel` objects along with a reference to your activity. This allows your broadcast receiver to notify your activity of interesting events and update it accordingly. It also lets you manipulate the device's Wi-Fi state if necessary:

```

WifiP2pManager mManager;
Channel mChannel;
BroadcastReceiver mReceiver;
...
@Override
protected void onCreate(Bundle savedInstanceState){
 ...
 mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
 mChannel = mManager.initialize(this, getMainLooper(), null);
 mReceiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
 ...
}

```

4. Create an intent filter and add the same intents that your broadcast receiver checks for:

```

IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState){
 ...
 mIntentFilter = new IntentFilter();
 mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
 mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
 mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
 mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
 ...
}

```

5. Register the broadcast receiver in the `onResume()` method of your activity and unregister it in the `onPause()` method of your activity:

```

/* register the broadcast receiver with the intent values to be matched */
@Override
protected void onResume() {
 super.onResume();
 registerReceiver(mReceiver, mIntentFilter);
}
/* unregister the broadcast receiver */
@Override
protected void onPause() {
 super.onPause();
 unregisterReceiver(mReceiver);
}

```

When you have obtained a `WifiP2pManager.Channel` and set up a broadcast receiver, your application can make Wi-Fi P2P method calls and receive Wi-Fi P2P intents.

You can now implement your application and use the Wi-Fi P2P features by calling the methods in [WifiP2pManager](#). The next sections describe how to do common actions such as discovering and connecting to peers.

## Discovering peers

To discover peers that are available to connect to, call [discoverPeers\(\)](#) to detect available peers that are in range. The call to this function is asynchronous and a success or failure is communicated to your application with [onSuccess\(\)](#) and [onFailure\(\)](#) if you created a [WifiP2pManager.ActionListener](#). The [onSuccess\(\)](#) method only notifies you that the discovery process succeeded and does not provide any information about the actual peers that it discovered, if any:

```
mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
 @Override
 public void onSuccess() {
 ...
 }

 @Override
 public void onFailure(int reasonCode) {
 ...
 }
});
```

If the discovery process succeeds and detects peers, the system broadcasts the [WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#) intent, which you can listen for in a broadcast receiver to obtain a list of peers. When your application receives the [WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#) intent, you can request a list of the discovered peers with [requestPeers\(\)](#). The following code shows how to set this up:

```
PeerListListener myPeerListListener;
...
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

 // request available peers from the wifi p2p manager. This is an
 // asynchronous call and the calling activity is notified with a
 // callback on PeerListListener.onPeersAvailable()
 if (mManager != null) {
 mManager.requestPeers(mChannel, myPeerListListener);
 }
}
```

The [requestPeers\(\)](#) method is also asynchronous and can notify your activity when a list of peers is available with [onPeersAvailable\(\)](#), which is defined in the [WifiP2pManager.PeerListListener](#) interface. The [onPeersAvailable\(\)](#) method provides you with an [WifiP2pDeviceList](#), which you can iterate through to find the peer that you want to connect to.

## Connecting to peers

When you have figured out the device that you want to connect to after obtaining a list of possible peers, call the [connect\(\)](#) method to connect to the device. This method call requires a [WifiP2pConfig](#) object that contains the information of the device to connect to. You can be notified of a connection success or failure through the [WifiP2pManager.ActionListener](#). The following code shows you how to create a connection to a desired device:

```

//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
mManager.connect(mChannel, config, new ActionListener() {

 @Override
 public void onSuccess() {
 //success logic
 }

 @Override
 public void onFailure(int reason) {
 //failure logic
 }
});

```

## Transferring data

Once a connection is established, you can transfer data between the devices with sockets. The basic steps of transferring data are as follows:

1. Create a [ServerSocket](#). This socket waits for a connection from a client on a specified port and blocks until it happens, so do this in a background thread.
2. Create a client [Socket](#). The client uses the IP address and port of the server socket to connect to the server device.
3. Send data from the client to the server. When the client socket successfully connects to the server socket, you can send data from the client to the server with byte streams.
4. The server socket waits for a client connection (with the [accept\(\)](#) method). This call blocks until a client connects, so call this in another thread. When a connection happens, the server device can receive the data from the client. Carry out any actions with this data, such as saving it to a file or presenting it to the user.

The following example, modified from the [Wi-Fi P2P Demo](#) sample, shows you how to create this client-server socket communication and transfer JPEG images from a client to a server with a service. For a complete working example, compile and run the [Wi-Fi P2P Demo](#) sample.

```

public static class FileServerAsyncTask extends AsyncTask {

 private Context context;
 private TextView statusText;

 public FileServerAsyncTask(Context context, View statusText) {
 this.context = context;
 this.statusText = (TextView) statusText;
 }

 @Override
 protected String doInBackground(Void... params) {
 try {

 /**
 * Create a server socket and wait for client connections. This
 * call blocks until a connection is accepted from a client
 */
 ServerSocket serverSocket = new ServerSocket(8888);
 Socket client = serverSocket.accept();

 /**
 * If this code is reached, a client has connected and transferred data
 * Save the input stream from the client as a JPEG file
 */
 final File f = new File(Environment.getExternalStorageDirectory() + "/"
 + context.getPackageName() + "/wifip2pshared-" + System.currentTimeMillis()
 + ".jpg");

 File dirs = new File(f.getParent());
 if (!dirs.exists())
 dirs.mkdirs();
 f.createNewFile();
 InputStream inputstream = client.getInputStream();
 copyFile(inputstream, new FileOutputStream(f));
 serverSocket.close();
 return f.getAbsolutePath();
 } catch (IOException e) {
 Log.e(WiFiDirectActivity.TAG, e.getMessage());
 return null;
 }
 }

 /**
 * Start activity that can handle the JPEG image
 */
 @Override
 protected void onPostExecute(String result) {
 if (result != null) {
 statusText.setText("File copied - " + result);
 Intent intent = new Intent();
 intent.setAction(android.content.Intent.ACTION_VIEW);
 intent.setDataAndType(Uri.parse("file://" + result), "image/*");
 context.startActivity(intent);
 }
 }
}

```

On the client, connect to the server socket with a client socket and transfer data. This example transfers a JPEG file on the client device's file system.

```
Context context = this.getApplicationContext();
String host;
int port;
int len;
Socket socket = new Socket();
byte buf[] = new byte[1024];
...
try {
 /**
 * Create a client socket with the host,
 * port, and timeout information.
 */
 socket.bind(null);
 socket.connect((new InetSocketAddress(host, port)), 500);

 /**
 * Create a byte stream from a JPEG file and pipe it to the output stream
 * of the socket. This data will be retrieved by the server device.
 */
 OutputStream outputStream = socket.getOutputStream();
 ContentResolver cr = context.getContentResolver();
 InputStream inputStream = null;
 inputStream = cr.openInputStream(Uri.parse("path/to/picture.jpg"));
 while ((len = inputStream.read(buf)) != -1) {
 outputStream.write(buf, 0, len);
 }
 outputStream.close();
 inputStream.close();
} catch (FileNotFoundException e) {
 //catch logic
} catch (IOException e) {
 //catch logic
}

/**
 * Clean up any open sockets when done
 * transferring or if an exception occurred.
 */
finally {
 if (socket != null) {
 if (socket.isConnected()) {
 try {
 socket.close();
 } catch (IOException e) {
 //catch logic
 }
 }
 }
}
```

# Session Initiation Protocol

## In this document

- › Requirements and Limitations
- › Classes and Interfaces
- › Creating the Manifest
- › Creating a SIP Manager
- › Registering with a SIP Server
- › Making an Audio Call
- › Receiving Calls
- › Testing SIP Applications

## Key classes

- › [SipManager](#)
- › [SipProfile](#)
- › [SipAudioCall](#)

## Related samples

- › [SipDemo](#)

Android provides an API that supports the Session Initiation Protocol (SIP). This lets you add SIP-based internet telephony features to your applications. Android includes a full SIP protocol stack and integrated call management services that let applications easily set up outgoing and incoming voice calls, without having to manage sessions, transport-level communication, or audio record or playback directly.

Here are examples of the types of applications that might use the SIP API:

- Video conferencing.
- Instant messaging.

## Requirements and Limitations

Here are the requirements for developing a SIP application:

- You must have a mobile device that is running Android 2.3 or higher.
- SIP runs over a wireless data connection, so your device must have a data connection (with a mobile data service or Wi-Fi). This means that you can't test on AVD—you can only test on a physical device. For details, see [Testing SIP Applications](#).
- Each participant in the application's communication session must have a SIP account. There are many different SIP providers that offer SIP accounts.

## SIP API Classes and Interfaces

Here is a summary of the classes and one interface ([SipRegistrationListener](#)) that are included in the Android SIP API:

Class/Interface	Description
<a href="#">SipAudioCall</a>	Handles an Internet audio call over SIP.

<a href="#">SipAudioCall.Listener</a>	Description events relating to a SIP call, such as when a call is being received ("on ringing") or a call is outgoing ("on calling").
<a href="#">SipErrorCode</a>	Defines error codes returned during SIP actions.
<a href="#">SipManager</a>	Provides APIs for SIP tasks, such as initiating SIP connections, and provides access to related SIP services.
<a href="#">SipProfile</a>	Defines a SIP profile, including a SIP account, domain and server information.
<a href="#">SipProfile.Builder</a>	Helper class for creating a SipProfile.
<a href="#">SipSession</a>	Represents a SIP session that is associated with a SIP dialog or a standalone transaction not within a dialog.
<a href="#">SipSession.Listener</a>	Listener for events relating to a SIP session, such as when a session is being registered ("on registering") or a call is outgoing ("on calling").
<a href="#">SipSession.State</a>	Defines SIP session states, such as "registering", "outgoing call", and "in call".
<a href="#">SipRegistrationListener</a>	An interface that is a listener for SIP registration events.

## Creating the Manifest

If you are developing an application that uses the SIP API, remember that the feature is supported only on Android 2.3 (API level 9) and higher versions of the platform. Also, among devices running Android 2.3 (API level 9) or higher, not all devices will offer SIP support.

To use SIP, add the following permissions to your application's manifest:

- `android.permission.USE_SIP`
- `android.permission.INTERNET`

To ensure that your application can only be installed on devices that are capable of supporting SIP, add the following to your application's manifest:

```
<uses-sdk android:minSdkVersion="9" />
```

This indicates that your application requires Android 2.3 or higher. For more information, see [API Levels](#) and the documentation for the `<uses-sdk>` element.

To control how your application is filtered from devices that do not support SIP (for example, on Google Play), add the following to your application's manifest:

```
<uses-feature android:name="android.hardware.sip.voip" />
```

This states that your application uses the SIP API. The declaration should include an `android:required` attribute that indicates whether you want the application to be filtered from devices that do not offer SIP support. Other `<uses-feature>` declarations may also be needed, depending on your implementation. For more information, see the documentation for the `<uses-feature>` element.

If your application is designed to receive calls, you must also define a receiver ([BroadcastReceiver](#) subclass) in the application's manifest:

```
<receiver android:name=".IncomingCallReceiver" android:label="Call Receiver" />
```

Here are excerpts from the **SipDemo** manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.android.sip">
 ...
 <receiver android:name=".IncomingCallReceiver" android:label="Call Receiver" />
 ...
 <uses-sdk android:minSdkVersion="9" />
 <uses-permission android:name="android.permission.USE_SIP" />
 <uses-permission android:name="android.permission.INTERNET" />
 ...
 <uses-feature android:name="android.hardware.sip.voip" android:required="true" />
 <uses-feature android:name="android.hardware.wifi" android:required="true" />
 <uses-feature android:name="android.hardware.microphone" android:required="true" />
</manifest>
```

## Creating a SipManager

To use the SIP API, your application must create a `SipManager` object. The `SipManager` takes care of the following in your application:

- Initiating SIP sessions.
- Initiating and receiving calls.
- Registering and unregistering with a SIP provider.
- Verifying session connectivity.

You instantiate a new `SipManager` as follows:

```
public SipManager mSipManager = null;
...
if (mSipManager == null) {
 mSipManager = SipManager.newInstance(this);
}
```

## Registering with a SIP Server

A typical Android SIP application involves one or more users, each of whom has a SIP account. In an Android SIP application, each SIP account is represented by a `SipProfile` object.

A `SipProfile` defines a SIP profile, including a SIP account, and domain and server information. The profile associated with the SIP account on the device running the application is called the *local profile*. The profile that the session is connected to is called the *peer profile*. When your SIP application logs into the SIP server with the local `SipProfile`, this effectively registers the device as the location to send SIP calls to for your SIP address.

This section shows how to create a `SipProfile`, register it with a SIP server, and track registration events.

You create a `SipProfile` object as follows:

```
public SipProfile mSipProfile = null;
...
SipProfile.Builder builder = new SipProfile.Builder(username, domain);
builder.setPassword(password);
mSipProfile = builder.build();
```

The following code excerpt opens the local profile for making calls and/or receiving generic SIP calls. The caller can make subsequent calls through `mSipManager.makeAudioCall`. This excerpt also sets the action `android.SipDemo.INCOMING_CALL`, which will be used by an intent filter when the device receives a call (see [Setting up an intent filter to receive calls](#)). This is the registration step:

```
Intent intent = new Intent();
intent.setAction("android.SipDemo.INCOMING_CALL");
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, Intent.FILL_IN_DATA);
mSipManager.open(mSipProfile, pendingIntent, null);
```

Finally, this code sets a `SipRegistrationListener` on the `SipManager`. This tracks whether the `SipProfile` was successfully registered with your SIP service provider:

```
mSipManager.setRegistrationListener(mSipProfile.getUriString(), new SipRegistrationListener() {

 public void onRegistering(String localProfileUri) {
 updateStatus("Registering with SIP Server...");
 }

 public void onRegistrationDone(String localProfileUri, long expiryTime) {
 updateStatus("Ready");
 }

 public void onRegistrationFailed(String localProfileUri, int errorCode,
 String errorMessage) {
 updateStatus("Registration failed. Please check settings.");
 }
})
```

When your application is done using a profile, it should close it to free associated objects into memory and unregister the device from the server. For example:

```
public void closeLocalProfile() {
 if (mSipManager == null) {
 return;
 }
 try {
 if (mSipProfile != null) {
 mSipManager.close(mSipProfile.getUriString());
 }
 } catch (Exception ee) {
 Log.d("WalkieTalkieActivity/onDestroy", "Failed to close local profile.", ee);
 }
}
```

## Making an Audio Call

To make an audio call, you must have the following in place:

- A `SipProfile` that is making the call (the "local profile"), and a valid SIP address to receive the call (the "peer profile").
- A `SipManager` object.

To make an audio call, you should set up a `SipAudioCall.Listener`. Much of the client's interaction with the SIP stack happens through listeners. In this snippet, you see how the `SipAudioCall.Listener` sets things up after the call is established:

```
SipAudioCall.Listener listener = new SipAudioCall.Listener() {
 @Override
 public void onCallEstablished(SipAudioCall call) {
 call.startAudio();
 call.setSpeakerMode(true);
 call.toggleMute();
 ...
 }

 @Override
 public void onCallEnded(SipAudioCall call) {
 // Do something.
 }
};
```

Once you've set up the `SipAudioCall.Listener`, you can make the call. The `SipManager` method `makeAudioCall` takes the following parameters:

- A local SIP profile (the caller).
- A peer SIP profile (the user being called).
- A `SipAudioCall.Listener` to listen to the call events from `SipAudioCall`. This can be `null`, but as shown above, the listener is used to set things up once the call is established.
- The timeout value, in seconds.

For example:

```
call = mSipManager.makeAudioCall(mSipProfile.getUriString(), sipAddress, listener, 30);
```

## Receiving Calls

To receive calls, a SIP application must include a subclass of `BroadcastReceiver` that has the ability to respond to an intent indicating that there is an incoming call. Thus, you must do the following in your application:

- In `AndroidManifest.xml`, declare a `<receiver>`. In `SipDemo`, this is `<receiver android:name=".IncomingCallReceiver" android:label="Call Receiver" />`.
- Implement the receiver, which is a subclass of `BroadcastReceiver`. In `SipDemo`, this is `IncomingCallReceiver`.
- Initialize the local profile (`SipProfile`) with a pending intent that fires your receiver when someone calls the local profile.
- Set up an intent filter that filters by the action that represents an incoming call. In `SipDemo`, this action is `android.SipDemo.INCOMING_CALL`.

### Subclassing BroadcastReceiever

To receive calls, your SIP application must subclass `BroadcastReceiver`. The Android system handles incoming SIP calls and broadcasts an "incoming call" intent (as defined by the application) when it receives a call. Here is the subclassed `BroadcastReceiver` code from the `SipDemo` sample.

```

/**
 * Listens for incoming SIP calls, intercepts and hands them off to WalkieTalkieActivity.
 */
public class IncomingCallReceiver extends BroadcastReceiver {
 /**
 * Processes the incoming call, answers it, and hands it over to the
 * WalkieTalkieActivity.
 * @param context The context under which the receiver is running.
 * @param intent The intent being received.
 */
 @Override
 public void onReceive(Context context, Intent intent) {
 SipAudioCall incomingCall = null;
 try {
 SipAudioCall.Listener listener = new SipAudioCall.Listener() {
 @Override
 public void onRinging(SipAudioCall call, SipProfile caller) {
 try {
 call.answerCall(30);
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
 };
 WalkieTalkieActivity wtActivity = (WalkieTalkieActivity) context;
 incomingCall = wtActivity.mSipManager.takeAudioCall(intent, listener);
 incomingCall.answerCall(30);
 incomingCall.startAudio();
 incomingCall.setSpeakerMode(true);
 if(incomingCall.isMuted()) {
 incomingCall.toggleMute();
 }
 wtActivity.call = incomingCall;
 wtActivity.updateStatus(incomingCall);
 } catch (Exception e) {
 if (incomingCall != null) {
 incomingCall.close();
 }
 }
 }
}

```

## Setting up an intent filter to receive calls

When the SIP service receives a new call, it sends out an intent with the action string provided by the application. In **SipDemo**, this action string is `android.SipDemo.INCOMING_CALL`.

This code excerpt from **SipDemo** shows how the `SipProfile` object gets created with a pending intent based on the action string `android.SipDemo.INCOMING_CALL`. The `PendingIntent` object will perform a broadcast when the `SipProfile` receives a call:

```

public SipManager mSipManager = null;
public SipProfile mSipProfile = null;
...

Intent intent = new Intent();
intent.setAction("android.SipDemo.INCOMING_CALL");
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, Intent.FILL_IN_DATA);
mSipManager.open(mSipProfile, pendingIntent, null);

```

The broadcast will be intercepted by the intent filter, which will then fire the receiver (`IncomingCallReceiver`). You can specify an intent filter in your application's manifest file, or do it in code as in the **SipDemo** sample application's `onCreate()` method of the application's `Activity`:

```
public class WalkieTalkieActivity extends Activity implements View.OnTouchListener {
 ...
 public IncomingCallReceiver callReceiver;
 ...

 @Override
 public void onCreate(Bundle savedInstanceState) {

 IntentFilter filter = new IntentFilter();
 filter.addAction("android.SipDemo.INCOMING_CALL");
 callReceiver = new IncomingCallReceiver();
 this.registerReceiver(callReceiver, filter);
 ...
 }
 ...
}
```

## Testing SIP Applications

---

To test SIP applications, you need the following:

- A mobile device that is running Android 2.3 or higher. SIP runs over wireless, so you must test on an actual device. Testing on AVD won't work.
- A SIP account. There are many different SIP providers that offer SIP accounts.
- If you are placing a call, it must also be to a valid SIP account.

To test a SIP application:

1. On your device, connect to wireless (**Settings > Wireless & networks > Wi-Fi > Wi-Fi settings**).
2. Set up your mobile device for testing, as described in [Developing on a Device](#).
3. Run your application on your mobile device, as described in [Developing on a Device](#).
4. If you are using Android Studio, you can view the application log output by opening the Event Log console (**View > Tool Windows > Event Log**).
5. Ensure your application is configured to launch Logcat automatically when it runs:
  - a. Select **Run > Edit Configurations**.
  - b. Select the **Miscellaneous** tab in the **Run/Debug Configurations** window.
  - c. Under **Logcat**, select **Show logcat automatically** then select **OK**.

# 蓝牙

## 本文内容

- › [基础知识](#)
- › [蓝牙权限](#)
- › [设置蓝牙](#)
- › [查找设备](#)
  - › [查询配对的设备](#)
  - › [发现设备](#)
- › [连接设备](#)
  - › [连接为服务器](#)
  - › [连接为客户端](#)
- › [管理连接](#)
- › [使用配置文件](#)
  - › [供应商特定的 AT 命令](#)
  - › [健康设备配置文件](#)

## 关键类

- › [BluetoothAdapter](#)
- › [BluetoothDevice](#)
- › [BluetoothSocket](#)
- › [BluetoothServerSocket](#)

## 相关示例

- › [蓝牙聊天](#)
- › [蓝牙 HDP（健康设备配置文件）](#)

Android 平台包含蓝牙网络堆栈支持，凭借此项支持，设备能以无线方式与其他蓝牙设备交换数据。应用框架提供了通过 Android Bluetooth API 访问蓝牙功能的途径。这些 API 允许应用以无线方式连接到其他蓝牙设备，从而实现点到点和多点无线功能。

使用 Bluetooth API，Android 应用可执行以下操作：

- 扫描其他蓝牙设备
- 查询本地蓝牙适配器的配对蓝牙设备
- 建立 RFCOMM 通道
- 通过服务发现连接到其他设备
- 与其他设备进行双向数据传输
- 管理多个连接

本文将介绍如何使用传统蓝牙。传统蓝牙适用于电池使用强度较大的操作，例如 Android 设备之间的流式传输和通信等。针对具有低功耗要求的蓝牙设备，Android 4.3（API 级别 18）中引入了面向低功耗蓝牙的 API 支持。如需了解更多信息，请参阅[低功耗蓝牙](#)。

## 基础知识

本文将介绍如何使用 Android Bluetooth API 来完成使用蓝牙进行通信的四项主要任务：设置蓝牙、查找局部区域内的配对设备或可用设备、连接设备，以及在设备之间传输数据。

`android.bluetooth` 包中提供了所有 Bluetooth API。下面概要列出了创建蓝牙连接所需的类和接口：

#### `BluetoothAdapter`

表示本地蓝牙适配器（蓝牙无线装置）。`BluetoothAdapter` 是所有蓝牙交互的入口点。利用它可以发现其他蓝牙设备，查询绑定（配对）设备的列表，使用已知的 MAC 地址实例化 `BluetoothDevice`，以及创建 `BluetoothServerSocket` 以侦听来自其他设备的通信。

#### `BluetoothDevice`

表示远程蓝牙设备。利用它可以通过 `BluetoothSocket` 请求与某个远程设备建立连接，或查询有关该设备的信息，例如设备的名称、地址、类和绑定状态等。

#### `BluetoothSocket`

表示蓝牙套接字接口（与 TCP `Socket` 相似）。这是允许应用通过 `InputStream` 和 `OutputStream` 与其他蓝牙设备交换数据的连接点。

#### `BluetoothServerSocket`

表示用于侦听传入请求的开放服务器套接字（类似于 TCP `ServerSocket`）。要连接两台 Android 设备，其中一台设备必须使用此类开放一个服务器套接字。当一台远程蓝牙设备向此设备发出连接请求时，`BluetoothServerSocket` 将会在接受连接后返回已连接的 `BluetoothSocket`。

#### `BluetoothClass`

描述蓝牙设备的一般特征和功能。这是一组只读属性，用于定义设备的主要和次要设备类及其服务。不过，它不能可靠地描述设备支持的所有蓝牙配置文件和服务，而是适合作为设备类型提示。

#### `BluetoothProfile`

表示蓝牙配置文件的接口。蓝牙配置文件是适用于设备间蓝牙通信的无线接口规范。免提配置文件便是一个示例。如需了解有关配置文件的详细讨论，请参阅[使用配置文件](#)

#### `BluetoothHeadset`

提供蓝牙耳机支持，以便与手机配合使用。其中包括蓝牙耳机和免提（1.5 版）配置文件。

#### `BluetoothA2dp`

定义高质量音频如何通过蓝牙连接和流式传输，从一台设备传输到另一台设备。“A2DP”代表高级音频分发配置文件。

#### `BluetoothHealth`

表示用于控制蓝牙服务的健康设备配置文件代理。

#### `BluetoothHealthCallback`

用于实现 `BluetoothHealth` 回调的抽象类。您必须扩展此类并实现回调方法，以接收关于应用注册状态和蓝牙通道状态变化的更新内容。

#### `BluetoothHealthAppConfiguration`

表示第三方蓝牙健康应用注册的应用配置，以便与远程蓝牙健康设备通信。

#### `BluetoothProfile.ServiceListener`

在 `BluetoothProfile` IPC 客户端连接到服务（即，运行特定配置文件的内部服务）或断开服务连接时向其发送通知的接口。

# 蓝牙权限

要在应用中使用蓝牙功能，必须声明蓝牙权限 `BLUETOOTH`。您需要此权限才能执行任何蓝牙通信，例如请求连接、接受连接和传输数据等。

如果您希望您的应用启动设备发现或操作蓝牙设置，则还必须声明 `BLUETOOTH_ADMIN` 权限。大多数应用需要此权限仅仅为了能够发现本地蓝牙设备。除非该应用是将要应用户请求修改蓝牙设置的“超级管理员”，否则不应使用此权限所授予的其他能力。**注：**如果要使用 `BLUETOOTH_ADMIN` 权限，则还必须拥有 `BLUETOOTH` 权限。

在您的应用清单文件中声明蓝牙权限。例如：

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

如需了解有关声明应用权限的更多信息，请参阅 [<uses-permission>](#) 参考资料。

# 设置蓝牙

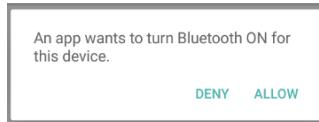


图 1：启用蓝牙对话框。

您需要验证设备支持蓝牙，并且如果支持确保将其启用，然后您的应用才能通过蓝牙进行通信。

如果不支持蓝牙，则应正常停用任何蓝牙功能。如果支持蓝牙但已停用此功能，则可以请求用户在不离开应用的同时启用蓝牙。可使用 `BluetoothAdapter`，分两步完成此设置：

## 1. 获取 `BluetoothAdapter`

所有蓝牙 Activity 都需要 `BluetoothAdapter`。要获取 `BluetoothAdapter`，请调用静态 `getOrDefaultAdapter()` 方法。这将返回一个表示设备自身的蓝牙适配器（蓝牙无线装置）的 `BluetoothAdapter`。整个系统有一个蓝牙适配器，并且您的应用可使用此对象与之交互。如果 `getOrDefaultAdapter()` 返回 null，则该设备不支持蓝牙，您的操作到此为止。例如：

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
 // Device does not support Bluetooth
}
```

## 2. 启用蓝牙

下一步，您需要确保已启用蓝牙。调用 `isEnabled()` 以检查当前是否已启用蓝牙。如果此方法返回 false，则表示蓝牙处于停用状态。要请求启用蓝牙，请使用 `ACTION_REQUEST_ENABLE` 操作 Intent 调用 `startActivityForResult()`。这将通过系统设置发出启用蓝牙的请求（无需停止您的应用）。例如：

```
if (!mBluetoothAdapter.isEnabled()) {
 Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
 startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

如图 1 所示，将显示对话框，请求用户允许启用蓝牙。如果用户响应“Yes”，系统将开始启用蓝牙，并在该进程完成（或失败）后将焦点返回到您的应用。

传递给 `startActivityForResult()` 的 `REQUEST_ENABLE_BT` 常量是在局部定义的整型（必须大于 0），系统会将其作为 `requestCode` 参数传递回您的 `onActivityResult()` 实现。

如果成功启用蓝牙，您的 Activity 将会在 `onActivityResult()` 回调中收到 `RESULT_OK` 结果代码。如果由于某个错误（或用户响应“No”）而没有启用蓝牙，则结果代码为 `RESULT_CANCELED`。

您的应用还可以选择侦听 `ACTION_STATE_CHANGED` 广播 Intent，每当蓝牙状态发生变化时，系统都会广播此 Intent。此广播包含额外字段

`EXTRA_STATE` 和 `EXTRA_PREVIOUS_STATE`，二者分别包含新的和旧的蓝牙状态。这些额外字段可能的值包括 `STATE_TURNING_ON`、`STATE_ON`、`STATE_TURNING_OFF` 和 `STATE_OFF`。侦听此广播适用于检测在您的应用运行期间对蓝牙状态所做的更改。

**提示：**启用可检测性将会自动启用蓝牙。如果您计划在执行蓝牙 Activity 之前一直启用设备的可检测性，则可以跳过上述步骤 2。阅读下面的[启用可检测性](#)。

## 查找设备

使用 `BluetoothAdapter`，您可以通过设备发现或通过查询配对（绑定）设备的列表来查找远程蓝牙设备。

设备发现是一个扫描过程，它会搜索局部区域内已启用蓝牙功能的设备，然后请求一些关于各台设备的信息（有时也被称为“发现”、“查询”或“扫描”）。但局部区域内的蓝牙设备仅在其当前已启用可检测性时才会响应发现请求。如果设备可检测到，它将通过共享一些信息（例如设备名称、类及其唯一 MAC 地址）来响应发现请求。利用此信息，执行发现的设备可以选择发起到被发现设备的连接。

在首次与远程设备建立连接后，将会自动向用户显示配对请求。设备完成配对后，将会保存关于该设备的基本信息（例如设备名称、类和 MAC 地址），并且可使用 Bluetooth API 读取这些信息。利用远程设备的已知 MAC 地址可随时向其发起连接，而无需执行发现操作（假定该设备处于有效范围内）。

请记住，被配对与被连接之间存在差别。被配对意味着两台设备知晓彼此的存在，具有可用于身份验证的共享链路密钥，并且能够与彼此建立加密连接。被连接意味着设备当前共享一个 RFCOMM 通道，并且能够向彼此传输数据。当前的 Android Bluetooth API 要求对设备进行配对，然后才能建立 RFCOMM 连接。（在使用 Bluetooth API 发起加密连接时，会自动执行配对）。

以下几节介绍如何查找已配对的设备，或使用设备发现来发现新设备。

**注：**Android 设备默认处于不可检测到状态。用户可通过系统设置将设备设为在有限的时间内处于可检测到状态，或者，应用可请求用户在不离开应用的同时启用可检测性。下面讨论如何[启用可检测性](#)。

## 查询配对的设备

在执行设备发现之前，有必要查询已配对的设备集，以了解所需的设备是否处于已知状态。为此，请调用 `getBondedDevices()`。这将返回表示已配对设备的一组 `BluetoothDevice`。例如，您可以查询所有已配对设备，然后使用  `ArrayAdapter` 向用户显示每台设备的名称：

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
 // Loop through paired devices
 for (BluetoothDevice device : pairedDevices) {
 // Add the name and address to an array adapter to show in a ListView
 mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
 }
}
```

要发起连接，`BluetoothDevice` 对象仅仅需要提供 MAC 地址。在此示例中，它将保存为显示给用户的  `ArrayAdapter` 的一部分。之后可提取该 MAC 地址，以便发起连接。您可以在有关[连接设备](#)的部分详细了解如何创建连接。

## 发现设备

要开始发现设备，只需调用 `startDiscovery()`。该进程为异步进程，并且该方法会立即返回一个布尔值，指示是否已成功启动发现操作。发现进程通常包含约 12 秒钟的查询扫描，之后对每台发现的设备进行页面扫描，以检索其蓝牙名称。

您的应用必须针对 `ACTION_FOUND` Intent 注册一个 `BroadcastReceiver`，以便接收每台发现的设备的相关信息。针对每台设备，系统将会广播 `ACTION_FOUND` Intent。此 Intent 将携带额外字段 `EXTRA_DEVICE` 和 `EXTRA_CLASS`，二者分别包含 `BluetoothDevice` 和 `BluetoothClass`。例如，下面说明了在发现设备时如何注册以处理广播。

```

// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
 public void onReceive(Context context, Intent intent) {
 String action = intent.getAction();
 // When discovery finds a device
 if (BluetoothDevice.ACTION_FOUND.equals(action)) {
 // Get the BluetoothDevice object from the Intent
 BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
 // Add the name and address to an array adapter to show in a ListView
 mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
 }
 }
};

// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy

```

要发起连接，`BluetoothDevice` 对象仅仅需要提供 MAC 地址。在此示例中，它将保存为显示给用户的 ArrayAdapter 的一部分。之后可提取该 MAC 地址，以便发起连接。您可以在有关[连接设备](#)的部分详细了解如何创建连接。

**注意：**执行设备发现对于蓝牙适配器而言是一个非常繁重的操作过程，并且会消耗大量资源。在找到要连接的设备后，确保始终使用 `cancelDiscovery()` 停止发现，然后再尝试连接。此外，如果您已经保持与某台设备的连接，那么执行发现操作可能会大幅减少可用于该连接的带宽，因此不应该在处于连接状态时执行发现操作。

## 启用可检测性

如果您希望将本地设备设为可被其他设备检测到，请使用 `ACTION_REQUEST_DISCOVERABLE` 操作 Intent 调用 `startActivityForResult(Intent, int)`。这将通过系统设置发出启用可检测到模式的请求（无需停止您的应用）。默认情况下，设备将变为可检测到并持续 120 秒钟。您可以通过添加 `EXTRA_DISCOVERABLE_DURATION` Intent Extra 来定义不同的持续时间。应用可以设置的最大持续时间为 3600 秒，值为 0 则表示设备始终可检测到。任何小于 0 或大于 3600 的值都会自动设为 120 秒。例如，以下片段会将持续时间设为 300 秒：

```

Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);

```

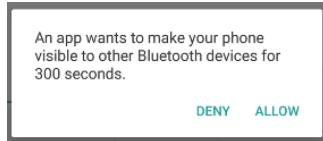


图 2：启用可检测性对话框。

如图 2 所示，将显示对话框，请求用户允许将设备设为可检测到。如果用户响应“Yes”，则设备将变为可检测到并持续指定的时间量。然后，您的 Activity 将会收到对 `onActivityResult()` 回调的调用，其结果代码等于设备可检测到的持续时间。如果用户响应“No”或出现错误，结果代码将为 `RESULT_CANCELED`。

**注：**如果尚未在设备上启用蓝牙，则启用设备可检测性将会自动启用蓝牙。

设备将在分配的时间内以静默方式保持可检测到模式。如果您希望在可检测到模式发生变化时收到通知，您可以针对 `ACTION_SCAN_MODE_CHANGED` Intent 注册 BroadcastReceiver。它将包含额外字段 `EXTRA_SCAN_MODE` 和 `EXTRA_PREVIOUS_SCAN_MODE`，二者分别告知您新的和旧的扫描模式。每个字段可能的值包括 `SCAN_MODE_CONNECTABLE_DISCOVERABLE`、`SCAN_MODE_CONNECTABLE` 或 `SCAN_MODE_NONE`，这些值分别指示设备处于可检测到模式、未处于可检测到模式但仍能接收连接，或未处于可检测到模式并且无法接收连接。

如果您将要发起到远程设备的连接，则无需启用设备可检测性。仅当您希望您的应用托管将用于接受传入连接的服务器套接字时，才有必要启用可检测性，因为远程设备必须能够发现该设备，然后才能发起连接。

## 连接设备

要在两台设备上的应用之间创建连接，必须同时实现服务器端和客户端机制，因为其中一台设备必须开放服务器套接字，而另一台设备必须发起连接（使用服务器设备的 MAC 地址发起连接）。当服务器和客户端在同一 RFCOMM 通道上分别拥有已连接的 [BluetoothSocket](#) 时，二者将被视为彼此连接。这种情况下，每台设备都能获得输入和输出流式传输，并且可以开始传输数据，在有关[管理连接](#)的部分将会讨论这一主题。本部分介绍如何在两台设备之间发起连接。

服务器设备和客户端设备分别以不同的方法获得需要的 [BluetoothSocket](#)。服务器将在传入连接被接受时收到套接字。客户端将在其打开到服务器的 RFCOMM 通道时收到该套接字。

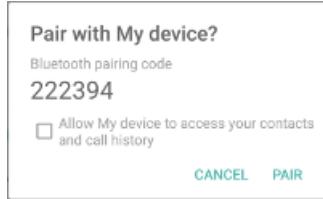


图 3：蓝牙配对对话框。

一种实现技术是自动将每台设备准备为一个服务器，从而使每台设备开放一个服务器套接字并侦听连接。然后任一设备可以发起与另一台设备的连接，并成为客户端。或者，其中一台设备可显式“托管”连接并按需开放一个服务器套接字，而另一台设备则直接发起连接。

**注：**如果两台设备之前尚未配对，则在连接过程中，Android 框架会自动向用户显示配对请求通知或对话框（如图 3 所示）。因此，在尝试连接设备时，您的应用无需担心设备是否已配对。您的 RFCOMM 连接尝试将被阻塞，直至用户成功完成配对或配对失败（包括用户拒绝配对、配对失败或超时）。

## 连接为服务器

当您需要连接两台设备时，其中一台设备必须通过保持开放的 [BluetoothServerSocket](#) 来充当服务器。服务器套接字的用途是侦听传入的连接请求，并在接受一个请求后提供已连接的 [BluetoothSocket](#)。从 [BluetoothServerSocket](#) 获取 [BluetoothSocket](#) 后，可以（并且应该）舍弃 [BluetoothServerSocket](#)，除非您需要接受更多连接。

以下是设置服务器套接字并接受连接的基本过程：

1. 通过调用 `listenUsingRfcommWithServiceRecord(String, UUID)` 获取 [BluetoothServerSocket](#)。

该字符串是您的服务的可识别名称，系统会自动将其写入到设备上的新服务发现协议 (SDP) 数据库条目（可使用任意名称，也可直接使用您的应用名称）。UUID 也包含在 SDP 条目中，并且将作为与客户端设备的连接协议的基础。也就是说，当客户端尝试连接此设备时，它会携带能够唯一标识其想要连接的服务的 UUID。两个 UUID 必须匹配，在下一步中，连接才会被接受。

2. 通过调用 `accept()` 开始侦听连接请求。

这是一个阻塞调用。它将在连接被接受或发生异常时返回。仅当远程设备发送的连接请求中所包含的 UUID 与向此侦听服务器套接字注册的 UUID 相匹配时，连接才会被接受。操作成功后，`accept()` 将会返回已连接的 [BluetoothSocket](#)。

3. 除非您想要接受更多连接，否则请调用 `close()`。

这将释放服务器套接字及其所有资源，但不会关闭 `accept()` 所返回的已连接的 [BluetoothSocket](#)。与 TCP/IP 不同，RFCOMM 一次只允许每个通道有一个已连接的客户端，因此大多数情况下，在接受已连接的套接字后立即在 [BluetoothServerSocket](#) 上调用 `close()` 是行得通的。

`accept()` 调用不应在主 Activity UI 线程中执行，因为它是阻塞调用，并会阻止与应用的任何其他交互。在您的应用所管理的新线程中使用 [BluetoothServerSocket](#) 或 [BluetoothSocket](#) 完成所有工作，这通常是一种行之有效的做法。要终止 `accept()` 等被阻塞的调用，请通过另一个线程在 [BluetoothServerSocket](#)（或 [BluetoothSocket](#)）上调用 `close()`，被阻塞的调用将会立即返回。请注意，[BluetoothServerSocket](#) 或 [BluetoothSocket](#) 中的所有方法都是线程安全的方法。

### 关于 UUID

通用唯一标识符 (UUID) 是用于唯一标识信息的字符串 ID 的 128 位标准化格式。UUID 的特点是其足够庞大，因此您可以选择任意随机值而不会发生冲突。在此示例中，它被用于唯一标识应用的蓝牙服务。要获取 UUID 以用于您的应用，您可以使用网络上的众多随机 UUID 生成器之一，然后使用 `fromString(String)` 初始化一个 UUID。

## 示例

以下是一个用于接受传入连接的服务器组件的简化线程：

```

private class AcceptThread extends Thread {
 private final BluetoothServerSocket mmServerSocket;

 public AcceptThread() {
 // Use a temporary object that is later assigned to mmServerSocket,
 // because mmServerSocket is final
 BluetoothServerSocket tmp = null;
 try {
 // MY_UUID is the app's UUID string, also used by the client code
 tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
 } catch (IOException e) { }
 mmServerSocket = tmp;
 }

 public void run() {
 BluetoothSocket socket = null;
 // Keep listening until exception occurs or a socket is returned
 while (true) {
 try {
 socket = mmServerSocket.accept();
 } catch (IOException e) {
 break;
 }
 // If a connection was accepted
 if (socket != null) {
 // Do work to manage the connection (in a separate thread)
 manageConnectedSocket(socket);
 mmServerSocket.close();
 break;
 }
 }
 }

 /** Will cancel the listening socket, and cause the thread to finish */
 public void cancel() {
 try {
 mmServerSocket.close();
 } catch (IOException e) { }
 }
}

```

在此示例中，只需要一个传入连接，因此在接受连接并获取 `BluetoothSocket` 之后，应用会立即将获取的 `BluetoothSocket` 发送到单独的线程，关闭 `BluetoothServerSocket` 并中断循环。

请注意，当 `accept()` 返回 `BluetoothSocket` 时，表示套接字已连接好，因此您不应该像在客户端那样调用 `connect()`。

`manageConnectedSocket()` 是应用中的虚构方法，它将启动用于传输数据的线程，在有关 [管理连接](#)的部分将会讨论这一主题。

在完成传入连接的侦听后，通常应立即关闭您的 `BluetoothServerSocket`。在此示例中，获取 `BluetoothSocket` 后立即调用 `close()`。您也可能希望在您的线程中提供一个公共方法，以便在需要停止侦听服务器套接字时关闭私有 `BluetoothSocket`。

## 连接为客户端

要发起与远程设备（保持开放的服务器套接字的设备）的连接，必须首先获取表示该远程设备的 `BluetoothDevice` 对象。（在前面有关 [查找设备](#)的部分介绍了如何获取 `BluetoothDevice`）。然后必须使用 `BluetoothDevice` 来获取 `BluetoothSocket` 并发起连接。

以下是基本过程：

1. 使用 `BluetoothDevice`，通过调用 `createRfcommSocketToServiceRecord(UUID)` 获取 `BluetoothSocket`。

这将初始化将要连接到 `BluetoothDevice` 的 `BluetoothSocket`。此处传递的 UUID 必须与服务器设备在使用 `listenUsingRfcommWithServiceRecord(String, UUID)` 开放其 `BluetoothServerSocket` 时所用的 UUID 相匹配。要使用相同的 UUID，只需将该 UUID 字符串以硬编码方式编入应用，然后通过服务器代码和客户端代码引用该字符串。

2. 通过调用 `connect()` 发起连接。

执行此调用时，系统将会在远程设备上执行 SDP 查找，以便匹配 UUID。如果查找成功并且远程设备接受了该连接，它将共享 RFCOMM 通道以便在连接期间使用，并且 `connect()` 将会返回。此方法为阻塞调用。如果由于任何原因连接失败或 `connect()` 方法超时（大约 12

秒之后），它将会引发异常。

由于 `connect()` 为阻塞调用，因此该连接过程应始终在主 Activity 线程以外的线程中执行。

注：在调用 `connect()` 时，应始终确保设备未在执行设备发现。如果正在进行发现操作，则会大幅降低连接尝试的速度，并增加连接失败的可能性。

## 示例

以下是发起蓝牙连接的线程的基本示例：

```
private class ConnectThread extends Thread {
 private final BluetoothSocket mmSocket;
 private final BluetoothDevice mmDevice;

 public ConnectThread(BluetoothDevice device) {
 // Use a temporary object that is later assigned to mmSocket,
 // because mmSocket is final
 BluetoothSocket tmp = null;
 mmDevice = device;

 // Get a BluetoothSocket to connect with the given BluetoothDevice
 try {
 // MY_UUID is the app's UUID string, also used by the server code
 tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
 } catch (IOException e) { }
 mmSocket = tmp;
 }

 public void run() {
 // Cancel discovery because it will slow down the connection
 mBluetoothAdapter.cancelDiscovery();

 try {
 // Connect the device through the socket. This will block
 // until it succeeds or throws an exception
 mmSocket.connect();
 } catch (IOException connectException) {
 // Unable to connect; close the socket and get out
 try {
 mmSocket.close();
 } catch (IOException closeException) { }
 return;
 }

 // Do work to manage the connection (in a separate thread)
 manageConnectedSocket(mmSocket);
 }

 /** Will cancel an in-progress connection, and close the socket */
 public void cancel() {
 try {
 mmSocket.close();
 } catch (IOException e) { }
 }
}
```

请注意，在建立连接之前会调用 `cancelDiscovery()`。在进行连接之前应始终执行此调用，而且调用时无需实际检查其是否正在运行（但如果您确实想要执行检查，请调用 `isDiscovering()`）。

`manageConnectedSocket()` 是应用中的虚构方法，它将启动用于传输数据的线程，在有关 [管理连接](#)的部分将会讨论这一主题。

在完成 `BluetoothSocket` 后，应始终调用 `close()` 以执行清理操作。这样做会立即关闭已连接的套接字并清理所有内部资源。

## 管理连接

在成功连接两台（或更多台）设备后，每台设备都会有一个已连接的 `BluetoothSocket`。这一点非常有趣，因为这表示您可以在设备之间共享数据。利用 `BluetoothSocket`，传输任意数据的一般过程非常简单：

1. 获取 `InputStream` 和 `OutputStream`，二者分别通过套接字以及 `getInputStream()` 和 `getOutputStream()` 来处理数据传输。
2. 使用 `read(byte[])` 和 `write(byte[])` 读取数据并写入到流式传输。

就这么简单。

当然，还需要考虑实现细节。首要的是，应该为所有流式传输读取和写入操作使用专门的线程。这一点很重要，因为 `read(byte[])` 和 `write(byte[])` 方法都是阻塞调用。`read(byte[])` 将会阻塞，直至从流式传输中读取内容。`write(byte[])` 通常不会阻塞，但如果远程设备没有足够快地调用 `read(byte[])`，并且中间缓冲区已满，则其可能会保持阻塞状态以实现流量控制。因此，线程中的主循环应专门用于读取 `InputStream`。可使用线程中单独的公共方法来发起对 `OutputStream` 的写入操作。

## 示例

以下是可能的显示内容示例：

```
private class ConnectedThread extends Thread {
 private final BluetoothSocket mmSocket;
 private final InputStream mmInStream;
 private final OutputStream mmOutStream;

 public ConnectedThread(BluetoothSocket socket) {
 mmSocket = socket;
 mmInStream = null;
 mmOutStream = null;

 // Get the input and output streams, using temp objects because
 // member streams are final
 try {
 mmInStream = socket.getInputStream();
 mmOutStream = socket.getOutputStream();
 } catch (IOException e) { }

 mmInStream = tmpIn;
 mmOutStream = tmpOut;
 }

 public void run() {
 byte[] buffer = new byte[1024]; // buffer store for the stream
 int bytes; // bytes returned from read()

 // Keep listening to the InputStream until an exception occurs
 while (true) {
 try {
 // Read from the InputStream
 bytes = mmInStream.read(buffer);
 // Send the obtained bytes to the UI activity
 mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
 .sendToTarget();
 } catch (IOException e) {
 break;
 }
 }
 }

 /* Call this from the main activity to send data to the remote device */
 public void write(byte[] bytes) {
 try {
 mmOutStream.write(bytes);
 } catch (IOException e) { }
 }

 /* Call this from the main activity to shutdown the connection */
 public void cancel() {
 try {
 mmSocket.close();
 } catch (IOException e) { }
 }
}
```

构造函数获取必要的流式传输，并且一旦执行，线程将会等待通过 `InputStream` 传入的数据。当 `read(byte[])` 返回流式传输中的字节时，将

使用来自父类的成员处理程序将数据发送到主 Activity。然后该方法返回并等待流式传输提供更多字节。

发送传出数据不外乎从主 Activity 调用线程的 `write()` 方法，并传入要发送的字节。然后，此方法直接调用 `write(byte[])`，将数据发送到远程设备。

线程的 `cancel()` 方法很重要，它能通过关闭 `BluetoothSocket` 随时终止连接。当您结束蓝牙连接的使用时，应始终调用此方法。

有关使用 Bluetooth API 的演示，请参阅[蓝牙聊天示例应用](#)。

## 使用配置文件

---

从 Android 3.0 开始，Bluetooth API 便支持使用蓝牙配置文件。蓝牙配置文件是适用于设备间蓝牙通信的无线接口规范。免提配置文件便是一个示例。对于连接到无线耳机的手机，两台设备都必须支持免提配置文件。

您可以实现接口 `BluetoothProfile`，通过写入自己的类来支持特定的蓝牙配置文件。Android Bluetooth API 提供了以下蓝牙配置文件的实现：

- **耳机。**耳机配置文件提供了蓝牙耳机支持，以便与手机配合使用。Android 提供了 `BluetoothHeadset` 类，它是用于通过进程间通信 (IPC) 来控制蓝牙耳机服务的代理。这包括蓝牙耳机和免提（1.5 版）配置文件。`BluetoothHeadset` 类包含 AT 命令支持。有关此主题的详细讨论，请参阅[供应商特定的 AT 命令](#)
- **A2DP。**高级音频分发配置文件 (A2DP) 定义了高质量音频如何通过蓝牙连接和流式传输，从一个设备传输到另一个设备。Android 提供了 `BluetoothA2dp` 类，它是用于通过 IPC 来控制蓝牙 A2DP 服务的代理。
- **健康设备。**Android 4.0 (API 级别 14) 引入了对蓝牙健康设备配置文件 (HDP) 的支持。这允许您创建应用，使用蓝牙与支持蓝牙功能的健康设备进行通信，例如心率监测仪、血糖仪、温度计、台秤等等。有关支持的设备列表及其相应的设备数据专业化代码，请参阅 [www.bluetooth.org](http://www.bluetooth.org) 上的[蓝牙分配编号](#)。请注意，这些值在 ISO/IEEE 11073-20601 [7] 规范的“命名法规附录”中也被称为 MDC\_DEV\_SPEC\_PROFILE\_\*。有关 HDP 的详细讨论，请参阅[健康设备配置文件](#)。

以下是使用配置文件的基本步骤：

1. 获取默认适配器（请参阅[设置蓝牙](#)）。
2. 使用 `getProfileProxy()` 建立到配置文件所关联的配置文件代理对象的连接。在以下示例中，配置文件代理对象是一个 `BluetoothHeadset` 的实例。
3. 设置 `BluetoothProfile.ServiceListener`。此侦听程序会在 `BluetoothProfile` IPC 客户端连接到服务或断开服务连接时向其发送通知。
4. 在 `onServiceConnected()` 中，获取配置文件代理对象的句柄。
5. 获得配置文件代理对象后，可以立即将其用于监视连接状态和执行其他与该配置文件相关操作。

例如，以下代码片段显示了如何连接到 `BluetoothHeadset` 代理对象，以便能够控制耳机配置文件：

```

BluetoothHeadset mBluetoothHeadset;

// Get the default adapter
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Establish connection to the proxy.
mBluetoothAdapter.getProfileProxy(context, mProfileListener, BluetoothProfile.HEADSET);

private BluetoothProfile.ServiceListener mProfileListener = new BluetoothProfile.ServiceListener() {
 public void onServiceConnected(int profile, BluetoothProfile proxy) {
 if (profile == BluetoothProfile.HEADSET) {
 mBluetoothHeadset = (BluetoothHeadset) proxy;
 }
 }
 public void onServiceDisconnected(int profile) {
 if (profile == BluetoothProfile.HEADSET) {
 mBluetoothHeadset = null;
 }
 }
};

// ... call functions on mBluetoothHeadset

// Close proxy connection after use.
mBluetoothAdapter.closeProfileProxy(mBluetoothHeadset);

```

## 供应商特定的 AT 命令

从 Android 3.0 开始，应用可以注册接收耳机所发送的预定义的供应商特定 AT 命令的系统广播（例如 Plantronics +XEVENT 命令）。例如，应用可以接收指示所连接设备的电池电量的广播，并根据需要通知用户或采取其他操作。为 `ACTION_VENDOR_SPECIFIC_HEADSET_EVENT` intent 创建广播接收器，以处理耳机的供应商特定 AT 命令。

## 健康设备配置文件

Android 4.0 (API 级别 14) 引入了对蓝牙健康设备配置文件 (HDP) 的支持。这允许您创建应用，使用蓝牙与支持蓝牙功能的健康设备进行通信，例如心率监测仪、血糖仪、温度计、台秤等等。Bluetooth Health API 包括类 `BluetoothHealth`、`BluetoothHealthCallback` 和 `BluetoothHealthAppConfiguration`，在[基础知识](#)部分介绍了这些类。

在使用 Bluetooth Health API 时，了解以下关键 HDP 概念很有帮助：

概念	说明
<b>源设备</b>	在 HDP 中定义的角色。源设备是将医疗数据传输到 Android 手机或平板电脑等智能设备的健康设备（体重秤、血糖仪、温度计等）。
<b>汇集设备</b>	在 HDP 中定义的角色。在 HDP 中，汇集设备是接收医疗数据的智能设备。在 Android HDP 应用中，汇集设备表示为 <code>BluetoothHealthAppConfiguration</code> 对象。
<b>注册</b>	指的是注册特定健康设备的汇集设备。
<b>连接</b>	指的是开放健康设备与 Android 手机或平板电脑等智能设备之间的通道。

## 创建 HDP 应用

以下是创建 Android HDP 应用所涉及的基本步骤：

1. 获取 `BluetoothHealth` 代理对象的引用。

与常规耳机和 A2DP 配置文件设备相似，您必须使用 `BluetoothProfile.ServiceListener` 和 `HEALTH` 配置文件类型来调用 `getProfileProxy()`，以便与配置文件代理对象建立连接。

2. 创建 `BluetoothHealthCallback` 并注册充当健康汇集设备的应用配置 (`BluetoothHealthAppConfiguration`)。

3. 建立到健康设备的连接。一些设备将会发起该连接。对于这类设备，无需执行该步骤。

4. 成功连接到健康设备后，使用文件描述符对健康设备执行读/写操作。

接收的数据需要使用实现了 IEEE 11073-xxxxx 规范的健康管理器进行解释。

5. 完成后，关闭健康通道并取消注册该应用。该通道在长期闲置时也会关闭。

有关描述上述步骤的完整代码示例，请参阅[蓝牙 HDP（健康设备配置文件）](#)。



# Bluetooth Low Energy

In this document

- › [Key Terms and Concepts](#)
  - › [Roles and Responsibilities](#)
  - › [BLE Permissions](#)
    - › [Requesting User Permissions](#)
  - › [Setting Up BLE](#)
  - › [Finding BLE Devices](#)
  - › [Connecting to a GATT Server](#)
  - › [Reading BLE Attributes](#)
  - › [Receiving GATT Notifications](#)
  - › [Closing the Client App](#)

Key classes

- › [BluetoothGatt](#)
- › [BluetoothGattCallback](#)
- › [BluetoothGattCharacteristic](#)
- › [BluetoothGattService](#)

Related samples

- › [Android BluetoothLeGatt Sample](#)
- › [Android BluetoothAdvertisements Sample](#)

See Also

- › [Best Practices for Bluetooth Development \(video\)](#)



VIDEO

[DevBytes: Bluetooth Low Energy API](#)

Android 4.3 (API level 18) introduces built-in platform support for Bluetooth Low Energy (BLE) in the *central role* and provides APIs that apps can use to discover devices, query for services, and transmit information.

Common use cases include the following:

- Transferring small amounts of data between nearby devices.
- Interacting with proximity sensors like [Google Beacons](#) to give users a customized experience based on their current location.

In contrast to [Classic Bluetooth](#), Bluetooth Low Energy (BLE) is designed to provide significantly lower power consumption. This allows Android apps to communicate with BLE devices that have stricter power requirements, such as proximity sensors, heart rate monitors, and fitness devices.

## Key Terms and Concepts

Here is a summary of key BLE terms and concepts:

- **Generic Attribute Profile (GATT)**—The GATT profile is a general specification for sending and receiving short pieces of data known as

"attributes" over a BLE link. All current Low Energy application profiles are based on GATT.

- The Bluetooth SIG defines many [profiles](#) for Low Energy devices. A profile is a specification for how a device works in a particular application. Note that a device can implement more than one profile. For example, a device could contain a heart rate monitor and a battery level detector.
- **Attribute Protocol (ATT)**—GATT is built on top of the Attribute Protocol (ATT). This is also referred to as GATT/ATT. ATT is optimized to run on BLE devices. To this end, it uses as few bytes as possible. Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information. The *attributes* transported by ATT are formatted as *characteristics* and *services*.
- **Characteristic**—A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.
- **Descriptor**—Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value.
- **Service**—A service is a collection of characteristics. For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement." You can find a list of existing GATT-based profiles and services on [bluetooth.org](#).

## Roles and Responsibilities

Here are the roles and responsibilities that apply when an Android device interacts with a BLE device:

- Central vs. peripheral. This applies to the BLE connection itself. The device in the central role scans, looking for advertisement, and the device in the peripheral role makes the advertisement.
- GATT server vs. GATT client. This determines how two devices talk to each other once they've established the connection.

To understand the distinction, imagine that you have an Android phone and an activity tracker that is a BLE device. The phone supports the central role; the activity tracker supports the peripheral role (to establish a BLE connection you need one of each—two things that only support peripheral couldn't talk to each other, nor could two things that only support central).

Once the phone and the activity tracker have established a connection, they start transferring GATT metadata to one another. Depending on the kind of data they transfer, one or the other might act as the server. For example, if the activity tracker wants to report sensor data to the phone, it might make sense for the activity tracker to act as the server. If the activity tracker wants to receive updates from the phone, then it might make sense for the phone to act as the server.

In the example used in this document, the Android app (running on an Android device) is the GATT client. The app gets data from the GATT server, which is a BLE heart rate monitor that supports the [Heart Rate Profile](#). But you could alternatively design your Android app to play the GATT server role. See [BluetoothGattServer](#) for more information.

## BLE Permissions

In order to use Bluetooth features in your application, you must declare the Bluetooth permission [BLUETOOTH](#). You need this permission to perform any Bluetooth communication, such as requesting a connection, accepting a connection, and transferring data.

If you want your app to initiate device discovery or manipulate Bluetooth settings, you must also declare the [BLUETOOTH\\_ADMIN](#) permission.

**Note:** If you use the [BLUETOOTH\\_ADMIN](#) permission, then you must also have the [BLUETOOTH](#) permission.

Declare the Bluetooth permission(s) in your application manifest file. For example:

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

If you want to declare that your app is available to BLE-capable devices only, include the following in your app's manifest:

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

However, if you want to make your app available to devices that don't support BLE, you should still include this element in your app's

manifest, but set `required="false"`. Then at run-time you can determine BLE availability by using

`PackageManager.hasSystemFeature()`:

```
// Use this check to determine whether BLE is supported on the device. Then
// you can selectively disable BLE-related features.
if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
 Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).show();
 finish();
}
```

**Note:** LE Beacons are often associated with location. In order to use `BluetoothLeScanner` without a filter, you must request the user's permission by declaring either the `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION` permission in your app's manifest file. Without these permissions, scans won't return any results.

## Setting Up BLE

Before your application can communicate over BLE, you need to verify that BLE is supported on the device, and if so, ensure that it is enabled. Note that this check is only necessary if `<uses-feature.../>` is set to false.

If BLE is not supported, then you should gracefully disable any BLE features. If BLE is supported, but disabled, then you can request that the user enable Bluetooth without leaving your application. This setup is accomplished in two steps, using the `BluetoothAdapter`.

### 1. Get the `BluetoothAdapter`

The `BluetoothAdapter` is required for any and all Bluetooth activity. The `BluetoothAdapter` represents the device's own Bluetooth adapter (the Bluetooth radio). There's one Bluetooth adapter for the entire system, and your application can interact with it using this object. The snippet below shows how to get the adapter. Note that this approach uses `getSystemService()` to return an instance of `BluetoothManager`, which is then used to get the adapter. Android 4.3 (API Level 18) introduces `BluetoothManager`:

```
private BluetoothAdapter mBluetoothAdapter;
...
// Initializes Bluetooth adapter.
final BluetoothManager bluetoothManager =
 (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
```

### 2. Enable Bluetooth

Next, you need to ensure that Bluetooth is enabled. Call `isEnabled()` to check whether Bluetooth is currently enabled. If this method returns false, then Bluetooth is disabled. The following snippet checks whether Bluetooth is enabled. If it isn't, the snippet displays an error prompting the user to go to Settings to enable Bluetooth:

```
// Ensures Bluetooth is available on the device and it is enabled. If not,
// displays a dialog requesting user permission to enable Bluetooth.
if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
 Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
 startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

**Note:** The `REQUEST_ENABLE_BT` constant passed to `startActivityForResult(android.content.Intent, int)` is a locally-defined integer (which must be greater than 0) that the system passes back to you in your `onActivityResult(int, int, android.content.Intent)` implementation as the `requestCode` parameter.

## Finding BLE Devices

To find BLE devices, you use the `startLeScan()` method. This method takes a `BluetoothAdapter.LeScanCallback` as a parameter. You must implement this callback, because that is how scan results are returned. Because scanning is battery-intensive, you should observe the following guidelines:

- As soon as you find the desired device, stop scanning.

- Never scan on a loop, and set a time limit on your scan. A device that was previously available may have moved out of range, and continuing to scan drains the battery.

The following snippet shows how to start and stop a scan:

```
/*
 * Activity for scanning and displaying available BLE devices.
 */
public class DeviceScanActivity extends ListActivity {

 private BluetoothAdapter mBluetoothAdapter;
 private boolean mScanning;
 private Handler mHandler;

 // Stops scanning after 10 seconds.
 private static final long SCAN_PERIOD = 10000;
 ...

 private void scanLeDevice(final boolean enable) {
 if (enable) {
 // Stops scanning after a pre-defined scan period.
 mHandler.postDelayed(new Runnable() {
 @Override
 public void run() {
 mScanning = false;
 mBluetoothAdapter.stopLeScan(mLeScanCallback);
 }
 }, SCAN_PERIOD);

 mScanning = true;
 mBluetoothAdapter.startLeScan(mLeScanCallback);
 } else {
 mScanning = false;
 mBluetoothAdapter.stopLeScan(mLeScanCallback);
 }
 ...
}
...
}
```

If you want to scan for only specific types of peripherals, you can instead call `startLeScan(UUID[], BluetoothAdapter.LeScanCallback)`, providing an array of `UUID` objects that specify the GATT services your app supports.

Here is an implementation of the `BluetoothAdapter.LeScanCallback`, which is the interface used to deliver BLE scan results:

```
private LeDeviceListAdapter mLeDeviceListAdapter;
...
// Device scan callback.
private BluetoothAdapter.LeScanCallback mLeScanCallback =
 new BluetoothAdapter.LeScanCallback() {
 @Override
 public void onLeScan(final BluetoothDevice device, int rssi,
 byte[] scanRecord) {
 runOnUiThread(new Runnable() {
 @Override
 public void run() {
 mLeDeviceListAdapter.addDevice(device);
 mLeDeviceListAdapter.notifyDataSetChanged();
 }
 });
 }
 };
};


```

**Note:** You can only scan for Bluetooth LE devices or scan for Classic Bluetooth devices, as described in [Bluetooth](#). You cannot scan for both Bluetooth LE and classic devices at the same time.

## Connecting to a GATT Server

The first step in interacting with a BLE device is connecting to it— more specifically, connecting to the GATT server on the device. To connect to a GATT server on a BLE device, you use the `connectGatt()` method. This method takes three parameters: a `Context` object, `autoConnect` (boolean indicating whether to automatically connect to the BLE device as soon as it becomes available), and a reference to a `BluetoothGattCallback`:

```
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

This connects to the GATT server hosted by the BLE device, and returns a `BluetoothGatt` instance, which you can then use to conduct GATT client operations. The caller (the Android app) is the GATT client. The `BluetoothGattCallback` is used to deliver results to the client, such as connection status, as well as any further GATT client operations.

In this example, the BLE app provides an activity (`DeviceControlActivity`) to connect, display data, and display GATT services and characteristics supported by the device. Based on user input, this activity communicates with a `Service` called `BluetoothLeService`, which interacts with the BLE device via the Android BLE API:

```
// A service that interacts with the BLE device via the Android BLE API.
public class BluetoothLeService extends Service {
 private final static String TAG = BluetoothLeService.class.getSimpleName();

 private BluetoothManager mBluetoothManager;
 private BluetoothAdapter mBluetoothAdapter;
 private String mBluetoothDeviceAddress;
 private BluetoothGatt mBluetoothGatt;
 private int mConnectionState = STATE_DISCONNECTED;

 private static final int STATE_DISCONNECTED = 0;
 private static final int STATE_CONNECTING = 1;
 private static final int STATE_CONNECTED = 2;

 public final static String ACTION_GATT_CONNECTED =
 "com.example.bluetooth.le.ACTION_GATT_CONNECTED";
 public final static String ACTION_GATT_DISCONNECTED =
 "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED";
 public final static String ACTION_GATT_SERVICES_DISCOVERED =
 "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED";
 public final static String ACTION_DATA_AVAILABLE =
 "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";
 public final static String EXTRA_DATA =
 "com.example.bluetooth.le.EXTRA_DATA";

 public final static UUID UUID_HEART_RATE_MEASUREMENT =
 UUID.fromString(SampleGattAttributes.HEART_RATE_MEASUREMENT);

 // Various callback methods defined by the BLE API.
 private final BluetoothGattCallback mGattCallback =
 new BluetoothGattCallback() {
 @Override
 public void onConnectionStateChange(BluetoothGatt gatt, int status,
 int newState) {
 String intentAction;
 if (newState == BluetoothProfile.STATE_CONNECTED) {
 intentAction = ACTION_GATT_CONNECTED;
 mConnectionState = STATE_CONNECTED;
 broadcastUpdate(intentAction);
 Log.i(TAG, "Connected to GATT server.");
 Log.i(TAG, "Attempting to start service discovery:" +
 mBluetoothGatt.discoverServices());

 } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
 intentAction = ACTION_GATT_DISCONNECTED;
 mConnectionState = STATE_DISCONNECTED;
 Log.i(TAG, "Disconnected from GATT server.");
 broadcastUpdate(intentAction);
 }
 }

 @Override
 // New services discovered
 public void onServicesDiscovered(BluetoothGatt gatt, int status) {
 if (status == BluetoothGatt.GATT_SUCCESS) {
 Log.i(TAG, "onServicesDiscovered - Status: " + status);
 broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
 } else {
 Log.e(TAG, "onServicesDiscovered - Status: " + status);
 }
 }
 }
}
```

```

public void onServicesDiscovered(BluetoothGatt gatt, int status) {
 if (status == BluetoothGatt.GATT_SUCCESS) {
 broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
 } else {
 Log.w(TAG, "onServicesDiscovered received: " + status);
 }
}

@Override
// Result of a characteristic read operation
public void onCharacteristicRead(BluetoothGatt gatt,
 BluetoothGattCharacteristic characteristic,
 int status) {
 if (status == BluetoothGatt.GATT_SUCCESS) {
 broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
 }
}
...
};

}

```

When a particular callback is triggered, it calls the appropriate `broadcastUpdate()` helper method and passes it an action. Note that the data parsing in this section is performed in accordance with the Bluetooth Heart Rate Measurement [profile specifications](#):

```

private void broadcastUpdate(final String action) {
 final Intent intent = new Intent(action);
 sendBroadcast(intent);
}

private void broadcastUpdate(final String action,
 final BluetoothGattCharacteristic characteristic) {
 final Intent intent = new Intent(action);

 // This is special handling for the Heart Rate Measurement profile. Data
 // parsing is carried out as per profile specifications.
 if (UUID_HEART_RATE_MEASUREMENT.equals(characteristic.getUuid())) {
 int flag = characteristic.getProperties();
 int format = -1;
 if ((flag & 0x01) != 0) {
 format = BluetoothGattCharacteristic.FORMAT_UINT16;
 Log.d(TAG, "Heart rate format UINT16.");
 } else {
 format = BluetoothGattCharacteristic.FORMAT_UINT8;
 Log.d(TAG, "Heart rate format UINT8.");
 }
 final int heartRate = characteristic.getIntValue(format, 1);
 Log.d(TAG, String.format("Received heart rate: %d", heartRate));
 intent.putExtra(EXTRA_DATA, String.valueOf(heartRate));
 } else {
 // For all other profiles, writes the data formatted in HEX.
 final byte[] data = characteristic.getValue();
 if (data != null && data.length > 0) {
 final StringBuilder stringBuilder = new StringBuilder(data.length);
 for(byte byteChar : data)
 stringBuilder.append(String.format("%02X ", byteChar));
 intent.putExtra(EXTRA_DATA, new String(data) + "\n" +
 stringBuilder.toString());
 }
 }
 sendBroadcast(intent);
}

```

Back in `DeviceControlActivity`, these events are handled by a [BroadcastReceiver](#):

```
// Handles various events fired by the Service.
// ACTION_GATT_CONNECTED: connected to a GATT server.
// ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
// ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
// ACTION_DATA_AVAILABLE: received data from the device. This can be a
// result of read or notification operations.
private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
 @Override
 public void onReceive(Context context, Intent intent) {
 final String action = intent.getAction();
 if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
 mConnected = true;
 updateConnectionState(R.string.connected);
 invalidateOptionsMenu();
 } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
 mConnected = false;
 updateConnectionState(R.string.disconnected);
 invalidateOptionsMenu();
 clearUI();
 } else if (BluetoothLeService.
 ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
 // Show all the supported services and characteristics on the
 // user interface.
 displayGattServices(mBluetoothLeService.getSupportedGattServices());
 } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
 displayData(intent.getStringExtra(BluetoothLeService.EXTRA_DATA));
 }
 }
};
```

## Reading BLE Attributes

---

Once your Android app has connected to a GATT server and discovered services, it can read and write attributes, where supported. For example, this snippet iterates through the server's services and characteristics and displays them in the UI:

```

public class DeviceControlActivity extends Activity {
 ...
 // Demonstrates how to iterate through the supported GATT
 // Services/Characteristics.
 // In this sample, we populate the data structure that is bound to the
 // ExpandableListView on the UI.
 private void displayGattServices(List<BluetoothGattService> gattServices) {
 if (gattServices == null) return;
 String uuid = null;
 String unknownServiceString = getResources().
 getString(R.string.unknown_service);
 String unknownCharaString = getResources().
 getString(R.string.unknown_characteristic);
 ArrayList<HashMap<String, String>> gattServiceData =
 new ArrayList<HashMap<String, String>>();
 ArrayList<ArrayList<HashMap<String, String>>> gattCharacteristicData
 = new ArrayList<ArrayList<HashMap<String, String>>>();
 mGattCharacteristics =
 new ArrayList<ArrayList<BluetoothGattCharacteristic>>();

 // Loops through available GATT Services.
 for (BluetoothGattService gattService : gattServices) {
 HashMap<String, String> currentServiceData =
 new HashMap<String, String>();
 uuid = gattService.getUuid().toString();
 currentServiceData.put(
 LIST_NAME, SampleGattAttributes.
 lookup(uuid, unknownServiceString));
 currentServiceData.put(LIST_UUID, uuid);
 gattServiceData.add(currentServiceData);

 ArrayList<HashMap<String, String>> gattCharacteristicGroupData =
 new ArrayList<HashMap<String, String>>();
 List<BluetoothGattCharacteristic> gattCharacteristics =
 gattService.getCharacteristics();
 ArrayList<BluetoothGattCharacteristic> charas =
 new ArrayList<BluetoothGattCharacteristic>();
 // Loops through available Characteristics.
 for (BluetoothGattCharacteristic gattCharacteristic :
 gattCharacteristics) {
 charas.add(gattCharacteristic);
 HashMap<String, String> currentCharaData =
 new HashMap<String, String>();
 uuid = gattCharacteristic.getUuid().toString();
 currentCharaData.put(
 LIST_NAME, SampleGattAttributes.lookup(uuid,
 unknownCharaString));
 currentCharaData.put(LIST_UUID, uuid);
 gattCharacteristicGroupData.add(currentCharaData);
 }
 mGattCharacteristics.add(charas);
 gattCharacteristicData.add(gattCharacteristicGroupData);
 }
 ...
}

```

## Receiving GATT Notifications

It's common for BLE apps to ask to be notified when a particular characteristic changes on the device. This snippet shows how to set a notification for a characteristic, using the `setCharacteristicNotification()` method:

```
private BluetoothGatt mBluetoothGatt;
BluetoothGattCharacteristic characteristic;
boolean enabled;
...
mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);
...
BluetoothGattDescriptor descriptor = characteristic.getDescriptor(
 UUID.fromString(SampleGattAttributes.CLIENT_CHARACTERISTIC_CONFIG));
descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
mBluetoothGatt.writeDescriptor(descriptor);
```

Once notifications are enabled for a characteristic, an `onCharacteristicChanged()` callback is triggered if the characteristic changes on the remote device:

```
@Override
// Characteristic notification
public void onCharacteristicChanged(BluetoothGatt gatt,
 BluetoothGattCharacteristic characteristic) {
 broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
}
```

## Closing the Client App

Once your app has finished using a BLE device, it should call `close()` so the system can release resources appropriately:

```
public void close() {
 if (mBluetoothGatt == null) {
 return;
 }
 mBluetoothGatt.close();
 mBluetoothGatt = null;
}
```



# Near Field Communication

Near Field Communication (NFC) is a set of short-range wireless technologies, typically requiring a distance of 4cm or less to initiate a connection. NFC allows you to share small payloads of data between an NFC tag and an Android-powered device, or between two Android-powered devices.

Tags can range in complexity. Simple tags offer just read and write semantics, sometimes with one-time-programmable areas to make the card read-only. More complex tags offer math operations, and have cryptographic hardware to authenticate access to a sector. The most sophisticated tags contain operating environments, allowing complex interactions with code executing on the tag. The data stored in the tag can also be written in a variety of formats, but many of the Android framework APIs are based around a [NFC Forum](#) standard called NDEF (NFC Data Exchange Format).

Android-powered devices with NFC simultaneously support three main modes of operation:

1. **Reader/writer mode**, allowing the NFC device to read and/or write passive NFC tags and stickers.
2. **P2P mode**, allowing the NFC device to exchange data with other NFC peers; this operation mode is used by [Android Beam](#).
3. **Card emulation mode**, allowing the NFC device itself to act as an NFC card. The emulated NFC card can then be accessed by an external NFC reader, such as an NFC point-of-sale terminal.

## NFC Basics

This document describes how Android handles discovered NFC tags and how it notifies applications of data that is relevant to the application. It also goes over how to work with the NDEF data in your applications and gives an overview of the framework APIs that support the basic NFC feature set of Android.

## Advanced NFC

This document goes over the APIs that enable use of the various tag technologies that Android supports. When you are not working with NDEF data, or when you are working with NDEF data that Android cannot fully understand, you have to manually read or write to the tag in raw bytes using your own protocol stack. In these cases, Android provides support to detect certain tag technologies and to open communication with the tag using your own protocol stack.

## Host-based Card Emulation

This document describes how Android devices can perform as NFC cards without using a secure element, allowing any Android application to emulate a card and talk directly to the NFC reader.

# NFC Basics

In this document

- [The Tag Dispatch System](#)
  - [How NFC tags are mapped to MIME types and URIs](#)
  - [How NFC Tags are Dispatched to Applications](#)
  - [Requesting NFC Access in the Android Manifest](#)
  - [Filtering for Intents](#)
    - [ACTION\\_NDEF\\_DISCOVERED](#)
    - [ACTION\\_TECH\\_DISCOVERED](#)
    - [ACTION\\_TAG\\_DISCOVERED](#)
    - [Obtaining information from intents](#)
  - [Creating Common Types of NDEF Records](#)
    - [TNF\\_ABSOLUTE\\_URI](#)
    - [TNF\\_MIME\\_MEDIA](#)
    - [TNF\\_WELL\\_KNOWN with RTD\\_TEXT](#)
    - [TNF\\_WELL\\_KNOWN with RTD\\_URI](#)
    - [TNF\\_EXTERNAL\\_TYPE](#)
    - [Android Application Records](#)
  - [Beaming NDEF Messages to Other Devices](#)

This document describes the basic NFC tasks you perform in Android. It explains how to send and receive NFC data in the form of NDEF messages and describes the Android framework APIs that support these features. For more advanced topics, including a discussion of working with non-NDEF data, see [Advanced NFC](#).

There are two major uses cases when working with NDEF data and Android:

- Reading NDEF data from an NFC tag
- Beaming NDEF messages from one device to another with [Android Beam™](#)

Reading NDEF data from an NFC tag is handled with the [tag dispatch system](#), which analyzes discovered NFC tags, appropriately categorizes the data, and starts an application that is interested in the categorized data. An application that wants to handle the scanned NFC tag can [declare an intent filter](#) and request to handle the data.

The [Android Beam™](#) feature allows a device to push an NDEF message onto another device by physically tapping the devices together. This interaction provides an easier way to send data than other wireless technologies like Bluetooth, because with NFC, no manual device discovery or pairing is required. The connection is automatically started when two devices come into range. [Android Beam](#) is available through a set of NFC APIs, so any application can transmit information between devices. For example, the Contacts, Browser, and YouTube applications use [Android Beam](#) to share contacts, web pages, and videos with other devices.

## The Tag Dispatch System

Android-powered devices are usually looking for NFC tags when the screen is unlocked, unless NFC is disabled in the device's Settings menu. When an Android-powered device discovers an NFC tag, the desired behavior is to have the most appropriate activity handle the intent without asking the user what application to use. Because devices scan NFC tags at a very short range, it is likely that making users manually select an activity would force them to move the device away from the tag and break the connection. You should develop your activity to only handle the NFC tags that your activity cares about to prevent the Activity Chooser from appearing.

To help you with this goal, Android provides a special tag dispatch system that analyzes scanned NFC tags, parses them, and tries to locate applications that are interested in the scanned data. It does this by:

1. Parsing the NFC tag and figuring out the MIME type or a URI that identifies the data payload in the tag.
2. Encapsulating the MIME type or URI and the payload into an intent. These first two steps are described in [How NFC tags are mapped to MIME types and URIs](#).
3. Starts an activity based on the intent. This is described in [How NFC Tags are Dispatched to Applications](#).

## How NFC tags are mapped to MIME types and URIs

Before you begin writing your NFC applications, it is important to understand the different types of NFC tags, how the tag dispatch system parses NFC tags, and the special work that the tag dispatch system does when it detects an NDEF message. NFC tags come in a wide array of technologies and can also have data written to them in many different ways. Android has the most support for the NDEF standard, which is defined by the [NFC Forum](#).

NDEF data is encapsulated inside a message ([NdefMessage](#)) that contains one or more records ([NdefRecord](#)). Each NDEF record must be well-formed according to the specification of the type of record that you want to create. Android also supports other types of tags that do not contain NDEF data, which you can work with by using the classes in the [android.nfc.tech](#) package. To learn more about these technologies, see the [Advanced NFC](#) topic. Working with these other types of tags involves writing your own protocol stack to communicate with the tags, so we recommend using NDEF when possible for ease of development and maximum support for Android-powered devices.

**Note:** To download complete NDEF specifications, go to the [NFC Forum Specifications & Application Documents](#) site and see [Creating common types of NDEF records](#) for examples of how to construct NDEF records.

Now that you have some background in NFC tags, the following sections describe in more detail how Android handles NDEF formatted tags. When an Android-powered device scans an NFC tag containing NDEF formatted data, it parses the message and tries to figure out the data's MIME type or identifying URI. To do this, the system reads the first [NdefRecord](#) inside the [NdefMessage](#) to determine how to interpret the entire NDEF message (an NDEF message can have multiple NDEF records). In a well-formed NDEF message, the first [NdefRecord](#) contains the following fields:

### 3-bit TNF (Type Name Format)

Indicates how to interpret the variable length type field. Valid values are described in described in [Table 1](#).

### Variable length type

Describes the type of the record. If using [TNF\\_WELL\\_KNOWN](#), use this field to specify the Record Type Definition (RTD). Valid RTD values are described in [Table 2](#).

### Variable length ID

A unique identifier for the record. This field is not used often, but if you need to uniquely identify a tag, you can create an ID for it.

### Variable length payload

The actual data payload that you want to read or write. An NDEF message can contain multiple NDEF records, so don't assume the full payload is in the first NDEF record of the NDEF message.

The tag dispatch system uses the TNF and type fields to try to map a MIME type or URI to the NDEF message. If successful, it encapsulates that information inside of a [ACTION\\_NDEF\\_DISCOVERED](#) intent along with the actual payload. However, there are cases when the tag dispatch system cannot determine the type of data based on the first NDEF record. This happens when the NDEF data cannot be mapped to a MIME type or URI, or when the NFC tag does not contain NDEF data to begin with. In such cases, a [Tag](#) object that has information about the tag's technologies and the payload are encapsulated inside of a [ACTION\\_TECH\\_DISCOVERED](#) intent instead.

[Table 1](#) describes how the tag dispatch system maps TNF and type fields to MIME types or URIs. It also describes which TNFs cannot be mapped to a MIME type or URI. In these cases, the tag dispatch system falls back to [ACTION\\_TECH\\_DISCOVERED](#).

For example, if the tag dispatch system encounters a record of type [TNF\\_ABSOLUTE\\_URI](#), it maps the variable length type field of that record

into a URI. The tag dispatch system encapsulates that URI in the data field of an [ACTION\\_NDEF\\_DISCOVERED](#) intent along with other information about the tag, such as the payload. On the other hand, if it encounters a record of type [TNF\\_UNKNOWN](#), it creates an intent that encapsulates the tag's technologies instead.

**Table 1.** Supported TNFs and their mappings

Type Name Format (TNF)	Mapping
<a href="#">TNF_ABSOLUTE_URI</a>	URI based on the type field.
<a href="#">TNF_EMPTY</a>	Falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">TNF_EXTERNAL_TYPE</a>	URI based on the URN in the type field. The URN is encoded into the NDEF type field in a shortened form: <domain_name>:<service_name>. Android maps this to a URI in the form: vnd.android.nfc://ext/<domain_name>:<service_name>.
<a href="#">TNF_MIME_MEDIA</a>	MIME type based on the type field.
<a href="#">TNF_UNCHANGED</a>	Invalid in the first record, so falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">TNF_UNKNOWN</a>	Falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">TNF_WELL_KNOWN</a>	MIME type or URI depending on the Record Type Definition (RTD), which you set in the type field. See <a href="#">Table 2</a> for more information on available RTDs and their mappings.

**Table 2.** Supported RTDs for [TNF\\_WELL\\_KNOWN](#) and their mappings

Record Type Definition (RTD)	Mapping
<a href="#">RTD_ALTERNATIVE_CARRIER</a>	Falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">RTD_HANDOVER_CARRIER</a>	Falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">RTD_HANDOVER_REQUEST</a>	Falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">RTD_HANDOVER_SELECT</a>	Falls back to <a href="#">ACTION_TECH_DISCOVERED</a> .
<a href="#">RTD_SMART_POSTER</a>	URI based on parsing the payload.
<a href="#">RTD_TEXT</a>	MIME type of <code>text/plain</code> .
<a href="#">RTD_URI</a>	URI based on payload.

## How NFC Tags are Dispatched to Applications

When the tag dispatch system is done creating an intent that encapsulates the NFC tag and its identifying information, it sends the intent to an interested application that filters for the intent. If more than one application can handle the intent, the Activity Chooser is presented so the user can select the Activity. The tag dispatch system defines three intents, which are listed in order of highest to lowest priority:

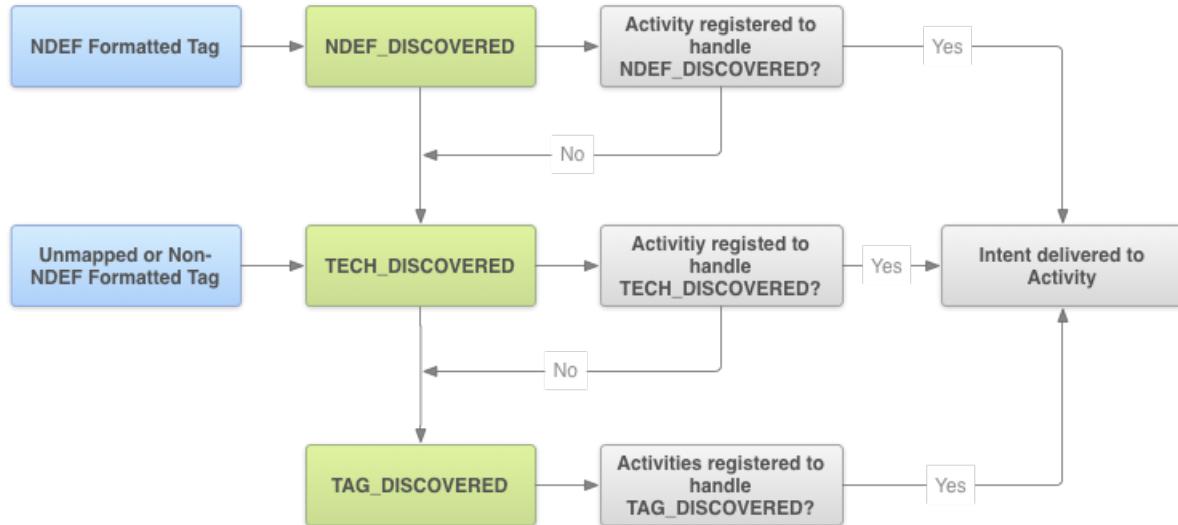
1. [ACTION\\_NDEF\\_DISCOVERED](#): This intent is used to start an Activity when a tag that contains an NDEF payload is scanned and is of a recognized type. This is the highest priority intent, and the tag dispatch system tries to start an Activity with this intent before any other intent, whenever possible.
2. [ACTION\\_TECH\\_DISCOVERED](#): If no activities register to handle the [ACTION\\_NDEF\\_DISCOVERED](#) intent, the tag dispatch system tries to start an application with this intent. This intent is also directly started (without starting [ACTION\\_NDEF\\_DISCOVERED](#) first) if the tag that is scanned contains NDEF data that cannot be mapped to a MIME type or URI, or if the tag does not contain NDEF data but is of a known tag technology.
3. [ACTION\\_TAG\\_DISCOVERED](#): This intent is started if no activities handle the [ACTION\\_NDEF\\_DISCOVERED](#) or [ACTION\\_TECH\\_DISCOVERED](#) intents.

The basic way the tag dispatch system works is as follows:

1. Try to start an Activity with the intent that was created by the tag dispatch system when parsing the NFC tag (either [ACTION\\_NDEF\\_DISCOVERED](#) or [ACTION\\_TECH\\_DISCOVERED](#)).
2. If no activities filter for that intent, try to start an Activity with the next lowest priority intent (either [ACTION\\_TECH\\_DISCOVERED](#) or

`ACTION_TAG_DISCOVERED`) until an application filters for the intent or until the tag dispatch system tries all possible intents.

3. If no applications filter for any of the intents, do nothing.



**Figure 1.** Tag Dispatch System

Whenever possible, work with NDEF messages and the `ACTION_NDEF_DISCOVERED` intent, because it is the most specific out of the three. This intent allows you to start your application at a more appropriate time than the other two intents, giving the user a better experience.

## Requesting NFC Access in the Android Manifest

Before you can access a device's NFC hardware and properly handle NFC intents, declare these items in your `AndroidManifest.xml` file:

- The NFC `<uses-permission>` element to access the NFC hardware:

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

- The minimum SDK version that your application can support. API level 9 only supports limited tag dispatch via `ACTION_TAG_DISCOVERED`, and only gives access to NDEF messages via the `EXTRA_NDEF_MESSAGES` extra. No other tag properties or I/O operations are accessible. API level 10 includes comprehensive reader/writer support as well as foreground NDEF pushing, and API level 14 provides an easier way to push NDEF messages to other devices with Android Beam and extra convenience methods to create NDEF records.

```
<uses-sdk android:minSdkVersion="10"/>
```

- The `uses-feature` element so that your application shows up in Google Play only for devices that have NFC hardware:

```
<uses-feature android:name="android.hardware.nfc" android:required="true" />
```

If your application uses NFC functionality, but that functionality is not crucial to your application, you can omit the `uses-feature` element and check for NFC availability at runtime by checking to see if `getOrDefaultAdapter()` is null.

## Filtering for NFC Intents

To start your application when an NFC tag that you want to handle is scanned, your application can filter for one, two, or all three of the NFC intents in the Android manifest. However, you usually want to filter for the `ACTION_NDEF_DISCOVERED` intent for the most control of when your application starts. The `ACTION_TECH_DISCOVERED` intent is a fallback for `ACTION_NDEF_DISCOVERED` when no applications filter for `ACTION_NDEF_DISCOVERED` or for when the payload is not NDEF. Filtering for `ACTION_TAG_DISCOVERED` is usually too general of a category to filter on. Many applications will filter for `ACTION_NDEF_DISCOVERED` or `ACTION_TECH_DISCOVERED` before `ACTION_TAG_DISCOVERED`, so your application has a low probability of starting. `ACTION_TAG_DISCOVERED` is only available as a last resort for applications to filter for in the cases where no other applications are installed to handle the `ACTION_NDEF_DISCOVERED` or `ACTION_TECH_DISCOVERED` intent.

Because NFC tag deployments vary and are many times not under your control, this is not always possible, which is why you can fallback to

the other two intents when necessary. When you have control over the types of tags and data written, it is recommended that you use NDEF to format your tags. The following sections describe how to filter for each type of intent.

## ACTION\_NDEF\_DISCOVERED

To filter for `ACTION_NDEF_DISCOVERED` intents, declare the intent filter along with the type of data that you want to filter for. The following example filters for `ACTION_NDEF_DISCOVERED` intents with a MIME type of `text/plain`:

```
<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
 <category android:name="android.intent.category.DEFAULT"/>
 <data android:mimeType="text/plain" />
</intent-filter>
```

The following example filters for a URI in the form of `http://developer.android.com/index.html`.

```
<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
 <category android:name="android.intent.category.DEFAULT"/>
 <data android:scheme="http"
 android:host="developer.android.com"
 android:pathPrefix="/index.html" />
</intent-filter>
```

## ACTION\_TECH\_DISCOVERED

If your activity filters for the `ACTION_TECH_DISCOVERED` intent, you must create an XML resource file that specifies the technologies that your activity supports within a `tech-list` set. Your activity is considered a match if a `tech-list` set is a subset of the technologies that are supported by the tag, which you can obtain by calling `getTechList()`.

For example, if the tag that is scanned supports MifareClassic, NdefFormattable, and NfcA, your `tech-list` set must specify all three, two, or one of the technologies (and nothing else) in order for your activity to be matched.

The following sample defines all of the technologies. You can remove the ones that you do not need. Save this file (you can name it anything you wish) in the `<project-root>/res/xml` folder.

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
 <tech-list>
 <tech>android.nfc.tech.IsoDep</tech>
 <tech>android.nfc.tech.NfcA</tech>
 <tech>android.nfc.tech.NfcB</tech>
 <tech>android.nfc.tech.NfcF</tech>
 <tech>android.nfc.tech.NfcV</tech>
 <tech>android.nfc.tech.Ndef</tech>
 <tech>android.nfc.tech.NdefFormattable</tech>
 <tech>android.nfc.tech.MifareClassic</tech>
 <tech>android.nfc.tech.MifareUltralight</tech>
 </tech-list>
</resources>
```

You can also specify multiple `tech-list` sets. Each of the `tech-list` sets is considered independently, and your activity is considered a match if any single `tech-list` set is a subset of the technologies that are returned by `getTechList()`. This provides AND and OR semantics for matching technologies. The following example matches tags that can support the NfcA and Ndef technologies or can support the NfcB and Ndef technologies:

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
 <tech-list>
 <tech>android.nfc.tech.NfcA</tech>
 <tech>android.nfc.tech.Ndef</tech>
 </tech-list>
</resources>

<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
 <tech-list>
 <tech>android.nfc.tech.NfcB</tech>
 <tech>android.nfc.tech.Ndef</tech>
 </tech-list>
</resources>
```

In your `AndroidManifest.xml` file, specify the resource file that you just created in the `<meta-data>` element inside the `<activity>` element like in the following example:

```
<activity>
 ...
 <intent-filter>
 <action android:name="android.nfc.action.TECH_DISCOVERED"/>
 </intent-filter>

 <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
 android:resource="@xml/nfc_tech_filter" />
 ...
</activity>
```

For more information about working with tag technologies and the `ACTION_TECH_DISCOVERED` intent, see [Working with Supported Tag Technologies](#) in the Advanced NFC document.

## ACTION\_TAG\_DISCOVERED

To filter for `ACTION_TAG_DISCOVERED` use the following intent filter:

```
<intent-filter>
 <action android:name="android.nfc.action.TAG_DISCOVERED"/>
</intent-filter>
```

## Obtaining information from intents

If an activity starts because of an NFC intent, you can obtain information about the scanned NFC tag from the intent. Intents can contain the following extras depending on the tag that was scanned:

- `EXTRA_TAG` (required): A `Tag` object representing the scanned tag.
- `EXTRA_NDEF_MESSAGES` (optional): An array of NDEF messages parsed from the tag. This extra is mandatory on `ACTION_NDEF_DISCOVERED` intents.
- `EXTRA_ID` (optional): The low-level ID of the tag.

To obtain these extras, check to see if your activity was launched with one of the NFC intents to ensure that a tag was scanned, and then obtain the extras out of the intent. The following example checks for the `ACTION_NDEF_DISCOVERED` intent and gets the NDEF messages from an intent extra.

```

@Override
protected void onNewIntent(Intent intent) {
 super.onNewIntent(intent);
 ...
 if (intent != null && NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())) {
 Parcelable[] rawMessages =
 intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
 if (rawMessages != null) {
 NdefMessage[] messages = new NdefMessage[rawMessages.length];
 for (int i = 0; i < rawMessages.length; i++) {
 messages[i] = (NdefMessage) rawMessages[i];
 }
 // Process the messages array.
 ...
 }
 }
}

```

Alternatively, you can obtain a `Tag` object from the intent, which will contain the payload and allow you to enumerate the tag's technologies:

```
Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
```

## Creating Common Types of NDEF Records

This section describes how to create common types of NDEF records to help you when writing to NFC tags or sending data with Android Beam. Starting with Android 4.0 (API level 14), the `createUri()` method is available to help you create URI records automatically. Starting in Android 4.1 (API level 16), `createExternal()` and `createMime()` are available to help you create MIME and external type NDEF records. Use these helper methods whenever possible to avoid mistakes when manually creating NDEF records.

This section also describes how to create the corresponding intent filter for the record. All of these NDEF record examples should be in the first NDEF record of the NDEF message that you are writing to a tag or beaming.

### TNF\_ABSOLUTE\_URI

**Note:** We recommend that you use the `RTD_URI` type instead of `TNF_ABSOLUTE_URI`, because it is more efficient.

You can create a `TNF_ABSOLUTE_URI` NDEF record in the following way :

```
NdefRecord uriRecord = new NdefRecord(
 NdefRecord.TNF_ABSOLUTE_URI ,
 "http://developer.android.com/index.html".getBytes(Charset.forName("US-ASCII")),
 new byte[0], new byte[0]);
```

The intent filter for the previous NDEF record would look like this:

```
<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED" />
 <category android:name="android.intent.category.DEFAULT" />
 <data android:scheme="http"
 android:host="developer.android.com"
 android:pathPrefix="/index.html" />
</intent-filter>
```

### TNF\_MIME\_MEDIA

You can create a `TNF_MIME_MEDIA` NDEF record in the following ways:

Using the `createMime()` method:

```
NdefRecord mimeRecord = NdefRecord.createMime("application/vnd.com.example.android.beam",
 "Beam me up, Android".getBytes(Charset.forName("US-ASCII")));
```

Creating the `NdefRecord` manually:

```
NdefRecord mimeRecord = new NdefRecord(
 NdefRecord.TNF_MIME_MEDIA,
 "application/vnd.com.example.android.beam".getBytes(Charset.forName("US-ASCII")),
 new byte[0], "Beam me up, Android!".getBytes(Charset.forName("US-ASCII")));
```

The intent filter for the previous NDEF record would look like this:

```
<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED" />
 <category android:name="android.intent.category.DEFAULT" />
 <data android:mimeType="application/vnd.com.example.android.beam" />
</intent-filter>
```

## TNF\_WELL\_KNOWN with RTD\_TEXT

You can create a `TNF_WELL_KNOWN` NDEF record in the following way:

```
public NdefRecord createTextRecord(String payload, Locale locale, boolean encodeInUtf8) {
 byte[] langBytes = locale.getLanguage().getBytes(Charset.forName("US-ASCII"));
 Charset utfEncoding = encodeInUtf8 ? Charset.forName("UTF-8") : Charset.forName("UTF-16");
 byte[] textBytes = payload.getBytes(utfEncoding);
 int utfBit = encodeInUtf8 ? 0 : (1 << 7);
 char status = (char) (utfBit + langBytes.length);
 byte[] data = new byte[1 + langBytes.length + textBytes.length];
 data[0] = (byte) status;
 System.arraycopy(langBytes, 0, data, 1, langBytes.length);
 System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.length);
 NdefRecord record = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
 NdefRecord.RTD_TEXT, new byte[0], data);
 return record;
}
```

The intent filter for the previous NDEF record would look like this:

```
<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED" />
 <category android:name="android.intent.category.DEFAULT" />
 <data android:mimeType="text/plain" />
</intent-filter>
```

## TNF\_WELL\_KNOWN with RTD\_URI

You can create a `TNF_WELL_KNOWN` NDEF record in the following ways:

Using the `createUri(String)` method:

```
NdefRecord rtdUriRecord1 = NdefRecord.createUri("http://example.com");
```

Using the `createUri(Uri)` method:

```
Uri uri = new Uri("http://example.com");
NdefRecord rtdUriRecord2 = NdefRecord.createUri(uri);
```

Creating the `NdefRecord` manually:

```

byte[] uriField = "example.com".getBytes(Charset.forName("US-ASCII"));
byte[] payload = new byte[uriField.length + 1]; //add 1 for the URI Prefix
byte payload[0] = 0x01; //prefixes http://www. to the URI
System.arraycopy(uriField, 0, payload, 1, uriField.length); //appends URI to payload
NdefRecord rtdUriRecord = new NdefRecord(
 NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_URI, new byte[0], payload);

```

The intent filter for the previous NDEF record would look like this:

```

<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED" />
 <category android:name="android.intent.category.DEFAULT" />
 <data android:scheme="http"
 android:host="example.com"
 android:pathPrefix="/" />
</intent-filter>

```

## TNF\_EXTERNAL\_TYPE

You can create a `TNF_EXTERNAL_TYPE` NDEF record in the following ways:

Using the `createExternal()` method:

```

byte[] payload; //assign to your data
String domain = "com.example"; //usually your app's package name
String type = "externalType";
NdefRecord extRecord = NdefRecord.createExternal(domain, type, payload);

```

Creating the `NdefRecord` manually:

```

byte[] payload;
...
NdefRecord extRecord = new NdefRecord(
 NdefRecord.TNF_EXTERNAL_TYPE, "com.example:externalType", new byte[0], payload);

```

The intent filter for the previous NDEF record would look like this:

```

<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED" />
 <category android:name="android.intent.category.DEFAULT" />
 <data android:scheme="vnd.android.nfc"
 android:host="ext"
 android:pathPrefix="/com.example:externalType"/>
</intent-filter>

```

Use `TNF_EXTERNAL_TYPE` for more generic NFC tag deployments to better support both Android-powered and non-Android-powered devices.

**Note:** URNs for `TNF_EXTERNAL_TYPE` have a canonical format of: `urn:nfc:ext:example.com:externalType`, however the NFC Forum RTD specification declares that the `urn:nfc:ext:` portion of the URN must be omitted from the NDEF record. So all you need to provide is the domain (`example.com` in the example) and type (`externalType` in the example) separated by a colon. When dispatching `TNF_EXTERNAL_TYPE`, Android converts the `urn:nfc:ext:example.com:externalType` URN to a `vnd.android.nfc://ext/example.com:externalType` URI, which is what the intent filter in the example declares.

## Android Application Records

Introduced in Android 4.0 (API level 14), an Android Application Record (AAR) provides a stronger certainty that your application is started when an NFC tag is scanned. An AAR has the package name of an application embedded inside an NDEF record. You can add an AAR to any NDEF record of your NDEF message, because Android searches the entire NDEF message for AARs. If it finds an AAR, it starts the application based on the package name inside the AAR. If the application is not present on the device, Google Play is launched to download

the application.

AARs are useful if you want to prevent other applications from filtering for the same intent and potentially handling specific tags that you have deployed. AARs are only supported at the application level, because of the package name constraint, and not at the Activity level as with intent filtering. If you want to handle an intent at the Activity level, [use intent filters](#).

If a tag contains an AAR, the tag dispatch system dispatches in the following manner:

1. Try to start an Activity using an intent filter as normal. If the Activity that matches the intent also matches the AAR, start the Activity.
2. If the Activity that filters for the intent does not match the AAR, if multiple Activities can handle the intent, or if no Activity handles the intent, start the application specified by the AAR.
3. If no application can start with the AAR, go to Google Play to download the application based on the AAR.

**Note:** You can override AARs and the intent dispatch system with the foreground dispatch system, which allows a foreground activity to have priority when an NFC tag is discovered. With this method, the activity must be in the foreground to override AARs and the intent dispatch system.

If you still want to filter for scanned tags that do not contain an AAR, you can declare intent filters as normal. This is useful if your application is interested in other tags that do not contain an AAR. For example, maybe you want to guarantee that your application handles proprietary tags that you deploy as well as general tags deployed by third parties. Keep in mind that AARs are specific to Android 4.0 devices or later, so when deploying tags, you most likely want to use a combination of AARs and MIME types/URIs to support the widest range of devices. In addition, when you deploy NFC tags, think about how you want to write your NFC tags to enable support for the most devices (Android-powered and other devices). You can do this by defining a relatively unique MIME type or URI to make it easier for applications to distinguish.

Android provides a simple API to create an AAR, [createApplicationRecord\(\)](#). All you need to do is embed the AAR anywhere in your [NdefMessage](#). You do not want to use the first record of your [NdefMessage](#), unless the AAR is the only record in the [NdefMessage](#). This is because the Android system checks the first record of an [NdefMessage](#) to determine the MIME type or URI of the tag, which is used to create an intent for applications to filter. The following code shows you how to create an AAR:

```
NdefMessage msg = new NdefMessage(
 new NdefRecord[] {
 ...,
 NdefRecord.createApplicationRecord("com.example.android.beam")
 }
)
```

## Beaming NDEF Messages to Other Devices

Android Beam allows simple peer-to-peer data exchange between two Android-powered devices. The application that wants to beam data to another device must be in the foreground and the device receiving the data must not be locked. When the beaming device comes in close enough contact with a receiving device, the beaming device displays the "Touch to Beam" UI. The user can then choose whether or not to beam the message to the receiving device.

**Note:** Foreground NDEF pushing was available at API level 10, which provides similar functionality to Android Beam. These APIs have since been deprecated, but are available to support older devices. See [enableForegroundNdefPush\(\)](#) for more information.

You can enable Android Beam for your application by calling one of the two methods:

- [setNdefPushMessage\(\)](#): Accepts an [NdefMessage](#) to set as the message to beam. Automatically beams the message when two devices are in close enough proximity.
- [setNdefPushMessageCallback\(\)](#): Accepts a callback that contains a [createNdefMessage\(\)](#) which is called when a device is in range to beam data to. The callback lets you create the NDEF message only when necessary.

An activity can only push one NDEF message at a time, so [setNdefPushMessageCallback\(\)](#) takes precedence over [setNdefPushMessage\(\)](#) if both are set. To use Android Beam, the following general guidelines must be met:

- The activity that is beaming the data must be in the foreground. Both devices must have their screens unlocked.
- You must encapsulate the data that you are beaming in an [NdefMessage](#) object.
- The NFC device that is receiving the beamed data must support the [com.android.npp](#) NDEF push protocol or NFC Forum's SNEP

(Simple NDEF Exchange Protocol). The `com.android.npp` protocol is required for devices on API level 9 (Android 2.3) to API level 13 (Android 3.2). `com.android.npp` and SNEP are both required on API level 14 (Android 4.0) and later.

**Note:** If your activity enables Android Beam and is in the foreground, the standard intent dispatch system is disabled. However, if your activity also enables [foreground dispatching](#), then it can still scan tags that match the intent filters set in the foreground dispatching.

To enable Android Beam:

1. Create an `NdefMessage` that contains the `NdefRecords` that you want to push onto the other device.
2. Call `setNdefPushMessage()` with a `NdefMessage` or call `setNdefPushMessageCallback` passing in a `NfcAdapter.CreateNdefMessageCallback` object in the `onCreate()` method of your activity. These methods require at least one activity that you want to enable with Android Beam, along with an optional list of other activities to activate.  
In general, you normally use `setNdefPushMessage()` if your Activity only needs to push the same NDEF message at all times, when two devices are in range to communicate. You use `setNdefPushMessageCallback` when your application cares about the current context of the application and wants to push an NDEF message depending on what the user is doing in your application.

The following sample shows how a simple activity calls `NfcAdapter.CreateNdefMessageCallback` in the `onCreate()` method of an activity (see [AndroidBeamDemo](#) for the complete sample). This example also has methods to help you create a MIME record:

```
package com.example.android.beam;

import android.app.Activity;
import android.content.Intent;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.nfc.NfcAdapter.CreateNdefMessageCallback;
import android.nfc.NfcEvent;
import android.os.Bundle;
import android.os.Parcelable;
import android.widget.TextView;
import android.widget.Toast;
import java.nio.charset.Charset;

public class Beam extends Activity implements CreateNdefMessageCallback {
 NfcAdapter mNfcAdapter;
 TextView textView;

 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 TextView textView = (TextView) findViewById(R.id.textView);
 // Check for available NFC Adapter
 mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
 if (mNfcAdapter == null) {
 Toast.makeText(this, "NFC is not available", Toast.LENGTH_LONG).show();
 finish();
 return;
 }
 // Register callback
 mNfcAdapter.setNdefPushMessageCallback(this, this);
 }

 @Override
 public NdefMessage createNdefMessage(NfcEvent event) {
 String text = ("Beam me up, Android!\n\n" +
 "Beam Time: " + System.currentTimeMillis());
 NdefMessage msg = new NdefMessage(
 new NdefRecord[] { createMime(
 "application/vnd.com.example.android.beam", text.getBytes())
 });
 /**
 * The Android Application Record (AAR) is commented out. When a device
 * receives a push with an AAR in it, the application specified in the AAR
 * is guaranteed to run. The AAR overrides the tag dispatch system.
 * You can add it back in to guarantee that this
 }
}
```

```

 * activity starts when receiving a beamed message. For now, this code
 * uses the tag dispatch system.
 */
 //,NdefRecord.createApplicationRecord("com.example.android.beam")
);
return msg;
}

@Override
public void onResume() {
 super.onResume();
 // Check to see that the Activity started due to an Android Beam
 if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
 processIntent(getIntent());
 }
}

@Override
public void onNewIntent(Intent intent) {
 // onResume gets called after this to handle the intent
 setIntent(intent);
}

/**
 * Parses the NDEF Message from the intent and prints to the TextView
 */
void processIntent(Intent intent) {
 textView = (TextView) findViewById(R.id.textView);
 Parcelable[] rawMsgs = intent.getParcelableArrayExtra(
 NfcAdapter.EXTRA_NDEF_MESSAGES);
 // only one message sent during the beam
 NdefMessage msg = (NdefMessage) rawMsgs[0];
 // record 0 contains the MIME type, record 1 is the AAR, if present
 textView.setText(new String(msg.getRecords()[0].getPayload()));
}
}

```

Note that this code comments out an AAR, which you can remove. If you enable the AAR, the application specified in the AAR always receives the Android Beam message. If the application is not present, Google Play is started to download the application. Therefore, the following intent filter is not technically necessary for Android 4.0 devices or later if the AAR is used:

```

<intent-filter>
 <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
 <category android:name="android.intent.category.DEFAULT"/>
 <data android:mimeType="application/vnd.com.example.android.beam"/>
</intent-filter>

```

With this intent filter, the `com.example.android.beam` application now can be started when it scans an NFC tag or receives an Android Beam with an AAR of type `com.example.android.beam`, or when an NDEF formatted message contains a MIME record of type `application/vnd.com.example.android.beam`.

Even though AARs guarantee an application is started or downloaded, intent filters are recommended, because they let you start an Activity of your choice in your application instead of always starting the main Activity within the package specified by an AAR. AARs do not have Activity level granularity. Also, because some Android-powered devices do not support AARs, you should also embed identifying information in the first NDEF record of your NDEF messages and filter for that as well, just in case. See [Creating Common Types of NDEF records](#) for more information on how to create records.



# Advanced NFC

In this document

- [Working with Supported Tag Technologies](#)
- [Working with tag technologies and the ACTION\\_TECH\\_DISCOVERED intent](#)
- [Reading and writing to tags](#)
- [Using the Foreground Dispatch System](#)

This document describes advanced NFC topics, such as working with various tag technologies, writing to NFC tags, and foreground dispatching, which allows an application in the foreground to handle intents even when other applications filter for the same ones.

## Working with Supported Tag Technologies

When working with NFC tags and Android-powered devices, the main format you use to read and write data on tags is NDEF. When a device scans a tag with NDEF data, Android provides support in parsing the message and delivering it in an [NdefMessage](#) when possible. There are cases, however, when you scan a tag that does not contain NDEF data or when the NDEF data could not be mapped to a MIME type or URI. In these cases, you need to open communication directly with the tag and read and write to it with your own protocol (in raw bytes). Android provides generic support for these use cases with the [android.nfc.tech](#) package, which is described in [Table 1](#). You can use the [getTechList\(\)](#) method to determine the technologies supported by the tag and create the corresponding [TagTechnology](#) object with one of classes provided by [android.nfc.tech](#)

**Table 1.** Supported tag technologies

Class	Description
<a href="#">TagTechnology</a>	The interface that all tag technology classes must implement.
<a href="#">NfcA</a>	Provides access to NFC-A (ISO 14443-3A) properties and I/O operations.
<a href="#">NfcB</a>	Provides access to NFC-B (ISO 14443-3B) properties and I/O operations.
<a href="#">NfcF</a>	Provides access to NFC-F (JIS 6319-4) properties and I/O operations.
<a href="#">NfcV</a>	Provides access to NFC-V (ISO 15693) properties and I/O operations.
<a href="#">IsoDep</a>	Provides access to ISO-DEP (ISO 14443-4) properties and I/O operations.
<a href="#">Ndef</a>	Provides access to NDEF data and operations on NFC tags that have been formatted as NDEF.
<a href="#">NdefFormattable</a>	Provides a format operations for tags that may be NDEF formattable.

The following tag technologies are not required to be supported by Android-powered devices.

**Table 2.** Optional supported tag technologies

Class	Description
<a href="#">MifareClassic</a>	Provides access to MIFARE Classic properties and I/O operations, if this Android device supports MIFARE.
<a href="#">MifareUltralight</a>	Provides access to MIFARE Ultralight properties and I/O operations, if this Android device supports MIFARE.

## Working with tag technologies and the ACTION\_TECH\_DISCOVERED intent

When a device scans a tag that has NDEF data on it, but could not be mapped to a MIME or URI, the tag dispatch system tries to start an activity with the [ACTION\\_TECH\\_DISCOVERED](#) intent. The [ACTION\\_TECH\\_DISCOVERED](#) is also used when a tag with non-NDEF data is scanned. Having this fallback allows you to work with the data on the tag directly if the tag dispatch system could not parse it for you. The basic steps when working with tag technologies are as follows:

1. Filter for an [ACTION\\_TECH\\_DISCOVERED](#) intent specifying the tag technologies that you want to handle. See [Filtering for NFC intents](#) for more information. In general, the tag dispatch system tries to start a [ACTION\\_TECH\\_DISCOVERED](#) intent when an NDEF message cannot be mapped to a MIME type or URI, or if the tag scanned did not contain NDEF data. For more information on how this is determined, see [The Tag Dispatch System](#).
2. When your application receives the intent, obtain the [Tag](#) object from the intent:

```
Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
```

3. Obtain an instance of a [TagTechnology](#), by calling one of the [get](#) factory methods of the classes in the [android.nfc.tech](#) package. You can enumerate the supported technologies of the tag by calling [getTechList\(\)](#) before calling a [get](#) factory method. For example, to obtain an instance of [MifareUltralight](#) from a [Tag](#), do the following:

```
MifareUltralight.get(intent.getParcelableExtra(NfcAdapter.EXTRA_TAG));
```

## Reading and writing to tags

Reading and writing to an NFC tag involves obtaining the tag from the intent and opening communication with the tag. You must define your own protocol stack to read and write data to the tag. Keep in mind, however, that you can still read and write NDEF data when working directly with a tag. It is up to you how you want to structure things. The following example shows how to work with a MIFARE Ultralight tag.

```

package com.example.android.nfc;

import android.nfc.Tag;
import android.nfc.tech.MifareUltralight;
import android.util.Log;
import java.io.IOException;
import java.nio.charset.Charset;

public class MifareUltralightTagTester {

 private static final String TAG = MifareUltralightTagTester.class.getSimpleName();

 public void writeTag(Tag tag, String tagText) {
 MifareUltralight ultralight = MifareUltralight.get(tag);
 try {
 ultralight.connect();
 ultralight.writePage(4, "abcd".getBytes(Charset.forName("US-ASCII")));
 ultralight.writePage(5, "efgh".getBytes(Charset.forName("US-ASCII")));
 ultralight.writePage(6, "ijkl".getBytes(Charset.forName("US-ASCII")));
 ultralight.writePage(7, "mnop".getBytes(Charset.forName("US-ASCII")));
 } catch (IOException e) {
 Log.e(TAG, "IOException while closing MifareUltralight...", e);
 } finally {
 try {
 ultralight.close();
 } catch (IOException e) {
 Log.e(TAG, "IOException while closing MifareUltralight...", e);
 }
 }
 }

 public String readTag(Tag tag) {
 MifareUltralight mifare = MifareUltralight.get(tag);
 try {
 mifare.connect();
 byte[] payload = mifare.readPages(4);
 return new String(payload, Charset.forName("US-ASCII"));
 } catch (IOException e) {
 Log.e(TAG, "IOException while writing MifareUltralight message...", e);
 } finally {
 if (mifare != null) {
 try {
 mifare.close();
 } catch (IOException e) {
 Log.e(TAG, "Error closing tag...", e);
 }
 }
 }
 return null;
 }
}

```

## Using the Foreground Dispatch System

The foreground dispatch system allows an activity to intercept an intent and claim priority over other activities that handle the same intent. Using this system involves constructing a few data structures for the Android system to be able to send the appropriate intents to your application. To enable the foreground dispatch system:

1. Add the following code in the `onCreate()` method of your activity:
  1. Create a `PendingIntent` object so the Android system can populate it with the details of the tag when it is scanned.

```

PendingIntent pendingIntent = PendingIntent.getActivity(
 this, 0, new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);

```

2. Declare intent filters to handle the intents that you want to intercept. The foreground dispatch system checks the specified intent filters

with the intent that is received when the device scans a tag. If it matches, then your application handles the intent. If it does not match, the foreground dispatch system falls back to the intent dispatch system. Specifying a `null` array of intent filters and technology filters, specifies that you want to filter for all tags that fallback to the `TAG_DISCOVERED` intent. The code snippet below handles all MIME types for `NDEF_DISCOVERED`. You should only handle the ones that you need.

```
IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
try {
 ndef.addDataType("*/*"); /* Handles all MIME based dispatches.
 You should specify only the ones that you need. */
}
catch (MalformedMimeTypeException e) {
 throw new RuntimeException("fail", e);
}
intentFiltersArray = new IntentFilter[] {ndef, };
```

3. Set up an array of tag technologies that your application wants to handle. Call the `Object.class.getName()` method to obtain the class of the technology that you want to support.

```
techListsArray = new String[][] { new String[] { NfcF.class.getName() } };
```

2. Override the following activity lifecycle callbacks and add logic to enable and disable the foreground dispatch when the activity loses (`onPause()`) and regains (`onResume()`) focus. `enableForegroundDispatch()` must be called from the main thread and only when the activity is in the foreground (calling in `onResume()` guarantees this). You also need to implement the `onNewIntent` callback to process the data from the scanned NFC tag.

```
public void onPause() {
 super.onPause();
 mAdapter.disableForegroundDispatch(this);
}

public void onResume() {
 super.onResume();
 mAdapter.enableForegroundDispatch(this, pendingIntent, intentFiltersArray, techListsArray);
}

public void onNewIntent(Intent intent) {
 Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
 //do something with tagFromIntent
}
```

See the [ForegroundDispatch](#) sample from API Demos for the complete sample.

# Host-based Card Emulation

In this document

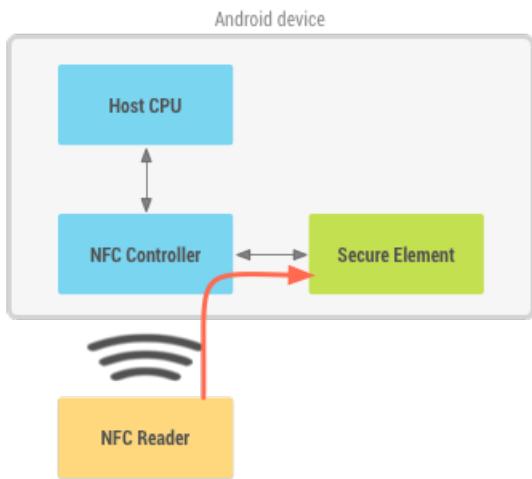
- [Card Emulation with a Secure Element](#)
- [Host-based Card Emulation](#)
- [Supported NFC Cards and Protocols](#)
- [HCE Services](#)
- [Implementing an HCE Service](#)
- [AID Conflict Resolution](#)
- [Payment Applications](#)
- [Screen Off and Lock-screen Behavior](#)
- [Coexistence with Secure Element Cards](#)
- [HCE and Security](#)
- [Protocol parameters and details](#)

Many Android-powered devices that offer NFC functionality already support NFC card emulation. In most cases, the card is emulated by a separate chip in the device, called a *secure element*. Many SIM cards provided by wireless carriers also contain a secure element.

Android 4.4 introduces an additional method of card emulation that does not involve a secure element, called *host-based card emulation*. This allows any Android application to emulate a card and talk directly to the NFC reader. This document describes how host-based card emulation (HCE) works on Android and how you can develop an app that emulates an NFC card using this technique.

## Card Emulation with a Secure Element

When NFC card emulation is provided using a secure element, the card to be emulated is provisioned into the secure element on the device through an Android application. Then, when the user holds the device over an NFC terminal, the NFC controller in the device routes all data from the reader directly to the secure element. Figure 1 illustrates this concept.



**Figure 1.** NFC card emulation with a secure element.

The secure element itself performs the communication with the NFC terminal, and no Android application is involved in the transaction at all. After the transaction is complete, an Android application can query the secure element directly for the transaction status and notify the user.

## Host-based Card Emulation

When an NFC card is emulated using host-based card emulation, the data is routed to the host CPU on which Android applications are running directly, instead of routing the NFC protocol frames to a secure element. Figure 2 illustrates how host-based card emulation works.

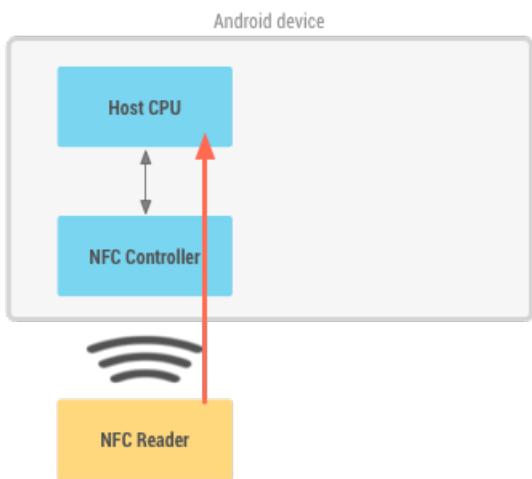


Figure 2. NFC card emulation without a secure element.

## Supported NFC Cards and Protocols

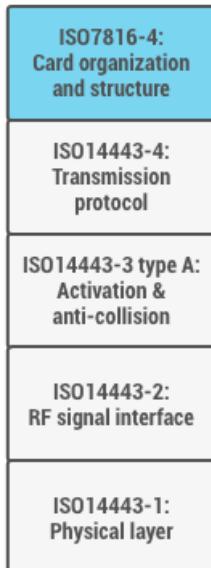


Figure 3. Android's HCE protocol stack.

The NFC standards offer support for many different protocols, and there are different types of cards that can be emulated.

Android 4.4 supports several protocols that are common in the market today. Many existing contactless cards are already based on these protocols, such as contactless payment cards. These protocols are also supported by many NFC readers in the market today, including Android NFC devices functioning as readers themselves (see the [IsoDep](#) class). This allows you to build and deploy an end-to-end NFC solution around HCE using only Android-powered devices.

Specifically, Android 4.4 supports emulating cards that are based on the NFC-Forum ISO-DEP specification (based on ISO/IEC 14443-4) and process Application Protocol Data Units (APDUs) as defined in the ISO/IEC 7816-4 specification. Android mandates emulating ISO-DEP only on top of the Nfc-A (ISO/IEC 14443-3 Type A) technology. Support for Nfc-B (ISO/IEC 14443-4 Type B) technology is optional. The layering of all these specifications is shown in the figure 3.

## HCE Services

The HCE architecture in Android is based around Android [Service](#) components (known as "HCE services"). One of the key advantages of a service is that it can run in the background without any user interface. This is a natural fit for many HCE applications like loyalty or transit

cards, with which the user shouldn't need to launch the app to use it. Instead, tapping the device against the NFC reader starts the correct service (if not already running) and executes the transaction in the background. Of course, you are free to launch additional UI (such as user notifications) from your service if that makes sense.

## Service selection

When the user taps a device to an NFC reader, the Android system needs to know which HCE service the NFC reader actually wants to talk to. This is where the ISO/IEC 7816-4 specification comes in: it defines a way to select applications, centered around an Application ID (AID). An AID consists of up to 16 bytes. If you are emulating cards for an existing NFC reader infrastructure, the AIDs that those readers are looking for are typically well-known and publicly registered (for example, the AIDs of payment networks such as Visa and MasterCard).

If you want to deploy new reader infrastructure for your own application, you will need to register your own AID(s). The registration procedure for AIDs is defined in the ISO/IEC 7816-5 specification. Google recommends registering an AID as per 7816-5 if you are deploying a HCE application for Android, as it will avoid collisions with other applications.

## AID groups

In some cases, an HCE service may need to register multiple AIDs to implement a certain application, and it needs to be sure that it is the default handler for all of these AIDs (as opposed to some AIDs in the group going to another service).

An AID group is a list of AIDs that should be considered as belonging together by the OS. For all AIDs in an AID group, Android guarantees one of the following:

- All AIDs in the group are routed to this HCE service
- No AIDs in the group are routed to this HCE service (for example, because the user preferred another service which requested one or more AIDs in your group as well)

In other words, there is no in-between state, where some AIDs in the group can be routed to one HCE service, and some to another.

## AID groups and categories

Each AID group can be associated with a category. This allows Android to group HCE services together by category, and that in turn allows the user to set defaults at the category level instead of the AID level. In general, avoid mentioning AIDs in any user-facing parts of your application: they do not mean anything to the average user.

Android 4.4 supports two categories: [CATEGORY\\_PAYMENT](#) (covering industry-standard payment apps) and [CATEGORY\\_OTHER](#) (for all other HCE apps).

**Note:** Only one AID group in the [CATEGORY\\_PAYMENT](#) category may be enabled in the system at any given time. Typically, this will be an app that understands major credit card payment protocols and which can work at any merchant.

For *closed-loop* payment apps that only work at one merchant (such as stored-value cards), you should use [CATEGORY\\_OTHER](#). AID groups in this category can be always active, and can be given priority by NFC readers during AID selection if necessary.

## Implementing an HCE Service

To emulate an NFC card using host-based card emulation, you need to create a [Service](#) component that handles the NFC transactions.

### Checking for HCE support

Your application can check whether a device supports HCE by checking for the [FEATURE\\_NFC\\_HOST\\_CARD\\_EMULATION](#) feature. You should use the `<uses-feature>` tag in the manifest of your application to declare that your app uses the HCE feature, and whether it is required for the app to function or not.

### Service implementation

Android 4.4 comes with a convenience [Service](#) class that can be used as a basis for implementing a HCE service: the [HostApduService](#) class.

The first step is therefore to extend [HostApduService](#).

```
public class MyHostApduService extends HostApduService {
 @Override
 public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
 ...
 }
 @Override
 public void onDeactivated(int reason) {
 ...
 }
}
```

[HostApduService](#) declares two abstract methods that need to be overridden and implemented.

[processCommandApdu\(\)](#) is called whenever a NFC reader sends an Application Protocol Data Unit (APDU) to your service. APDUs are defined in the ISO/IEC 7816-4 specification as well. APDUs are the application-level packets being exchanged between the NFC reader and your HCE service. That application-level protocol is half-duplex: the NFC reader will send you a command APDU, and it will wait for you to send a response APDU in return.

**Note:** The ISO/IEC 7816-4 specification also defines the concept of multiple logical channels, where you can have multiple parallel APDU exchanges on separate logical channels. Android's HCE implementation however only supports a single logical channel, so there's only a single-threaded exchange of APDUs.

As mentioned previously, Android uses the AID to determine which HCE service the reader wants to talk to. Typically, the first APDU an NFC reader sends to your device is a "SELECT AID" APDU; this APDU contains the AID that the reader wants to talk to. Android extracts that AID from the APDU, resolves it to an HCE service, then forwards that APDU to the resolved service.

You can send a response APDU by returning the bytes of the response APDU from [processCommandApdu\(\)](#). Note that this method will be called on the main thread of your application, which shouldn't be blocked. So if you can't compute and return a response APDU immediately, return null. You can then do the necessary work on another thread, and use the [sendResponseApdu\(\)](#) method defined in the [HostApduService](#) class to send the response when you are done.

Android will keep forwarding new APDUs from the reader to your service, until either:

1. The NFC reader sends another "SELECT AID" APDU, which the OS resolves to a different service;
2. The NFC link between the NFC reader and your device is broken.

In both of these cases, your class's [onDeactivated\(\)](#) implementation is called with an argument indicating which of the two happened.

If you are working with existing reader infrastructure, you need to implement the existing application-level protocol that the readers expect in your HCE service.

If you are deploying new reader infrastructure which you control as well, you can define your own protocol and APDU sequence. In general try to limit the amount of APDUs and the size of the data that needs to be exchanged: this makes sure that your users will only have to hold their device over the NFC reader for a short amount of time. A sane upper bound is about 1KB of data, which can usually be exchanged within 300ms.

## Service manifest declaration and AID registration

Your service must be declared in the manifest as usual, but some additional pieces must be added to the service declaration as well.

First, to tell the platform that it is a HCE service implementing a [HostApduService](#) interface, your service declaration must contain an intent filter for the [SERVICE\\_INTERFACE](#) action.

Additionally, to tell the platform which AIDs groups are requested by this service, a [SERVICE\\_META\\_DATA <meta-data>](#) tag must be included in the declaration of the service, pointing to an XML resource with additional information about the HCE service.

Finally, you must set the [android:exported](#) attribute to true, and require the ["android.permission.BIND\\_NFC\\_SERVICE"](#) permission in your service declaration. The former ensures that the service can be bound to by external applications. The latter then enforces that only external applications that hold the ["android.permission.BIND\\_NFC\\_SERVICE"](#) permission can bind to your service. Since ["android.permission.BIND\\_NFC\\_SERVICE"](#) is a system permission, this effectively enforces that only the Android OS can bind to your

service.

Here's an example of a [HostApduService](#) manifest declaration:

```
<service android:name=".MyHostApduService" android:exported="true"
 android:permission="android.permission.BIND_NFC_SERVICE">
 <intent-filter>
 <action android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE"/>
 </intent-filter>
 <meta-data android:name="android.nfc.cardemulation.host_apdu_service"
 android:resource="@xml/apduservice"/>
</service>
```

This meta-data tag points to an [apduservice.xml](#) file. An example of such a file with a single AID group declaration containing two proprietary AIDs is shown below:

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
 android:description="@string/servicedesc"
 android:requireDeviceUnlock="false">
 <aid-group android:description="@string/aiddescription"
 android:category="other">
 <aid-filter android:name="F0010203040506"/>
 <aid-filter android:name="F0394148148100"/>
 </aid-group>
</host-apdu-service>
```

The `<host-apdu-service>` tag is required to contain a `<android:description>` attribute that contains a user-friendly description of the service that may be shown in UI. The `requireDeviceUnlock` attribute can be used to specify that the device must be unlocked before this service can be invoked to handle APDUs.

The `<host-apdu-service>` must contain one or more `<aid-group>` tags. Each `<aid-group>` tag is required to:

- Contain an `android:description` attribute that contains a user-friendly description of the AID group, suitable for display in UI.
- Have its `android:category` attribute set to indicate the category the AID group belongs to, e.g. the string constants defined by `CATEGORY_PAYMENT` or `CATEGORY_OTHER`.
- Each `<aid-group>` must contain one or more `<aid-filter>` tags, each of which contains a single AID. The AID must be specified in hexadecimal format, and contain an even number of characters.

As a final note, your application also needs to hold the [NFC](#) permission to be able to register as a HCE service.

## AID Conflict Resolution

Multiple [HostApduService](#) components may be installed on a single device, and the same AID can be registered by more than one service. The Android platform resolves AID conflicts depending on which category an AID belongs to. Each category may have a different conflict resolution policy.

For example, for some categories (like payment) the user may be able to select a default service in the Android settings UI. For other categories, the policy may be to always ask the user which service is to be invoked in case of conflict. To query the conflict resolution policy for a certain category, see [getSelectionModeForCategory\(\)](#).

## Checking if your service is the default

Applications can check whether their HCE service is the default service for a certain category by using the [isDefaultServiceForCategory\(ComponentName, String\)](#) API.

If your service is not the default, you can request it to be made the default. See [ACTION\\_CHANGE\\_DEFAULT](#).

# Payment Applications

Android considers HCE services that have declared an AID group with the "payment" category as payment applications. The Android 4.4 release contains a top-level Settings menu entry called "tap & pay", which enumerates all such payment applications. In this settings menu, the user can select the default payment application that will be invoked when a payment terminal is tapped.

## Required assets for payment applications

To provide a more visually attractive user experience, HCE payment applications are required to provide an additional asset for their service: a so-called service banner.

This asset should be sized 260x96 dp, and can be specified in your meta-data XML file by adding the `android:apduServiceBanner` attribute to the `<host-apdu-service>` tag, which points to the drawable resource. An example is shown below:

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
 android:description="@string/servicedesc"
 android:requireDeviceUnlock="false"
 android:apduServiceBanner="@drawable/my_banner">
 <aid-group android:description="@string/aiddescription"
 android:category="payment">
 <aid-filter android:name="F0010203040506"/>
 <aid-filter android:name="F0394148148100"/>
 </aid-group>
</host-apdu-service>
```

## Screen Off and Lock-screen Behavior

Current Android implementations turn the NFC controller and the application processor off completely when the screen of the device is turned off. HCE services will therefore not work when the screen is off.

HCE services can function from the lock-screen however: this is controlled by the `android:requireDeviceUnlock` attribute in the `<host-apdu-service>` tag of your HCE service. By default, device unlock is not required, and your service will be invoked even if the device is locked.

If you set the `android:requireDeviceUnlock` attribute to "true" for your HCE service, Android will prompt the user to unlock the device when you tap an NFC reader that selects an AID that is resolved to your service. After unlocking, Android will show a dialog prompting the user to tap again to complete the transaction. This is necessary because the user may have moved the device away from the NFC reader in order to unlock it.

## Coexistence with Secure Element Cards

This section is of interest for developers that have deployed an application that relies on a secure element for card emulation. Android's HCE implementation is designed to work in parallel with other methods of implementing card emulation, including the use of secure elements.

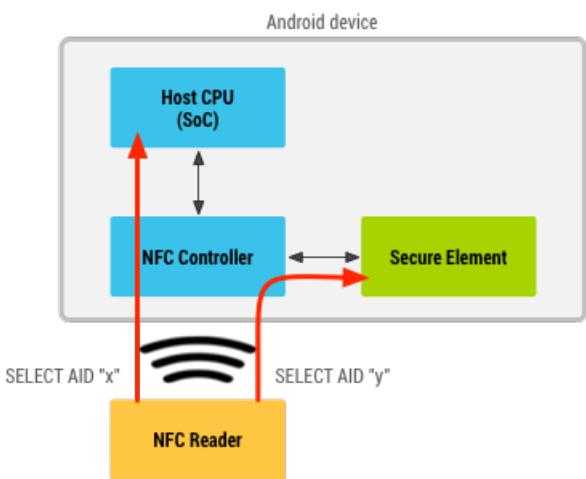
**Note:** Android does not offer APIs for directly communicating with a secure element itself.

This coexistence is based on a principle called "AID routing": the NFC controller keeps a routing table that consists of a (finite) list of routing rules. Each routing rule contains an AID and a destination. The destination can either be the host CPU (where Android apps are running), or a connected secure element.

When the NFC reader sends an APDU with a "SELECT AID", the NFC controller parses it and checks whether the AIDs matches with any AID in its routing table. If it matches, that APDU and all APDUs following it will be sent to the destination associated with the AID, until another "SELECT AID" APDU is received or the NFC link is broken.

**Note:** While ISO/IEC 7816-4 defines the concept of "partial matches" as well, this is currently not supported by Android HCE devices.

This architecture is illustrated in figure 4.



**Figure 4.** Android operating with both secure element and host-card emulation.

The NFC controller typically also contains a default route for APDUs. When an AID is not found in the routing table, the default route is used. While this setting might differ from device to device, Android devices are required to ensure that the AIDs being registered by your app are properly routed to the host.

Android applications that implement a HCE service or that use a secure element don't have to worry about configuring the routing table - that is taken care of by Android automatically. Android merely needs to know which AIDs can be handled by HCE services and which ones can be handled by the secure element. Based on which services are installed and which the user has configured as preferred, the routing table is configured automatically.

We've already described how to declare AIDs for HCE services. The following section explains how to declare AIDs for applications that use a secure element for card emulation.

## Secure element AID registration

Applications using a secure element for card emulation can declare a so-called "off host service" in their manifest. The declaration of such a service is almost identical to the declaration of a HCE service. The exceptions are:

- The action used in the intent-filter must be set to `SERVICE_INTERFACE`.
- The meta-data name attribute must be set to `SERVICE_META_DATA`.
- The meta-data XML file must use the `<offhost-apdu-service>` root tag.

```

<service android:name=".MyOffHostApduService" android:exported="true"
 android:permission="android.permission.BIND_NFC_SERVICE">
 <intent-filter>
 <action android:name="android.nfc.cardemulation.action.OFF_HOST_APDU_SERVICE"/>
 </intent-filter>
 <meta-data android:name="android.nfc.cardemulation.off_host_apdu_service"
 android:resource="@xml/apduservice"/>
</service>

```

An example of the corresponding `apduservice.xml` file registering two AIDs:

```

<offhost-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
 android:description="@string/servicedesc">
 <aid-group android:description="@string/subscription" android:category="other">
 <aid-filter android:name="F0010203040506"/>
 <aid-filter android:name="F0394148148100"/>
 </aid-group>
</offhost-apdu-service>

```

The `android:requireDeviceUnlock` attribute does not apply to off host services, because the host CPU is not involved in the transaction and therefore cannot prevent the secure element from executing transactions when the device is locked.

The `android:apduServiceBanner` attribute must be used for off host services that are payment applications as well in order to be selectable as a default payment application.

## Off host service invocation

Android itself will never start or bind to a service that is declared as "off host". This is because the actual transactions are executed by the secure element and not by the Android service itself. The service declaration merely allows applications to register AIDs present on the secure element.

## HCE and Security

---

The HCE architecture itself provides one core piece of security: because your service is protected by the [BIND\\_NFC\\_SERVICE](#) system permission, only the OS can bind to and communicate with your service. This ensures that any APDU you receive is actually an APDU that was received by the OS from the NFC controller, and that any APDU you send back will only go to the OS, which in turn directly forwards the APDUs to the NFC controller.

The core remaining piece is where you get your data that your app sends to the NFC reader. This is intentionally decoupled in the HCE design: it does not care where the data comes from, it just makes sure that it is safely transported to the NFC controller and out to the NFC reader.

For securely storing and retrieving the data that you want to send from your HCE service, you can, for example, rely on the Android Application Sandbox, which isolates your app's data from other apps. For more details on Android security, read [Security Tips](#).

## Protocol parameters and details

---

This section is of interest for developers that want to understand what protocol parameters HCE devices use during the anti-collision and activation phases of the NFC protocols. This allows building a reader infrastructure that is compatible with Android HCE devices.

### Nfc-A (ISO/IEC 14443 type A) protocol anti-collision and activation

As part of the Nfc-A protocol activation, multiple frames are exchanged.

In the first part of the exchange the HCE device will present its UID; HCE devices should be assumed to have a random UID. This means that on every tap, the UID that is presented to the reader will be a randomly generated UID. Because of this, NFC readers should not depend on the UID of HCE devices as a form of authentication or identification.

The NFC reader can subsequently select the HCE device by sending a SEL\_REQ command. The SEL\_RES response of the HCE device will at least have the 6th bit (0x20) set, indicating that the device supports ISO-DEP. Note that other bits in the SEL\_RES may be set as well, indicating for example support for the NFC-DEP (p2p) protocol. Since other bits may be set, readers wanting to interact with HCE devices should explicitly check for the 6th bit only, and not compare the complete SEL\_RES with a value of 0x20.

### ISO-DEP activation

After the Nfc-A protocol is activated, the ISO-DEP protocol activation is initiated by the NFC reader. It sends a "RATS" (Request for Answer To Select) command. The RATS response, the ATS, is completely generated by the NFC controller and not configurable by HCE services. However, HCE implementations are required to meet NFC Forum requirements for the ATS response, so NFC readers can count on these parameters being set in accordance with NFC Forum requirements for any HCE device.

The section below provides more details on the individual bytes of the ATS response provided by the NFC controller on a HCE device:

- TL: length of the ATS response. Must not indicate a length greater than 20 bytes.
- T0: bits 5, 6 and 7 must be set on all HCE devices, indicating TA(1), TB(1) and TC(1) are included in the ATS response. Bits 1 to 4 indicate the FSCI, coding the maximum frame size. On HCE devices the value of FSCI must be between 0h and 8h.
- T(A)1: defines bitrates between reader and emulator, and whether they can be asymmetric. There are no bitrate requirements or guarantees for HCE devices.
- T(B)1: bits 1 to 4 indicate the Start-up Frame Guard time Integer (SFGI). On HCE devices, SFGI must be <= 8h. Bits 5 to 8 indicate the Frame Waiting time Integer (FWI) and codes the Frame Waiting Time (FWT). On HCE devices, FWI must be <= 8h.
- T(C)1: bit 5 indicates support for "Advanced Protocol features". HCE devices may or may not support "Advanced Protocol features". Bit 2

indicates support for DID. HCE devices may or may not support DID. Bit 1 indicates support for NAD. HCE devices must not support NAD and set bit 1 to zero.

- Historical bytes: HCE devices may return up to 15 historical bytes. NFC readers willing to interact with HCE services should make no assumptions about the contents of the historical bytes or their presence.

Note that many HCE devices are likely made compliant with protocol requirements that the payment networks united in EMVCo have specified in their "Contactless Communication Protocol" specification. In particular:

- FSCI in T0 must be between 2h and 8h.
- T(A)1 must be set to 0x80, indicating only the 106 kbit/s bitrate is supported, and asymmetric bitrates between reader and emulator are not supported.
- FWI in T(B)1 must be <= 7h.

## APDU data exchange

As noted earlier, HCE implementations only support a single logical channel. Attempting to select applications on different logical channels will not work on a HCE device.

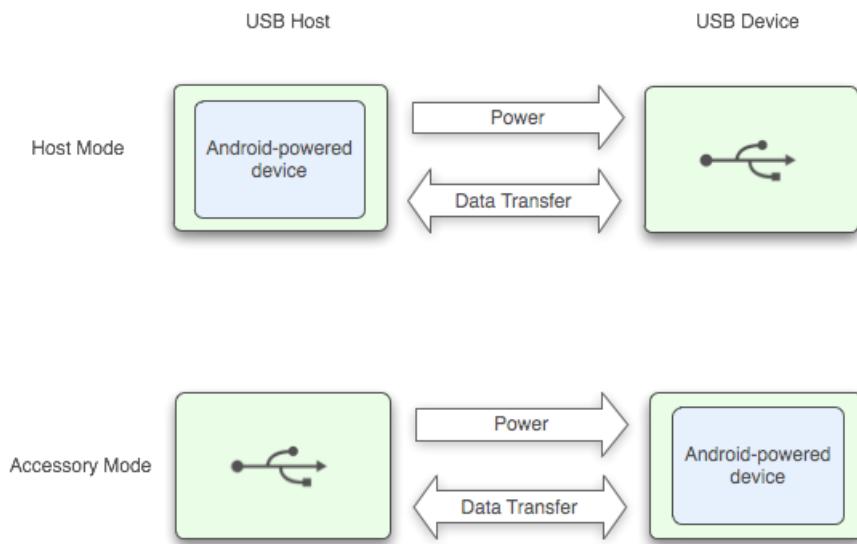
# USB Host and Accessory

## Topics

- [USB Accessory](#)
- [USB Host](#)

Android supports a variety of USB peripherals and Android USB accessories (hardware that implements the Android accessory protocol) through two modes: USB accessory and USB host. In USB accessory mode, the external USB hardware act as the USB hosts. Examples of accessories might include robotics controllers; docking stations; diagnostic and musical equipment; kiosks; card readers; and much more. This gives Android-powered devices that do not have host capabilities the ability to interact with USB hardware. Android USB accessories must be designed to work with Android-powered devices and must adhere to the [Android accessory communication protocol](#). In USB host mode, the Android-powered device acts as the host. Examples of devices include digital cameras, keyboards, mice, and game controllers. USB devices that are designed for a wide range of applications and environments can still interact with Android applications that can correctly communicate with the device.

Figure 1 shows the differences between the two modes. When the Android-powered device is in host mode, it acts as the USB host and powers the bus. When the Android-powered device is in USB accessory mode, the connected USB hardware (an Android USB accessory in this case) acts as the host and powers the bus.



**Figure 1.** USB Host and Accessory Modes

USB accessory and host modes are directly supported in Android 3.1 (API level 12) or newer platforms. USB accessory mode is also backported to Android 2.3.4 (API level 10) as an add-on library to support a broader range of devices. Device manufacturers can choose whether or not to include the add-on library on the device's system image.

**Note:** Support for USB host and accessory modes are ultimately dependant on the device's hardware, regardless of platform level. You can filter for devices that support USB host and accessory through a `<uses-feature>` element. See the [USB accessory](#) and [host](#) documentation for more details.

## Debugging considerations

When debugging applications that use USB accessory or host features, you most likely will have USB hardware connected to your Android-

powered device. This will prevent you from having an `adb` connection to the Android-powered device via USB. You can still access `adb` over a network connection. To enable `adb` over a network connection:

1. Connect the Android-powered device via USB to your computer.
2. From your SDK `platform-tools/` directory, enter `adb tcpip 5555` at the command prompt.
3. Enter `adb connect <device-ip-address>:5555` You should now be connected to the Android-powered device and can issue the usual `adb` commands like `adb logcat`.
4. To set your device to listen on USB, enter `adb usb`.



# USB Accessory

In this document

- › [Choosing the Right USB Accessory APIs](#)
  - › [Installing the Google APIs add-on library](#)
  - › [API Overview](#)
    - › [Usage differences between the add-on library and the platform APIs](#)
  - › [Android Manifest Requirements](#)
  - › [Working with accessories](#)
    - › [Discovering an accessory](#)
    - › [Obtaining permission to communicate with an accessory](#)
    - › [Communicating with an accessory](#)
    - › [Terminating communication with an accessory](#)

See also

- › [Android USB Accessory Development Kit](#)

USB accessory mode allows users to connect USB host hardware specifically designed for Android-powered devices. The accessories must adhere to the Android accessory protocol outlined in the [Android Accessory Development Kit](#) documentation. This allows Android-powered devices that cannot act as a USB host to still interact with USB hardware. When an Android-powered device is in USB accessory mode, the attached Android USB accessory acts as the host, provides power to the USB bus, and enumerates connected devices. Android 3.1 (API level 12) supports USB accessory mode and the feature is also backported to Android 2.3.4 (API level 10) to enable support for a broader range of devices.

## Choosing the Right USB Accessory APIs

Although the USB accessory APIs were introduced to the platform in Android 3.1, they are also available in Android 2.3.4 using the Google APIs add-on library. Because these APIs were backported using an external library, there are two packages that you can import to support USB accessory mode. Depending on what Android-powered devices you want to support, you might have to use one over the other:

- **`com.android.future.usb`**: To support USB accessory mode in Android 2.3.4, the [Google APIs add-on library](#) includes the backported USB accessory APIs and they are contained in this namespace. Android 3.1 also supports importing and calling the classes within this namespace to support applications written with the add-on library. This add-on library is a thin wrapper around the [`android.hardware.usb`](#) accessory APIs and does not support USB host mode. If you want to support the widest range of devices that support USB accessory mode, use the add-on library and import this package. It is important to note that not all Android 2.3.4 devices are required to support the USB accessory feature. Each individual device manufacturer decides whether or not to support this capability, which is why you must declare it in your manifest file.
- **`android.hardware.usb`**: This namespace contains the classes that support USB accessory mode in Android 3.1. This package is included as part of the framework APIs, so Android 3.1 supports USB accessory mode without the use of an add-on library. Use this package if you only care about Android 3.1 or newer devices that have hardware support for USB accessory mode, which you can declare in your manifest file.

## Installing the Google APIs add-on library

If you want to install the add-on, you can do so by installing the Google APIs Android API 10 package with the SDK Manager. See [Installing the Google APIs Add-on](#) for more information on installing the add-on library.

# API Overview

Because the add-on library is a wrapper for the framework APIs, the classes that support the USB accessory feature are similar. You can use the reference documentation for the [android.hardware.usb](#) even if you are using the add-on library.

**Note:** There is, however, a minor [usage difference](#) between the add-on library and framework APIs that you should be aware of.

The following table describes the classes that support the USB accessory APIs:

Class	Description
<a href="#">UsbManager</a>	Allows you to enumerate and communicate with connected USB accessories.
<a href="#">UsbAccessory</a>	Represents a USB accessory and contains methods to access its identifying information.

## Usage differences between the add-on library and platform APIs

There are two usage differences between using the Google APIs add-on library and the platform APIs.

If you are using the add-on library, you must obtain the [UsbManager](#) object in the following manner:

```
UsbManager manager = UsbManager.getInstance(this);
```

If you are not using the add-on library, you must obtain the [UsbManager](#) object in the following manner:

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
```

When you filter for a connected accessory with an intent filter, the [UsbAccessory](#) object is contained inside the intent that is passed to your application. If you are using the add-on library, you must obtain the [UsbAccessory](#) object in the following manner:

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

If you are not using the add-on library, you must obtain the [UsbAccessory](#) object in the following manner:

```
UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

## Android Manifest requirements

The following list describes what you need to add to your application's manifest file before working with the USB accessory APIs. The [manifest and resource file examples](#) show how to declare these items:

- Because not all Android-powered devices are guaranteed to support the USB accessory APIs, include a `<uses-feature>` element that declares that your application uses the `android.hardware.usb.accessory` feature.
- If you are using the [add-on library](#), add the `<uses-library>` element specifying `com.android.future.usb.accessory` for the library.
- Set the minimum SDK of the application to API Level 10 if you are using the add-on library or 12 if you are using the `android.hardware.usb` package.
- If you want your application to be notified of an attached USB accessory, specify an `<intent-filter>` and `<meta-data>` element pair for the `android.hardware.usb.action.USB_ACCESSORY_ATTACHED` intent in your main activity. The `<meta-data>` element points to an external XML resource file that declares identifying information about the accessory that you want to detect.

In the XML resource file, declare `<usb-accessory>` elements for the accessories that you want to filter. Each `<usb-accessory>` can have the following attributes:

- `manufacturer`
- `model`
- `version`

Save the resource file in the `res/xml/` directory. The resource file name (without the `.xml` extension) must be the same as the one you specified in the `<meta-data>` element. The format for the XML resource file is also shown in the [example](#) below.

## Manifest and resource file examples

The following example shows a sample manifest and its corresponding resource file:

```
<manifest ...>
 <uses-feature android:name="android.hardware.usb.accessory" />

 <uses-sdk android:minSdkVersion="<version>" />
 ...
 <application>
 <uses-library android:name="com.android.future.usb.accessory" />
 <activity ...>
 ...
 <intent-filter>
 <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
 </intent-filter>

 <meta-data android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
 android:resource="@xml/accessory_filter" />
 </activity>
 </application>
</manifest>
```

In this case, the following resource file should be saved in `res/xml/accessory_filter.xml` and specifies that any accessory that has the corresponding model, manufacturer, and version should be filtered. The accessory sends these attributes the Android-powered device:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
 <usb-accessory model="DemoKit" manufacturer="Google" version="1.0"/>
</resources>
```

## Working with Accessories

When users connect USB accessories to an Android-powered device, the Android system can determine whether your application is interested in the connected accessory. If so, you can set up communication with the accessory if desired. To do this, your application has to:

1. Discover connected accessories by using an intent filter that filters for accessory attached events or by enumerating connected accessories and finding the appropriate one.
2. Ask the user for permission to communicate with the accessory, if not already obtained.
3. Communicate with the accessory by reading and writing data on the appropriate interface endpoints.

### Discovering an accessory

Your application can discover accessories by either using an intent filter to be notified when the user connects an accessory or by enumerating accessories that are already connected. Using an intent filter is useful if you want to be able to have your application automatically detect a desired accessory. Enumerating connected accessories is useful if you want to get a list of all connected accessories or if your application did not filter for an intent.

#### Using an intent filter

To have your application discover a particular USB accessory, you can specify an intent filter to filter for the `android.hardware.usb.action.USB_ACCESSORY_ATTACHED` intent. Along with this intent filter, you need to specify a resource file that specifies properties of the USB accessory, such as manufacturer, model, and version. When users connect an accessory that matches your accessory filter,

The following example shows how to declare the intent filter:

```
<activity ...>
 ...
 <intent-filter>
 <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
 </intent-filter>

 <meta-data android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
 android:resource="@xml/accessory_filter" />
</activity>
```

The following example shows how to declare the corresponding resource file that specifies the USB accessories that you're interested in:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
 <usb-accessory manufacturer="Google, Inc." model="DemoKit" version="1.0" />
</resources>
```

In your activity, you can obtain the [UsbAccessory](#) that represents the attached accessory from the intent like this (with the add-on library):

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

or like this (with the platform APIs):

```
UsbAccessory accessory = (UsbAccessory)intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

## Enumerating accessories

You can have your application enumerate accessories that have identified themselves while your application is running.

Use the [getAccessoryList\(\)](#) method to get an array all the USB accessories that are connected:

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
UsbAccessory[] accessoryList = manager.getAccessoryList();
```

**Note:** Only one connected accessory is supported at a time.

## Obtaining permission to communicate with an accessory

Before communicating with the USB accessory, your application must have permission from your users.

**Note:** If your application [uses an intent filter](#) to discover accessories as they're connected, it automatically receives permission if the user allows your application to handle the intent. If not, you must request permission explicitly in your application before connecting to the accessory.

Explicitly asking for permission might be necessary in some situations such as when your application enumerates accessories that are already connected and then wants to communicate with one. You must check for permission to access an accessory before trying to communicate with it. If not, you will receive a runtime error if the user denied permission to access the accessory.

To explicitly obtain permission, first create a broadcast receiver. This receiver listens for the intent that gets broadcast when you call [requestPermission\(\)](#). The call to [requestPermission\(\)](#) displays a dialog to the user asking for permission to connect to the accessory.

The following sample code shows how to create the broadcast receiver:

```
private static final String ACTION_USB_PERMISSION =
 "com.android.example.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {

 public void onReceive(Context context, Intent intent) {
 String action = intent.getAction();
 if (ACTION_USB_PERMISSION.equals(action)) {
 synchronized (this) {
 UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);

 if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
 if(accessory != null){
 //call method to set up accessory communication
 }
 } else {
 Log.d(TAG, "permission denied for accessory " + accessory);
 }
 }
 }
 }
};
```

To register the broadcast receiver, put this in your `onCreate()` method in your activity:

```
UsbManager mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
private static final String ACTION_USB_PERMISSION =
 "com.android.example.USB_PERMISSION";
...
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);
```

To display the dialog that asks users for permission to connect to the accessory, call the `requestPermission()` method:

```
UsbAccessory accessory;
...
mUsbManager.requestPermission(accessory, mPermissionIntent);
```

When users reply to the dialog, your broadcast receiver receives the intent that contains the `EXTRA_PERMISSION_GRANTED` extra, which is a boolean representing the answer. Check this extra for a value of true before connecting to the accessory.

## Communicating with an accessory

You can communicate with the accessory by using the `UsbManager` to obtain a file descriptor that you can set up input and output streams to read and write data to descriptor. The streams represent the accessory's input and output bulk endpoints. You should set up the communication between the device and accessory in another thread, so you don't lock the main UI thread. The following example shows how to open an accessory to communicate with:

```

UsbAccessory mAccessory;
ParcelFileDescriptor mFileDescriptor;
FileInputStream mInputStream;
FileOutputStream mOutputStream;

...

private void openAccessory() {
 Log.d(TAG, "openAccessory: " + accessory);
 mFileDescriptor = mUsbManager.openAccessory(mAccessory);
 if (mFileDescriptor != null) {
 FileDescriptor fd = mFileDescriptor.getFileDescriptor();
 mInputStream = new FileInputStream(fd);
 mOutputStream = new FileOutputStream(fd);
 Thread thread = new Thread(null, this, "AccessoryThread");
 thread.start();
 }
}

```

In the thread's `run()` method, you can read and write to the accessory by using the `FileInputStream` or `FileOutputStream` objects. When reading data from an accessory with a `FileInputStream` object, ensure that the buffer that you use is big enough to store the USB packet data. The Android accessory protocol supports packet buffers up to 16384 bytes, so you can choose to always declare your buffer to be of this size for simplicity.

**Note:** At a lower level, the packets are 64 bytes for USB full-speed accessories and 512 bytes for USB high-speed accessories. The Android accessory protocol bundles the packets together for both speeds into one logical packet for simplicity.

For more information about using threads in Android, see [Processes and Threads](#).

## Terminating communication with an accessory

When you are done communicating with an accessory or if the accessory was detached, close the file descriptor that you opened by calling `close()`. To listen for detached events, create a broadcast receiver like below:

```

BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
 public void onReceive(Context context, Intent intent) {
 String action = intent.getAction();

 if (UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {
 UsbAccessory accessory = (UsbAccessory)intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
 if (accessory != null) {
 // call your method that cleans up and closes communication with the accessory
 }
 }
 }
};

```

Creating the broadcast receiver within the application, and not the manifest, allows your application to only handle detached events while it is running. This way, detached events are only sent to the application that is currently running and not broadcast to all applications.



# USB Host

In this document

- › [API Overview](#)
- › [Android Manifest Requirements](#)
- › [Working with devices](#)
  - › [Discovering a device](#)
  - › [Obtaining permission to communicate with a device](#)
  - › [Communicating with a device](#)
  - › [Terminating communication with a device](#)

Related Samples

- › [AdbTest](#)
- › [MissileLauncher](#)

When your Android-powered device is in USB host mode, it acts as the USB host, powers the bus, and enumerates connected USB devices. USB host mode is supported in Android 3.1 and higher.

## API Overview

Before you begin, it is important to understand the classes that you need to work with. The following table describes the USB host APIs in the [android.hardware.usb](#) package.

**Table 1.** USB Host APIs

Class	Description
<a href="#">UsbManager</a>	Allows you to enumerate and communicate with connected USB devices.
<a href="#">UsbDevice</a>	Represents a connected USB device and contains methods to access its identifying information, interfaces, and endpoints.
<a href="#">UsbInterface</a>	Represents an interface of a USB device, which defines a set of functionality for the device. A device can have one or more interfaces on which to communicate on.
<a href="#">UsbEndpoint</a>	Represents an interface endpoint, which is a communication channel for this interface. An interface can have one or more endpoints, and usually has input and output endpoints for two-way communication with the device.
<a href="#">UsbDeviceConnection</a>	Represents a connection to the device, which transfers data on endpoints. This class allows you to send data back and forth synchronously or asynchronously.
<a href="#">UsbRequest</a>	Represents an asynchronous request to communicate with a device through a <a href="#">UsbDeviceConnection</a> .
<a href="#">UsbConstants</a>	Defines USB constants that correspond to definitions in linux/usb/ch9.h of the Linux kernel.

In most situations, you need to use all of these classes ([UsbRequest](#) is only required if you are doing asynchronous communication) when communicating with a USB device. In general, you obtain a [UsbManager](#) to retrieve the desired [UsbDevice](#). When you have the device, you need to find the appropriate [UsbInterface](#) and the [UsbEndpoint](#) of that interface to communicate on. Once you obtain the correct endpoint, open a [UsbDeviceConnection](#) to communicate with the USB device.

## Android Manifest Requirements

The following list describes what you need to add to your application's manifest file before working with the USB host APIs:

- Because not all Android-powered devices are guaranteed to support the USB host APIs, include a `<uses-feature>` element that declares that your application uses the `android.hardware.usb.host` feature.
- Set the minimum SDK of the application to API Level 12 or higher. The USB host APIs are not present on earlier API levels.
- If you want your application to be notified of an attached USB device, specify an `<intent-filter>` and `<meta-data>` element pair for the `android.hardware.usb.action.USB_DEVICE_ATTACHED` intent in your main activity. The `<meta-data>` element points to an external XML resource file that declares identifying information about the device that you want to detect.

In the XML resource file, declare `<usb-device>` elements for the USB devices that you want to filter. The following list describes the attributes of `<usb-device>`. In general, use vendor and product ID if you want to filter for a specific device and use class, subclass, and protocol if you want to filter for a group of USB devices, such as mass storage devices or digital cameras. You can specify none or all of these attributes. Specifying no attributes matches every USB device, so only do this if your application requires it:

- `vendor-id`
- `product-id`
- `class`
- `subclass`
- `protocol` (device or interface)

Save the resource file in the `res/xml/` directory. The resource file name (without the `.xml` extension) must be the same as the one you specified in the `<meta-data>` element. The format for the XML resource file is in the [example](#) below.

## Manifest and resource file examples

The following example shows a sample manifest and its corresponding resource file:

```
<manifest ...>
 <uses-feature android:name="android.hardware.usb.host" />
 <uses-sdk android:minSdkVersion="12" />
 ...
 <application>
 <activity ...>
 ...
 <intent-filter>
 <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
 </intent-filter>

 <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
 android:resource="@xml/device_filter" />
 </activity>
 </application>
</manifest>
```

In this case, the following resource file should be saved in `res/xml/device_filter.xml` and specifies that any USB device with the specified attributes should be filtered:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
 <usb-device vendor-id="1234" product-id="5678" class="255" subclass="66" protocol="1" />
</resources>
```

## Working with Devices

When users connect USB devices to an Android-powered device, the Android system can determine whether your application is interested in the connected device. If so, you can set up communication with the device if desired. To do this, your application has to:

1. Discover connected USB devices by using an intent filter to be notified when the user connects a USB device or by enumerating USB devices that are already connected.
2. Ask the user for permission to connect to the USB device, if not already obtained.
3. Communicate with the USB device by reading and writing data on the appropriate interface endpoints.

## Discovering a device

Your application can discover USB devices by either using an intent filter to be notified when the user connects a device or by enumerating USB devices that are already connected. Using an intent filter is useful if you want to be able to have your application automatically detect a desired device. Enumerating connected USB devices is useful if you want to get a list of all connected devices or if your application did not filter for an intent.

### Using an intent filter

To have your application discover a particular USB device, you can specify an intent filter to filter for the `android.hardware.usb.action.USB_DEVICE_ATTACHED` intent. Along with this intent filter, you need to specify a resource file that specifies properties of the USB device, such as product and vendor ID. When users connect a device that matches your device filter, the system presents them with a dialog that asks if they want to start your application. If users accept, your application automatically has permission to access the device until the device is disconnected.

The following example shows how to declare the intent filter:

```
<activity ...>
...
<intent-filter>
 <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>

<meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
 android:resource="@xml/device_filter" />
</activity>
```

The following example shows how to declare the corresponding resource file that specifies the USB devices that you're interested in:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <usb-device vendor-id="1234" product-id="5678" />
</resources>
```

In your activity, you can obtain the `UsbDevice` that represents the attached device from the intent like this:

```
UsbDevice device = (UsbDevice) intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
```

### Enumerating devices

If your application is interested in inspecting all of the USB devices currently connected while your application is running, it can enumerate devices on the bus. Use the `getDeviceList()` method to get a hash map of all the USB devices that are connected. The hash map is keyed by the USB device's name if you want to obtain a device from the map.

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
...
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
UsbDevice device = deviceList.get("deviceName");
```

If desired, you can also just obtain an iterator from the hash map and process each device one by one:

```

UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
...
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
while(deviceIterator.hasNext()){
 UsbDevice device = deviceIterator.next();
 //your code
}

```

## Obtaining permission to communicate with a device

Before communicating with the USB device, your application must have permission from your users.

**Note:** If your application [uses an intent filter](#) to discover USB devices as they're connected, it automatically receives permission if the user allows your application to handle the intent. If not, you must request permission explicitly in your application before connecting to the device.

Explicitly asking for permission might be necessary in some situations such as when your application enumerates USB devices that are already connected and then wants to communicate with one. You must check for permission to access a device before trying to communicate with it. If not, you will receive a runtime error if the user denied permission to access the device.

To explicitly obtain permission, first create a broadcast receiver. This receiver listens for the intent that gets broadcast when you call `requestPermission()`. The call to `requestPermission()` displays a dialog to the user asking for permission to connect to the device. The following sample code shows how to create the broadcast receiver:

```

private static final String ACTION_USB_PERMISSION =
 "com.android.example.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {

 public void onReceive(Context context, Intent intent) {
 String action = intent.getAction();
 if (ACTION_USB_PERMISSION.equals(action)) {
 synchronized (this) {
 UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);

 if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
 if(device != null){
 //call method to set up device communication
 }
 }
 else {
 Log.d(TAG, "permission denied for device " + device);
 }
 }
 }
 }
};


```

To register the broadcast receiver, add this in your `onCreate()` method in your activity:

```

UsbManager mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
private static final String ACTION_USB_PERMISSION =
 "com.android.example.USB_PERMISSION";
...
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);

```

To display the dialog that asks users for permission to connect to the device, call the `requestPermission()` method:

```

UsbDevice device;
...
mUsbManager.requestPermission(device, mPermissionIntent);

```

When users reply to the dialog, your broadcast receiver receives the intent that contains the `EXTRA_PERMISSION_GRANTED` extra, which is a boolean representing the answer. Check this extra for a value of true before connecting to the device.

## Communicating with a device

Communication with a USB device can be either synchronous or asynchronous. In either case, you should create a new thread on which to carry out all data transmissions, so you don't block the UI thread. To properly set up communication with a device, you need to obtain the appropriate `UsbInterface` and `UsbEndpoint` of the device that you want to communicate on and send requests on this endpoint with a `UsbDeviceConnection`. In general, your code should:

- Check a `UsbDevice` object's attributes, such as product ID, vendor ID, or device class to figure out whether or not you want to communicate with the device.
- When you are certain that you want to communicate with the device, find the appropriate `UsbInterface` that you want to use to communicate along with the appropriate `UsbEndpoint` of that interface. Interfaces can have one or more endpoints, and commonly will have an input and output endpoint for two-way communication.
- When you find the correct endpoint, open a `UsbDeviceConnection` on that endpoint.
- Supply the data that you want to transmit on the endpoint with the `bulkTransfer()` or `controlTransfer()` method. You should carry out this step in another thread to prevent blocking the main UI thread. For more information about using threads in Android, see [Processes and Threads](#).

The following code snippet is a trivial way to do a synchronous data transfer. Your code should have more logic to correctly find the correct interface and endpoints to communicate on and also should do any transferring of data in a different thread than the main UI thread:

```
private Byte[] bytes;
private static int TIMEOUT = 0;
private boolean forceClaim = true;

...

UsbInterface intf = device.getInterface(0);
UsbEndpoint endpoint = intf.getEndpoint(0);
UsbDeviceConnection connection = mUsbManager.openDevice(device);
connection.claimInterface(intf, forceClaim);
connection.bulkTransfer(endpoint, bytes, bytes.length, TIMEOUT); //do in another thread
```

To send data asynchronously, use the `UsbRequest` class to `initialize` and `queue` an asynchronous request, then wait for the result with `requestWait()`.

For more information, see the [AdbTest sample](#), which shows how to do asynchronous bulk transfers, and the [MissileLauncher sample](#), which shows how to listen on an interrupt endpoint asynchronously.

## Terminating communication with a device

When you are done communicating with a device or if the device was detached, close the `UsbInterface` and `UsbDeviceConnection` by calling `releaseInterface()` and `close()`. To listen for detached events, create a broadcast receiver like below:

```
BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
 public void onReceive(Context context, Intent intent) {
 String action = intent.getAction();

 if (UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action)) {
 UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
 if (device != null) {
 // call your method that cleans up and closes communication with the device
 }
 }
 }
};
```

Creating the broadcast receiver within the application, and not the manifest, allows your application to only handle detached events while it is

running. This way, detached events are only sent to the application that is currently running and not broadcast to all applications.



# Text and Input

Use text services to add convenient features such as copy/paste and spell checking to your app. You can also develop your own text services to offer custom IMEs, dictionaries, and spelling checkers that you can distribute to users as applications.

## BLOG ARTICLES

---

### Add Voice Typing To Your IME

A new feature available in Android 4.0 is voice typing: the difference for users is that the recognition results appear in the text box while they are still speaking. If you are an IME developer, you can easily integrate with voice typing.

### Speech Input API for Android

We believe speech can fundamentally change the mobile experience. We would like to invite every Android application developer to consider integrating speech input capabilities via the Android SDK.

### Making Sense of Multitouch

The word "multitouch" gets thrown around quite a bit and it's not always clear what people are referring to. For some it's about hardware capability, for others it refers to specific gesture support in software. Whatever you decide to call it, today we're going to look at how to make your apps and views behave nicely with multiple fingers on the screen.

# Copy and Paste

## Quickview

- › A clipboard-based framework for copying and pasting data.
- › Supports both simple and complex data, including text strings, complex data structures, text and binary stream data, and application assets.
- › Copies and pastes simple text directly to and from the clipboard.
- › Copies and pastes complex data using a content provider.
- › Requires API 11.

## In this document

- › [The Clipboard Framework](#)
- › [Clipboard Classes](#)
  - › [ClipboardManager](#)
  - › [ClipData, ClipDescription, and ClipData.Item](#)
  - › [ClipData convenience methods](#)
  - › [Coercing the clipboard data to text](#)
- › [Copying to the Clipboard](#)
- › [Pasting from the Clipboard](#)
  - › [Pasting plain text](#)
  - › [Pasting data from a content URI](#)
  - › [Pasting an Intent](#)
- › [Using Content Providers to Copy Complex Data](#)
  - › [Encoding an identifier on the URI](#)
  - › [Copying data structures](#)
  - › [Copying data streams](#)
- › [Designing Effective Copy/Paste Functionality](#)

## Key classes

- › [ClipboardManager](#)
- › [ClipData](#)
- › [ClipData.Item](#)
- › [ClipDescription](#)
- › [Uri](#)
- › [ContentProvider](#)
- › [Intent](#)

## Related Samples

- › [Note Pad sample application](#)

## See also

- › [Content Providers](#)

Android provides a powerful clipboard-based framework for copying and pasting. It supports both simple and complex data types, including text strings, complex data structures, text and binary stream data, and even application assets. Simple text data is stored directly in the clipboard, while complex data is stored as a reference that the pasting application resolves with

a content provider. Copying and pasting works both within an application and between applications that implement the framework.

Since a part of the framework uses content providers, this topic assumes some familiarity with the Android Content Provider API, which is described in the topic [Content Providers](#).

## The Clipboard Framework

---

When you use the clipboard framework, you put data into a clip object, and then put the clip object on the system-wide clipboard. The clip object can take one of three forms:

### Text

A text string. You put the string directly into the clip object, which you then put onto the clipboard. To paste the string, you get the clip object from the clipboard and copy the string to into your application's storage.

### URI

A [Uri](#) object representing any form of URI. This is primarily for copying complex data from a content provider. To copy data, you put a [Uri](#) object into a clip object and put the clip object onto the clipboard. To paste the data, you get the clip object, get the [Uri](#) object, resolve it to a data source such as a content provider, and copy the data from the source into your application's storage.

### Intent

An [Intent](#). This supports copying application shortcuts. To copy data, you create an Intent, put it into a clip object, and put the clip object onto the clipboard. To paste the data, you get the clip object and then copy the Intent object into your application's memory area.

The clipboard holds only one clip object at a time. When an application puts a clip object on the clipboard, the previous clip object disappears.

If you want to allow users to paste data into your application, you don't have to handle all types of data. You can examine the data on the clipboard before you give users the option to paste it. Besides having a certain data form, the clip object also contains metadata that tells you what MIME type or types are available. This metadata helps you decide if your application can do something useful with the clipboard data. For example, if you have an application that primarily handles text, you may want to ignore clip objects that contain a URI or Intent.

You may also want to allow users to paste text regardless of the form of data on the clipboard. To do this, you can force the clipboard data into a text representation, and then paste this text. This is described in the section [Coercing the clipboard to text](#).

## Clipboard Classes

---

This section describes the classes used by the clipboard framework.

### ClipboardManager

In the Android system, the system clipboard is represented by the global [ClipboardManager](#) class. You do not instantiate this class directly; instead, you get a reference to it by invoking [getSystemService\(CLIPBOARD\\_SERVICE\)](#).

### ClipData, ClipData.Item, and ClipDescription

To add data to the clipboard, you create a [ClipData](#) object that contains both a description of the data and the data itself. The clipboard holds only one [ClipData](#) at a time. A [ClipData](#) contains a [ClipDescription](#) object and one or more [ClipData.Item](#) objects.

A [ClipDescription](#) object contains metadata about the clip. In particular, it contains an array of available MIME types for the clip's data. When you put a clip on the clipboard, this array is available to pasting applications, which can examine it to see if they can handle any of available the MIME types.

A [ClipData.Item](#) object contains the text, URI, or Intent data:

### Text

A `CharSequence`.

## URI

A `Uri`. This usually contains a content provider URI, although any URI is allowed. The application that provides the data puts the URI on the clipboard. Applications that want to paste the data get the URI from the clipboard and use it to access the content provider (or other data source) and retrieve the data.

## Intent

An `Intent`. This data type allows you to copy an application shortcut to the clipboard. Users can then paste the shortcut into their applications for later use.

You can add more than one `ClipData.Item` object to a clip. This allows users to copy and paste multiple selections as a single clip. For example, if you have a list widget that allows the user to select more than one item at a time, you can copy all the items to the clipboard at once. To do this, you create a separate `ClipData.Item` for each list item, and then you add the `ClipData.Item` objects to the `ClipData` object.

## ClipData convenience methods

The `ClipData` class provides static convenience methods for creating a `ClipData` object with a single `ClipData.Item` object and a simple `ClipDescription` object:

```
newPlainText(label, text)
```

Returns a `ClipData` object whose single `ClipData.Item` object contains a text string. The `ClipDescription` object's label is set to `label`. The single MIME type in `ClipDescription` is `MIMETYPE_TEXT_PLAIN`.

Use `newPlainText()` to create a clip from a text string.

```
newUri(resolver, label, URI)
```

Returns a `ClipData` object whose single `ClipData.Item` object contains a URI. The `ClipDescription` object's label is set to `label`. If the URI is a content URI (`Uri.getScheme()` returns `content`), the method uses the `ContentResolver` object provided in `resolver` to retrieve the available MIME types from the content provider and store them in `ClipDescription`. For a URI that is not a `content`: URI, the method sets the MIME type to `MIMETYPE_TEXT_URLLIST`.

Use `newUri()` to create a clip from a URI, particularly a `content`: URI.

```
newIntent(label, intent)
```

Returns a `ClipData` object whose single `ClipData.Item` object contains an `Intent`. The `ClipDescription` object's label is set to `label`. The MIME type is set to `MIMETYPE_TEXT_INTENT`.

Use `newIntent()` to create a clip from an Intent object.

## Coercing the clipboard data to text

Even if your application only handles text, you can copy non-text data from the clipboard by converting it with the method `ClipData.Item.coerceToText()`.

This method converts the data in `ClipData.Item` to text and returns a `CharSequence`. The value that `ClipData.Item.coerceToText()` returns is based on the form of data in `ClipData.Item`:

### Text

If `ClipData.Item` is text (`getText()` is not null), `coerceToText()` returns the text.

### URI

If `ClipData.Item` is a URI (`getUri()` is not null), `coerceToText()` tries to use it as a content URI:

- If the URI is a content URI and the provider can return a text stream, `coerceToText()` returns a text stream.
- If the URI is a content URI but the provider does not offer a text stream, `coerceToText()` returns a representation of the URI. The representation is the same as that returned by `Uri.toString()`.
- If the URI is not a content URI, `coerceToText()` returns a representation of the URI. The representation is the same as that returned by `Uri.toString()`.

### *Intent*

If `ClipData.Item` is an Intent (`getIntent()` is not null), `coerceToText()` converts it to an Intent URI and returns it. The representation is the same as that returned by `Intent.toUri(URI_INTENT_SCHEME)`.

The clipboard framework is summarized in Figure 1. To copy data, an application puts a `ClipData` object on the `ClipboardManager` global clipboard. The `ClipData` contains one or more `ClipData.Item` objects and one `ClipDescription` object. To paste data, an application gets the `ClipData`, gets its MIME type from the `ClipDescription`, and gets the data either from the `ClipData.Item` or from the content provider referred to by `ClipData.Item`.

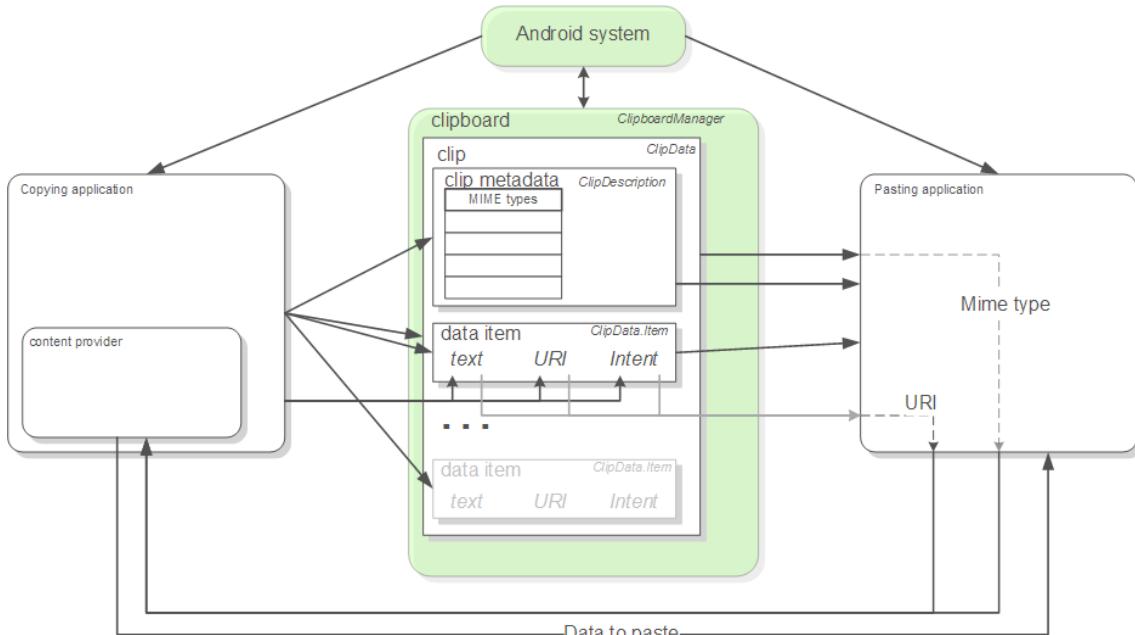


Figure 1. The Android clipboard framework

## Copying to the Clipboard

As described previously, to copy data to the clipboard you get a handle to the global `ClipboardManager` object, create a `ClipData` object, add a `ClipDescription` and one or more `ClipData.Item` objects to it, and add the finished `ClipData` object to the `ClipboardManager` object. This is described in detail in the following procedure:

1. If you are copying data using a content URI, set up a content provider.

The `Note Pad` sample application is an example of using a content provider for copying and pasting. The `NotePadProvider` class implements the content provider. The `NotePad` class defines a contract between the provider and other applications, including the supported MIME types.

2. Get the system clipboard:

```
...
// if the user selects copy
case R.id.menu_copy:

 // Gets a handle to the clipboard service.
 ClipboardManager clipboard = (ClipboardManager)
 getSystemService(Context.CLIPBOARD_SERVICE);
```

3. Copy the data to a new `ClipData` object:

- For text

```
// Creates a new text clip to put on the clipboard
ClipData clip = ClipData.newPlainText("simple text", "Hello, World!");
```

- For a URI

This snippet constructs a URI by encoding a record ID onto the content URI for the provider. This technique is covered in more detail in the section [Encoding an identifier on the URI](#):

```
// Creates a Uri based on a base Uri and a record ID based on the contact's last name
// Declares the base URI string
private static final String CONTACTS = "content://com.example.contacts";

// Declares a path string for URIs that you use to copy data
private static final String COPY_PATH = "/copy";

// Declares the Uri to paste to the clipboard
Uri copyUri = Uri.parse(CONTACTS + COPY_PATH + "/" + lastName);

...

// Creates a new URI clip object. The system uses the anonymous getContentResolver() object to
// get MIME types from provider. The clip object's label is "URI", and its data is
// the Uri previously created.
ClipData clip = ClipData.newUri(getContentResolver(), "URI", copyUri);
```

- For an Intent

This snippet constructs an Intent for an application and then puts it in the clip object:

```
// Creates the Intent
Intent appIntent = new Intent(this, com.example.demo.myapplication.class);

...

// Creates a clip object with the Intent in it. Its label is "Intent" and its data is
// the Intent object created previously
ClipData clip = ClipData.newIntent("Intent", appIntent);
```

4. Put the new clip object on the clipboard:

```
// Set the clipboard's primary clip.
clipboard.setPrimaryClip(clip);
```

## Pasting from the Clipboard

As described previously, you paste data from the clipboard by getting the global clipboard object, getting the clip object, looking at its data, and if possible copying the data from the clip object to your own storage. This section describes in detail how to do this for the three forms of clipboard data.

### Pasting plain text

To paste plain text, first get the global clipboard and verify that it can return plain text. Then get the clip object and copy its text to your own storage using `getText()`, as described in the following procedure:

1. Get the global `ClipboardManager` object using `getSystemService(CLIPBOARD_SERVICE)`. Also declare a global variable to contain the pasted text:

```
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
String pasteData = "";
```

2. Next, determine if you should enable or disable the "paste" option in the current Activity. You should verify that the clipboard contains a clip and that you can handle the type of data represented by the clip:

```
// Gets the ID of the "paste" menu item
MenuItem mPasteItem = menu.findItem(R.id.menu_paste);

// If the clipboard doesn't contain data, disable the paste menu item.
// If it does contain data, decide if you can handle the data.
if (!(clipboard.hasPrimaryClip())) {

 mPasteItem.setEnabled(false);

} else if (!(clipboard.getPrimaryClipDescription().hasMimeType(MIMETYPE_TEXT_PLAIN))) {

 // This disables the paste menu item, since the clipboard has data but it is not plain text
 mPasteItem.setEnabled(false);
} else {

 // This enables the paste menu item, since the clipboard contains plain text.
 mPasteItem.setEnabled(true);
}
```

3. Copy the data from the clipboard. This point in the program is only reachable if the "paste" menu item is enabled, so you can assume that the clipboard contains plain text. You do not yet know if it contains a text string or a URI that points to plain text. The following snippet tests this, but it only shows the code for handling plain text:

```
// Responds to the user selecting "paste"
case R.id.menu_paste:

// Examines the item on the clipboard. If getText() does not return null, the clip item contains the
// text. Assumes that this application can only handle one item at a time.
ClipData.Item item = clipboard.getPrimaryClip().getItemAt(0);

// Gets the clipboard as text.
pasteData = item.getText();

// If the string contains data, then the paste operation is done
if (pasteData != null) {
 return;

// The clipboard does not contain text. If it contains a URI, attempts to get data from it
} else {
 Uri pasteUri = item.getUri();

 // If the URI contains something, try to get text from it
 if (pasteUri != null) {

 // calls a routine to resolve the URI and get data from it. This routine is not
 // presented here.
 pasteData = resolveUri(Uri);
 return;
 } else {

 // Something is wrong. The MIME type was plain text, but the clipboard does not contain either
 // text or a Uri. Report an error.
 Log.e("Clipboard contains an invalid data type");
 return;
 }
}
```

## Pasting data from a content URI

If the `ClipData.Item` object contains a content URI and you have determined that you can handle one of its MIME types, create a `ContentResolver` and then call the appropriate content provider method to retrieve the data.

The following procedure describes how to get data from a content provider based on a content URI on the clipboard. It checks that a MIME type that the application can use is available from the provider:

1. Declare a global variable to contain the MIME type:

```
// Declares a MIME type constant to match against the MIME types offered by the provider
public static final String MIME_TYPE_CONTACT = "vnd.android.cursor.item/vnd.example.contact";
```

2. Get the global clipboard. Also get a content resolver so you can access the content provider:

```
// Gets a handle to the Clipboard Manager
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

// Gets a content resolver instance
ContentResolver cr = getContentResolver();
```

3. Get the primary clip from the clipboard, and get its contents as a URI:

```
// Gets the clipboard data from the clipboard
ClipData clip = clipboard.getPrimaryClip();

if (clip != null) {

 // Gets the first item from the clipboard data
 ClipData.Item item = clip.getItemAt(0);

 // Tries to get the item's contents as a Uri
 Uri pasteUri = item.getUri();
```

4. Test to see if the URI is a content URI by calling `getType(Uri)`. This method returns null if `Uri` does not point to a valid content provider:

```
// If the clipboard contains a Uri reference
if (pasteUri != null) {

 // Is this a content Uri?
 String uriMimeType = cr.getType(pasteUri);
```

5. Test to see if the content provider supports a MIME type that the current application understands. If it does, call `ContentResolver.query()` to get the data. The return value is a `Cursor`:

```
// If the return value is not null, the Uri is a content Uri
if (uriMimeType != null) {

 // Does the content provider offer a MIME type that the current application can use?
 if (uriMimeType.equals(MIME_TYPE_CONTACT)) {

 // Get the data from the content provider.
 Cursor pasteCursor = cr.query(uri, null, null, null, null);

 // If the Cursor contains data, move to the first record
 if (pasteCursor != null) {
 if (pasteCursor.moveToFirst()) {

 // get the data from the Cursor here. The code will vary according to the
 // format of the data model.
 }
 }

 // close the Cursor
 pasteCursor.close();
 }
}
}
```

## Pasting an Intent

To paste an Intent, first get the global clipboard. Examine the `ClipData.Item` object to see if it contains an Intent. Then call `getIntent()` to copy the Intent to your own storage. The following snippet demonstrates this:

```
// Gets a handle to the Clipboard Manager
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

// Checks to see if the clip item contains an Intent, by testing to see if getIntent() returns null
Intent pasteIntent = clipboard.getPrimaryClip().getItemAt(0).getIntent();

if (pasteIntent != null) {

 // handle the Intent

} else {

 // ignore the clipboard, or issue an error if your application was expecting an Intent to be
 // on the clipboard
}
```

## Using Content Providers to Copy Complex Data

Content providers support copying complex data such as database records or file streams. To copy the data, you put a content URI on the clipboard. Pasting applications then get this URI from the clipboard and use it to retrieve database data or file stream descriptors.

Since the pasting application only has the content URI for your data, it needs to know which piece of data to retrieve. You can provide this information by encoding an identifier for the data on the URI itself, or you can provide a unique URI that will return the data you want to copy. Which technique you choose depends on the organization of your data.

The following sections describe how to set up URLs, how to provide complex data, and how to provide file streams. The descriptions assume that you are familiar with the general principles of content provider design.

## Encoding an identifier on the URI

A useful technique for copying data to the clipboard with a URI is to encode an identifier for the data on the URI itself. Your content provider can then get the identifier from the URI and use it to retrieve the data. The pasting application doesn't have to know that the identifier exists; all it has to do is get your "reference" (the URI plus the identifier) from the clipboard, give it your content provider, and get back the data.

You usually encode an identifier onto a content URI by concatenating it to the end of the URI. For example, suppose you define your provider URI as the following string:

```
"content://com.example.contacts"
```

If you want to encode a name onto this URI, you would use the following snippet:

```
String uriString = "content://com.example.contacts" + "/" + "Smith";

// uriString now contains content://com.example.contacts/Smith.

// Generates a Uri object from the string representation
Uri copyUri = Uri.parse(uriString);
```

If you are already using a content provider, you may want to add a new URI path that indicates the URI is for copying. For example, suppose you already have the following URI paths:

```
"content://com.example.contacts"/people
"content://com.example.contacts"/people/detail
"content://com.example.contacts"/people/images
```

You could add another path that is specific to copy URIs:

```
"content://com.example.contacts/copying"
```

You could then detect a "copy" URI by pattern-matching and handle it with code that is specific for copying and pasting.

You normally use the encoding technique if you're already using a content provider, internal database, or internal table to organize your data. In these cases, you have multiple pieces of data you want to copy, and presumably a unique identifier for each piece. In response to a query from the pasting application, you can look up the data by its identifier and return it.

If you don't have multiple pieces of data, then you probably don't need to encode an identifier. You can simply use a URI that is unique to your provider. In response to a query, your provider would return the data it currently contains.

Getting a single record by ID is used in the [Note Pad](#) sample application to open a note from the notes list. The sample uses the `_id` field from an SQL database, but you can have any numeric or character identifier you want.

## Copying data structures

You set up a content provider for copying and pasting complex data as a subclass of the [ContentProvider](#) component. You should also encode the URI you put on the clipboard so that it points to the exact record you want to provide. In addition, you have to consider the existing state of your application:

- If you already have a content provider, you can add to its functionality. You may only need to modify its `query()` method to handle URIs coming from applications that want to paste data. You will probably want to modify the method to handle a "copy" URI pattern.
- If your application maintains an internal database, you may want to move this database into a content provider to facilitate copying from it.
- If you are not currently using a database, you can implement a simple content provider whose sole purpose is to offer data to applications that are pasting from the clipboard.

In the content provider, you will want to override at least the following methods:

`query()`

Pasting applications will assume that they can get your data by using this method with the URI you put on the clipboard. To support copying, you should have this method detect URIs that contain a special "copy" path. Your application can then create a "copy" URI to put on the clipboard, containing the copy path and a pointer to the exact record you want to copy.

`getType()`

This method should return the MIME type or types for the data you intend to copy. The method `newUri()` calls `getType()` in order to put the MIME types into the new `ClipData` object.

MIME types for complex data are described in the topic [Content Providers](#).

Notice that you don't have to have any of the other content provider methods such as `insert()` or `update()`. A pasting application only needs to get your supported MIME types and copy data from your provider. If you already have these methods, they won't interfere with copy operations.

The following snippets demonstrate how to set up your application to copy complex data:

1. In the global constants for your application, declare a base URI string and a path that identifies URI strings you are using to copy data.

Also declare a MIME type for the copied data:

```
// Declares the base URI string
private static final String CONTACTS = "content://com.example.contacts";

// Declares a path string for URIs that you use to copy data
private static final String COPY_PATH = "/copy";

// Declares a MIME type for the copied data
public static final String MIME_TYPE_CONTACT = "vnd.android.cursor.item/vnd.example.contact";
```

2. In the Activity from which users copy data, set up the code to copy data to the clipboard. In response to a copy request, put the URI on the clipboard:

```
public class MyCopyActivity extends Activity {

 ...

 // The user has selected a name and is requesting a copy.
 case R.id.menu_copy:

 // Appends the last name to the base URI
 // The name is stored in "lastName"
 uriString = CONTACTS + COPY_PATH + "/" + lastName;

 // Parses the string into a URI
 Uri copyUri = Uri.parse(uriString);

 // Gets a handle to the clipboard service.
 ClipboardManager clipboard = (ClipboardManager)
 getSystemService(Context.CLIPBOARD_SERVICE);

 ClipData clip = ClipData.newUri(getContentResolver(), "URI", copyUri);

 // Set the clipboard's primary clip.
 clipboard.setPrimaryClip(clip);
```

3. In the global scope of your content provider, create a URI matcher and add a URI pattern that will match URIs you put on the clipboard:

```
public class MyCopyProvider extends ContentProvider {

 ...

 // A Uri Match object that simplifies matching content URIs to patterns.
 private static final UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

 // An integer to use in switching based on the incoming URI pattern
 private static final int GET_SINGLE_CONTACT = 0;

 ...

 // Adds a matcher for the content URI. It matches
 // "content://com.example.contacts/copy/*"
 sUriMatcher.addURI(CONTACTS, "names/*", GET_SINGLE_CONTACT);
```

4. Set up the `query()` method. This method can handle different URI patterns, depending on how you code it, but only the pattern for the

clipboard copying operation is shown:

```
// Sets up your provider's query() method.
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,
 String sortOrder) {
 ...
 // Switch based on the incoming content URI
 switch (sUriMatcher.match(uri)) {
 case GET_SINGLE_CONTACT:
 // query and return the contact for the requested name. Here you would decode
 // the incoming URI, query the data model based on the last name, and return the result
 // as a Cursor.
 ...
 }
}
```

5. Set up the `getType()` method to return an appropriate MIME type for copied data:

```
// Sets up your provider's getType() method.
public String getType(Uri uri) {
 ...
 switch (sUriMatcher.match(uri)) {
 case GET_SINGLE_CONTACT:
 return (MIME_TYPE_CONTACT);
 ...
 }
}
```

The section [Pasting data from a content URI](#) describes how to get a content URI from the clipboard and use it to get and paste data.

## Copying data streams

You can copy and paste large amounts of text and binary data as streams. The data can have forms such as the following:

- Files stored on the actual device.
- Streams from sockets.
- Large amounts of data stored in a provider's underlying database system.

A content provider for data streams provides access to its data with a file descriptor object such as `AssetFileDescriptor` instead of a `Cursor` object. The pasting application reads the data stream using this file descriptor.

To set up your application to copy a data stream with a provider, follow these steps:

1. Set up a content URI for the data stream you are putting on the clipboard. Options for doing this include the following:
  - Encode an identifier for the data stream onto the URI, as described in the section [Encoding an identifier on the URI](#), and then maintain a table in your provider that contains identifiers and the corresponding stream name.
  - Encode the stream name directly on the URI.
  - Use a unique URI that always returns the current stream from the provider. If you use this option, you have to remember to update your provider to point to a different stream whenever you copy the stream to the clipboard via the URI.
2. Provide a MIME type for each type of data stream you plan to offer. Pasting applications need this information to determine if they can paste the data on the clipboard.
3. Implement one of the `ContentProvider` methods that returns a file descriptor for a stream. If you encode identifiers on the content URI, use this method to determine which stream to open.
4. To copy the data stream to the clipboard, construct the content URI and place it on the clipboard.

To paste a data stream, an application gets the clip from the clipboard, gets the URI, and uses it in a call to a `ContentResolver` file descriptor method that opens the stream. The `ContentResolver` method calls the corresponding `ContentProvider` method, passing it the content URI. Your provider returns the file descriptor to `ContentResolver` method. The pasting application then has the responsibility to read the data from the stream.

The following list shows the most important file descriptor methods for a content provider. Each of these has a corresponding `ContentResolver` method with the string "Descriptor" appended to the method name; for example, the `ContentResolver` analog of `openAssetFile()` is `openAssetDescriptor()`:

#### `openTypedAssetFile()`

This method should return an asset file descriptor, but only if the provided MIME type is supported by the provider. The caller (the application doing the pasting) provides a MIME type pattern. The content provider (of the application that has copied a URI to the clipboard) returns an `AssetFileDescriptor` file handle if it can provide that MIME type, or throws an exception if it can not.

This method handles subsections of files. You can use it to read assets that the content provider has copied to the clipboard.

#### `openAssetFile()`

This method is a more general form of `openTypedAssetFile()`. It does not filter for allowed MIME types, but it can read subsections of files.

#### `openFile()`

This is a more general form of `openAssetFile()`. It can't read subsections of files.

You can optionally use the `openPipeHelper()` method with your file descriptor method. This allows the pasting application to read the stream data in a background thread using a pipe. To use this method, you need to implement the `ContentProvider.PipeDataWriter` interface. An example of doing this is given in the [Note Pad](#) sample application, in the `openTypedAssetFile()` method of `NotePadProvider.java`.

## Designing Effective Copy/Paste Functionality

---

To design effective copy and paste functionality for your application, remember these points:

- At any time, there is only one clip on the clipboard. A new copy operation by any application in the system overwrites the previous clip. Since the user may navigate away from your application and do a copy before returning, you can't assume that the clipboard contains the clip that the user previously copied in *your* application.
- The intended purpose of multiple `ClipData.Item` objects per clip is to support copying and pasting of multiple selections rather than different forms of reference to a single selection. You usually want all of the `ClipData.Item` objects in a clip to have the same form, that is, they should all be simple text, content URI, or `Intent`, but not a mixture.
- When you provide data, you can offer different MIME representations. Add the MIME types you support to the `ClipDescription`, and then implement the MIME types in your content provider.
- When you get data from the clipboard, your application is responsible for checking the available MIME types and then deciding which one, if any, to use. Even if there is a clip on the clipboard and the user requests a paste, your application is not required to do the paste. You *should* do the paste if the MIME type is compatible. You may choose to coerce the data on the clipboard to text using `coerceToText()` if you choose. If your application supports more than one of the available MIME types, you can allow the user to choose which one to use.

# Creating an Input Method

## In This Document

- [The IME Lifecycle](#)
- [Declaring IME Components in the Manifest](#)
- [The Input Method API](#)
- [Designing the Input Method UI](#)
- [Sending Text to the Application](#)
- [Creating an IME Subtype](#)
- [Switching among IME Subtypes](#)
- [General IME Considerations](#)

## See also

- [Onscreen Input Methods](#)

## Sample

- [SoftKeyboard](#)

An input method editor (IME) is a user control that enables users to enter text. Android provides an extensible input-method framework that allows applications to provide users alternative input methods, such as on-screen keyboards or even speech input. After installing the desired IMEs, a user can select which one to use from the system settings, and use it across the entire system; only one IME may be enabled at a time.

To add an IME to the Android system, you create an Android application containing a class that extends [InputMethodService](#). In addition, you usually create a "settings" activity that passes options to the IME service. You can also define a settings UI that's displayed as part of the system settings.

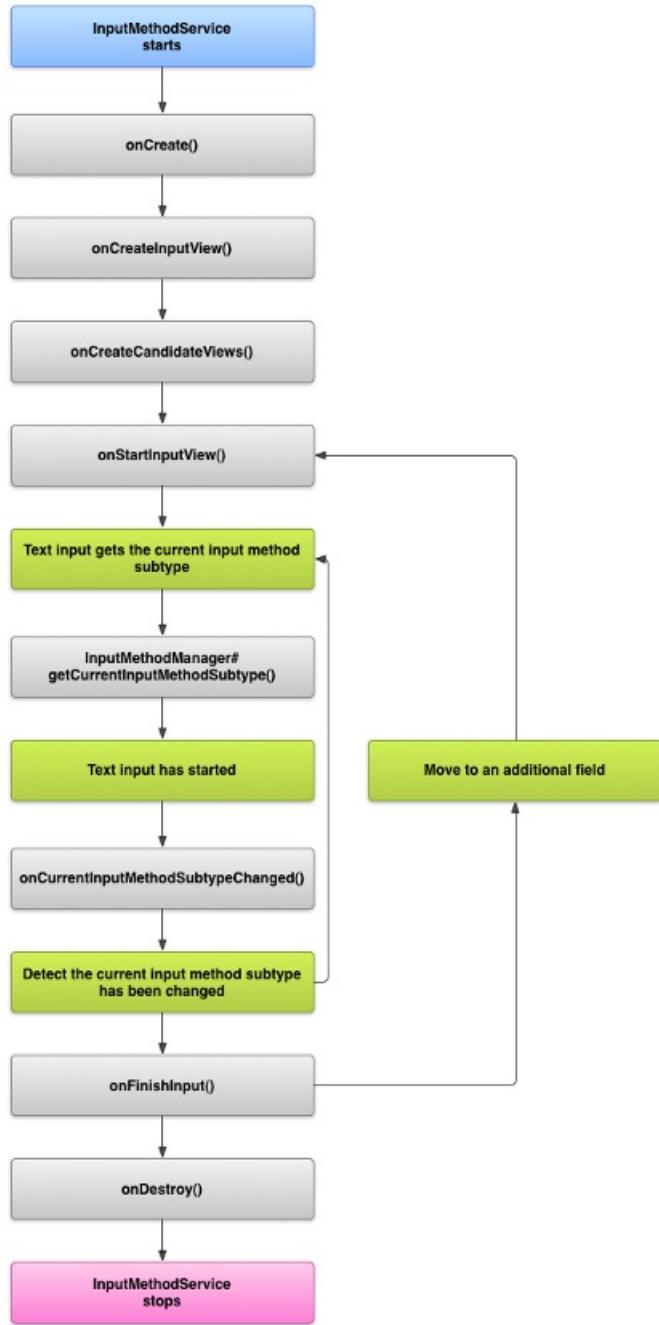
This guide covers the following:

- The IME lifecycle
- Declaring IME components in the application manifest
- The IME API
- Designing an IME UI
- Sending text from an IME to an application
- Working with IME subtypes

If you haven't worked with IMEs before, you should read the introductory article [Onscreen Input Methods](#) first. Also, the [SoftKeyboard](#) sample app included in the SDK contains sample code that you can modify to start building your own IME.

## The IME Lifecycle

The following diagram describes the life cycle of an IME:



**Figure 1.** The life cycle of an IME.

The following sections describe how to implement the UI and code associated with an IME that follows this lifecycle.

## Declaring IME Components in the Manifest

In the Android system, an IME is an Android application that contains a special IME service. The application's manifest file must declare the service, request the necessary permissions, provide an intent filter that matches the action `action.view.InputMethod`, and provide metadata that defines characteristics of the IME. In addition, to provide a settings interface that allows the user to modify the behavior of the IME, you can define a "settings" activity that can be launched from System Settings.

The following snippet declares an IME service. It requests the permission `BIND_INPUT_METHOD` to allow the service to connect the IME to the system, sets up an intent filter that matches the action `android.view.InputMethod`, and defines metadata for the IME:

```
<!-- Declares the input method service -->
<service android:name="FastInputIME"
 android:label="@string/fast_input_label"
 android:permission="android.permission.BIND_INPUT_METHOD">
 <intent-filter>
 <action android:name="android.view.InputMethod" />
 </intent-filter>
 <meta-data android:name="android.view.im"
 android:resource="@xml/method" />
</service>
```

This next snippet declares the settings activity for the IME. It has an intent filter for `ACTION_MAIN` that indicates this activity is the main entry point for the IME application:

```
<!-- Optional: an activity for controlling the IME settings -->
<activity android:name="FastInputIMESettings"
 android:label="@string/fast_input_settings">
 <intent-filter>
 <action android:name="android.intent.action.MAIN"/>
 </intent-filter>
</activity>
```

You can also provide access to the IME's settings directly from its UI.

## The Input Method API

Classes specific to IMEs are found in the `android.inputmethodservice` and `android.view.inputmethod` packages. The `KeyEvent` class is important for handling keyboard characters.

The central part of an IME is a service component, a class that extends `InputMethodService`. In addition to implementing the normal service lifecycle, this class has callbacks for providing your IME's UI, handling user input, and delivering text to the field that currently has focus. By default, the `InputMethodService` class provides most of the implementation for managing the state and visibility of the IME and communicating with the current input field.

The following classes are also important:

### `BaseInputConnection`

Defines the communication channel from an `InputMethod` back to the application that is receiving its input. You use it to read text around the cursor, commit text to the text box, and send raw key events to the application. Applications should extend this class rather than implementing the base interface `InputConnection`.

### `KeyboardView`

An extension of `View` that renders a keyboard and responds to user input events. The keyboard layout is specified by an instance of `Keyboard`, which you can define in an XML file.

## Designing the Input Method UI

There are two main visual elements for an IME: the `input` view and the `candidates` view. You only have to implement the elements that are relevant to the input method you're designing.

### Input view

The input view is the UI where the user inputs text in the form of keyclicks, handwriting or gestures. When the IME is displayed for the first time, the system calls the `onCreateInputView()` callback. In your implementation of this method, you create the layout you want to display in the IME window and return the layout to the system. This snippet is an example of implementing the `onCreateInputView()` method:

```

@Override
public View onCreateInputView() {
 MyKeyboardView inputView =
 (MyKeyboardView) getLayoutInflater().inflate(R.layout.input, null);

 inputView.setOnKeyboardActionListener(this);
 inputView.setKeyboard(mLatinKeyboard);

 return mInputView;
}

```

In this example, `MyKeyboardView` is an instance of a custom implementation of `KeyboardView` that renders a `Keyboard`. If you're building a traditional QWERTY keyboard, see the `KeyboardView` class.

## Candidates view

The candidates view is the UI where the IME displays potential word corrections or suggestions for the user to select. In the IME lifecycle, the system calls `onCreateCandidatesView()` when it's ready to display the candidates view. In your implementation of this method, return a layout that shows word suggestions, or return null if you don't want to show anything. A null response is the default behavior, so you don't have to implement this if you don't provide suggestions.

For an example implementation that provides user suggestions, see the [SoftKeyboard](#) sample app.

## UI design considerations

This section describes some specific UI design considerations for IMEs.

### Handling multiple screen sizes

The UI for your IME must be able to scale for different screen sizes, and it also must handle both landscape and portrait orientations. In non-fullscreen IME mode, leave sufficient space for the application to show the text field and any associated context, so that no more than half the screen is occupied by the IME. In fullscreen IME mode this is not an issue.

### Handling different input types

Android text fields allow you to set a specific input type, such as free-form text, numbers, URLs, email addresses, and search strings. When you implement a new IME, you need to detect the input type of each field and provide the appropriate interface for it. However, you don't have to set up your IME to check that the user entered valid text for the input type; that's the responsibility of the application that owns the text field.

For example, here are screenshots of the interfaces that the Latin IME provided with the Android platform provides for text and phone number inputs:



**Figure 2.** Latin IME input types.

When an input field receives focus and your IME starts, the system calls `onStartInputView()`, passing in an `EditorInfo` object that contains details about the input type and other attributes of the text field. In this object, the `inputType` field contains the text field's input type.

The `inputType` field is an `int` that contains bit patterns for various input type settings. To test it for the text field's input type, mask it with the constant `TYPE_MASK_CLASS`, like this:

```
inputType & InputType.TYPE_MASK_CLASS
```

The input type bit pattern can have one of several values, including:

#### `TYPE_CLASS_NUMBER`

A text field for entering numbers. As illustrated in the previous screen shot, the Latin IME displays a number pad for fields of this type.

#### `TYPE_CLASS_DATETIME`

A text field for entering a date and time.

#### `TYPE_CLASS_PHONE`

A text field for entering telephone numbers.

#### `TYPE_CLASS_TEXT`

A text field for entering all supported characters.

These constants are described in more detail in the reference documentation for [InputType](#).

The `inputType` field can contain other bits that indicate a variant of the text field type, such as:

#### `TYPE_TEXT_VARIATION_PASSWORD`

A variant of `TYPE_CLASS_TEXT` for entering passwords. The input method will display dingbats instead of the actual text.

#### `TYPE_TEXT_VARIATION_URI`

A variant of `TYPE_CLASS_TEXT` for entering web URLs and other Uniform Resource Identifiers (URIs).

#### `TYPE_TEXT_FLAG_AUTO_COMPLETE`

A variant of `TYPE_CLASS_TEXT` for entering text that the application "auto-completes" from a dictionary, search, or other facility.

Remember to mask `inputType` with the appropriate constant when you test for these variants. The available mask constants are listed in the reference documentation for [InputType](#).

**Caution:** In your own IME, make sure you handle text correctly when you send it to a password field. Hide the password in your UI both in the input view and in the candidates view. Also remember that you shouldn't store passwords on a device. To learn more, see the [Designing for Security](#) guide.

## Sending Text to the Application

As the user inputs text with your IME, you can send text to the application by sending individual key events or by editing the text around the cursor in the application's text field. In either case, you use an instance of [InputConnection](#) to deliver the text. To get this instance, call `InputMethodService.getCurrentInputConnection()`.

### Editing the text around the cursor

When you're handling the editing of existing text in a text field, some of the more useful methods in [BaseInputConnection](#) are:

#### `getTextBeforeCursor()`

Returns a [CharSequence](#) containing the number of requested characters before the current cursor position.

#### `getTextAfterCursor()`

Returns a [CharSequence](#) containing the number of requested characters following the current cursor position.

#### `deleteSurroundingText()`

Deletes the specified number of characters before and following the current cursor position.

## commitText()

Commit a `CharSequence` to the text field and set a new cursor position.

For example, the following snippet shows how to replace the four characters to the left of the cursor with the text "Hello!":

```
InputConnection ic = getCurrentInputConnection();
ic.deleteSurroundingText(4, 0);
ic.commitText("Hello", 1);
ic.commitText("!", 1);
```

## Composing text before committing

If your IME does text prediction or requires multiple steps to compose a glyph or word, you can show the progress in the text field until the user commits the word, and then you can replace the partial composition with the completed text. You may give special treatment to the text by adding a "span" to it when you pass it to `setComposingText()`.

The following snippet shows how to show progress in a text field:

```
InputConnection ic = getCurrentInputConnection();
ic.setComposingText("Composi", 1);
ic.setComposingText("Composin", 1);
ic.commitText("Composing ", 1);
```

The following screenshots show how this appears to the user:



Figure 3. Composing text before committing.

## Intercepting hardware key events

Even though the input method window doesn't have explicit focus, it receives hardware key events first and can choose to consume them or forward them along to the application. For example, you may want to consume the directional keys to navigate within your UI for candidate selection during composition. You may also want to trap the back key to dismiss any popups originating from the input method window.

To intercept hardware keys, override `onKeyDown()` and `onKeyUp()`. See the [SoftKeyboard](#) sample app for an example.

Remember to call the `super()` method for keys you don't want to handle yourself.

## Creating an IME Subtype

Subtypes allow the IME to expose multiple input modes and languages supported by an IME. A subtype can represent:

- A locale such as `en_US` or `fr_FR`.
- An input mode such as voice, keyboard, or handwriting.
- Other input styles, forms, or properties specific to the IME, such as 10-key or qwerty keyboard layouts.

Basically, the mode can be any text such as "keyboard", "voice", and so forth. A subtype can also expose a combination of these.

Subtype information is used for an IME switcher dialog that's available from the notification bar and also for IME settings. The information also allows the framework to bring up a specific subtype of an IME directly. When you build an IME, use the subtype facility, because it helps the user identify and switch between different IME languages and modes.

You define subtypes in one of the input method's XML resource files, using the `<subtype>` element. The following snippet defines an IME with two subtypes: a keyboard subtype for the US English locale, and another keyboard subtype for the French language locale for France:

```

<input-method xmlns:android="http://schemas.android.com/apk/res/android"
 android:settingsActivity="com.example.softkeyboard.Settings"
 android:icon="@drawable/ime_icon">
 <subtype android:name="@string/display_name_english_keyboard_ime"
 android:icon="@drawable/subtype_icon_english_keyboard_ime"
 android:imeSubtypeLanguage="en_US"
 android:imeSubtypeMode="keyboard"
 android:imeSubtypeExtraValue="somePrivateOption=true" />
 <subtype android:name="@string/display_name_french_keyboard_ime"
 android:icon="@drawable/subtype_icon_french_keyboard_ime"
 android:imeSubtypeLanguage="fr_FR"
 android:imeSubtypeMode="keyboard"
 android:imeSubtypeExtraValue="foobar=30, someInternalOption=false" />
 <subtype android:name="@string/display_name_german_keyboard_ime" ... />
</input-method>

```

To ensure that your subtypes are labeled correctly in the UI, use %s to get a subtype label that is the same as the subtype's locale label. This is demonstrated in the next two snippets. The first snippet shows part of the input method's XML file:

```

<subtype
 android:label="@string/label_subtype_generic"
 android:imeSubtypeLocale="en_US"
 android:icon="@drawable/icon_en_us"
 android:imeSubtypeMode="keyboard" />

```

The next snippet is part of the IME's `strings.xml` file. The string resource `label_subtype_generic`, which is used by the input method UI definition to set the subtype's label, is defined as:

```
<string name="label_subtype_generic">%s</string>
```

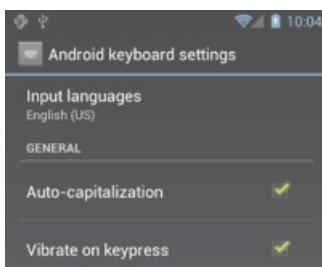
This setting causes the subtype's display name to match the locale setting. For example, in any English locale, the display name is "English (United States)".

## Choosing IME subtypes from the notification bar

The Android system manages all subtypes exposed by all IMEs. IME subtypes are treated as modes of the IME they belong to. In the notification bar, a user can select an available subtype for the currently-set IME, as shown in the following screenshot:



**Figure 4.** Choosing an IME subtype from the notification bar.



**Figure 5.** Setting subtype preferences in System Settings.

## Choosing IME subtypes from System Settings

A user can control how subtypes are used in the "Language & input" settings panel in the System Settings area. In the `SoftKeyboard` sample app, the file `InputMethodSettingsFragment.java` contains an implementation that facilitates a subtype enabler in the IME settings. Refer to the `SoftKeyboard` sample app in the Android SDK for more information about how to support Input Method Subtypes in your IME.

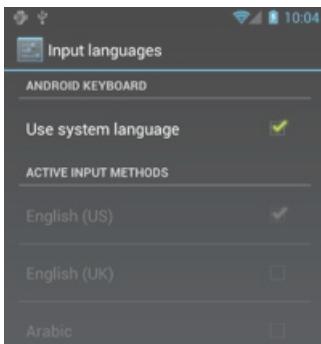


Figure 6. Choosing a language for the IME.

## Switching among IME Subtypes

You can allow users to switch easily among multiple IME subtypes by providing a switching key, such as the globe-shaped language icon, as part of the keyboard. Doing so greatly improves the keyboard's usability, and can help avoid user frustration. To enable such switching, perform the following steps:

1. Declare `supportsSwitchingToNextInputMethod = "true"` in the input method's XML resource files. Your declaration should look similar to the following snippet:

```
<input-method xmlns:android="http://schemas.android.com/apk/res/android"
 android:settingsActivity="com.example.softkeyboard.Settings"
 android:icon="@drawable/ime_icon"
 android:supportsSwitchingToNextInputMethod="true">
```

2. Call the `shouldOfferSwitchingToNextInputMethod()` method.
3. If the method returns true, display a switching key.
4. When the user taps the switching key, call `switchToNextInputMethod()`, passing false to the second parameter. A value of false tells the system to treat all subtypes equally, regardless of what IME they belong to. Specifying true requires the system to cycle through subtypes in the current IME.

**Caution:** Prior to Android 5.0 (API level 21), `switchToNextInputMethod()` is not aware of the `supportsSwitchingToNextInputMethod` attribute. If the user switches into an IME without a switching key, he or she may get stuck in that IME, unable to switch out of it easily.

## General IME Considerations

Here are some other things to consider as you're implementing your IME:

- Provide a way for users to set options directly from the IME's UI.
- Because multiple IMEs may be installed on the device, provide a way for the user to switch to a different IME directly from the input method UI.
- Bring up the IME's UI quickly. Preload or load on demand any large resources so that users see the IME as soon as they tap on a text field. Cache resources and views for subsequent invocations of the input method.
- Conversely, you should release large memory allocations soon after the input method window is hidden, so that applications can have sufficient memory to run. Consider using a delayed message to release resources if the IME is in a hidden state for a few seconds.
- Make sure that users can enter as many characters as possible for the language or locale associated with the IME. Remember that users may use punctuation in passwords or user names, so your IME has to provide many different characters to allow users to enter a password and get access to the device.

# Spelling Checker Framework

## In This Document

- › [Spell Check Lifecycle](#)
- › [Implementing a Spell Checker Service](#)
- › [Implementing a Spell Checker Client](#)

## See also

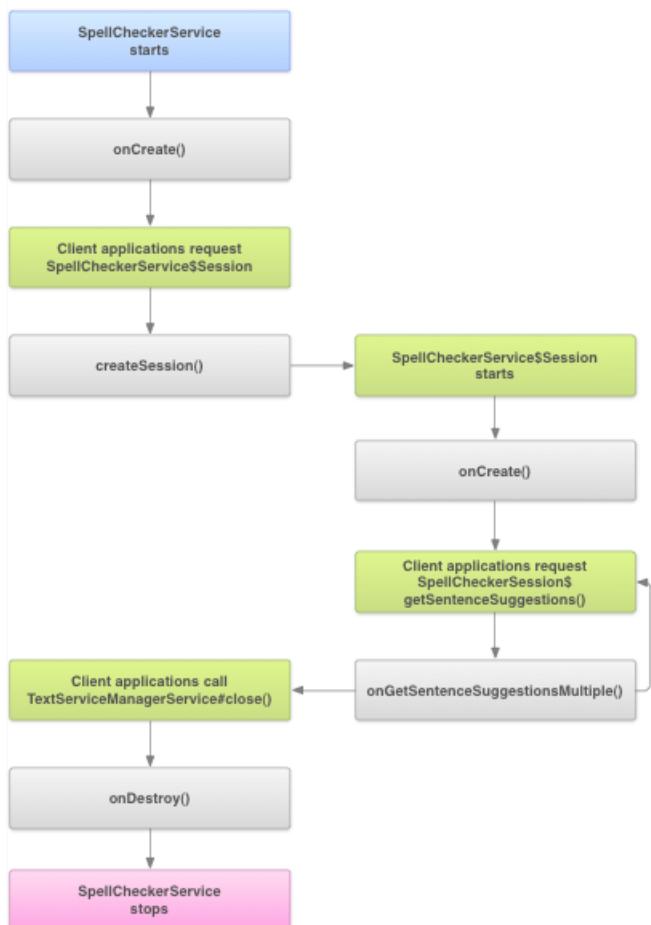
- › [Spell Checker Service sample app](#)
- › [Spell Checker Client sample app](#)

The Android platform offers a spelling checker framework that lets you implement and access spell checking in your application. The framework is one of the Text Service APIs offered by the Android platform.

To use the framework in your app, you create a special type of Android service that generates a spelling checker **session** object. Based on text you provide, the session object returns spelling suggestions generated by the spelling checker.

## Spell Checker Lifecycle

The following diagram shows the lifecycle of the spelling checker service:



**Figure 1.** The spelling checker service lifecycle.

To initiate spell checking, your app starts its implementation of the spelling checker service. Clients in your app, such as activities or individual UI elements, request a spelling checker session from the service, then use the session to get suggestions for text. As a client terminates its operation, it closes its spelling checker session. If necessary, your app can shut down the spelling checker service at any time.

## Implementing a Spell Checker Service

To use the spelling checker framework in your app, add a spelling checker service component including the session object definition. You can also add to your app an optional activity that controls settings. You must also add an XML metadata file that describes the spelling checker service, and add the appropriate elements to your manifest file.

### Spell checker classes

Define the service and session object with the following classes:

A subclass of `SpellCheckerService`

The `SpellCheckerService` implements both the `Service` class and the spelling checker framework interface. Within your subclass, you must implement the following method:

`createSession()`

A factory method that returns a `SpellCheckerService.Session` object to a client that wants to do spell checking.

See the [Spell Checker Service](#) sample app to learn more about implementing this class.

An implementation of `SpellCheckerService.Session`

An object that the spelling checker service provides to clients, to let them pass text to the spelling checker and receive suggestions. Within this class, you must implement the following methods:

`onCreate()`

Called by the system in response to `createSession()`. In this method, you can initialize the `SpellCheckerService.Session` object based on the current locale and so forth.

`onGetSentenceSuggestionsMultiple()`

Does the actual spell checking. This method returns an array of `SentenceSuggestionsInfo` containing suggestions for the sentences passed to it.

Optionally, you can implement `onCancel()`, which handles requests to cancel spell checking, `onGetSuggestions()`, which handles a word suggestion request, or `onGetSuggestionsMultiple()`, which handles batches of word suggestion requests.

See the [Spell Checker Client](#) sample app to learn more about implementing this class.

**Note:** You must implement all aspects of spell checking as asynchronous and thread-safe. A spelling checker may be called simultaneously by different threads running on different cores. The `SpellCheckerService` and `SpellCheckerService.Session` take care of this automatically.

### Spell checker manifest and metadata

In addition to code, you need to provide the appropriate manifest file and a metadata file for the spelling checker.

The manifest file defines the application, the service, and the activity for controlling settings, as shown in the following snippet:

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.android.samplespellcheckerservice" >
 <application
 android:label="@string/app_name" >
 <service
 android:label="@string/app_name"
 android:name=".SampleSpellCheckerService"
 android:permission="android.permission.BIND_TEXT_SERVICE" >
 <intent-filter >
 <action android:name="android.service.textservice.SpellCheckerService" />
 </intent-filter>

 <meta-data
 android:name="android.view.textservice.scs"
 android:resource="@xml/spellchecker" />
 </service>

 <activity
 android:label="@string/sample_settings"
 android:name="SpellCheckerSettingsActivity" >
 <intent-filter >
 <action android:name="android.intent.action.MAIN" />
 </intent-filter>
 </activity>
 </application>
</manifest>

```

Notice that components that want to use the service must request the permission `BIND_TEXT_SERVICE` to ensure that only the system binds to the service. The service's definition also specifies the `spellchecker.xml` metadata file, which is described in the next section.

The metadata file `spellchecker.xml` contains the following XML:

```

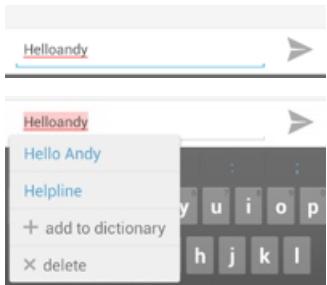
<spell-checker xmlns:android="http://schemas.android.com/apk/res/android"
 android:label="@string/spellchecker_name"
 android:settingsActivity="com.example.SpellCheckerSettingsActivity">
 <subtype
 android:label="@string/subtype_generic"
 android:subtypeLocale="en"
 />
 <subtype
 android:label="@string/subtype_generic"
 android:subtypeLocale="fr"
 />
</spell-checker>

```

The metadata specifies the activity that the spelling checker uses for controlling settings. It also defines subtypes for the spelling checker; in this case, the subtypes define locales that the spelling checker can handle.

## Accessing the Spell Checker Service from a Client

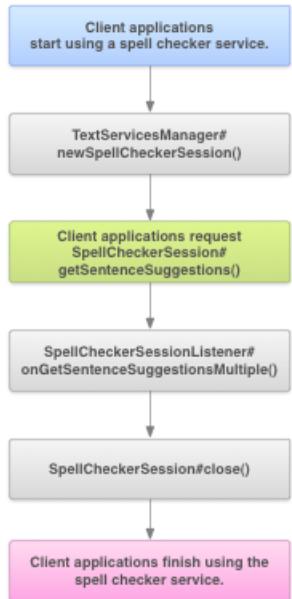
Applications that use `TextView` views automatically benefit from spell checking, because `TextView` automatically uses a spelling checker. The following screenshots show this:



**Figure 2.** Spell checking in `TextView`.

However, you may want to interact directly with a spelling checker service in other cases as well. The following diagram shows the flow of

control for interacting with a spelling checker service:



**Figure 3.** Interacting with a spelling checker service.

The [Spell Checker Client](#) sample app shows how to interact with a spelling checker service. The LatinIME input method editor in the Android Open Source Project also contains an example of spell checking.



# Data Storage

Store application data in databases, files, or preferences, in internal or removable storage. You can also add a data backup service to restore identity, settings, and app data to new devices.

# 存储选项

## 存储快览

- › 将共享首选项用于原始数据
- › 将内部设备存储用于私有数据
- › 将外部存储用于非私有大型数据集
- › 将 SQLite 数据库用于结构化存储

## 本文内容

- › [使用共享首选项](#)
- › [使用内部存储](#)
- › [使用外部存储](#)
- › [使用数据库](#)
- › [使用网络连接](#)

## 另请参阅

- › [内容提供程序和内容解析程序](#)

Android 为您提供了多种选项来保存永久性应用数据。您所选择的解决方案取决于您的特定需求，例如数据应该是应用的私有数据，还是可供其他应用（和用户）访问，以及您的数据需要多少空间等。

您的数据存储选项如下：

### [共享首选项](#)

在键值对中存储私有原始数据。

### [内部存储](#)

在设备内存中存储私有数据。

### [外部存储](#)

在共享的外部存储中存储公共数据。

### [SQLite 数据库](#)

在私有数据库中存储结构化数据。

### [网络连接](#)

在网络中使用您自己的网络服务器存储数据。

Android 为您提供了一种方法 — 使用[内容提供程序](#)将您的数据（甚至是您的私有数据）公开给其他应用。 内容提供程序是一个可选组件，可根据您希望施加的任何限制公开您的应用数据的读/写访问权限。 如需了解有关使用内容提供程序的更多信息，请参阅[内容提供程序](#)文档。

## 使用共享首选项

[SharedPreferences](#) 类提供了一个通用框架，以便您能够保存和检索原始数据类型的永久性键值对。 您可以使用 [SharedPreferences](#) 来保存任何原始数据：布尔值、浮点值、整型值、长整型和字符串。 此数据将跨多个用户会话永久保留（即使您的应用已终止亦如此）。

要获取应用的 `SharedPreferences` 对象，请使用以下两个方法之一：

- `getSharedPreferences()` - 如果您需要多个按名称（使用第一个参数指定）识别的首选项文件，请使用此方法。
- `getPreferences()` - 如果您只需要一个用于 Activity 的首选项文件，请使用此方法。由于这将是用于 Activity 的唯一首选项文件，因此无需提供名称。

要写入值：

1. 调用 `edit()` 以获取 `SharedPreferences.Editor`。
2. 使用 `putBoolean()` 和 `putString()` 等方法添加值。
3. 使用 `commit()` 提交新值

要读取值，请使用 `getBoolean()` 和 `getString()` 等 `SharedPreferences` 方法。

以下是在计算器中保存静音按键模式首选项的示例：

```
public class Calc extends Activity {
 public static final String PREFS_NAME = "MyPrefsFile";

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

 // Restore preferences
 SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
 boolean silent = settings.getBoolean("silentMode", false);
 setSilent(silent);
 }

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

 // We need an Editor object to make preference changes.
 // All objects are from android.context.Context
 SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
 SharedPreferences.Editor editor = settings.edit();
 editor.putBoolean("silentMode", mSilentMode);

 // Commit the edits!
 editor.commit();
 }
}
```

## 使用内部存储

您可以直接在设备的内部存储中保存文件。默认情况下，保存到内部存储的文件是应用的私有文件，其他应用（和用户）不能访问这些文件。当用户卸载您的应用时，这些文件也会被移除。

要创建私有文件并写入到内部存储：

1. 使用文件名称和操作模式调用 `openFileOutput()`。这将返回一个 `FileOutputStream`。
2. 使用 `write()` 写入到文件。
3. 使用 `close()` 关闭流式传输。

例如：

### 用户首选项

严格来说，共享首选项并非用于保存“用户首选项”，例如用户所选择的铃声。如果您有兴趣为您的应用创建用户首选项，请参阅 `PreferenceActivity`，其中为您提供了一个 Activity 框架，用于创建将会自动永久保留（通过共享首选项）的用户首选项。

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

`MODE_PRIVATE` 将会创建文件（或替换具有相同名称的文件），并将其设为应用的私有文件。其他可用模式包括：`MODE_APPEND`、`MODE_WORLD_READABLE` 和 `MODE_WORLD_WRITEABLE`。

**注：**自 API 级别 17 以来，常量 `MODE_WORLD_READABLE` 和 `MODE_WORLD_WRITEABLE` 已被弃用。从 Android N 开始，使用这些常量将会导致引发 `SecurityException`。这意味着，面向 Android N 和更高版本的应用无法按名称共享私有文件，尝试共享“file://”URI 将会导致引发 `FileUriExposedException`。如果您的应用需要与其他应用共享私有文件，则可以将 `FileProvider` 与 `FLAG_GRANT_READ_URI_PERMISSION` 配合使用。另请参阅[共享文件](#)。

要从内部存储读取文件：

1. 调用 `openFileInput()` 并向其传递要读取的文件名称。这将返回一个 `InputStream`。
2. 使用 `read()` 读取文件字节。
3. 然后使用 `close()` 关闭流式传输。

**提示：**如果在编译时想要保存应用中的静态文件，请在项目的 `res/raw/` 目录中保存该文件。可以使用 `openRawResource()` 打开该资源并传递 `R.raw.<filename>` 资源 ID。此方法将返回一个 `InputStream`，您可以使用该流式传输读取文件（但不能写入到原始文件）。

## 保存缓存文件

如果您想要缓存一些数据，而不是永久存储这些数据，应该使用 `getCacheDir()` 来打开一个 `File`，它表示您的应用应该将临时缓存文件保存到的内部目录。

当设备的内部存储空间不足时，Android 可能会删除这些缓存文件以回收空间。但您不应该依赖系统来为您清理这些文件，而应该始终自行维护缓存文件，使其占用的空间保持在合理的限制范围内（例如 1 MB）。当用户卸载您的应用时，这些文件也会被移除。

## 其他实用方法

### `getFilesDir()`

获取在其中存储内部文件的文件系统目录的绝对路径。

### `getDir()`

在您的内部存储空间内创建（或打开现有的）目录。

### `deleteFile()`

删除保存在内部存储的文件。

### `fileList()`

返回您的应用当前保存的一系列文件。

## 使用外部存储

每个兼容 Android 的设备都支持可用于保存文件的共享“外部存储”。该存储可能是可移除的存储介质（例如 SD 卡）或内部（不可移除）存储。保存到外部存储的文件是全局可读取文件，而且，在计算机上启用 USB 大容量存储以传输文件后，可由用户修改这些文件。

**注意：**如果用户在计算机上装载了外部存储或移除了介质，则外部存储可能变为不可用状态，并且在您保存到外部存储的文件上没有实施任何安全性。所有应用都能读取和写入放置在外部存储上的文件，并且用户可以移除这些文件。

# 使用作用域目录访问

在 Android 7.0 或更高版本中，如果您需要访问外部存储上的特定目录，请使用作用域目录访问。作用域目录访问可简化您的应用访问标准外部存储目录（例如 `Pictures` 目录）的方式，并提供简单的权限 UI，清楚地详细介绍应用正在请求访问的目录。有关作用域目录访问的更多详情，请参阅[使用作用域目录访问](#)。

## 获取外部存储的访问权限

要读取或写入外部存储上的文件，您的应用必须获取 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 系统权限。例如：

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

如果您同时需要读取和写入文件，则只需请求 `WRITE_EXTERNAL_STORAGE` 权限，因为此权限也隐含了读取权限要求。

**注：**从 Android 4.4 开始，如果您仅仅读取或写入应用的私有文件，则不需要这些权限。如需了解更多信息，请参阅下面有关[保存应用私有文件](#)的部分。

## 检查介质可用性

在使用外部存储执行任何工作之前，应始终调用 `getExternalStorageState()` 以检查介质是否可用。介质可能已装载到计算机，处于缺失、只读或其他某种状态。例如，以下是可用于检查可用性的几种方法：

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
 String state = Environment.getExternalStorageState();
 if (Environment.MEDIA_MOUNTED.equals(state)) {
 return true;
 }
 return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
 String state = Environment.getExternalStorageState();
 if (Environment.MEDIA_MOUNTED.equals(state) ||
 Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
 return true;
 }
 return false;
}
```

`getExternalStorageState()` 方法将返回您可能需要检查的其他状态（例如介质是否处于共享 [连接到计算]、完全缺失、错误移除等状态）。当您的应用需要访问介质时，您可以使用这些状态向用户通知更多信息。

## 保存可与其他应用共享的文件

一般而言，应该将用户可通过您的应用获取的新文件保存到设备上的“公共”位置，以便其他应用能够在其中访问这些文件，并且用户也能轻松地从该设备复制这些文件。执行此操作时，应使用共享的公共目录之一，例如 `Music/`、`Pictures/` 和 `Ringtones/` 等。

要获取表示相应的公共目录的 `File`，请调用 `getExternalStoragePublicDirectory()`，向其传递您需要的目录类型，例如 `DIRECTORY_MUSIC`、`DIRECTORY_PICTURES`、`DIRECTORY_RINGTONES` 或其他类型。通过将您的文件保存到相应的媒体类型目录，系统的媒体扫描程序可以在系统中正确地归类您的文件（例如铃声在系统设置中显示为铃声而不是音乐）。

例如，以下方法在公共图片目录中创建了一个用于新相册的目录：

### 在媒体扫描程序中隐藏您的文件

在您的外部文件目录中包含名为 `.nomedia` 的空文件（注意文件名中的点前缀）。这将阻止媒体扫描程序读取您的媒体文件，并通过 `MediaStore` 内容提供程序将其提供给其他应用。但如果您的文件真正是应用的私有文件，则应该[将其保存在应用私有的目录中](#)。

```
public File getAlbumStorageDir(String albumName) {
 // Get the directory for the user's public pictures directory.
 File file = new File(Environment.getExternalStoragePublicDirectory(
 Environment.DIRECTORY_PICTURES), albumName);
 if (!file.mkdirs()) {
 Log.e(LOG_TAG, "Directory not created");
 }
 return file;
}
```

## 保存应用私有文件

如果您正在处理的文件不适合其他应用使用（例如仅供您的应用使用的图形纹理或音效），则应该通过调用 `getExternalFilesDir()` 来使用外部存储上的私有存储目录。此方法还会采用 `type` 参数指定子目录的类型（例如 `DIRECTORY_MOVIES`）。如果您不需要特定的媒体目录，请传递 `null` 以接收应用私有目录的根目录。

从 Android 4.4 开始，读取或写入应用私有目录中的文件不再需要 `READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 权限。因此，您可以通过添加 `maxSdkVersion` 属性来声明，只能在较低版本的 Android 中请求该权限：

```
<manifest ...>
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
 android:maxSdkVersion="18" />
 ...
</manifest>
```

**注：**当用户卸载您的应用时，此目录及其内容将被删除。此外，系统媒体扫描程序不会读取这些目录中的文件，因此不能从 `MediaStore` 内容提供程序访问这些文件。同样，**不应将这些目录** 用于最终属于用户的媒体，例如使用您的应用拍摄或编辑的照片或用户使用您的应用购买的音乐等 — 这些文件应**保存在公共目录中**。

有时，已分配某个内部存储器分区用作外部存储的设备可能还提供了 SD 卡槽。在使用运行 Android 4.3 和更低版本的这类设备时，`getExternalFilesDir()` 方法将仅提供内部分区的访问权限，而您的应用无法读取或写入 SD 卡。不过，从 Android 4.4 开始，可通过调用 `getExternalFilesDirs()` 来同时访问两个位置，该方法将会返回包含各个位置条目的 `File` 数组。数组中的第一个条目被视为外部主存储；除非该位置已满或不可用，否则应该使用该位置。如果您希望在支持 Android 4.3 和更低版本的同时访问两个可能的位置，请使用**支持库**中的静态方法 `ContextCompat.getExternalFilesDirs()`。在 Android 4.3 和更低版本中，此方法也会返回一个 `File` 数组，但其中始终仅包含一个条目。

**注意** 尽管 `MediaStore` 内容提供程序不能访问 `getExternalFilesDir()` 和 `getExternalFilesDirs()` 所提供的目录，但其他具有 `READ_EXTERNAL_STORAGE` 权限的应用仍可访问外部存储上的所有文件，包括上述文件。如果您需要完全限制您的文件的访问权限，则应该转而将您的文件写入到**内部存储**。

## 保存缓存文件

要打开表示应该将缓存文件保存到的外部存储目录的 `File`，请调用 `getExternalCacheDir()`。如果用户卸载您的应用，这些文件也会被自动删除。

与前述 `ContextCompat.getExternalFilesDirs()` 相似，您也可以通过调用 `ContextCompat.getExternalCacheDirs()` 来访问辅助外部存储（如果可用）上的缓存目录。

**提示：**为节省文件空间并保持应用性能，您应该在应用的整个生命周期内仔细管理您的缓存文件并移除其中不再需要的文件，这一点非常重要。

## 使用数据库

Android 提供了对 `SQLite` 数据库的完全支持。应用中的任何类（不包括应用外部的类）均可按名称访问您所创建的任何数据库。

创建新 `SQLite` 数据库的推荐方法是创建 `SQLiteOpenHelper` 的子类并覆盖 `onCreate()` 方法，在此方法中，您可以执行 `SQLite` 命令以创建数据库中的表。例如：

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

 private static final int DATABASE_VERSION = 2;
 private static final String DICTIONARY_TABLE_NAME = "dictionary";
 private static final String DICTIONARY_TABLE_CREATE =
 "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
 KEY_WORD + " TEXT, " +
 KEY_DEFINITION + " TEXT);";

 DictionaryOpenHelper(Context context) {
 super(context, DATABASE_NAME, null, DATABASE_VERSION);
 }

 @Override
 public void onCreate(SQLiteDatabase db) {
 db.execSQL(DICTIONARY_TABLE_CREATE);
 }
}
```

然后您可以使用已定义的构造函数获取 `SQLiteOpenHelper` 实现的实例。要从数据库执行写入和读取操作，请分别调用 `getWritableDatabase()` 和 `getReadableDatabase()`。二者都会返回一个表示数据库的 `SQLiteDatabase` 对象，并提供用于 SQLite 操作的方法。

您可以使用 `SQLiteDatabase query()` 方法来执行 SQLite 查询，这些方法可接受各种查询参数，例如要查询的表、投影、选择、列、分组和其他参数。对于复杂的查询，例如需要列别名的查询，应该使用 `SQLQueryBuilder`，它将提供多种便捷的方法来构建查询。

每个 SQLite 查询都会返回一个指向该查询找到的所有行的 `Cursor`。您始终可以使用 `Cursor` 机制来浏览数据库查询结果，以及读取行和列。

如需演示 Android 中的 SQLite 数据库使用方法的示例应用，请参阅 [记事本](#) 和 [可搜索字典](#) 应用。

Android 没有实施标准 SQLite 概念之外的任何限制。我们推荐包含一个可用作唯一 ID 的自动增量值关键字段，以便快速查找记录。私有数据不要求这样做，但如果您实现了一个 [内容提供程序](#)，则必须包含使用 `BaseColumns._ID` 常量的唯一 ID。

## 数据库调试

Android SDK 包含一项 `sqlite3` 数据库工具，利用此工具可以浏览表内容，运行 SQL 命令，以及在 SQLite 数据库上执行其他实用功能。请参阅[从远程 shell 检查 sqlite3 数据库](#)，了解如何运行此工具。

## 使用网络连接

您可以使用网络（如果可用）来存储和检索有关您自己的网络服务的数据。要执行网络操作，请使用以下包中的类：

- `java.net.*`
- `android.net.*`

# Restoring User Data on New Devices

In this document

- [Preserving settings data](#)
- [Choosing a backup approach for app settings](#)

Users often invest significant time and effort creating an identity, adding data and customizing settings and preferences within your app. Preserving this data and personalization for users when they upgrade to a new device or replace a broken one is an important part of ensuring a great user experience. This section covers techniques for backing up data to the cloud and provides information on migrating identity, app data, settings data, and permissions for returning users.

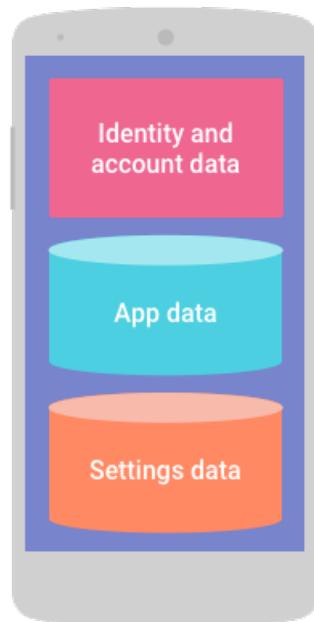
Make sure you minimize the number of steps required for a user to pick up where they left off on their previous device. You can gain a number of potential benefits by ensuring a great user experience for returning users on new devices:

- Reduce user frustration.
- Increase login-rate.
- Reduce support calls.
- Minimize user attrition.
- Sustain user engagement.
- Increase user retention rate.

You can help maintain existing user engagement on a new device by providing a seamless login experience. You can integrate [Smart Lock for Passwords](#) into your Android app to restore user sign-ins on a device. Smart Lock for Passwords supports saving both username-password credentials and federated identity provider credentials.

App data may include user-generated content, such as text, images, and other media. For restoring app data, see [Transferring Data Using Sync Adapters](#) or [Google Drive Android API](#). You can use either approach to synchronize app data between Android-powered devices and save data which you'd like to use during the normal app lifecycle. You can also use either approach to restore a returning user's data onto a new device.

Make sure you also back up and restore settings data to preserve a returning user's personalized preferences on a new device. You can restore settings data even if a user doesn't log in to your app. You can back up settings that a user explicitly sets in your app's UI, as well as transparent data, such as a flag indicating whether a user has seen a setup wizard. For a description of the settings to back up, see [Preserving settings data](#). For a comparison of the available backup solutions for settings data, see [Choosing a backup solution for app settings](#).



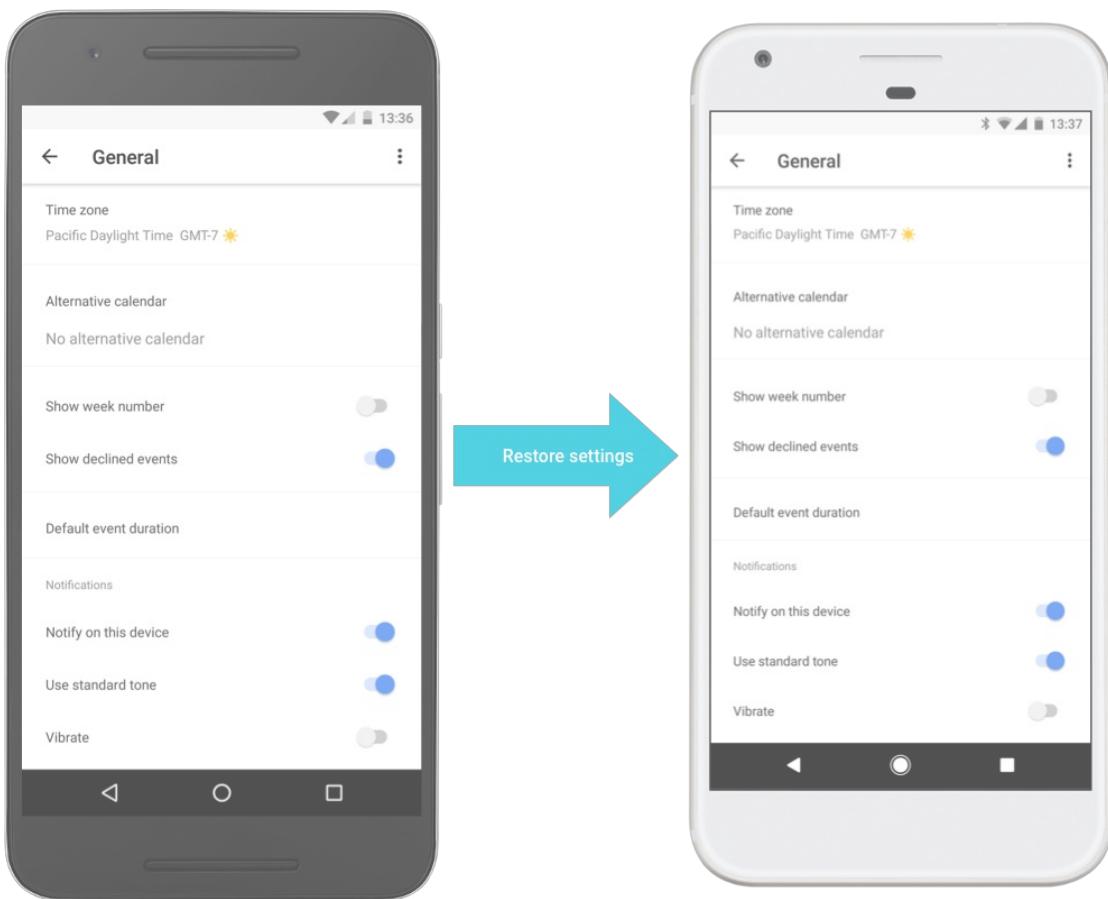
**Figure 1.** Make sure you restore identity, app data, and settings data for users returning to your app.

## Preserving settings data

To preserve as much of an existing user's experience on a new device as possible, make sure you back up the following user settings:

**Note:** Any permissions a user grants to your app are automatically backed up and restored by the system on devices running Android 7.0 (API 24) or newer. However, if a user uninstalls your app, then the system clears any granted permission and the user must grant them again.

- Any settings modified by the user, for example using a [PreferenceScreen](#).
- Whether the user has turned notification and ringtones on or off.
- Boolean flags which indicate if the user has seen welcome screens or introductory tooltips.



**Figure 2.** Restoring settings on new devices ensures a great user experience.

You must not back up URLs for ringtones because URLs are unstable. In some cases a restoration to a new mobile device may result in a URI which points to no ringtone, or a different ringtone from the one intended. Instead, you can back up ringtones using either their title or a hash of the ringtone.

## Choosing a backup approach for app settings

Android provides two ways for apps to back up their data to the cloud: [Auto Backup for Apps](#) and [Key/Value Backup](#). Auto Backup, which is available starting Android 6.0 (API level 23), preserves data by uploading it to the user's Google Drive account. Auto Backup [includes files](#) in most of the directories that are assigned to your app by the system. Auto Backup can store up to 25 MB of file-based data per app. The Key/Value Backup feature (formerly known as the Backup API and the Android Backup Service) preserves settings data in the form of key/value pairs by uploading it to the [Android Backup Service](#).

Generally, we recommend Auto Backup because it requires no work to implement. Apps that target Android 6.0 (API level 23) or higher are automatically enabled for Auto Backup. The Auto Backup feature is a file-based approach to backing up app data. While Auto Backup is simple to implement, you may consider using the Key/Value Backup feature if you have more specific needs for backing up data.

**Note:** If your app doesn't have a backup mechanism for app contents and the size of your app contents is unlikely to exceed the 25 MB limit, then Auto Backup may be sufficient for your needs.

The following table describes some of the key differences between Key/Value Backup and Auto Backup:

Category	Key/Value Backup	Auto Backup
Android API level	Available in API 8, Android 2.2 and higher.	Available in API 23, Android 6.0 and higher.
Implementation	Apps must implement a <a href="#">BackupAgent</a> . The backup	By default, Auto Backup includes almost all of the app's files.

	agent defines what data to back up and how to restore data.	You can use XML to <a href="#">include and exclude files</a> . Under the hood, Auto Backup relies on a backup agent that is bundled into the SDK.
Frequency	Apps must issue a request when there is data that is ready to be backed up. Requests from multiple apps are batched and executed every few hours.	Backups happen automatically roughly once a day.
Transmission	Backup data can be transmitted via Wi-Fi or cellular data.	Backup data is transmitted only via Wi-Fi. If the device is never connected to a Wi-Fi network, then Auto Backup never occurs.
App shut down	Apps are not shut down during backup.	The system shuts down the app during backup.
Backup storage	Backup data is stored in <a href="#">Android Backup Service</a> and limited to 5MB per app. Google treats this data as personal information in accordance with Google's <a href="#">Privacy Policy</a> .	Backup data is stored in the user's Google Drive limited to 25MB per app. Google treats this data as personal information in accordance with Google's <a href="#">Privacy Policy</a> .
User login	Doesn't require a user to be logged into your app. The user must be logged into the device with a Google account.	Doesn't require a user to be logged into your app. The user must be logged into the device with a Google account.
API	Related API methods are entity-based: <ul style="list-style-type: none"> <li>• <a href="#">onBackup()</a></li> <li>• <a href="#">onRestore()</a></li> </ul>	Related API methods are file-based: <ul style="list-style-type: none"> <li>• <a href="#">onFullBackup()</a></li> <li>• <a href="#">onRestoreFile()</a></li> </ul>

**Note:** If Wi-Fi isn't available, Key/Value Backup may use mobile data. Key/Value Backup is therefore typically not suitable for app data contents, such as media, downloaded files, and caches, unless the amount of data is very small.

# App Install Location

## Quickview

- You can allow your application to install on the device's external storage.
- Some types of applications should **not** allow installation on the external storage.
- Installing on the external storage is ideal for large applications that are not tightly integrated with the system (most commonly, games).

## In this document

- [Backward Compatibility](#)
- [Applications That Should NOT Install on External Storage](#)
- [Applications That Should Install on External Storage](#)

## See also

- [`<manifest>`](#)

Beginning with API Level 8, you can allow your application to be installed on the external storage (for example, the device's SD card). This is an optional feature you can declare for your application with the `android:installLocation` manifest attribute. If you do *not* declare this attribute, your application will be installed on the internal storage only and it cannot be moved to the external storage.

To allow the system to install your application on the external storage, modify your manifest file to include the `android:installLocation` attribute in the `<manifest>` element, with a value of either "`preferExternal`" or "`auto`". For example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 android:installLocation="preferExternal"
 ... >
```

If you declare "`preferExternal`", you request that your application be installed on the external storage, but the system does not guarantee that your application will be installed on the external storage. If the external storage is full, the system will install it on the internal storage. The user can also move your application between the two locations.

If you declare "`auto`", you indicate that your application may be installed on the external storage, but you don't have a preference of install location. The system will decide where to install your application based on several factors. The user can also move your application between the two locations.

When your application is installed on the external storage:

- There is no effect on the application performance so long as the external storage is mounted on the device.
- The `.apk` file is saved on the external storage, but all private user data, databases, optimized `.dex` files, and extracted native code are saved on the internal device memory.
- The unique container in which your application is stored is encrypted with a randomly generated key that can be decrypted only by the device that originally installed it. Thus, an application installed on an SD card works for only one device.
- The user can move your application to the internal storage through the system settings.

**Warning:** When the user enables USB mass storage to share files with a computer or unmounts the SD card via the system settings, the external storage is unmounted from the device and all applications running on the external storage are immediately killed.

## Backward Compatibility

The ability for your application to install on the external storage is a feature available only on devices running API Level 8 (Android 2.2) or greater. Existing applications that were built prior to API Level 8 will always install on the internal storage and cannot be moved to the external storage (even on devices with API Level 8). However, if your application is designed to support an API Level *lower than* 8, you can choose to support this feature for devices with API Level 8 or greater and still be compatible with devices using an API Level lower than 8.

To allow installation on external storage and remain compatible with versions lower than API Level 8:

1. Include the `android:installLocation` attribute with a value of "auto" or "preferExternal" in the `<manifest>` element.
2. Leave your `android:minSdkVersion` attribute as is (something *less than* "8") and be certain that your application code uses only APIs compatible with that level.
3. In order to compile your application, change your build target to API Level 8. This is necessary because older Android libraries don't understand the `android:installLocation` attribute and will not compile your application when it's present.

When your application is installed on a device with an API Level lower than 8, the `android:installLocation` attribute is ignored and the application is installed on the internal storage.

**Caution:** Although XML markup such as this will be ignored by older platforms, you must be careful not to use programming APIs introduced in API Level 8 while your `minSdkVersion` is less than "8", unless you perform the work necessary to provide backward compatibility in your code.

## Applications That Should NOT Install on External Storage

When the user enables USB mass storage to share files with their computer (or otherwise unmounts or removes the external storage), any application installed on the external storage and currently running is killed. The system effectively becomes unaware of the application until mass storage is disabled and the external storage is remounted on the device. Besides killing the application and making it unavailable to the user, this can break some types of applications in a more serious way. In order for your application to consistently behave as expected, you **should not** allow your application to be installed on the external storage if it uses any of the following features, due to the cited consequences when the external storage is unmounted:

### Services

Your running `Service` will be killed and will not be restarted when external storage is remounted. You can, however, register for the `ACTION_EXTERNAL_APPLICATIONS_AVAILABLE` broadcast Intent, which will notify your application when applications installed on external storage have become available to the system again. At which time, you can restart your Service.

### Alarm Services

Your alarms registered with `AlarmManager` will be cancelled. You must manually re-register any alarms when external storage is remounted.

### Input Method Engines

Your `IME` will be replaced by the default IME. When external storage is remounted, the user can open system settings to enable your IME again.

### Live Wallpapers

Your running `Live Wallpaper` will be replaced by the default Live Wallpaper. When external storage is remounted, the user can select your Live Wallpaper again.

### App Widgets

Your `App Widget` will be removed from the home screen. When external storage is remounted, your App Widget will *not* be available for the user to select until the system resets the home application (usually not until a system reboot).

### Account Managers

Your accounts created with [AccountManager](#) will disappear until external storage is remounted.

#### Sync Adapters

Your [AbstractThreadedSyncAdapter](#) and all its sync functionality will not work until external storage is remounted.

#### Device Administrators

Your [DeviceAdminReceiver](#) and all its admin capabilities will be disabled, which can have unforeseeable consequences for the device functionality, which may persist after external storage is remounted.

#### Broadcast Receivers listening for "boot completed"

The system delivers the [ACTION\\_BOOT\\_COMPLETED](#) broadcast before the external storage is mounted to the device. If your application is installed on the external storage, it can never receive this broadcast.

If your application uses any of the features listed above, you **should not** allow your application to install on external storage. By default, the system *will not* allow your application to install on the external storage, so you don't need to worry about your existing applications. However, if you're certain that your application should never be installed on the external storage, then you should make this clear by declaring [android:installLocation](#) with a value of "[internalOnly](#)". Though this does not change the default behavior, it explicitly states that your application should only be installed on the internal storage and serves as a reminder to you and other developers that this decision has been made.

## Applications That Should Install on External Storage

---

In simple terms, anything that does not use the features listed in the previous section are safe when installed on external storage. Large games are more commonly the types of applications that should allow installation on external storage, because games don't typically provide additional services when inactive. When external storage becomes unavailable and a game process is killed, there should be no visible effect when the storage becomes available again and the user restarts the game (assuming that the game properly saved its state during the normal [Activity lifecycle](#)).

If your application requires several megabytes for the APK file, you should carefully consider whether to enable the application to install on the external storage so that users can preserve space on their internal storage.



# Android 库

本部分介绍了未纳入 Android 框架的几个非常有用的 Android 库。

# Data Binding Library

In this document:

- › [Build Environment](#)
- › [Data Binding Layout Files](#)
  - › [Writing your first set of data binding expressions](#)
  - › [Data Object](#)
  - › [Binding Data](#)
  - › [Event Handling](#)
    - › [Method References](#)
    - › [Listener Bindings](#)
- › [Layout Details](#)
  - › [Imports](#)
  - › [Variables](#)
  - › [Custom Binding Class Names](#)
  - › [Includes](#)
  - › [Expression Language](#)
- › [Data Objects](#)
  - › [Observable Objects](#)
  - › [ObservableFields](#)
  - › [Observable Collections](#)
- › [Generated Binding](#)
  - › [Creating](#)
  - › [Views With IDs](#)
  - › [Variables](#)
  - › [ViewStubs](#)
  - › [Advanced Binding](#)
- › [Attribute Setters](#)
  - › [Automatic Setters](#)
  - › [Renamed Setters](#)
  - › [Custom Setters](#)
- › [Converters](#)
  - › [Object Conversions](#)
  - › [Custom Conversions](#)
- › [Android Studio Support for Data Binding](#)

This document explains how to use the Data Binding Library to write declarative layouts and minimize the glue code necessary to bind your application logic and layouts.

The Data Binding Library offers both flexibility and broad compatibility — it's a support library, so you can use it with all Android platform versions back to **Android 2.1** (API level 7+).

To use data binding, Android Plugin for Gradle **1.5.0-alpha1** or higher is required. See how to [update the Android Plugin for Gradle](#).

# Build Environment

To get started with Data Binding, download the library from the Support repository in the Android SDK manager.

To configure your app to use data binding, add the `dataBinding` element to your `build.gradle` file in the app module.

Use the following code snippet to configure data binding:

```
android {
 ...
 dataBinding {
 enabled = true
 }
}
```

If you have an app module that depends on a library which uses data binding, your app module must configure data binding in its `build.gradle` file as well.

Also, make sure you are using a compatible version of Android Studio. [Android Studio 1.3](#) and later provides support for data binding as described in [Android Studio Support for Data Binding](#).

## Data Binding Layout Files

### Writing your first set of data binding expressions

Data-binding layout files are slightly different and start with a root tag of `layout` followed by a `data` element and a `view` root element. This `view` element is what your root would be in a non-binding layout file. A sample file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
 <data>
 <variable name="user" type="com.example.User"/>
 </data>
 <LinearLayout
 android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.firstName}"/>
 <TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.lastName}"/>
 </LinearLayout>
</layout>
```

The user `variable` within `data` describes a property that may be used within this layout.

```
<variable name="user" type="com.example.User"/>
```

Expressions within the layout are written in the attribute properties using the `"@{ }"` syntax. Here, the `TextView`'s `text` is set to the `firstName` property of `user`:

```
<TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.firstName}"/>
```

## Data Object

Let's assume for now that you have a plain-old Java object (POJO) for User:

```
public class User {
 public final String firstName;
 public final String lastName;
 public User(String firstName, String lastName) {
 this.firstName = firstName;
 this.lastName = lastName;
 }
}
```

This type of object has data that never changes. It is common in applications to have data that is read once and never changes thereafter. It is also possible to use a JavaBeans objects:

```
public class User {
 private final String firstName;
 private final String lastName;
 public User(String firstName, String lastName) {
 this.firstName = firstName;
 this.lastName = lastName;
 }
 public String getFirstName() {
 return this.firstName;
 }
 public String getLastName() {
 return this.lastName;
 }
}
```

From the perspective of data binding, these two classes are equivalent. The expression `@{user.firstName}` used for the TextView's `android:text` attribute will access the `firstName` field in the former class and the `getFirstName()` method in the latter class. Alternatively, it will also be resolved to `firstName()` if that method exists.

## Binding Data

By default, a Binding class will be generated based on the name of the layout file, converting it to Pascal case and suffixing "Binding" to it. The above layout file was `main_activity.xml` so the generate class was `MainActivityBinding`. This class holds all the bindings from the layout properties (e.g. the `user` variable) to the layout's Views and knows how to assign values for the binding expressions. The easiest means for creating the bindings is to do it while inflating:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 MainActivityBinding binding = DataBindingUtil.setContentView(this, R.layout.main_activity);
 User user = new User("Test", "User");
 binding.setUser(user);
}
```

You're done! Run the application and you'll see Test User in the UI. Alternatively, you can get the view via:

```
MainActivityBinding binding = MainActivityBinding.inflate(LayoutInflater());
```

If you are using data binding items inside a ListView or RecyclerView adapter, you may prefer to use:

```
ListItemBinding binding = ListItemBinding.inflate(layoutInflater, ViewGroup, false);
//or
ListItemBinding binding = DataBindingUtil.inflate(layoutInflater, R.layout.list_item, ViewGroup, false);
```

## Event Handling

Data Binding allows you to write expressions handling events that are dispatched from the views (e.g. onClick). Event attribute names are governed by the name of the listener method with a few exceptions. For example, `View.OnLongClickListener` has a method `onLongClick()`, so the attribute for this event is `android:onClick`. There are two ways to handle an event.

- **Method References:** In your expressions, you can reference methods that conform to the signature of the listener method. When an expression evaluates to a method reference, Data Binding wraps the method reference and owner object in a listener, and sets that listener on the target view. If the expression evaluates to null, Data Binding does not create a listener and sets a null listener instead.
- **Listener Bindings:** These are lambda expressions that are evaluated when the event happens. Data Binding always creates a listener, which it sets on the view. When the event is dispatched, the listener evaluates the lambda expression.

## Method References

Events can be bound to handler methods directly, similar to the way `android:onClick` can be assigned to a method in an Activity. One major advantage compared to the `View#onClick` attribute is that the expression is processed at compile time, so if the method does not exist or its signature is not correct, you receive a compile time error.

The major difference between Method References and Listener Bindings is that the actual listener implementation is created when the data is bound, not when the event is triggered. If you prefer to evaluate the expression when the event happens, you should use [listener binding](#).

To assign an event to its handler, use a normal binding expression, with the value being the method name to call. For example, if your data object has two methods:

```
public class MyHandlers {
 public void onClickFriend(View view) { ... }
}
```

The binding expression can assign the click listener for a View:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
 <data>
 <variable name="handlers" type="com.example.MyHandlers"/>
 <variable name="user" type="com.example.User"/>
 </data>
 <LinearLayout
 android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.firstName}"
 android:onClick="@{handlers::onClickFriend}"/>
 </LinearLayout>
</layout>
```

Note that the signature of the method in the expression must exactly match the signature of the method in the Listener object.

## Listener Bindings

Listener Bindings are binding expressions that run when an event happens. They are similar to method references, but they let you run arbitrary data binding expressions. This feature is available with Android Gradle Plugin for Gradle version 2.0 and later.

In method references, the parameters of the method must match the parameters of the event listener. In Listener Bindings, only your return value must match the expected return value of the listener (unless it is expecting void). For example, you can have a presenter class that has the following method:

```
public class Presenter {
 public void onSaveClick(Task task){}
}
```

Then you can bind the click event to your class as follows:

```

<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
 <data>
 <variable name="task" type="com.android.example.Task" />
 <variable name="presenter" type="com.android.example.Presenter" />
 </data>
 <LinearLayout android:layout_width="match_parent" android:layout_height="match_parent">
 <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
 android:onClick="@{() -> presenter.onSaveClick(task)}" />
 </LinearLayout>
</layout>

```

Listeners are represented by lambda expressions that are allowed only as root elements of your expressions. When a callback is used in an expression, Data Binding automatically creates the necessary listener and registers for the event. When the view fires the event, Data Binding evaluates the given expression. As in regular binding expressions, you still get the null and thread safety of Data Binding while these listener expressions are being evaluated.

Note that in the example above, we haven't defined the `view` parameter that is passed into `onClick(android.view.View)`. Listener bindings provide two choices for listener parameters: you can either ignore all parameters to the method or name all of them. If you prefer to name the parameters, you can use them in your expression. For example, the expression above could be written as:

```
android:onClick="@{(view) -> presenter.onSaveClick(task)}"
```

Or if you wanted to use the parameter in the expression, it could work as follows:

```

public class Presenter {
 public void onSaveClick(View view, Task task){}
}

```

```
android:onClick="@{(theView) -> presenter.onSaveClick(theView, task)}"
```

You can use a lambda expression with more than one parameter:

```

public class Presenter {
 public void onCompletedChanged(Task task, boolean completed){}
}

```

```
<CheckBox android:layout_width="wrap_content" android:layout_height="wrap_content"
 android:onCheckedChanged="@{(cb, isChecked) -> presenter.completeChanged(task, isChecked)}" />
```

If the event you are listening to returns a value whose type is not `void`, your expressions must return the same type of value as well. For example, if you want to listen for the long click event, your expression should return `boolean`.

```

public class Presenter {
 public boolean onLongClick(View view, Task task){}
}

```

```
android:onLongClick="@{(theView) -> presenter.onLongClick(theView, task)}"
```

If the expression cannot be evaluated due to `null` objects, Data Binding returns the default Java value for that type. For example, `null` for reference types, `0` for `int`, `false` for `boolean`, etc.

If you need to use an expression with a predicate (e.g. ternary), you can use `void` as a symbol.

```
android:onClick="@{(v) -> v.isVisible() ? doSomething() : void}"
```

## Avoid Complex Listeners

Listener expressions are very powerful and can make your code very easy to read. On the other hand, listeners containing complex

expressions make your layouts hard to read and unmaintainable. These expressions should be as simple as passing available data from your UI to your callback method. You should implement any business logic inside the callback method that you invoked from the listener expression.

Some specialized click event handlers exist and they need an attribute other than `android:onClick` to avoid a conflict. The following attributes have been created to avoid such conflicts:

Class	Listener Setter	Attribute
<code>SearchView</code>	<code>setOnSearchClickListener(View.OnClickListener)</code>	<code>android:onSearchClick</code>
<code>ZoomControls</code>	<code>setOnZoomInClickListener(View.OnClickListener)</code>	<code>android:onZoomIn</code>
<code>ZoomControls</code>	<code>setOnZoomOutClickListener(View.OnClickListener)</code>	<code>android:onZoomOut</code>

## Layout Details

### Imports

Zero or more `import` elements may be used inside the `data` element. These allow easy reference to classes inside your layout file, just like in Java.

```
<data>
 <import type="android.view.View"/>
</data>
```

Now, `View` may be used within your binding expression:

```
<TextView
 android:text="@{user.lastName}"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:visibility="@{user.isAdult ? View.VISIBLE : View.GONE}"/>
```

When there are class name conflicts, one of the classes may be renamed to an "alias."

```
<import type="android.view.View"/>
<import type="com.example.real.estate.View"
 alias="Vista"/>
```

Now, `Vista` may be used to reference the `com.example.real.estate.View` and `View` may be used to reference `android.view.View` within the layout file. Imported types may be used as type references in variables and expressions:

```
<data>
 <import type="com.example.User"/>
 <import type="java.util.List"/>
 <variable name="user" type="User"/>
 <variable name="userList" type="List<User>"/>
</data>
```

**Note:** Android Studio does not yet handle imports so the autocomplete for imported variables may not work in your IDE. Your application will still compile fine and you can work around the IDE issue by using fully qualified names in your variable definitions.

```
<TextView
 android:text="@{((User)(user.connection)).lastName}"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

Imported types may also be used when referencing static fields and methods in expressions:

```
<data>
 <import type="com.example.MyStringUtils"/>
 <variable name="user" type="com.example.User"/>
</data>
...
<TextView
 android:text="@{MyStringUtils.capitalize(user.lastName)}"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

Just as in Java, `java.lang.*` is imported automatically.

## Variables

Any number of `variable` elements may be used inside the `data` element. Each `variable` element describes a property that may be set on the layout to be used in binding expressions within the layout file.

```
<data>
 <import type="android.graphics.drawable.Drawable"/>
 <variable name="user" type="com.example.User"/>
 <variable name="image" type="Drawable"/>
 <variable name="note" type="String"/>
</data>
```

The variable types are inspected at compile time, so if a variable implements `Observable` or is an `observable collection`, that should be reflected in the type. If the variable is a base class or interface that does not implement the `Observable*` interface, the variables will **not be observed!**

When there are different layout files for various configurations (e.g. landscape or portrait), the variables will be combined. There must not be conflicting variable definitions between these layout files.

The generated binding class will have a setter and getter for each of the described variables. The variables will take the default Java values until the setter is called — `null` for reference types, `0` for `int`, `false` for `boolean`, etc.

A special variable named `context` is generated for use in binding expressions as needed. The value for `context` is the `Context` from the root View's `getContext()`. The `context` variable will be overridden by an explicit variable declaration with that name.

## Custom Binding Class Names

By default, a Binding class is generated based on the name of the layout file, starting it with upper-case, removing underscores (`_`) and capitalizing the following letter and then suffixing "Binding". This class will be placed in a databinding package under the module package. For example, the layout file `contact_item.xml` will generate `ContactItemBinding`. If the module package is `com.example.my.app`, then it will be placed in `com.example.my.app.databinding`.

Binding classes may be renamed or placed in different packages by adjusting the `class` attribute of the `data` element. For example:

```
<data class="ContactItem">
 ...
</data>
```

This generates the binding class as `ContactItem` in the databinding package in the module package. If the class should be generated in a different package within the module package, it may be prefixed with `".":`

```
<data class=".ContactItem">
 ...
</data>
```

In this case, `ContactItem` is generated in the module package directly. Any package may be used if the full package is provided:

```
<data class="com.example.ContactItem">
 ...
</data>
```

## Includes

Variables may be passed into an included layout's binding from the containing layout by using the application namespace and the variable name in an attribute:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:bind="http://schemas.android.com/apk/res-auto">
 <data>
 <variable name="user" type="com.example.User"/>
 </data>
 <LinearLayout
 android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <include layout="@layout/name"
 bind:user="@{user}"/>
 <include layout="@layout/contact"
 bind:user="@{user}"/>
 </LinearLayout>
</layout>
```

Here, there must be a `user` variable in both the `name.xml` and `contact.xml` layout files.

Data binding does not support include as a direct child of a merge element. For example, **the following layout is not supported:**

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:bind="http://schemas.android.com/apk/res-auto">
 <data>
 <variable name="user" type="com.example.User"/>
 </data>
 <merge>
 <include layout="@layout/name"
 bind:user="@{user}"/>
 <include layout="@layout/contact"
 bind:user="@{user}"/>
 </merge>
</layout>
```

## Expression Language

### Common Features

The expression language looks a lot like a Java expression. These are the same:

- Mathematical `+` `-` `/` `*` `%`
- String concatenation `+`
- Logical `&&` `||`
- Binary `&` `|` `^`
- Unary `+` `-` `!` `~`
- Shift `>>` `>>>` `<<`
- Comparison `==` `>` `<` `>=` `<=`
- `instanceof`

- Grouping ()
- Literals - character, String, numeric, `null`
- Cast
- Method calls
- Field access
- Array access []
- Ternary operator ?:

Examples:

```
android:text="@{String.valueOf(index + 1)}"
android:visibility="@{age < 13 ? View.GONE : View.VISIBLE}"
android:transitionName='@{"image_" + id}'
```

## Missing Operations

A few operations are missing from the expression syntax that you can use in Java.

- `this`
- `super`
- `new`
- Explicit generic invocation

## Null Coalescing Operator

The null coalescing operator (`??`) chooses the left operand if it is not null or the right if it is null.

```
android:text="@{user.displayName ?? user.lastName}"
```

This is functionally equivalent to:

```
android:text="@{user.displayName != null ? user.displayName : user.lastName}"
```

## Property Reference

The first was already discussed in the [Writing your first data binding expressions](#) above: short form JavaBean references. When an expression references a property on a class, it uses the same format for fields, getters, and ObservableFields.

```
android:text="@{user.lastName}"
```

## Avoiding NullPointerException

Generated data binding code automatically checks for nulls and avoid null pointer exceptions. For example, in the expression `@{user.name}`, if `user` is null, `user.name` will be assigned its default value (null). If you were referencing `user.age`, where `age` is an `int`, then it would default to 0.

## Collections

Common collections: arrays, lists, sparse lists, and maps, may be accessed using the [] operator for convenience.

```

<data>
 <import type="android.util.SparseArray"/>
 <import type="java.util.Map"/>
 <import type="java.util.List"/>
 <variable name="list" type="List<String>"/>
 <variable name="sparse" type="SparseArray<String>"/>
 <variable name="map" type="Map<String, String>"/>
 <variable name="index" type="int"/>
 <variable name="key" type="String"/>
</data>
...
 android:text="@{list[index]}"
...
 android:text="@{sparse[index]}"
...
 android:text="@{map[key]}"

```

## String Literals

When using single quotes around the attribute value, it is easy to use double quotes in the expression:

```
 android:text='@{map["firstName"]}'
```

It is also possible to use double quotes to surround the attribute value. When doing so, String literals should either use the ' or back quote (`).

```
 android:text="@{map['firstName']}"
 android:text="@{map['firstName']}"
```

## Resources

It is possible to access resources as part of expressions using the normal syntax:

```
 android:padding="@{large? @dimen/largePadding : @dimen/smallPadding}"
```

Format strings and plurals may be evaluated by providing parameters:

```
 android:text="@{@string/nameFormat(firstName, lastName)}"
 android:text="@{@plurals/banana(bananaCount)}"
```

When a plural takes multiple parameters, all parameters should be passed:

```

 Have an orange
 Have %d oranges

 android:text="@{@plurals/orange(orangeCount, orangeCount)}"
```

Some resources require explicit type evaluation.

Type	Normal Reference	Expression Reference
String[]	@array	@stringArray
int[]	@array	@intArray
TypedArray	@array	@typedArray
Animator	@animator	@animator
StateListAnimator	@animator	@stateListAnimator
color int	@color	@color
ColorStateList	@color	@colorStateList

# Data Objects

Any plain old Java object (POJO) may be used for data binding, but modifying a POJO will not cause the UI to update. The real power of data binding can be used by giving your data objects the ability to notify when data changes. There are three different data change notification mechanisms, [Observable objects](#), [observable fields](#), and [observable collections](#).

When one of these observable data object is bound to the UI and a property of the data object changes, the UI will be updated automatically.

## Observable Objects

A class implementing the [Observable](#) interface will allow the binding to attach a single listener to a bound object to listen for changes of all properties on that object.

The [Observable](#) interface has a mechanism to add and remove listeners, but notifying is up to the developer. To make development easier, a base class, [BaseObservable](#), was created to implement the listener registration mechanism. The data class implementer is still responsible for notifying when the properties change. This is done by assigning a [Bindable](#) annotation to the getter and notifying in the setter.

```
private static class User extends BaseObservable {
 private String firstName;
 private String lastName;
 @Bindable
 public String getFirstName() {
 return this.firstName;
 }
 @Bindable
 public String getLastName() {
 return this.lastName;
 }
 public void setFirstName(String firstName) {
 this.firstName = firstName;
 notifyPropertyChanged(BR.firstName);
 }
 public void setLastName(String lastName) {
 this.lastName = lastName;
 notifyPropertyChanged(BR.lastName);
 }
}
```

The [Bindable](#) annotation generates an entry in the BR class file during compilation. The BR class file will be generated in the module package. If the base class for data classes cannot be changed, the [Observable](#) interface may be implemented using the convenient [PropertyChangeRegistry](#) to store and notify listeners efficiently.

## ObservableFields

A little work is involved in creating [Observable](#) classes, so developers who want to save time or have few properties may use [ObservableField](#) and its siblings [ObservableBoolean](#), [ObservableByte](#), [ObservableChar](#), [ObservableShort](#), [ObservableInt](#), [ObservableLong](#), [ObservableFloat](#), [ObservableDouble](#), and [ObservableParcelble](#). [ObservableFields](#) are self-contained observable objects that have a single field. The primitive versions avoid boxing and unboxing during access operations. To use, create a public final field in the data class:

```
private static class User {
 public final ObservableField<String> firstName =
 new ObservableField<>();
 public final ObservableField<String> lastName =
 new ObservableField<>();
 public final ObservableInt age = new ObservableInt();
}
```

That's it! To access the value, use the set and get accessor methods:

```
user.firstName.set("Google");
int age = user.age.get();
```

## Observable Collections

Some applications use more dynamic structures to hold data. Observable collections allow keyed access to these data objects.

`ObservableArrayMap` is useful when the key is a reference type, such as String.

```
ObservableArrayMap<String, Object> user = new ObservableArrayMap<>();
user.put("firstName", "Google");
user.put("lastName", "Inc.");
user.put("age", 17);
```

In the layout, the map may be accessed through the String keys:

```
<data>
 <import type="android.databinding.ObservableMap"/>
 <variable name="user" type="ObservableMap<String, Object>"/>
</data>
...
<TextView
 android:text='@{user["lastName"]}'
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
<TextView
 android:text='@{String.valueOf(1 + (Integer)user["age"])}'
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

`ObservableArrayList` is useful when the key is an integer:

```
ObservableArrayList<Object> user = new ObservableArrayList<>();
user.add("Google");
user.add("Inc.");
user.add(17);
```

In the layout, the list may be accessed through the indices:

```
<data>
 <import type="android.databinding.ObservableList"/>
 <import type="com.example.my.app.Fields"/>
 <variable name="user" type="ObservableList<Object>"/>
</data>
...
<TextView
 android:text='@{user[Fields.LAST_NAME]}'
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
<TextView
 android:text='@{String.valueOf(1 + (Integer)user[Fields.AGE])}'
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

## Generated Binding

The generated binding class links the layout variables with the Views within the layout. As discussed earlier, the name and package of the Binding may be [customized](#). The Generated binding classes all extend `ViewDataBinding`.

## Creating

The binding should be created soon after inflation to ensure that the View hierarchy is not disturbed prior to binding to the Views with expressions within the layout. There are a few ways to bind to a layout. The most common is to use the static methods on the Binding class. The inflate method inflates the View hierarchy and binds to it all in one step. There is a simpler version that only takes a [LayoutInflator](#) and one that takes a [ViewGroup](#) as well:

```
MyLayoutBinding binding = MyLayoutBinding.inflate(layoutInflater);
MyLayoutBinding binding = MyLayoutBinding.inflate(layoutInflater, viewGroup, false);
```

If the layout was inflated using a different mechanism, it may be bound separately:

```
MyLayoutBinding binding = MyLayoutBinding.bind(viewRoot);
```

Sometimes the binding cannot be known in advance. In such cases, the binding can be created using the [DataBindingUtil](#) class:

```
 ViewDataBinding binding = DataBindingUtil.inflate(LayoutInflater, layoutId,
 parent, attachToParent);
ViewDataBinding binding = DataBindingUtil.bindTo(viewRoot, layoutId);
```

## Views With IDs

A public final field will be generated for each View with an ID in the layout. The binding does a single pass on the View hierarchy, extracting the Views with IDs. This mechanism can be faster than calling `findViewById` for several Views. For example:

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
 <data>
 <variable name="user" type="com.example.User"/>
 </data>
 <LinearLayout
 android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.firstName}"
 android:id="@+id/firstName"/>
 <TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.lastName}"
 android:id="@+id/lastName"/>
 </LinearLayout>
</layout>
```

Will generate a binding class with:

```
public final TextView firstName;
public final TextView lastName;
```

IDs are not nearly as necessary as without data binding, but there are still some instances where access to Views are still necessary from code.

## Variables

Each variable will be given accessor methods.

```
<data>
 <import type="android.graphics.drawable.Drawable"/>
 <variable name="user" type="com.example.User"/>
 <variable name="image" type="Drawable"/>
 <variable name="note" type="String"/>
</data>
```

will generate setters and getters in the binding:

```
public abstract com.example.User getUser();
public abstract void setUser(com.example.User user);
public abstract Drawable getImage();
public abstract void setImage(Drawable image);
public abstract String getNote();
public abstract void setNote(String note);
```

## ViewStubs

[ViewStubs](#) are a little different from normal Views. They start off invisible and when they either are made visible or are explicitly told to inflate, they replace themselves in the layout by inflating another layout.

Because the [ViewStub](#) essentially disappears from the View hierarchy, the View in the binding object must also disappear to allow collection.

Because the Views are final, a [ViewStubProxy](#) object takes the place of the [ViewStub](#), giving the developer access to the ViewStub when it exists and also access to the inflated View hierarchy when the [ViewStub](#) has been inflated.

When inflating another layout, a binding must be established for the new layout. Therefore, the [ViewStubProxy](#) must listen to the [ViewStub](#)'s [ViewStub.OnInflateListener](#) and establish the binding at that time. Since only one can exist, the [ViewStubProxy](#) allows the developer to set an [OnInflateListener](#) on it that it will call after establishing the binding.

## Advanced Binding

### Dynamic Variables

At times, the specific binding class won't be known. For example, a [RecyclerView.Adapter](#) operating against arbitrary layouts won't know the specific binding class. It still must assign the binding value during the [onBindViewHolder\(VH, int\)](#).

In this example, all layouts that the RecyclerView binds to have an "item" variable. The [BindingHolder](#) has a [getBinding](#) method returning the  [ViewDataBinding](#) base.

```
public void onBindViewHolder(BindingHolder holder, int position) {
 final T item = mItems.get(position);
 holder.getBinding().setVariable(BR.item, item);
 holder.getBinding().executePendingBindings();
}
```

### Immediate Binding

When a variable or observable changes, the binding will be scheduled to change before the next frame. There are times, however, when binding must be executed immediately. To force execution, use the [executePendingBindings\(\)](#) method.

### Background Thread

You can change your data model in a background thread as long as it is not a collection. Data binding will localize each variable / field while evaluating to avoid any concurrency issues.

## Attribute Setters

Whenever a bound value changes, the generated binding class must call a setter method on the View with the binding expression. The data binding framework has ways to customize which method to call to set the value.

## Automatic Setters

For an attribute, data binding tries to find the method `setAttribute`. The namespace for the attribute does not matter, only the attribute name itself.

For example, an expression associated with `TextView`'s attribute `android:text` will look for a `setText(String)`. If the expression returns an int, data binding will search for a `setText(int)` method. Be careful to have the expression return the correct type, casting if necessary. Note that data binding will work even if no attribute exists with the given name. You can then easily "create" attributes for any setter by using data binding. For example, support `DrawerLayout` doesn't have any attributes, but plenty of setters. You can use the automatic setters to use one of these.

```
<android.support.v4.widget.DrawerLayout
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 app:scrimColor="@{color/scrim}"
 app:drawerListener="@{fragment.drawerListener}">
```

## Renamed Setters

Some attributes have setters that don't match by name. For these methods, an attribute may be associated with the setter through `BindingMethods` annotation. This must be associated with a class and contains `BindingMethod` annotations, one for each renamed method. For example, the `android:tint` attribute is really associated with `setImageTintList(ColorStateList)`, not `setTint`.

```
@BindingMethods({
 @BindingMethod(type = "android.widget.ImageView",
 attribute = "android:tint",
 method = "setImageTintList"),
})
```

It is unlikely that developers will need to rename setters; the android framework attributes have already been implemented.

## Custom Setters

Some attributes need custom binding logic. For example, there is no associated setter for the `android:paddingLeft` attribute. Instead, `setPadding(left, top, right, bottom)` exists. A static binding adapter method with the `BindingAdapter` annotation allows the developer to customize how a setter for an attribute is called.

The android attributes have already had `BindingAdapters` created. For example, here is the one for `paddingLeft`:

```
@BindingAdapter("android:paddingLeft")
public static void setPaddingLeft(View view, int padding) {
 view.setPadding(padding,
 view.getPaddingTop(),
 view.getPaddingRight(),
 view.getPaddingBottom());
}
```

Binding adapters are useful for other types of customization. For example, a custom loader can be called off-thread to load an image.

Developer-created binding adapters will override the data binding default adapters when there is a conflict.

You can also have adapters that receive multiple parameters.

```
@BindingAdapter({"bind:imageUrl", "bind:error"})
public static void loadImage(ImageView view, String url, Drawable error) {
 Picasso.with(view.getContext()).load(url).error(error).into(view);
}
```

```
<ImageView app:imageUrl="@{venue.imageUrl}"
 app:error="@{@drawable/venueError}">
```

This adapter will be called if both `imageUrl` and `error` are used for an `ImageView` and `imageUrl` is a string and `error` is a drawable.

- Custom namespaces are ignored during matching.
- You can also write adapters for android namespace.

Binding adapter methods may optionally take the old values in their handlers. A method taking old and new values should have all old values for the attributes come first, followed by the new values:

```
@BindingAdapter("android:paddingLeft")
public static void setPaddingLeft(View view, int oldPadding, int newPadding) {
 if (oldPadding != newPadding) {
 view.setPadding(newPadding,
 view.getPaddingTop(),
 view.getPaddingRight(),
 view.getPaddingBottom());
 }
}
```

Event handlers may only be used with interfaces or abstract classes with one abstract method. For example:

```
@BindingAdapter("android:onLayoutChange")
public static void setOnLayoutChangeListener(View view, View.OnLayoutChangeListener oldValue,
 View.OnLayoutChangeListener newValue) {
 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
 if (oldValue != null) {
 view.removeOnLayoutChangeListener(oldValue);
 }
 if (newValue != null) {
 view.addOnLayoutChangeListener(newValue);
 }
 }
}
```

When a listener has multiple methods, it must be split into multiple listeners. For example, `View.OnAttachStateChangeListener` has two methods: `onViewAttachedToWindow()` and `onViewDetachedFromWindow()`. We must then create two interfaces to differentiate the attributes and handlers for them.

```
@TargetApi(VERSION_CODES.HONEYCOMB_MR1)
public interface OnViewDetachedFromWindow {
 void onViewDetachedFromWindow(View v);
}

@TargetApi(VERSION_CODES.HONEYCOMB_MR1)
public interface OnViewAttachedToWindow {
 void onViewAttachedToWindow(View v);
}
```

Because changing one listener will also affect the other, we must have three different binding adapters, one for each attribute and one for both, should they both be set.

```

@BindingAdapter("android:onViewAttachedToWindow")
public static void setListener(View view, OnViewAttachedToWindow attached) {
 setListener(view, null, attached);
}

@BindingAdapter("android:onViewDetachedFromWindow")
public static void setListener(View view, OnViewDetachedFromWindow detached) {
 setListener(view, detached, null);
}

@BindingAdapter({"android:onViewDetachedFromWindow", "android:onViewAttachedToWindow"})
public static void setListener(View view, final OnViewDetachedFromWindow detach,
 final OnViewAttachedToWindow attach) {
 if (VERSION.SDK_INT >= VERSION_CODES.HONEYCOMB_MR1) {
 final OnAttachStateChangeListener newListener;
 if (detach == null && attach == null) {
 newListener = null;
 } else {
 newListener = new OnAttachStateChangeListener() {
 @Override
 public void onViewAttachedToWindow(View v) {
 if (attach != null) {
 attach.onViewAttachedToWindow(v);
 }
 }

 @Override
 public void onViewDetachedFromWindow(View v) {
 if (detach != null) {
 detach.onViewDetachedFromWindow(v);
 }
 }
 };
 }
 final OnAttachStateChangeListener oldListener = ListenerUtil.trackListener(view,
 newListener, R.id.onAttachStateChangeListener);
 if (oldListener != null) {
 view.removeOnAttachStateChangeListener(oldListener);
 }
 if (newListener != null) {
 view.addOnAttachStateChangeListener(newListener);
 }
 }
}

```

The above example is slightly more complicated than normal because View uses add and remove for the listener instead of a set method for `View.OnAttachStateChangeListener`. The `android.databinding.adapters.ListenerUtil` class helps keep track of the previous listeners so that they may be removed in the Binding Adapter.

By annotating the interfaces `OnViewDetachedFromWindow` and `OnViewAttachedToWindow` with `@TargetApi(VERSION_CODES.HONEYCOMB_MR1)`, the data binding code generator knows that the listener should only be generated when running on Honeycomb MR1 and new devices, the same version supported by `addOnAttachStateChangeListener(View.OnAttachStateChangeListener)`.

## Converters

### Object Conversions

When an Object is returned from a binding expression, a setter will be chosen from the automatic, renamed, and custom setters. The Object will be cast to a parameter type of the chosen setter.

This is a convenience for those using ObservableMaps to hold data. for example:

```
<TextView
 android:text='@{userMap["lastName"]}'
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

The `userMap` returns an Object and that Object will be automatically cast to parameter type found in the setter `setText(CharSequence)`. When there may be confusion about the parameter type, the developer will need to cast in the expression.

## Custom Conversions

Sometimes conversions should be automatic between specific types. For example, when setting the background:

```
<View
 android:background="@{isError ? @color/red : @color/white}"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

Here, the background takes a `Drawable`, but the color is an integer. Whenever a `Drawable` is expected and an integer is returned, the `int` should be converted to a `ColorDrawable`. This conversion is done using a static method with a `BindingConversion` annotation:

```
@BindingConversion
public static ColorDrawable convertColorToDrawable(int color) {
 return new ColorDrawable(color);
}
```

Note that conversions only happen at the setter level, so it is **not allowed** to mix types like this:

```
<View
 android:background="@{isError ? @drawable/error : @color/white}"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>
```

## Android Studio Support for Data Binding

Android Studio supports many of the code editing features for data binding code. For example, it supports the following features for data binding expressions:

- Syntax highlighting
- Flagging of expression language syntax errors
- XML code completion
- References, including [navigation](#) (such as navigate to a declaration) and [quick documentation](#)

**Note:** Arrays and a [generic type](#), such as the `Observable` class, might display errors when there are no errors.

The Preview pane displays default values for data binding expressions if provided. In the following example excerpt of an element from a layout XML file, the Preview pane displays the `PLACEHOLDER` default text value in the `TextView`.

```
<TextView android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@{user.firstName, default=PLACEHOLDER}"/>
```

If you need to display a default value during the design phase of your project, you can also use tools attributes instead of default expression values, as described in [Designtime Layout Attributes](#).

# 支持库

## 本文内容

- › [概览](#)
- › [向后兼容性](#)
- › [支持一般布局模式](#)
- › [支持不同的设备类型](#)
- › [一般实用程序](#)

## 另请参阅

- › [支持库功能](#)
- › [支持库设置](#)
- › [支持库修订历史记录](#)

Android 支持库提供了诸多未内置于框架的功能。这些库提供向后兼容版本的新功能、框架中未包含的实用 UI 元素，以及应用可以利用的一系列实用程序。

## 概览

许多情况下，某项功能可能对应用开发者很有用，但是添加到 Android 框架却并不合适。例如，某个应用可能仅需要用于特定用例的某项功能，如在不同版本的 Android 系统之间顺畅切换。

为了解决这一问题，Android SDK 添加了多个库，这些库统称为 *Android 支持库*。如果应用开发者想要在应用中集成库功能，他们可以添加其中任意一个库。

支持库提供一系列不同的功能：

- 向后兼容版本的框架组件。
- 用于实现建议的 Android 布局模式的 UI 元素。
- 支持不同的设备类型。
- 其他实用程序功能。

## 向后兼容性



**图 1.** 由于此应用使用支持库 UI 元素，即使是在对 Material Design 不提供原生支持的 Android 4.4 上运行，其界面仍符合 Material Design 原则。

支持库可以让在旧版本 Android 平台上运行的应用支持为新版本平台推出的功能。例如，应用在依赖于框架类的 5.0 (API 级别 21) 版本以下的 Android 系统上运行时，将无法显示 Material Design 元素，因为该版本的 Android 框架不支持 Material Design。但是，如果此应用添加了支持库的 [appcompat 库](#)，则可以访问 API 级别 21 中具有的许多功能，其中包括对 Material Design 的支持。因此，您的应用可以在多个平台版本中提供更为一致的体验。

某些情况下，类的支持库版本很大程度上取决于框架提供的功能。因此，如果应用调用其中一个支持类的方法，则支持库的行为将取决于运行应用的 Android 版本。如果框架提供必要的功能，则支持库将通过调用框架执行任务。如果应用在旧版本的 Android 上运行，且框架未显示所需的功能，则支持库自身可能会尝试提供功能或什么都不做。无论是哪一种情形，应用通常都不需要检查其在哪一版本的 Android 上运行，而是通过支持库执行检查并选择适当的行为。通常情况下，名称以 `...Compat` (如 `ActivityCompat`) 结束的类即是如此。

而另外一些情况下，支持库类提供一个不依赖于任何框架 API 可用性的完整、独立版框架类。这些方法可以在支持的所有平台中提供一致的行为。

无论是哪一种情形，应用均无需在运行期间检查系统版本。应用可通过支持库类执行适当的系统检查，并在必要时修改其行为。

## 支持一般布局模式

支持库提供 Android 框架中未提供的用户界面元素。例如，Android 支持库提供其他布局类，如 `DrawerLayout`。这些类遵循建议的 Android 设计做法；例如，设计库以一种适合多个 Android 版本的方式遵循 Material Design 的原则。

通过使用这些支持库类，您可以避免做一些重复性工作；如果应用有特殊的用户界面要求，您可以利用现有代码，这些代码将提供用户已经熟悉的用户界面。这些元素还可以帮助您开发看起来像 Android 生态系统一部分的应用。例如，许多应用需要显示任意长的元素列表，且需要能够在列表发生变化时快速有效地重复使用这些元素；这可以是电子邮件列表、联系人列表以及音乐专辑列表，等等。这些应用可以使用支持库 `RecyclerView` 小部件显示列表。这既可以让应用开发者不必从头开始开发列表，又能确保用户看到一个外观和行为与其他应用中的列表类似的列表。

## 支持不同的设备类型

Android SDK 为 TV 和穿戴式设备等多种不同的设备类型提供库。应用可以通过相应的支持库为各种平台版本提供功能，且可以在外部屏幕、扬声器和其他目标设备上提供内容。

## 一般实用程序

---

Android 支持库提供后向兼容的实用程序功能。应用可以使用这些实用程序功能为各种 Android 系统版本提供相应的用户体验。例如，支持库的权限处理方式取决于运行应用的平台版本。如果平台支持运行时权限模式，这些方法会向用户请求相应的权限；在不支持运行时权限模式的平台版本中，这些方法将在安装时检查是否已获得相应的权限。



# 支持库功能

## 本文内容

- › [v4 支持库](#)
  - › [v4 compat 库](#)
  - › [v4 core-utils 库](#)
  - › [v4 core-ui 库](#)
  - › [v4 media-compat 库](#)
  - › [v4 fragment 库](#)
- › [Dalvik 可执行文件分包支持库](#)
- › [v7 支持库](#)
  - › [v7 appcompat 库](#)
  - › [v7 cardview 库](#)
  - › [v7 gridlayout 库](#)
  - › [v7 mediarouter 库](#)
  - › [v7 palette 库](#)
  - › [v7 recyclerview 库](#)
  - › [v7 preference 库](#)
- › [v8 支持库](#)
- › [v13 支持库](#)
- › [v14 Preference 支持库](#)
- › [v17 Leanback 库](#)
- › [适用于电视的 v17 Preference 库](#)
- › [注解支持库](#)
- › [设计支持库](#)
- › [自定义标签页支持库](#)
- › [百分比支持库](#)
- › [适用于电视的建议支持库](#)

## 另请参阅

- › [支持库修订](#)
- › [支持库设置](#)
- › [测试支持库](#)

Android 支持库软件包含可以添加至应用的多个库。每个库均支持特定范围的 Android 平台版本和功能。

本指南介绍了各支持库提供的重要功能和版本支持，从而帮助您决定在应用中添加哪些支持库。一般而言，我们建议添加 [v4 支持库](#) 和 [v7 appcompat 库](#)，因为它们支持一系列 Android 版本，并且可以为推荐的用户界面模式提供 API。

要使用以下任一库，您必须将库文件下载到 Android SDK 安装位置中。请按照[支持库设置](#)中下载支持库的说明完成此步骤。要在应用中添加特定支持库，您还必须执行其他步骤。有关如何在应用中添加支持库的重要信息，请参阅下面各个库内容的末尾部分。

## v4 支持库

这些库旨在与 Android 2.3 (API 级别 9) 及更高版本搭配使用。与其他支持库相比，它们包含的 API 集合最大，包括对应用组件、用户界面功能、辅助功能、数据处理、网络连接以及编程实用工具的支持。

如需了解有关 v4 支持库所提供类和方法的完整详细信息，请参阅 API 参考中的 [android.support.v4](#) 软件包。

**注：**在支持库修订版 24.2.0 之前，存在一个 v4 支持库。为了提高效率，此库拆分成多个模块。出于向后兼容的考虑，如果您在 Gradle 脚本中列出了 support-v4，您的 APK 将包含所有的 v4 模块。不过，要减少 APK 大小，我们建议仅列出应用需要的特定模块。

## v4 compat 库

为众多框架 API 提供兼容性包装器，例如 `Context.obtainDrawable()` 和 `View.performAccessibilityAction()`。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-compat:24.2.0
```

## v4 core-utils 库

提供大量实用程序类，例如 `AsyncTaskLoader` 和 `PermissionChecker`。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-core-utils:24.2.0
```

## v4 core-ui 库

实现各种 UI 相关组件，例如 `ViewPager`、`NestedScrollView` 和 `ExploreByTouchHelper`。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-core-ui:24.2.0
```

## v4 media-compat 库

向后移植部分媒体框架，包括 `MediaBrowser` 和 `MediaSession`。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-media-compat:24.2.0
```

## v4 fragment 库

添加对使用 `片段` 封装用户界面和功能的支持，从而使应用能够提供可以在大屏幕设备与小屏幕设备之间进行调节的布局。此模块依赖于 `compat`、`core-utils`、`core-ui` 和 `media-compat`。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-fragment:24.2.0
```

## Dalvik 可执行文件分包支持库

此库可以为使用多个 Dalvik Executable (DEX) 文件开发应用提供支持。引用超过 65536 个方法的应用须使用 Dalvik 可执行文件分包配置。如需了解有关使用 Dalvik 可执行文件分包的详细信息，请参阅[使用超过 6.4 万种方法开发应用](#)。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:multidex:1.0.0
```

## v7 支持库

一些库旨在与 Android 2.3 (API 级别 9) 及更高版本搭配使用。这些库提供了特定的功能集，并且可以单独添加到应用中，与其他库互不影响。

### v7 appcompat 库

此库添加了对[操作栏](#)用户界面[设计模式](#)的支持。此库包含对 Material Design 用户界面实现的支持。

**注：**此库依赖于 v4 支持库。

下面是 v7 appcompat 库中包含的一些关键类：

- [ActionBar](#) - 提供操作栏[用户界面模式](#)的实现。如需了解有关使用操作栏的详细信息，请参阅[操作栏开发者指南](#)。
- [AppCompatActivity](#) - 添加一个应用 Activity 类，此类可作为使用支持库操作栏实现的 Activity 的基础类。
- [AppCompatDialog](#) - 添加一个对话框类，此类可作为 AppCompat 主题对话框的基础类。
- [ShareActionProvider](#) - 包含对可以添加到操作栏中的标准化分享操作（例如电子邮件或发帖至社交应用）的支持。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:appcompat-v7:24.2.0
```

### v7 cardview 库

此库添加了对[CardView](#)小部件的支持，让您能够在卡片内显示信息，从而使应用具备一致的外观。这些卡片对 Material Design 实现非常有用，并在电视应用布局中广为使用。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:cardview-v7:24.2.0
```

### v7 gridlayout 库

下载 Android 支持库后，此库可以添加对[GridLayout](#)类的支持，让您能够使用网状方格安排用户界面元素。如需了解有关 v7 gridlayout 库 API 的详细信息，请参阅 API 参考中的[android.support.v7.widget](#)软件包。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:gridlayout-v7:24.2.0
```

### v7 mediarouter 库

此库可以提供[MediaRouter](#)、[MediaRouteProvider](#)和支持[Google Cast](#)的相关媒体类。

一般而言，利用 v7 mediarouter 库中的 API，您可以控制当前设备到外部屏幕、扬声器和其他目标设备的媒体渠道和流的路由。此库包含的 API 可以用于发布应用特定的媒体路由提供程序、发现和选择目标设备，以及检查媒体状态，等等。如需了解有关 v7 mediarouter 库 API 的详细信息，请参阅 API 参考中的[android.support.v7.media](#)软件包。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:mediarouter-v7:24.2.0
```

支持库 r18 中引入的 v7 mediarouter 库 API 在后续版本的支持库中可能会发生更改。目前，我们建议仅使用与[Google Cast](#)有关的库。

### v7 palette 库

v7 palette 支持库包含 [Palette](#) 类，此类让您可以从图像中抽取突出颜色。例如，音乐应用可以使用 [Palette](#) 对象从专辑封面抽取主要颜色，然后使用这些颜色创建一个色彩协调的歌曲标题卡。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:palette-v7:24.2.0
```

## v7 recyclerview 库

recyclerview 库添加了 [RecyclerView](#) 类。此类能够为 [RecyclerView](#) 小部件提供支持，[RecyclerView](#) 是一种通过提供有限的数据项窗口有效显示大数据集的视图。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:recyclerview-v7:24.2.0
```

## v7 Preference 支持库

[首选项](#) 软件包提供的 API 支持添加 preference 对象（例如 [CheckBoxPreference](#) 和 [ListPreference](#)），方便用户修改 UI 设置。

v7 Preference 库添加了对接口（例如 [Preference.OnPreferenceChangeListener](#) 和 [Preference.OnPreferenceClickListener](#)）以及类（例如 [CheckBoxPreference](#) 和 [ListPreference](#)）的支持。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:preference-v7:24.2.0
```

## v8 支持库

此库旨在与 Android 2.3 (API 级别 9) 及更高版本搭配使用。此库提供了特定的功能集，并且可以单独添加到应用中，与其他库互不影响。

## v8 renderscript 库

此库旨在与 Android 2.3 (API 级别 9) 及更高版本搭配使用。它添加了对 [RenderScript](#) 计算框架的支持。[android.support.v8.renderscript](#) 软件包中包含这些 API。请注意，在应用中添加这些 API 的步骤与添加其他支持库 API 迥然不同。如需了解有关在应用中使用这些 API 的详细信息，请参阅 [RenderScript](#) 开发者指南。

**注：**Android Studio 和 Gradle 构建支持使用带支持库的 RenderScript。renderscript 库位于 `build-tools/$VERSION/renderscript/` 文件夹中。

以下示例显示了此库的 Gradle 构建脚本属性：

```
defaultConfig {
 renderscriptTargetApi 18
 renderscriptSupportModeEnabled true
}
```

## v13 支持库

此库旨在用于 Android 3.2 (API 级别 13) 及更高版本。它添加了对带有 ([FragmentCompat](#)) 类和其他片段支持类的 [Fragment](#) 用户界面模式的支持。如需了解有关片段的详细信息，请参阅 [片段](#) 开发者指南。如需了解有关 v13 支持库 API 的详细信息，请参阅 API 参考中的 [android.support.v13](#) 软件包。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-v13:24.2.0
```

## v14 Preference 支持库

`android.support.v14.preference` 软件包提供的 API 添加了对首选项接口（例如 `PreferenceFragment.OnPreferenceStartFragmentCallback` 和 `PreferenceFragment.OnPreferenceStartScreenCallback`）以及类（例如 `MultiSelectListPreference` 和 `PreferenceFragment`）的支持。如需了解有关 v14 Preference 支持库 API 的详细信息，请参阅 API 参考中的[首选项](#)软件包。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:preference-v14:24.2.0
```

## 适用于电视的 v17 Preference 支持库

`android.support.v17.preference` 软件包提供的 API 可以在电视设备上提供首选项接口，包括对 `LeanbackListPreferenceDialogFragment.ViewHolder.OnItemClickListener` 接口和类的支持，例如 `BaseLeanbackPreferenceFragment` 和 `LeanbackPreferenceFragment`。如需了解有关 v17 Preference 支持库 API 的详细信息，请参阅 API 参考中的[首选项](#)软件包。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:preference-leanback-v17:24.2.0
```

## v17 Leanback 库

`android.support.v17.leanback` 软件包提供的 API 支持在电视设备上构建用户界面。它为电视应用提供了一些重要的小部件。一些值得注意的类包括：

- `BrowseFragment` - 一种用于创建主要布局的片段，主要布局用于浏览类别和媒体项目。
- `DetailsFragment` - 用于 Leanback 细节屏幕的包装器片段。
- `PlaybackOverlayFragment` - 用于显示播放控件及相关内容的 `DetailsFragment` 子类。
- `SearchFragment` - 用于处理搜索的片段。片段可以接收用户的搜索请求并将其传递给应用提供的 `SearchResultProvider`。`SearchResultProvider` 可以将搜索结果返回给 `SearchFragment`，后者会将结果渲染到 `RowsFragment` 中。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:leanback-v17:24.2.0
```

## 注解支持库

[注解](#)软件包提供的 API 支持向应用中添加注解元数据。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:support-annotations:24.2.0
```

## 设计支持库

[设计](#)软件包提供的 API 支持向应用中添加 Material Design 组件和模式。

设计支持库添加了对应用开发者依赖的各种 Material Design 组件和模式的支持，例如抽屉式导航栏、浮动操作按钮 (*FAB*)、快捷信息栏和标签页。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:design:24.2.0
```

## 自定义标签页支持库

[自定义标签页](#)软件包提供的 API 支持向应用中添加自定义标签页并对其进行管理。

自定义标签页支持库添加了对[自定义标签页服务](#)和[自定义标签页回调](#)等各种类的支持。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:customtabs:24.2.0
```

## 百分比支持库

[百分比](#)软件包提供的 API 支持向应用中添加基于百分比的尺寸并对其进行管理。

百分比支持库添加了对 `PercentLayoutHelper.PercentLayoutParams` 接口和各种类的支持，例如 `PercentFrameLayout` 和 `PercentRelativeLayout`。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:percent:24.2.0
```

## 适用于电视的应用建议支持库

[应用建议](#)软件包提供的 API 支持向电视设备上运行的应用中添加内容建议。

应用库添加了对注解（例如 `ContentRecommendation.ContentMaturity` 和各种类（例如 `ContentRecommendation` 和 `RecommendationExtender`）的支持。

此库的 Gradle 构建脚本依赖关系标识符如下所示：

```
com.android.support:recommendation:24.2.0
```



# 支持库设置

## 本文内容

- › [下载支持库](#)
- › [选择支持库](#)
- › [添加支持库](#)
- › [使用支持库 API](#)
- › [清单声明变更](#)

## 另请参阅

- › [支持库修订](#)
- › [支持库功能](#)

如何在开发项目中设置 Android 支持库取决于您想要使用的功能，以及您希望应用支持的 Android 平台版本范围。

本文档将指导您下载支持库软件包以及向开发环境中添加库。

## 下载支持库

Android 支持存储库软件包作为 Android SDK 的辅助组件提供，可以通过 [Android SDK 管理器](#) 下载。请按照以下说明操作，获取支持库文件。

要通过 SDK 管理器下载支持库，请执行以下操作：

1. 启动 [Android SDK 管理器](#)。
2. 在 SDK 管理器窗口中，滚动到 *Packages* 列表末尾，找到 *Extras* 文件夹并展开（如有必要）以显示其内容。
3. 选择 **Android Support Repository** 项。
4. 点击 **Install packages...** 按钮。

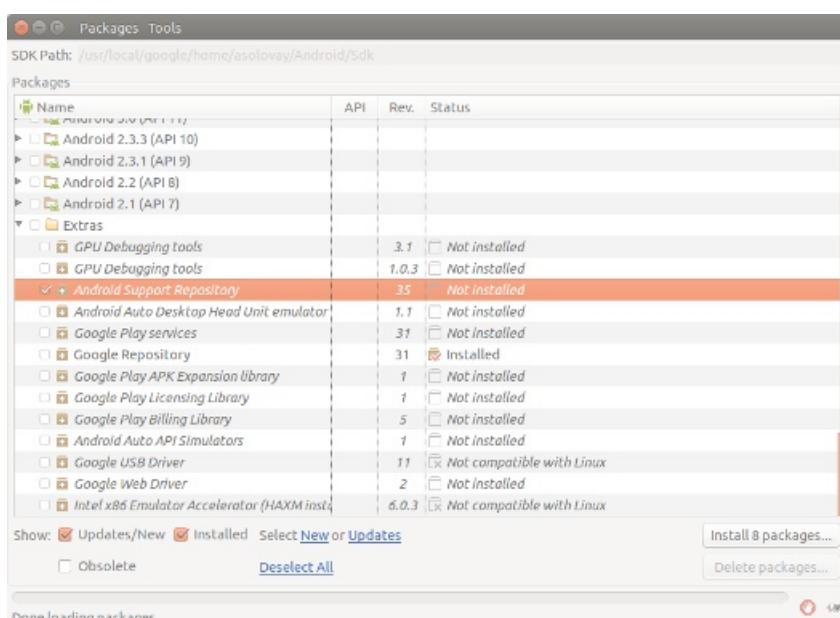


图 1. Android SDK 管理器以及选择的 Android 支持存储库。

下载后，此工具会将支持库文件安装到您现有的 Android SDK 目录中。库文件位于 SDK 的以下子目录

中：`<sdk>/extras/android/m2repository/com/android/support/` 目录。

## 选择支持库

在将支持库添加到应用之前，确定您想要包含的功能以及希望支持的最低 Android 版本。如需了解有关不同库所提供功能的详细信息，请参阅[支持库功能](#)。

## 添加支持库

要使用支持库，您必须在开发环境中修改应用项目的类路径依赖关系。必须针对想要使用的每个支持库执行该步骤。

要向应用项目中添加支持库，请执行以下操作：

1. 确保已使用[SDK 管理器](#)下载Android **支持存储库**。
2. 打开应用的 `build.gradle` 文件。
3. 将支持库添加到 `dependencies` 部分。例如，要添加 v4 core-utils 库，请添加以下行：

```
dependencies {
 ...
 compile "com.android.support:support-core-utils:24.2.0"
}
```

**注意**：使用动态依赖关系（例如 `palette-v7:23.0.+`）可能会导致意外的版本更新和回归不兼容问题。我们建议明确指定库版本（例如 `palette-v7:24.2.0`）。

## 使用支持库 API

为现有框架 API 提供支持的支持库类与框架类通常具有相同的名称，但前者位于 `android.support` 类软件包中，或带有 `*Compat` 后缀。

**注意**：使用支持库中的类时，确保从相应的软件包中导入类。例如，应用 `ActionBar` 类时：

- 使用支持库时为 `android.support.v7.app.ActionBar`。
- 仅为 API 级别 11 或以上开发时为 `android.app.ActionBar`。

**注**：将支持库添加到应用项目后，我们强烈建议使用 [ProGuard](#) 工具准备要发布的应用 APK。除了保护您的源代码外，ProGuard 工具还可以从添加到应用的任意库中移除不使用的类，从而使应用的下载大小尽可能小。如需了解详细信息，请参阅 [ProGuard](#)。

如需使用支持库某些功能的更多指导，请参见 Android 开发者[培训课程](#)、[指南](#)和示例。如需了解有关支持库中各个类和方法的详细信息，请参阅 API 参考中的 `android.support` 软件包。

## 清单声明变更

如果您计划增加现有应用对带有支持库的旧版本 Android API 的向后兼容性，请确保更新应用的清单。具体而言，您应将清单中 `<uses-sdk>` 标记的 `android:minSdkVersion` 元素更新为较低的新版本号，如下所示：

```
<uses-sdk
 android:minSdkVersion="14"
 android:targetSdkVersion="23" />
```

清单设置告知 Google Play 您的应用可以安装在 Android 4.0 (API 级别 14) 及更高版本的设备上。

如果您使用 Gradle 构建文件，构建文件中的 `minSdkVersion` 设置将替换清单设置。

```
apply plugin: 'com.android.application'

android {
 ...

 defaultConfig {
 minSdkVersion 16
 ...
 }
 ...
}
```

在这种情况下，构建文件设置将告知 Google Play 您应用的默认构建变体可以安装在 Android 4.1（API 级别 16）及更高版本的设备上。如需了解有关构建变体的详细信息，请参阅[构建系统概览](#)。

**注：**如果添加多个支持库，则最低的 SDK 版本必须是任意指定库所要求的最高版本。例如，如果您的应用中包含[v14 Preference 支持库](#)和[v17 Leanback 库](#)，则最低的 SDK 版本必须为 17 或更高。

# Recent Support Library Revisions

## Recent revisions

- › [26.0.0 Release](#)
- › [26.0.0 Beta 2](#)
- › [25.4.0](#)
- › [26.0.0 Beta 1](#)
- › [26.0.0 Alpha 1](#)
- › [25.3.1](#)
- › [25.3.0](#)
- › [25.2.0](#)
- › [25.1.1](#)
- › [25.1.0](#)
- › [25.0.1](#)
- › [25.0.0](#)

## See also

- › [Support Library Revisions Archive](#)
- › [Support Library Features](#)

This page provides details about the most recent Support Library package releases. For earlier releases, see the [Support Library Revisions Archive](#).

## Revision 26.0.0 Release

(July 2017)

**Important:** The support libraries are now available through Google's Maven repository. You do not need to download the support repository from the SDK Manager. For more information, see [Support Library Setup](#).

## Important changes

- The minimum SDK version has been increased to 14. As a result, many APIs that existed only for compatibility with pre-14 API levels have been deprecated. Clients of these APIs should migrate to their framework equivalents as noted in the reference page for each deprecated API.
- The Wear UI Library contains classes that help you implement patterns and layouts that work on Wear devices. For more information, see [Using the Wear UI Library](#).
- The [Percent Support module](#) has been deprecated. Clients of this module should migrate to the new [ConstraintLayout](#) widget, which is provided as a separate artifact in SDK Manager.
- [NotificationCompat](#) and its containing classes has been deprecated and will be removed in a future release:
  - Use [NotificationCompat.Builder](#) instead of [v7.app.NotificationCompat.Builder](#). Functionality that previously relied on using the v7 AppCompat Builder has now been folded into the v4 Compat Builder.
  - [DecoratedCustomViewStyle](#) has moved to the [android.support.v4.app](#) package.
  - [MediaStyle](#) and [DecoratedMediaCustomViewStyle](#) are now part of the [media-compat library](#) and can be found in the

[android.support.v4.media](#) package.

## New APIs

- New `fastScrollEnabled` boolean flag for `RecyclerView`. If enabled, `fastScrollHorizontalThumbDrawable`, `fastScrollHorizontalTrackDrawable`, `fastScrollVerticalThumbDrawable`, and `fastScrollVerticalTrackDrawable` must be set.

## API Diffs

- [26.0.0-beta2 → 26.0.0 Release](#)
- [25.4.0 → 26.0.0 Release](#)

## Bug fixes

- Infinite loop in `RecyclerView.toString()`
- `ResourceNotFoundException` running Kotlin project on API 16 AVD after upgrading to Canary 6
- `java.lang.AssertionError` in design view with support library 26.0.0-beta2
- Android Studio layout preview broken for Support Library widgets
- `Preference.setSingleLineTitle()` is ignored if the Preference was not created with attributes
- DAC "Since" annotations are wrong for 25.3.0 / 25.4.0 revisions of Support Library
- `ResourcesCompat.getFont()` throws exception
- Toolbar title not in bold font
- Auto sizing with `maxLines` produces unexpected results
- `NullPointerException` in `TextView.checkForRelayout()`
- `AppCompatTextViewAutoSizeHelper.setRawTextSize()` calls `requestLayout()` during layout
- `EmojiCompatTextView` crashes
- Autosize `TextView` does not adjust automatically when text is changed
- Screen corruption in Instacart
- `UnsupportedOperationException` in `MenuItemCompat`
- `NotificationCompat` doesn't fully extract actions on API 24 or higher
- CoordinatorLayout anchoring problems on layout updates

## Revision 26.0.0 Beta 2

---

(June 2017)

Please note that 26.0.0-beta2 is a pre-release version. Its API surface is subject to change, and it does not necessarily include features or bug fixes from the latest stable versions of Support Library.

**Important:** The support libraries are now available through Google's Maven repository. You do not need to download the support repository from the SDK Manager. For more information, see [Support Library Setup](#).

## New APIs

- New `JobIntentService` class, to help developers schedule tasks in a way that complies with the new Android O [background execution limits](#).

## API Diffs

- [26.0.0-beta1 -> 26.0.0-beta2](#)
- [25.4.0 -> 26.0.0-beta2](#)

## Bug fixes

- Android O SDK drop causes loss of italics in TextViews
- Null pointer exception when connecting to MediaBrowserServiceCompat
- TextInputLayout must set hints on onProvideAutofillStructure()
- Stack overflow when using TextView autosize on O

## Revision 25.4.0

(June 2017)

**Important:** The support libraries are now available through Google's Maven repository. You do not need to download the support repository from the SDK Manager. For more information, see [Support Library Setup](#).

## Important changes

- `executePendingTransactions()`, `commitNow()`, `popBackStackImmediate()`, and similar transaction calls are not allowed during `FragmentManager` state changes. Reentrant execution of transactions are unsafe and `FragmentManager` now enforces this during its state changes.
- Concurrent with this support library release, we are also releasing `multidex` version 1.0.2. This version includes the following important changes:
  - Allows multidexing of instrumentation APK.
  - Deprecates MultiDexTestRunner (AndroidJUnitRunner should be used instead).
  - Provides better protection against some bad archive extraction management of the app.
  - Fixes a bug that could lead to abandoned temporary files.
  - Provides faster installation when done in concurrent process.
  - Fixes an installation bug on API 19 and 20.

## New and Modified APIs

Path morphing and path interpolation are supported in `AnimatedVectorDrawableCompat`. Path morphing allow the shapes changing from one path (specified as `android:valueFrom`) to another path (specified as `android:valueTo`), in order to provide complex and attractive visual effects. Path interpolation allows the interpolators for `AnimatedVectorDrawableCompat` to be specified as paths (specified as `android:pathData` in the interpolator's XML).

## API Diffs

- [25.3.0 -> 25.4.0](#)

## Fixed issues

- Null pointer exception when connecting to MediaBrowserServiceCompat
- MediaBrowserCompat.search() API does not work (AOSP issue [262170](#))

- BrowseFragment onItemClick callbacks broken in 25.3.0
- NullPointerException while scrolling up and down in VerticalGridView in 25.3.1
- ClassCastException in SimpleArrayMap.allocArrays()

## Revision 26.0.0 Beta 1

(May 2017)

Please note that 26.0.0-beta1 is a pre-release version. Its API surface is subject to change, and it does not necessarily include features or bug fixes from the latest stable versions of Support Library.

**Important:** The support libraries are now available through Google's Maven repository. You do not need to download the support repository from the SDK Manager. For more information, see [Support Library Setup](#).

### Important changes

- `FragmentActivity.setSupportMediaController()` and `FragmentActivity.getSupportMediaController()` have been removed. Please use the new static `MediaControllerCompat.setMediaController()` and `MediaControllerCompat.getMediaController()` methods.
- `BottomNavigationView` now calls `onNavigationItemSelectedReselected()` when an already-selected item is selected, rather than calling `onNavigationItemSelected()`.
- All instances of the `findViewById()` method now return `<T extends View> T` instead of `View`. This change has the following implications:
  - This may result in existing code now having ambiguous return type, for example if there is both `someMethod(View)` and `someMethod(TextView)` that takes the result of a call to `findViewById()`.
  - When using Java 8 source language, this requires an explicit cast to `View` when the return type is unconstrained (for example, `assertNotNull(findViewById(...)).someViewMethod()`).
  - Overrides of non-final `findViewById()` methods (for example, `Activity.findViewById()`) will need their return type updated.

### New APIs

- `FragmentManager` and `Fragment` have an `isStateSaved()` method to allow querying whether or not a transaction will be allowed without state loss. This is especially useful to check when handling an `onClick()` event before executing any transaction.
- Path motion is supported in `AnimatedVectorDrawableCompat`. Path motion allows one object animator to change two properties at the same time based on one path; the path is specified as `android:pathData` in the animator's XML).
- **Physics-based animation:**
  - New `FlingAnimation` that supports animating with an initial velocity and smoothly slows down.
  - Subclasses of `DynamicAnimation` support animating custom property for any object.
  - Both `SpringAnimation` and `FlingAnimation` can now animate a float value without requiring a `View` or an `Object` to be associated with it.

For more information, see the [Spring animation](#) and [Fling animation](#) preview pages.

- **Font support in XML:**
  - `ResourcesCompat.getFont` allows loading font resources—including font-family XML—that may be used with `TextView.setTypeface()`.
  - When using AppCompat, `TextView` supports specifying a font resource or font-family XML via the `android:fontFamily` XML attribute.
  - Use XML font-family to create families of fonts with style and weight variations. (If you use the support library classes to do this, use the `app:` attributes as well as the `android:` attributes.)

- Downloadable fonts:
  - New `FontsContractCompat` that allows you to request fonts from a font provider instead of bundling them in your app.
  - Fonts can also be requested in XML and used in layouts.
- Emoji compatibility library:
  - `EmojiCompat` can process a given `CharSequence` and add `EmojiSpans`.
  - `EmojiTextView` and other widgets to display emoji.
  - `FontRequestEmojiCompatConfig` to request emoji font from a font provider.
- Autosizing TextView:
  - New methods in `TextViewCompat` as well as XML attributes to control autosizing in `TextView`.
- Leanback playback controls with seek support:
  - New `PlaybackTransportRowPresenter` that renders playback controls with a SeekBar.
  - New `PlaybackTransportControlGlue` that works with `PlaybackTransportRowPresenter` and supports seek.
  - New base class `PlaybackSeekDataProvider` for app to provide seek thumbnails to `PlaybackTransportControlGlue`.
- Preferences Data Store:
  - `PreferenceDataStore` now allows you to implement your own preferences storage, set with new methods in `Preference` and `PreferenceManager`.

## Known Issues

- Downloadable Fonts and Emoji compatibility integration with Google Play Services only works on Google Play Services v11+, which is currently available through the [Google Play Services beta program](#).

## Bug fixes

- `MediaBrowserCompat.search()` API does not work (AOSP issue [262170](#))
- `ViewCompat.postInvalidateOnAnimation()` throws exception (AOSP issue [80146](#))
- `onActivityCreated()` called for fragments in destroyed Activity
- `RecyclerView.isComputingLayout()` should return true during prefetch
- When a `Fade` transition is interrupted and reversed, the `View` starts the animation from the beginning. (Fix ported from Android Framework.)
- `Transition.Fade` ignores initial alpha of `View` (AOSP issue [221820](#))

## Revision 26.0.0 Alpha 1

(March 2017)

Please note that 26.0.0-alpha1 is a pre-release version. Its API surface is subject to change, and it does not necessarily include features or bug fixes from the latest stable versions of Support Library.

## Important changes

**Note:** The minimum SDK version has been increased to 14. As a result, many APIs that existed only for API < 14 compatibility have been deprecated. Clients of these APIs should migrate to their framework equivalents as noted in the reference page for each deprecated API.

- The support-percent module has been deprecated. Clients of this module should migrate to the new ConstraintLayout widget, which is provided as a separate artifact in SDK Manager.

- The support-fragment module no longer has a dependency on the support-media-compat module.

## New APIs

Many new classes, methods, and constants added to provide backwards-compatible support for platform APIs added in O Preview.

- **IME\_FLAG\_NO\_PERSONALIZED\_LEARNING:** IMEs can listen for "no learning" flags for apps that have a private mode, such as browsers. This feature helps IMEs understand if an app is in a private mode, so they can disable their learning or adaptive functionality while the app is in that mode.

For a complete list of API changes between 25.2.0 and 26.0.0-alpha1, see the [support library API differences report](#).

## Bug fixes

- In some cases simple `AutoTransition` animation can be interrupted by view "jumps". (AOSP issue [221816](#))

## Revision 25.3.1

---

(March 2017)

### Fixed issues

- `SwitchCompat` requires minimum SDK version of 14 or higher. (AOSP issue [251302](#))
- Physics-based animation `updateListener` skips the first frame.
- `BottomNavigationView` label animation is broken.

## Revision 25.3.0

---

(March 2017)

### Important changes

Support Library version metadata will automatically be added to `AndroidManifest.xml` when building from Gradle, which simplifies tracking versions in public builds. For example:

```
<meta-data android:name="android.support.VERSION" android:value="25.3.0" />
```

## Deprecations

A number of methods and classes have been deprecated in this release. These deprecated APIs will be removed in a future version and developers should migrate away from them. For more information on how to migrate away from a specific API, refer to its documentation.

### `ExifInterface`

The boolean method `getLatLong(float[])` is deprecated. Instead, use the new method `getLatLong()`, which takes no arguments and returns `double[]`.

### `mediacompat`

`PlaybackStateCompat.Builder.setErrorMessage(CharSequence)` is deprecated. Instead, use the new method `setMessage(int, CharSequence)`, which is passed an error code and an optional description.

`EXTRA_SUGGESTION_KEYWORDS` is deprecated. Instead, use the `MediaBrowserCompat` search functionality.

### `v7.recyclerview`

`LinearLayoutManager.getInitialItemPrefetchCount()` has been renamed to

`LinearLayoutManager.getInitialPrefetchItemCount()`. The old name is still supported but will be removed in a future release.

## New and Modified APIs

### appcompat-v7

The new method `ActionBarDrawerToggle.setDrawerSlideAnimationEnabled(boolean)` simplifies disabling the navigation drawer toggle icon's animation.

### customtabs

Added support for message channels. See the `CustomTabsService.requestPostMessageChannel()` and `CustomTabsService.postMessage()` reference for details.

### dynamic-animation

New physics-based animation library that provides a set of APIs for building animations that dynamically react to user input.

### leanback-v17

Added support for parallax backgrounds. See the `Parallax` reference for details.

Added `TimePicker` widget for picking times on a TV interface.

### mediacompat

Added search functionality. See the `MediaBrowserCompat.search()` and `MediaBrowserServiceCompat.onSearch()` reference for details.

Added support for shuffle and repeat modes. See the `MediaSessionCompat.setRepeatMode()` and `setShuffleModeEnabled()` reference for details.

## Fixed issues

- `StaggeredGridLayoutManager` throws `IllegalArgumentException` (AOSP issue [230295](#))
- `RecyclerView` prefetch does not properly handle a `RecyclerView` that is attached but not onscreen
- `LinearLayout` not recognized by Robolectric
- When `Activity` is destroyed, `onActivityCreated()` is improperly called for its fragments
- `AppCompatImageView` constructor causes `ArrayIndexOutOfBoundsException`
- Poor UI performance in `Call.Details` activity transition

## Revision 25.2.0

---

(February 2017)

### Important Changes

#### Fixed issues

- This release fixes a severe mediarouter issue in which using an A2DP device and media routing APIs could cause the device to become unresponsive, requiring a reboot.
- The `FragmentManager.FragmentLifecycleCallbacks` class is now static.

#### Fixed issues

- Showing a slide presentation with screen mirroring causes device to disconnect from Wi-Fi
- Media button did not properly handle media apps that did not register themselves with `setMediaButtonReceiver()`
- `VectorDrawable` error with string resource (AOSP issue [232407](#))
- `TextInputLayout` overlays hint and text if text is set by XML (AOSP issue [230171](#))
- Memory leak in `MediaControllerCompat` (AOSP issue [231441](#))
- `RecyclerViewLayoutTest.triggerFocusSearchInOnRecycledCallback()` crashing
- `RecyclerView` crashes when recycling view holders (AOSP issue [225762](#))
- `getAllowGeneratedReplies()` incorrectly returns false for actions inside a `WearableExtender`

## Revision 25.1.1

(January 2017)

**Important:** There is a known bug in the `android.support.v7.media.MediaRouter` class in revision 25.1.1 and 25.1.0 of the Support Library. If your app uses the v7 `MediaRouter`, you should update to [Support Library Revision 25.2.0](#), which fixes this bug.

### Important Changes

- Fragment transactions can now be optimized within and across transactions. Optimizing fragment transaction operations can eliminate operations that cancel. For example, suppose two transactions are executed together, one that adds a fragment A and a second one that replaces fragment A with fragment B. In this case, the first operation might be canceled, and only fragment B added. That means that fragment A might not go through the creation/destruction lifecycle.

A side effect of this optimization is that fragments might have state changes out of the expected order. For example, suppose one transaction adds fragment A, a second adds fragment B, then a third removes fragment A. Without optimization, fragment B could expect that while it is being created, fragment A will also exist because fragment A will be removed after fragment B is added. With optimization, fragment B cannot be sure that fragment A will exist while B is being created, because fragment A's creation and destruction may be removed by the optimization.

This optimization is disabled by default. To enable the optimization, call `FragmentTransaction.setAllowOptimization(true)`.

- Fragments can now postpone their transitions and animations until they are ready using `Fragment.postponeEnterTransition()` and `Fragment.startPostponedEnterTransition()`. This API is similar to `Activity.postponeEnterTransition()` and `Activity.startPostponedEnterTransition()` used with Activity Transitions.

### Fixed issues

- `MediaSessionCompatTest` fails with `IllegalArgumentException`
- `DetailsFragment.installTitleView()` is not called in 25.1.0
- Fragment transaction keeps ghost view on exit (AOSP issue [230679](#))
- `BottomNavigationView` needs spacing between item icon and text (AOSP issue [230653](#))
- Selected listeners are missing from the new `PlaybackFragment` and `PlaybackSupportFragment`
- `TextInputLayout` focus does not change properly in emulator from support library version 25.1.0 (AOSP issue [230461](#))
- Cannot replace the menu of a `BottomNavigationView` (AOSP issue [230343](#))
- `RecyclerView` with `StaggeredGridLayoutManager` crashes with full-span items (AOSP issue [230295](#))
- Crash in `MediaSessionCompat` when using `setCallback(null)`
- `PlaybackGlueHostOld` and `PlaybackSupportGlueHostOld` don't notify callbacks when playback row changes

- `PlaybackOverlayFragment` example /test can not start playing
- `RecyclerViewFocusRecoveryTest` is failing on API 15
- "Screenshots" row is focused to the top of the screen
- `RecyclerViewLayoutTest.triggerFocusSearchInOnRecycledCallback()` crashes on API 15
- `setActions()` in `onSubactionClicked()` is broken
- `RecyclerView` crashes when recycling some view holders

## Revision 25.1.0

---

(December 2016)

**Important:** There is a known bug in the `android.support.v7.media.MediaRouter` class in revision 25.1.1 and 25.1.0 of the Support Library. If your app uses the v7 `MediaRouter`, you should update to [Support Library Revision 25.2.0](#), which fixes this bug.

### Important Changes

- Clients of nested `RecyclerView` widgets (for example, vertical scrolling list of horizontal scrolling lists) can get significant performance benefits by hinting the inner `RecyclerView` widgets' layout managers how many items to prepare before being scrolled on screen. Call `LinearLayoutManager.setInitialPrefetchItemCount(N)`, where `N` is the number of views visible per inner item. For example, if your inner, horizontal lists show a minimum of three and a half item views at a time, you can improve performance by calling `LinearLayoutManager.setInitialPrefetchItemCount(4)`. Doing so allows `RecyclerView` to create all relevant views early, while the outer `RecyclerView` is scrolling, which significantly reduces the amount of stuttering during scrolls.
- `FragmentActivity.setSupportMediaController()` and `FragmentActivity.getSupportMediaController()` have been deprecated. Please use the new static `MediaControllerCompat.setMediaController()` and `MediaControllerCompat.getMediaController()` methods.
- When a client specifies a widget tint via appcompat tinting (for example, `appcompat:buttonTint`), the client is responsible for providing all necessary states (such as "disabled", "pressed", etc.). This is consistent with how widget tints are specified when using framework tinting.

### New and Modified APIs

- Added [ExifInterface support library](#). This library unbundles support for reading Exif information from JPEG and raw formatted files and setting the Exif information on JPEG image files.
- [Snackbar](#) has been refactored to allow apps to display custom content. `BaseTransientBottomBar` is the new base class that exposes the general sliding and animations behavior.
- Added a new `leanback.media` package which contains helper classes to integrate media players into Android TV applications.
- Added `SeekBarPreference` with customizable layout and attributes to the [v7 preference support library](#).
- Added `ArraySet` class to the v4 support library. This class corresponds to the framework `ArraySet` class that was introduced with API level 23.
- `RecyclerView` RecyclerView item prefetching improvements:
  - Nested `RecyclerView` prefetch enables prefetching of content from a `RecyclerView` within another scrolling `RecyclerView`, with API to control how much prefetching is done:
    - `LinearLayoutManager.setInitialPrefetchItemCount()`
    - `LinearLayoutManager.getInitialPrefetchItemCount()`
  - APIs added for custom `LayoutManager` objects to implement to enable prefetching during scrolls and flings
    - `RecyclerView.LayoutManager.LayoutPrefetchRegistry()`

- `RecyclerView.LayoutManager.collectAdjacentPrefetchPositions()`
- `RecyclerView.LayoutManager.collectInitialPrefetchPositions()`
- Improvements to prefetching to do only as much create/bind work as possible in the time between frames

## Fixed issues

- Password visibility toggle fails accessibility tests.
- Appcompat doesn't respect `state_enabled` on pre-L devices.
- Added focus recovery mechanism to `RecyclerView`. This also fixed support pref fragments broken focus when using DPAD navigation such as on Android TV devices.
- Leanback: BrowseFragment crashes with headers disabled and empty adapter.
- Appcompat: `AlertDialog` is too wide.
- `InputContentInfoCompat` calls `requestPermission()` when it should call `releasePermission()`.
- `MediaBrowserCompat` crashes.
- CoordinatorLayout measures/lays out views when visibility is set to `GONE`.
- Could not tint `AnimatedVectorDrawableCompat` on API level below 24
- Leanback library triggers spurious lint errors
- Palette library caused test failures on every API level
- `RecyclerView` failed tests on Leanback
- `RecyclerView` crashes when recycling view holders (AOSP issue [225762](#))
- `Fragment.onDestroy()` not called for fragment in backstack
- `CollapsingToolbarLayout` scrim is not drawn when collapsed
- `CoordinatorLayout.offsetChildByInset()` throws `IllegalArgumentException`
- Animating `RecyclerView` items detach inner `RecyclerViews`, prevent future prefetches
- Attached `RecyclerView` items can't be nested-prefetched
- Prefetch data for nested `RecyclerView` items is discarded during first layout
- `RecyclerView` prefetch fails if two drag events arrive at same position
- `RecyclerView` should speculatively layout while RenderThread is rendering
- Night-configured color resources converted to Drawables are not always properly purged from Resources cache
- `FloatingActionButton`: Programmatically setting BackgroundTintList does not work properly (AOSP issue [227428](#))
- `TextInputLayout`: Typeface is not getting set for ErrorView (AOSP issue [227803](#))
- `TextInputLayout` always falls back to light error color below API 23 (AOSP issue [221992](#))
- `FloatingActionButton` shows as pressed when pointer leaves

A complete list of public bug fixes is available on the [AOSP Issue Tracker](#).

## Deprecations

A number of methods and classes have been deprecated in this release. These deprecated APIs will be removed in a future version and developers should migrate away from them. For more information on how to migrate away from a specific API, refer to its documentation.

- `android.support.design.widget`

- `Snackbar.setCallback()`
- `android.support.v17.leanback.app`
  - `BackgroundManager.getDefaultDimLayer()`
  - `BackgroundManager.getDimLayer()`
  - `BackgroundManager.setDimLayer()`
  - `MediaControllerGlue.MediaControllerGlue(Context, PlaybackOverlayFragment, int[])`
  - `MediaControllerGlue.MediaControllerGlue(Context, PlaybackOverlayFragment, int[], int[])`
  - `PlaybackControlGlue.PlaybackControlGlue(Context, PlaybackOverlayFragment, int[])`
  - `PlaybackControlGlue.PlaybackControlGlue(Context, PlaybackOverlayFragment, int[], int[])`
  - `PlaybackControlGlue.getFragment()`
  - `PlaybackControlGlue.setOnItemViewClickedListener()`
  - `PlaybackControlGlue.onRowChanged()`
  - `PlaybackControlGlue.pausePlayback()`
  - `PlaybackControlGlue.skipToNext()`
  - `PlaybackControlGlue.skipToPrevious()`
  - `PlaybackControlGlue.startPlayback()`
  - `PlaybackControlSupportGlue`
  - `PlaybackOverlayFragment`
  - `PlaybackOverlaySupportFragment`
- `android.support.v17.leanback.widget`
  - `BaseCardView.getExtraVisibility()`
  - `BaseCardView.setExtraVisibility()`
  - `GuidedActionsStylist.onEditingModeChange()`
  - `GuidedActionsStylist.setEditingMode()`
  - `GuidedActionsStylist.setExpandedViewHolder()`
  - `GuidedActionsStylist.startExpandedTransition()`
- `android.support.v4.app`
  - `FragmentActivity.getSupportMediaController()`
  - `FragmentActivity.setSupportMediaController()`

## Revision 25.0.1

---

(November 2016)

### Fixed issues

- The `TextInputLayout` password toggle is now disabled by default to avoid unnecessarily overwriting developer-specified end drawables. It may be manually enabled via the `passwordToggleEnabled` XML attribute.
- `BottomNavigationView` items are now single line to match Material spec.
- `RecyclerView` crashes during prefetch if layout manager is null.

- `BottomNavigationView` elevation is now set properly. (AOSP issue [226182](#))
- `BottomNavigationView` crashing when adding menu items programmatically. (AOSP issue [225731](#))
- Fix to `TextInputLayout` left+right compound drawables. (AOSP issue [225836](#))
- `RecyclerView` crashes when recycling view holders. (AOSP issue [225762](#))
- Leanback: TalkBack frequently says the word "null" in split-screen views.
- `RecyclerView`: Rendering problems in Android Studio. (AOSP issue [225753](#))
- `BottomNavigationView` still shows menu item as selected after `onNavigationItemSelected()` returns false. (AOSP issue [225898](#))
- ForwardingListener throws `NoSuchMethodError`. (AOSP issue [225647](#))
- `TextInputEditText` does not show hints in IME extract mode. (AOSP issue [221880](#))

A complete list of public bug fixes is available on the [AOSP Issue Tracker](#).

## Revision 25.0.0

---

(October 2016)

### Important changes

- `ContextCompat` constructor has been made protected. This class should not be publicly instantiated, but it may be extended by support libraries targeting newer API levels.
- `ActivityCompat` constructor has been made protected. This class should not be publicly instantiated, but it may be extended by support libraries targeting newer API levels.
- `getReferrer(Activity)` has been made static.
- `android.support.design.widget.CoordinatorLayout.Behavior.isDirty(CoordinatorLayout, V)` has been removed. Any client implementations of this method should be removed.
- `android.support.v4.media.session.MediaSessionCompat.obtain(Context, Object)` has been removed. Usages should be replaced with the more appropriately named method `fromMediaSession()`.
- `android.support.v4.media.session.MediaSessionCompat.QueueItem.obtain(Object)` has been removed. Usages should be replaced with the more appropriately named method `MediaSessionCompat.QueueItem#fromQueueItem`.
- `android.support.v7.widget.Space` has been removed. Usages should be replaced with `android.support.v4.widget.Space`.

### New APIs

- `android.support.design.widget.BottomNavigationView` class implements the `bottom navigation` pattern from the Material Design spec.
- New `android.support.v13.view.inputmethod` package includes classes for accessing `android.view.inputmethod.InputConnection` features introduced after API level 13.
- `android.v7.widget.RecyclerView.DividerItemDecoration` class provides a base implementation for vertical or horizontal dividers between items.
- New decorated styles in `NotificationCompat`, `DecoratedCustomViewStyle` and `DecoratedMediaCustomViewStyle`, mirror classes added in platform API 24.

### Fixed issues

A complete list of public bug fixes is available on the [AOSP Issue Tracker](#).



# 测试支持库

## 本文内容

- › [测试支持库功能](#)
  - › [AndroidJUnitRunner](#)
  - › [Espresso](#)
  - › [UI Automator](#)
- › [测试支持库设置](#)

## 另请参阅

- › [测试支持库 API 参考](#)
- › [代码示例](#) ↗

Android 测试支持库提供了大量用于测试 Android 应用的框架。此库提供了一组 API，让您可以为应用快速构建何运行测试代码，包括 JUnit 4 和功能性用户界面（UI）测试。您可以从 [Android Studio IDE](#) 或命令行运行使用这些 API 创建的测试。

Android 测试支持库通过 Android SDK 管理器提供。如需了解详细信息，请参阅[测试支持库设置](#)

本页介绍了 Android 测试支持库提供了哪些工具、如何在测试环境中使用这些工具，以及库版本的相关信息。

## 测试支持库功能

Android 测试支持库包括以下自动化测试工具：

- [AndroidJUnitRunner](#)：适用于 Android 且与 JUnit 4 兼容的测试运行器
- [Espresso](#)：UI 测试框架；适合应用中的功能性 UI 测试
- [UI Automator](#)：UI 测试框架；适合跨系统和已安装应用的跨应用功能性 UI 测试

## AndroidJUnitRunner

[AndroidJUnitRunner](#) 类是一个 [JUnit](#) ↗ 测试运行器，可让您在 Android 设备上运行 JUnit 3 或 JUnit 4 样式测试类，包括使用 [Espresso](#) 和 [UI Automator](#) 测试框架的设备。测试运行器可以将测试软件包和要测试的应用加载到设备、运行测试并报告测试结果。此类将替换 [InstrumentationTestRunner](#) 类，后者仅支持 JUnit 3 测试。

此测试运行器的主要功能包括：

- [JUnit 支持](#)
- [访问仪器信息](#)
- [测试筛选](#)
- [测试分片](#)

要求 Android 2.2（API 级别 8）或更高版本。

### JUnit 支持

测试运行器与 JUnit 3 和 JUnit 4（最高版本为 JUnit 4.10）兼容。不过，请勿在同一软件包中混用 JUnit 3 和 JUnit 4 测试代码，因为这可能会导致意外结果。如果要创建一个 JUnit 4 仪器测试类以在设备或模拟器上运行，则测试类必须以 `@RunWith(AndroidJUnit4.class)` 注解作为前缀。

以下代码段显示了如何编写 JUnit 4 仪器测试来验证 `CalculatorActivity` 类中的 `add` 操作是否正常工作。

```
import android.support.test.runner.AndroidJUnit4;
import android.support.test.runner.AndroidJUnitRunner;
import android.test.ActivityInstrumentationTestCase2;

@RunWith(AndroidJUnit4.class)
public class CalculatorInstrumentationTest
 extends ActivityInstrumentationTestCase2<CalculatorActivity> {

 @Before
 public void setUp() throws Exception {
 super.setUp();

 // Injecting the Instrumentation instance is required
 // for your test to run with AndroidJUnitRunner.
 injectInstrumentation(InstrumentationRegistry.getInstrumentation());
 mActivity = getActivity();
 }

 @Test
 public void typeOperandsAndPerformAddOperation() {
 // Call the CalculatorActivity add() method and pass in some operand values, then
 // check that the expected value is returned.
 }

 @After
 public void tearDown() throws Exception {
 super.tearDown();
 }
}
```

## 访问仪器信息

您可以使用 `InstrumentationRegistry` 类访问与测试运行相关的信息。此类包括 `Instrumentation` 对象、目标应用 `Context` 对象、测试应用 `Context` 对象，以及传递到测试中的命令行参数。使用 UI Automator 框架编写测试或编写依赖于 `Instrumentation` 或 `Context` 对象的测试时，此数据非常有用。

## 测试筛选

在 JUnit 4.x 测试中，您可以使用注解对测试运行进行配置。此功能可将向测试中添加样板文件和条件代码的需求降至最低。除了 JUnit 4 支持的标准注解外，测试运行器还支持 Android 特定的注解，包括：

- `@RequiresDevice`：指定测试仅在物理设备而不在模拟器上运行。
- `@SdkSupress`：禁止在低于给定级别的 Android API 级别上运行测试。例如，要禁止在低于 18 的所有 API 级别上运行测试，请使用注解 `@SDKSupress(minSdkVersion=18)`。
- `@SmallTest`、`@MediumTest` 和 `@LargeTest`：指定测试的运行时长以及运行频率。

## 测试分片

测试运行器支持将单一测试套件拆分成多个碎片，因此您可以将属于同一碎片的测试作为一个组在同一 `Instrumentation` 实例下运行。每个分片由一个索引号进行标识。运行测试时，使用 `-e numShards` 选项指定要创建的独立分片数量，并使用 `-e shardIndex` 选项指定要运行哪个分片。

例如，要将测试套件拆分成 10 个分片，且仅运行第二个碎片中的测试，请使用以下命令：

```
adb shell am instrument -w -e numShards 10 -e shardIndex 2
```

要详细了解如何使用此测试运行器，请参阅 [API 参考](#)。

## Espresso

Espresso 测试框架提供了一组 API 来构建 UI 测试，用于测试应用中的用户流。利用这些 API，您可以编写简洁、运行可靠的自动化 UI 测试。Espresso 非常适合编写白盒自动化测试，其中测试代码将利用所测试应用的实现代码详情。

Espresso 测试框架的主要功能包括：

- 灵活的 API，用于目标应用中的视图和适配器匹配。如需了解详细信息，请参阅[视图匹配](#)。
- 一组丰富的操作 API，用于自动化 UI 交互。如需了解详细信息，请参阅[操作 API](#)。
- UI 线程同步，用于提升测试可靠性。如需了解详细信息，请参阅[UI 线程同步](#)。

要求 Android 2.2 (API 级别 8) 或更高版本。

## 视图匹配

利用 `Espresso.onView()` 方法，您可以访问目标应用中的 UI 组件并与之交互。此方法接受 `Matcher` 参数并搜索视图层次结构，以找到符合给定条件的相应 `View` 实例。您可以通过指定以下条件来优化搜索：

- 视图的类名称
- 视图的内容描述
- 视图的 `R.id`
- 在视图中显示的文本

例如，要找到 ID 值为 `my_button` 的按钮，可以指定如下匹配器：

```
onView(withId(R.id.my_button));
```

如果搜索成功，`onView()` 方法将返回一个引用，让您可以执行用户操作并基于目标视图对断言进行测试。

## 适配器匹配

在 `AdapterView` 布局中，布局在运行时由子视图动态填充。如果目标视图位于某个布局内部，而该布局是从 `AdapterView` (例如 `ListView` 或 `GridView`) 派生出的子类，则 `onView()` 方法可能无法工作，因为只有布局视图的子集会加载到当前视图层次结构中。

因此，请使用 `Espresso.onData()` 方法访问目标视图元素。`Espresso.onData()` 方法将返回一个引用，让您可以执行用户操作并根据 `AdapterView` 中的元素对断言进行测试。

## 操作 API

通常情况下，您可以通过根据应用的用户界面执行某些用户交互来测试应用。借助 `ViewActions` API，您可以轻松地实现这些操作的自动化。您可以执行多种 UI 交互，例如：

- 视图点击
- 滑动
- 按下按键和按钮
- 键入文本
- 打开链接

例如，要模拟输入字符串值并按下按钮以提交该值，您可以像下面一样编写自动化测试脚本。`ViewInteraction.perform()` 和 `DataInteraction.perform()` 方法采用一个或多个 `ViewAction` 参数，并以提供的顺序运行操作。

```
// Type text into an EditText view, then close the soft keyboard
onView(withId(R.id.editTextUserInput))
 .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard());

// Press the button to submit the text change
onView(withId(R.id.changeTextBt)).perform(click());
```

## UI 线程同步

由于计时问题，Android 设备上的测试可能随机失败。此测试问题称为 [测试不稳定](#)。在 Espresso 之前，解决方法是在测试中插入足够长的休眠或超时期或添加代码，以便重试失败的操作。Espresso 测试框架可以处理 `Instrumentation` 与 UI 线程之间的同步；这就消除了对之前的计时解决方法的需求，并确保测试操作与断言更可靠地运行。

要详细了解如何使用 Espresso，请参阅 [API 参考](#)和[测试单个应用的 UI 培训](#)。

## UI Automator

UI Automator 测试框架提供了一组 API 来构建 UI 测试，用于在用户应用和系统应用中执行交互。利用 UI Automator API，您可以执行在测试设备中打开“设置”菜单或应用启动器等操作。UI Automator 测试框架非常适合编写黑盒自动化测试，其中的测试代码不依赖于目标应用的内部实现详情。

UI Automator 测试框架的主要功能包括：

- 用于检查布局层次结构的查看器。如需了解详细信息，请参阅 [UI Automator 查看器](#)。
- 在目标设备上检索状态信息并执行操作的 API。如需了解详细信息，请参阅 [访问设备状态](#)。
- 支持跨应用 UI 测试的 API。如需了解详细信息，请参阅 [UI Automator API](#)。

要求 Android 4.3（API 级别 18）或更高版本。

### UI Automator 查看器

`uiautomatorviewer` 工具提供了一个方便的 GUI，可以扫描和分析 Android 设备上当前显示的 UI 组件。您可以使用此工具检查布局层次结构，并查看在设备前台显示的 UI 组件属性。利用此信息，您可以使用 UI Automator（例如，通过创建与特定可见属性匹配的 UI 选择器）创建控制更加精确的测试。

`uiautomatorviewer` 工具位于 `<android-sdk>/tools/` 目录中。

### 访问设备状态

UI Automator 测试框架提供了一个 `UiDevice` 类，用于在目标应用运行的设备上访问和执行操作。您可以调用其方法来访问设备属性，如当前屏幕方向或显示尺寸。`UiDevice` 类还可用于执行以下操作：

- 更改设备旋转
- 按 D-pad 按钮
- 按“返回”、“主屏幕”或“菜单”按钮
- 打开通知栏
- 对当前窗口进行屏幕截图

例如，要模拟按下“主屏幕”按钮，请调用 `UiDevice.pressHome()` 方法。

### UI Automator API

利用 UI Automator API，您可以编写稳健可靠的测试，而无需了解目标应用的实现详情。您可以使用这些 API 在多个应用中捕获和操作 UI 组件：

- `UiCollection`：枚举容器的 UI 元素以便计算子元素个数，或者通过可见的文本或内容描述属性来指代子元素。
- `UiObject`：表示设备上可见的 UI 元素。
- `UiScrollable`：为在可滚动 UI 容器中搜索项目提供支持。
- `UiSelector`：表示在设备上查询一个或多个目标 UI 元素。
- `Configurator`：允许您设置运行 UI Automator 测试所需的关键参数。

例如，以下代码显示了如何编写可在设备中调用默认应用启动器的测试脚本：

```
// Initialize UiDevice instance
mDevice = UiDevice.getInstance(getApplicationContext());

// Perform a short press on the HOME button
mDevice.pressHome();

// Bring up the default launcher by searching for
// a UI component that matches the content-description for the launcher button
UiObject allAppsButton = mDevice
 .findObject(new UiSelector().description("Apps"));

// Perform a click on the button to bring up the launcher
allAppsButton.clickAndWaitForNewWindow();
```

要详细了解如何使用 UI Automator，请参阅 [API 参考](#) 和 [测试多个应用的 UI 培训](#)。

## 测试支持库设置

Android 测试支持库软件包在最新版本的 Android 支持存储库中提供，后者可作为辅助组件通过 Android SDK 管理器下载。

要通过 SDK 管理器下载 Android 支持存储库，请执行以下操作：

1. 启动 [Android SDK 管理器](#)。
2. 在 SDK 管理器窗口中，滚动到 *Packages* 列表末尾，找到 *Extras* 文件夹并展开（如有必要）以显示其内容。
3. 选择 **Android Support Repository** 项。
4. 点击 **Install packages...** 按钮。

下载后，此工具会将支持存储库文件安装到您现有的 Android SDK 目录中。库文件位于 SDK 的以下子目录中：`<sdk>/extras/android/m2repository` 目录。

Android 测试支持库的类位于 `android.support.test` 软件包中。

要在 Gradle 项目中使用 Android 测试支持库，请在 `build.gradle` 文件中添加这些依赖关系：

```
dependencies {
 androidTestCompile 'com.android.support.test:runner:0.4'
 // Set this dependency to use JUnit 4 rules
 androidTestCompile 'com.android.support.test:rules:0.4'
 // Set this dependency to build and run Espresso tests
 androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
 // Set this dependency to build and run UI Automator tests
 androidTestCompile 'com.android.support.test.uiautomator:uiautomator-v18:2.1.2'
}
```

要将 `AndroidJUnitRunner` 设置为 Gradle 项目中的默认测试仪器运行器，请在 `build.gradle` 文件中指定此依赖关系：

```
android {
 defaultConfig {
 testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
 }
}
```

强烈建议将 Android 测试支持库与 Android Studio IDE 搭配使用。Android Studio 提供支持测试开发的功能，例如：

- 基于 Gradle 的灵活构建系统，为测试代码提供依赖关系管理支持
- 单一的项目结构，包含单元与仪器测试代码以及应用源代码
- 支持在虚拟或物理设备上通过命令行或图形用户界面部署和运行测试

如需了解有关 Android Studio 及其下载的详细信息，请参阅[下载 Android Studio 和 SDK 工具](#)。



# android.support.test

## Classes

### [InstrumentationRegistry](#)

An exposed registry instance that holds a reference to the instrumentation running in the process and it's arguments.



# Administration

If you are an enterprise administrator, you can take advantage of APIs and system capabilities to manage Android devices and control access.

## BLOG ARTICLES

---

### Unifying Key Store Access in ICS

Android 4.0 (ICS) comes with a number of enhancements that make it easier for people to bring their personal Android devices to work. In this post, we're going to have a look at the key store functionality.



# Device Administration

In this document

- [Device Administration API Overview](#)
  - [How does it work?](#)
  - [Policies](#)
  - [Sample Application](#)
  - [Developing a Device Administration Application](#)
    - [Creating the manifest](#)
    - [Implementing the code](#)

Key classes

- [DeviceAdminReceiver](#)
- [DevicePolicyManager](#)
- [DeviceAdminInfo](#)

Android 2.2 introduces support for enterprise applications by offering the Android Device Administration API. The Device Administration API provides device administration features at the system level. These APIs allow you to create security-aware applications that are useful in enterprise settings, in which IT professionals require rich control over employee devices. For example, the built-in Android Email application has leveraged the new APIs to improve Exchange support. Through the Email application, Exchange administrators can enforce password policies — including alphanumeric passwords or numeric PINs — across devices. Administrators can also remotely wipe (that is, restore factory defaults on) lost or stolen handsets. Exchange users can sync their email and calendar data.

This document is intended for developers who want to develop enterprise solutions for Android-powered devices. It discusses the various features provided by the Device Administration API to provide stronger security for employee devices that are powered by Android.

**Note** For information on building a Work Policy Controller for Android for Work deployments, see [Build a Device Policy Controller](#).

## Device Administration API Overview

Here are examples of the types of applications that might use the Device Administration API:

- Email clients.
- Security applications that do remote wipe.
- Device management services and applications.

## How does it work?

You use the Device Administration API to write device admin applications that users install on their devices. The device admin application enforces the desired policies. Here's how it works:

- A system administrator writes a device admin application that enforces remote/local device security policies. These policies could be hard-coded into the app, or the application could dynamically fetch policies from a third-party server.
- The application is installed on users' devices. Android does not currently have an automated provisioning solution. Some of the ways a sysadmin might distribute the application to users are as follows:

- Google Play.
- Enabling installation from another store.
- Distributing the application through other means, such as email or websites.
- The system prompts the user to enable the device admin application. How and when this happens depends on how the application is implemented.
- Once users enable the device admin application, they are subject to its policies. Complying with those policies typically confers benefits, such as access to sensitive systems and data.

If users do not enable the device admin app, it remains on the device, but in an inactive state. Users will not be subject to its policies, and they will conversely not get any of the application's benefits—for example, they may not be able to sync data.

If a user fails to comply with the policies (for example, if a user sets a password that violates the guidelines), it is up to the application to decide how to handle this. However, typically this will result in the user not being able to sync data.

If a device attempts to connect to a server that requires policies not supported in the Device Administration API, the connection will not be allowed. The Device Administration API does not currently allow partial provisioning. In other words, if a device (for example, a legacy device) does not support all of the stated policies, there is no way to allow the device to connect.

If a device contains multiple enabled admin applications, the strictest policy is enforced. There is no way to target a particular admin application.

To uninstall an existing device admin application, users need to first unregister the application as an administrator.

## Policies

In an enterprise setting, it's often the case that employee devices must adhere to a strict set of policies that govern the use of the device. The Device Administration API supports the policies listed in Table 1. Note that the Device Administration API currently only supports passwords for screen lock:

**Table 1.** Policies supported by the Device Administration API.

Policy	Description
Password enabled	Requires that devices ask for PIN or passwords.
Minimum password length	Set the required number of characters for the password. For example, you can require PIN or passwords to have at least six characters.
Alphanumeric password required	Requires that passwords have a combination of letters and numbers. They may include symbolic characters.
Complex password required	Requires that passwords must contain at least a letter, a numerical digit, and a special symbol. Introduced in Android 3.0.
Minimum letters required in password	The minimum number of letters required in the password for all admins or a particular one. Introduced in Android 3.0.
Minimum lowercase letters required in password	The minimum number of lowercase letters required in the password for all admins or a particular one. Introduced in Android 3.0.
Minimum non-letter characters required in password	The minimum number of non-letter characters required in the password for all admins or a particular one. Introduced in Android 3.0.
Minimum numerical digits required in	The minimum number of numerical digits required in the password for all admins or a particular one. Introduced in Android 3.0.

password	
Minimum symbols required in password	The minimum number of symbols required in the password for all admins or a particular one. Introduced in Android 3.0.
Minimum uppercase letters required in password	The minimum number of uppercase letters required in the password for all admins or a particular one. Introduced in Android 3.0.
Password expiration timeout	When the password will expire, expressed as a delta in milliseconds from when a device admin sets the expiration timeout. Introduced in Android 3.0.
Password history restriction	This policy prevents users from reusing the last $n$ unique passwords. This policy is typically used in conjunction with <code>setPasswordExpirationTimeout()</code> , which forces users to update their passwords after a specified amount of time has elapsed. Introduced in Android 3.0.
Maximum failed password attempts	Specifies how many times a user can enter the wrong password before the device wipes its data. The Device Administration API also allows administrators to remotely reset the device to factory defaults. This secures data in case the device is lost or stolen.
Maximum inactivity time lock	Sets the length of time since the user last touched the screen or pressed a button before the device locks the screen. When this happens, users need to enter their PIN or passwords again before they can use their devices and access data. The value can be between 1 and 60 minutes.
Require storage encryption	Specifies that the storage area should be encrypted, if the device supports it. Introduced in Android 3.0.
Disable camera	Specifies that the camera should be disabled. Note that this doesn't have to be a permanent disabling. The camera can be enabled/disabled dynamically based on context, time, and so on. Introduced in Android 4.0.

## Other features

In addition to supporting the policies listed in the above table, the Device Administration API lets you do the following:

- Prompt user to set a new password.
- Lock device immediately.
- Wipe the device's data (that is, restore the device to its factory defaults).

## Sample Application

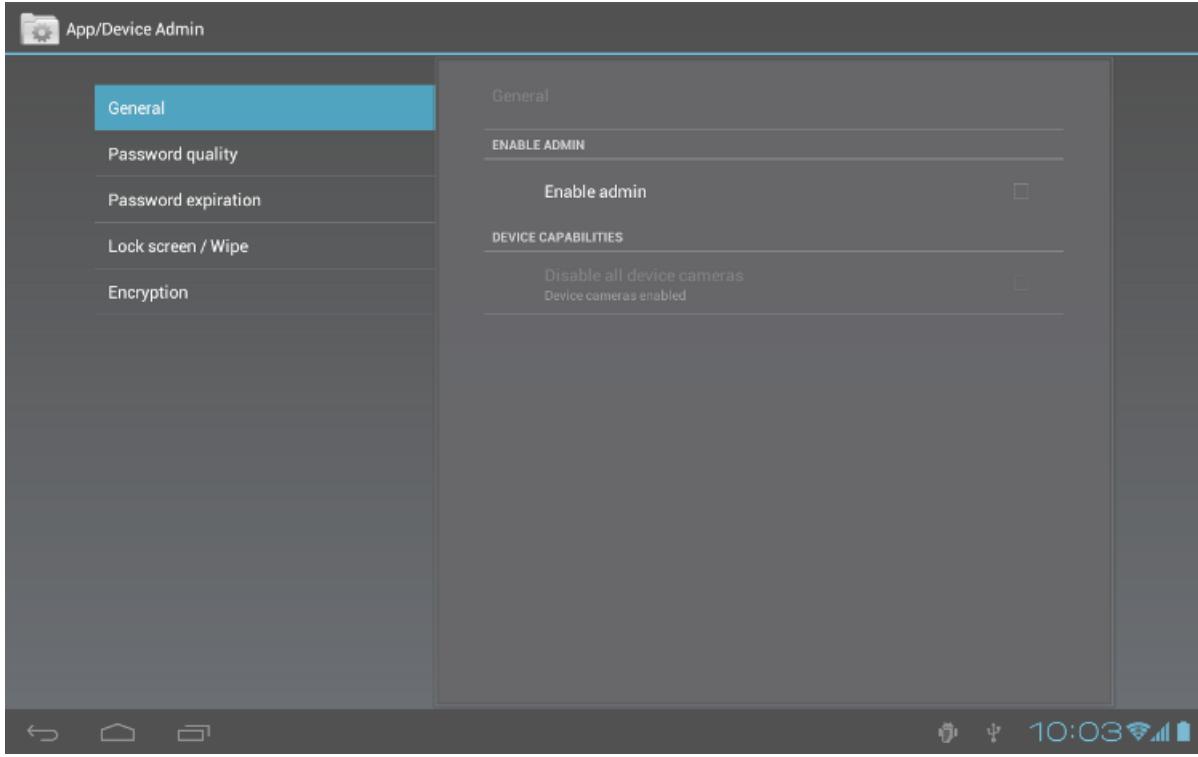
The examples used in this document are based on the Device Administration API sample, which is included in the SDK samples (available through the Android SDK Manager) and located on your system as

`<sdk_root>/ApiDemos/app/src/main/java/com/example/android/apis/app/DeviceAdminSample.java`.

The sample application offers a demo of device admin features. It presents users with a user interface that lets them enable the device admin application. Once they've enabled the application, they can use the buttons in the user interface to do the following:

- Set password quality.
- Specify requirements for the user's password, such as minimum length, the minimum number of numeric characters it must contain, and so on.
- Set the password. If the password does not conform to the specified policies, the system returns an error.
- Set how many failed password attempts can occur before the device is wiped (that is, restored to factory settings).
- Set how long from now the password will expire.
- Set the password history length (*length* refers to number of old passwords stored in the history). This prevents users from reusing one of the last  $n$  passwords they previously used.

- Specify that the storage area should be encrypted, if the device supports it.
- Set the maximum amount of inactive time that can elapse before the device locks.
- Make the device lock immediately.
- Wipe the device's data (that is, restore factory settings).
- Disable the camera.



**Figure 1.** Screenshot of the Sample Application

## Developing a Device Administration Application

System administrators can use the Device Administration API to write an application that enforces remote/local device security policy enforcement. This section summarizes the steps involved in creating a device administration application.

### Creating the manifest

To use the Device Administration API, the application's manifest must include the following:

- A subclass of `DeviceAdminReceiver` that includes the following:
  - The `BIND_DEVICE_ADMIN` permission.
  - The ability to respond to the `ACTION_DEVICE_ADMIN_ENABLED` intent, expressed in the manifest as an intent filter.
- A declaration of security policies used in metadata.

Here is an excerpt from the Device Administration sample manifest:

```

<activity android:name=".app.DeviceAdminSample"
 android:label="@string/activity_sample_device_admin">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.SAMPLE_CODE" />
 </intent-filter>
</activity>
<receiver android:name=".app.DeviceAdminSample$DeviceAdminSampleReceiver"
 android:label="@string/sample_device_admin"
 android:description="@string/sample_device_admin_description"
 android:permission="android.permission.BIND_DEVICE_ADMIN">
 <meta-data android:name="android.app.device_admin"
 android:resource="@xml/device_admin_sample" />
 <intent-filter>
 <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
 </intent-filter>
</receiver>

```

Note that:

- The following attributes refer to string resources that for the sample application reside in `ApiDemos/res/values/strings.xml`. For more information about resources, see [Application Resources](#).
  - `android:label="@string/activity_sample_device_admin"` refers to the user-readable label for the activity.
  - `android:label="@string/sample_device_admin"` refers to the user-readable label for the permission.
  - `android:description="@string/sample_device_admin_description"` refers to the user-readable description of the permission. A descriptor is typically longer and more informative than a label.
- `android:permission="android.permission.BIND_DEVICE_ADMIN"` is a permission that a `DeviceAdminReceiver` subclass must have, to ensure that only the system can interact with the receiver (no application can be granted this permission). This prevents other applications from abusing your device admin app.
- `android.app.action.DEVICE_ADMIN_ENABLED` is the primary action that a `DeviceAdminReceiver` subclass must handle to be allowed to manage a device. This is set to the receiver when the user enables the device admin app. Your code typically handles this in `onEnabled()`. To be supported, the receiver must also require the `BIND_DEVICE_ADMIN` permission so that other applications cannot abuse it.
- When a user enables the device admin application, that gives the receiver permission to perform actions in response to the broadcast of particular system events. When suitable event arises, the application can impose a policy. For example, if the user attempts to set a new password that doesn't meet the policy requirements, the application can prompt the user to pick a different password that does meet the requirements.
- Avoid changing the receiver name after publishing your app. If the name in the manifest changes, device admin is disabled when users update the app. To learn more, see [`<receiver>`](#).
- `android:resource="@xml/device_admin_sample"` declares the security policies used in metadata. The metadata provides additional information specific to the device administrator, as parsed by the `DeviceAdminInfo` class. Here are the contents of `device_admin_sample.xml`:

```

<device-admin xmlns:android="http://schemas.android.com/apk/res/android">
 <uses-policies>
 <limit-password />
 <watch-login />
 <reset-password />
 <force-lock />
 <wipe-data />
 <expire-password />
 <encrypted-storage />
 <disable-camera />
 </uses-policies>
</device-admin>

```

When designing your device administration application, you don't need to include all of the policies, just the ones that are relevant for your

app.

For more discussion of the manifest file, see the [Android Developers Guide](#).

## Implementing the code

The Device Administration API includes the following classes:

### [DeviceAdminReceiver](#)

Base class for implementing a device administration component. This class provides a convenience for interpreting the raw intent actions that are sent by the system. Your Device Administration application must include a [DeviceAdminReceiver](#) subclass.

### [DevicePolicyManager](#)

A class for managing policies enforced on a device. Most clients of this class must have published a [DeviceAdminReceiver](#) that the user has currently enabled. The [DevicePolicyManager](#) manages policies for one or more [DeviceAdminReceiver](#) instances

### [DeviceAdminInfo](#)

This class is used to specify metadata for a device administrator component.

These classes provide the foundation for a fully functional device administration application. The rest of this section describes how you use the [DeviceAdminReceiver](#) and [DevicePolicyManager](#) APIs to write a device admin application.

### Subclassing DeviceAdminReceiver

To create a device admin application, you must subclass [DeviceAdminReceiver](#). The [DeviceAdminReceiver](#) class consists of a series of callbacks that are triggered when particular events occur.

In its [DeviceAdminReceiver](#) subclass, the sample application simply displays a [Toast](#) notification in response to particular events. For example:

```
public class DeviceAdminSample extends DeviceAdminReceiver {

 void showToast(Context context, String msg) {
 String status = context.getString(R.string.admin_receiver_status, msg);
 Toast.makeText(context, status, Toast.LENGTH_SHORT).show();
 }

 @Override
 public void onEnabled(Context context, Intent intent) {
 showToast(context, context.getString(R.string.admin_receiver_status_enabled));
 }

 @Override
 public CharSequence onDisableRequested(Context context, Intent intent) {
 return context.getString(R.string.admin_receiver_status_disable_warning);
 }

 @Override
 public void onDisabled(Context context, Intent intent) {
 showToast(context, context.getString(R.string.admin_receiver_status_disabled));
 }

 @Override
 public void onPasswordChanged(Context context, Intent intent) {
 showToast(context, context.getString(R.string.admin_receiver_status_pw_changed));
 }
}
```

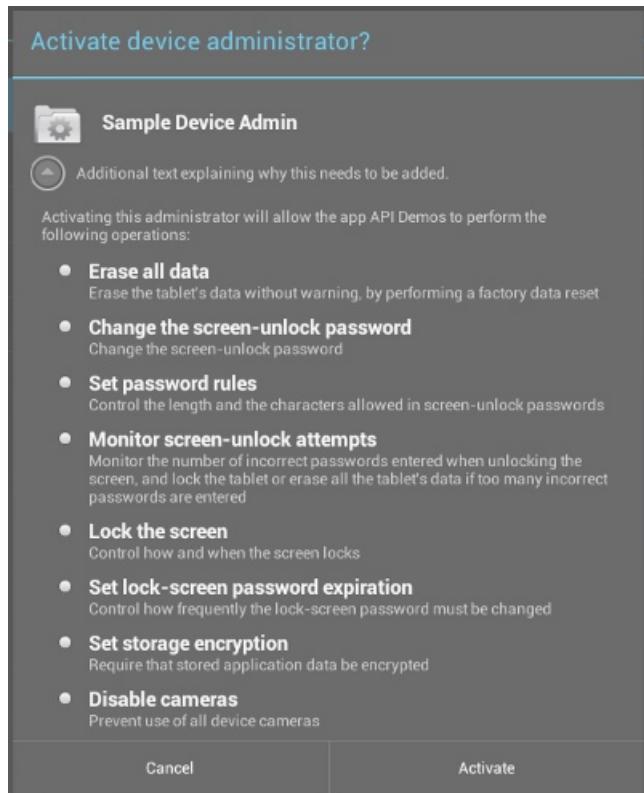
### Enabling the application

One of the major events a device admin application has to handle is the user enabling the application. The user must explicitly enable the application for the policies to be enforced. If the user chooses not to enable the application it will still be present on the device, but its policies

will not be enforced, and the user will not get any of the application's benefits.

The process of enabling the application begins when the user performs an action that triggers the `ACTION_ADD_DEVICE_ADMIN` intent. In the sample application, this happens when the user clicks the **Enable Admin** checkbox.

When the user clicks the **Enable Admin** checkbox, the display changes to prompt the user to activate the device admin application, as shown in figure 2.



**Figure 2.** Sample Application: Activating the Application

Below is the code that gets executed when the user clicks the **Enable Admin** checkbox. This has the effect of triggering the `onPreferenceChange()` callback. This callback is invoked when the value of this `Preference` has been changed by the user and is about to be set and/or persisted. If the user is enabling the application, the display changes to prompt the user to activate the device admin application, as shown in figure 2. Otherwise, the device admin application is disabled.

```

@Override
 public boolean onPreferenceChange(Preference preference, Object newValue) {
 if (super.onPreferenceChange(preference, newValue)) {
 return true;
 }
 boolean value = (Boolean) newValue;
 if (preference == mEnableCheckbox) {
 if (value != mAdminActive) {
 if (value) {
 // Launch the activity to have the user enable our admin.
 Intent intent = new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
 intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, mDeviceAdminSample);
 intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,
 mActivity.getString(R.string.add_admin_extra_app_text));
 startActivityForResult(intent, REQUEST_CODE_ENABLE_ADMIN);
 // return false - don't update checkbox until we're really active
 return false;
 } else {
 mDPM.removeActiveAdmin(mDeviceAdminSample);
 enableDeviceCapabilitiesArea(false);
 mAdminActive = false;
 }
 }
 } else if (preference == mDisableCameraCheckbox) {
 mDPM.setCameraDisabled(mDeviceAdminSample, value);
 ...
 }
 return true;
 }
}

```

The line `intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, mDeviceAdminSample)` states that `mDeviceAdminSample` (which is a `DeviceAdminReceiver` component) is the target policy. This line invokes the user interface shown in figure 2, which guides users through adding the device administrator to the system (or allows them to reject it).

When the application needs to perform an operation that is contingent on the device admin application being enabled, it confirms that the application is active. To do this it uses the `DevicePolicyManager` method `isAdminActive()`. Notice that the `DevicePolicyManager` method `isAdminActive()` takes a `DeviceAdminReceiver` component as its argument:

```

DevicePolicyManager mDPM;
...
private boolean isActiveAdmin() {
 return mDPM.isAdminActive(mDeviceAdminSample);
}

```

## Managing policies

`DevicePolicyManager` is a public class for managing policies enforced on a device. `DevicePolicyManager` manages policies for one or more `DeviceAdminReceiver` instances.

You get a handle to the `DevicePolicyManager` as follows:

```

DevicePolicyManager mDPM =
 (DevicePolicyManager) getSystemService(Context.DEVICE_POLICY_SERVICE);

```

This section describes how to use `DevicePolicyManager` to perform administrative tasks:

- Set password policies
- Set device lock
- Perform data wipe

## Set password policies

`DevicePolicyManager` includes APIs for setting and enforcing the device password policy. In the Device Administration API, the password only applies to screen lock. This section describes common password-related tasks.

### Set a password for the device

This code displays a user interface prompting the user to set a password:

```
Intent intent = new Intent(DevicePolicyManager.ACTION_SET_NEW_PASSWORD);
startActivity(intent);
```

### Set the password quality

The password quality can be one of the following `DevicePolicyManager` constants:

`PASSWORD_QUALITY_ALPHABETIC`

The user must enter a password containing at least alphabetic (or other symbol) characters.

`PASSWORD_QUALITY_ALPHANUMERIC`

The user must enter a password containing at least *both* numeric *and* alphabetic (or other symbol) characters.

`PASSWORD_QUALITY_NUMERIC`

The user must enter a password containing at least numeric characters.

`PASSWORD_QUALITY_COMPLEX`

The user must have entered a password containing at least a letter, a numerical digit and a special symbol.

`PASSWORD_QUALITY_SOMETHING`

The policy requires some kind of password, but doesn't care what it is.

`PASSWORD_QUALITY_UNSPECIFIED`

The policy has no requirements for the password.

For example, this is how you would set the password policy to require an alphanumeric password:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
mDPM.setPasswordQuality(mDeviceAdminSample, DevicePolicyManager.PASSWORD_QUALITY_ALPHANUMERIC);
```

### Set password content requirements

Beginning with Android 3.0, the `DevicePolicyManager` class includes methods that let you fine-tune the contents of the password. For example, you could set a policy that states that passwords must contain at least *n* uppercase letters. Here are the methods for fine-tuning a password's contents:

- `setPasswordMinimumLetters()`
- `setPasswordMinimumLowerCase()`
- `setPasswordMinimumUpperCase()`
- `setPasswordMinimumNonLetter()`
- `setPasswordMinimumNumeric()`
- `setPasswordMinimumSymbols()`

For example, this snippet states that the password must have at least 2 uppercase letters:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwMinUppercase = 2;
...
mDPM.setPasswordMinimumUpperCase(mDeviceAdminSample, pwMinUppercase);
```

## Set the minimum password length

You can specify that a password must be at least the specified minimum length. For example:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwLength;
...
mDPM.setPasswordMinimumLength(mDeviceAdminSample, pwLength);
```

## Set maximum failed password attempts

You can set the maximum number of allowed failed password attempts before the device is wiped (that is, reset to factory settings). For example:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int maxFailedPw;
...
mDPM.setMaximumFailedPasswordsForWipe(mDeviceAdminSample, maxFailedPw);
```

## Set password expiration timeout

Beginning with Android 3.0, you can use the `setPasswordExpirationTimeout()` method to set when a password will expire, expressed as a delta in milliseconds from when a device admin sets the expiration timeout. For example:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
long pwExpiration;
...
mDPM.setPasswordExpirationTimeout(mDeviceAdminSample, pwExpiration);
```

## Restrict password based on history

Beginning with Android 3.0, you can use the `setPasswordHistoryLength()` method to limit users' ability to reuse old passwords. This method takes a *length* parameter, which specifies how many old passwords are stored. When this policy is active, users cannot enter a new password that matches the last *n* passwords. This prevents users from using the same password over and over. This policy is typically used in conjunction with `setPasswordExpirationTimeout()`, which forces users to update their passwords after a specified amount of time has elapsed.

For example, this snippet prohibits users from reusing any of their last 5 passwords:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwHistoryLength = 5;
...
mDPM.setPasswordHistoryLength(mDeviceAdminSample, pwHistoryLength);
```

## Set device lock

You can set the maximum period of user inactivity that can occur before the device locks. For example:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
long timeMs = 1000L*Long.parseLong(mTimeout.getText().toString());
mDPM.setMaximumTimeToLock(mDeviceAdminSample, timeMs);
```

You can also programmatically tell the device to lock immediately:

```
DevicePolicyManager mDPM;
mDPM.lockNow();
```

## Perform data wipe

You can use the `DevicePolicyManager` method `wipeData()` to reset the device to factory settings. This is useful if the device is lost or stolen. Often the decision to wipe the device is the result of certain conditions being met. For example, you can use `setMaximumFailedPasswordsForWipe()` to state that a device should be wiped after a specific number of failed password attempts.

You wipe data as follows:

```
DevicePolicyManager mDPM;
mDPM.wipeData(0);
```

The `wipeData()` method takes as its parameter a bit mask of additional options. Currently the value must be 0.

## Disable camera

Beginning with Android 4.0, you can disable the camera. Note that this doesn't have to be a permanent disabling. The camera can be enabled/disabled dynamically based on context, time, and so on.

You control whether the camera is disabled by using the `setCameraDisabled()` method. For example, this snippet sets the camera to be enabled or disabled based on a checkbox setting:

```
private CheckBoxPreference mDisableCameraCheckbox;
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
mDPM.setCameraDisabled(mDeviceAdminSample, mDisableCameraCheckbox.isChecked());
```

## Storage encryption

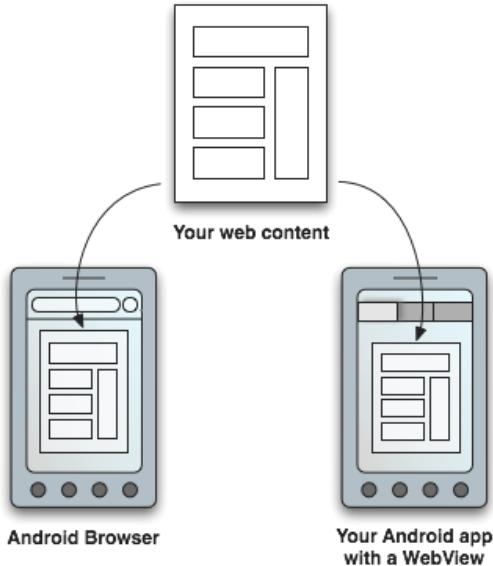
Beginning with Android 3.0, you can use the `setStorageEncryption()` method to set a policy requiring encryption of the storage area, where supported.

For example:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
mDPM.setStorageEncryption(mDeviceAdminSample, true);
```

See the Device Administration API sample for a complete example of how to enable storage encryption.

# Web Apps



**Figure 1.** You can make your web content available to users in two ways: in a traditional web browser and in an Android application, by including a WebView in the layout.

There are essentially two ways to deliver an application on Android: as a client-side application (developed using the Android SDK and installed on user devices in an APK) or as a web application (developed using web standards and accessed through a web browser—there's nothing to install on user devices).

If you chose to provide a web-based app for Android-powered devices, you can rest assured that major web browsers for Android (and the [WebView](#) framework) allow you to specify viewport and style properties that make your web pages appear at the proper size and scale on all screen configurations.

Figure 1 illustrates how you can provide access to your web pages from either a web browser or your own Android app. However, you shouldn't develop an Android app simply as a means to view your web site. Rather, the web pages you embed in your Android app should be designed especially for that environment. You can even define an interface between your Android application and your web pages that allows JavaScript in the web pages to call upon APIs in your Android application—providing Android APIs to your web-based application.

To start developing web pages for Android-powered devices, see the following documents:

## [Supporting Different Screens from Web Apps](#)

How to properly size your web app on Android-powered devices and support multiple screen densities. The information in this document is important if you're building a web application that you at least expect to be available on Android-powered devices (which you should assume for anything you publish on the web), but especially if you're targeting mobile devices or using [WebView](#).

## [Building Web Apps in WebView](#)

How to embed web pages into your Android application using [WebView](#) and bind JavaScript to Android APIs.

## [Debugging Web Apps](#)

How to debug web apps using JavaScript Console APIs.

## **Best Practices for Web Apps**

A list of practices you should follow, in order to provide an effective web application on Android-powered devices.



# Supporting Different Screens in Web Apps

In this document

- › [Using Viewport Metadata](#)
  - › [Defining the viewport size](#)
  - › [Defining the viewport scale](#)
  - › [Defining the viewport target density](#)
- › [Targeting Device Density with CSS](#)
- › [targeting Device Density with JavaScript](#)

See also

- › [Pixel-Perfect UI in the WebView](#)
- › [Creating a Mobile-First Responsive Web Design](#)
- › [High DPI Images for Variable Pixel Densities](#)

Because Android is available on devices with a variety of screen sizes and pixel densities, you should account for these factors in your web design so your web pages always appear at the appropriate size.

When targeting your web pages for Android devices, there are two major factors that you should account for:

## The viewport

The viewport is the rectangular area that provides a drawable region for your web page. You can specify several viewport properties, such as its size and initial scale. Most important is the view port width, which defines the total number of horizontal pixels available from the web page's point of view (the number of CSS pixels available).

## The screen density

The [WebView](#) class and most web browsers on Android convert your CSS pixel values to density-independent pixel values, so your web page appears at the same perceivable size as a medium-density screen (about 160dpi). However, if graphics are an important element of your web design, you should pay close attention to the scaling that occurs on different densities, because a 300px-wide image on a 320dpi screen will be scaled up (using more physical pixels per CSS pixel), which can produce artifacts (blurring and pixelation).

## Specifying Viewport Properties

The viewport is the area in which your web page is drawn. Although the viewport's total visible area matches the size of the screen when zoomed all the way out, the viewport has its own pixel dimensions that it makes available to a web page. For example, although a device screen might have physical a width of 480 pixels, the viewport can have a width of 800 pixels. This allows a web page designed at 800 pixels wide to be completely visible on the screen when the viewport scale is 1.0. Most web browsers on Android (including Chrome) set the viewport to a large size by default (known as "wide viewport mode" at about 980px wide). Many browsers also zoom out as far as possible, by default, to show the full viewport width (known as "overview mode").

**Note:** When your page is rendered in a [WebView](#), it does not use wide viewport mode (the page appears at full zoom) by default. You can enable wide viewport mode with `setUseWideViewPort()`.

You can define properties of the viewport for your web page, such as the width and initial zoom level, using the `<meta name="viewport" ...>` tag in your document `<head>`.

The following syntax shows all of the supported viewport properties and the types of values accepted by each one:

```
<meta name="viewport"
 content="
 height = [pixel_value | "device-height"] ,
 width = [pixel_value | "device-width"] ,
 initial-scale = float_value ,
 minimum-scale = float_value ,
 maximum-scale = float_value ,
 user-scalable = ["yes" | "no"]
 " />
```

For example, the following `<meta>` tag specifies that the viewport width should exactly match the device screen's width and that the ability to zoom should be disabled:

```
<head>
 <title>Example</title>
 <meta name="viewport" content="width=device-width, user-scalable=no" />
</head>
```

When optimizing your site for mobile devices, you should usually set the width to `"device-width"` so the size fits exactly on all devices, then use CSS media queries to flexibly adapt layouts to suit different screen sizes.

**Note:** You should disable user scaling only when you're certain that your web page layout is flexible and the content will fit the width of small screens.

## Targeting Device Density with CSS

The Android Browser and `WebView` support a CSS media feature that allows you to create styles for specific screen densities—the `-webkit-device-pixel-ratio` CSS media feature. The value you apply to this feature should be either "0.75", "1", or "1.5", to indicate that the styles are for devices with low density, medium density, or high density screens, respectively.

For example, you can create separate stylesheets for each density:

```
<link rel="stylesheet" media="screen and (-webkit-device-pixel-ratio: 1.5)" href="hdpi.css" />
<link rel="stylesheet" media="screen and (-webkit-device-pixel-ratio: 1.0)" href="mdpi.css" />
```

Or, specify the different styles in one stylesheet:

```
#header {
 background:url(medium-density-image.png);
}

@media screen and (-webkit-device-pixel-ratio: 1.5) {
 /* CSS for high-density screens */
 #header {
 background:url(high-density-image.png);
 }
}

@media screen and (-webkit-device-pixel-ratio: 0.75) {
 /* CSS for low-density screens */
 #header {
 background:url(low-density-image.png);
 }
}
```

For more information about handling different screen densities, especially images, see [High DPI Images for Variable Pixel Densities](#).

# Targeting Device Density with JavaScript

The Android Browser and [WebView](#) support a DOM property that allows you to query the density of the current device—the `window.devicePixelRatio` DOM property. The value of this property specifies the scaling factor used for the current device. For example, if the value of `window.devicePixelRatio` is "1.0", then the device is considered a medium density device and no scaling is applied by default; if the value is "1.5", then the device is considered a high density device and the page is scaled 1.5x by default; if the value is "0.75", then the device is considered a low density device and the page is scaled 0.75x by default. Of course, the scaling that the Android Browser and [WebView](#) apply is based on the web page's target density—as described in the section about [Defining the viewport target density](#), the default target is medium-density, but you can change the target to affect how your web page is scaled for different screen densities.

For example, here's how you can query the device density with JavaScript:

```
if (window.devicePixelRatio == 1.5) {
 alert("This is a high-density screen");
} else if (window.devicePixelRatio == 0.75) {
 alert("This is a low-density screen");
}
```



# Building Web Apps in WebView

## Quickview

- Use [WebView](#) to display web pages in your Android application layout
- You can create interfaces from your JavaScript to your client-side Android code

## In this document

- [Adding a WebView to Your Application](#)
- [Using JavaScript in WebView](#)
- [Enabling JavaScript](#)
- [Binding JavaScript code to Android code](#)
- [Handling Page Navigation](#)
- [Navigating web page history](#)

## Key classes

- [WebView](#)
- [WebSettings](#)
- [WebViewClient](#)

If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using [WebView](#). The [WebView](#) class is an extension of Android's [View](#) class that allows you to display web pages as a part of your activity layout. It does *not* include any features of a fully developed web browser, such as navigation controls or an address bar. All that [WebView](#) does, by default, is show a web page.

A common scenario in which using [WebView](#) is helpful is when you want to provide information in your application that you might need to update, such as an end-user agreement or a user guide. Within your Android application, you can create an [Activity](#) that contains a [WebView](#), then use that to display your document that's hosted online.

Another scenario in which [WebView](#) can help is if your application provides data to the user that always requires an Internet connection to retrieve data, such as email. In this case, you might find that it's easier to build a [WebView](#) in your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout. Instead, you can design a web page that's tailored for Android devices and then implement a [WebView](#) in your Android application that loads the web page.

This document shows you how to get started with [WebView](#) and how to do some additional things, such as handle page navigation and bind JavaScript from your web page to client-side code in your Android application.

## Adding a WebView to Your Application

To add a [WebView](#) to your Application, simply include the `<WebView>` element in your activity layout. For example, here's a layout file in which the [WebView](#) fills the screen:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/webview"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
/>
```

To load a web page in the [WebView](#), use [loadUrl\(\)](#). For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

Before this will work, however, your application must have access to the Internet. To get Internet access, request the `INTERNET` permission in your manifest file. For example:

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

That's all you need for a basic `WebView` that displays a web page.

## Using JavaScript in WebView

If the web page you plan to load in your `WebView` use JavaScript, you must enable JavaScript for your `WebView`. Once JavaScript is enabled, you can also create interfaces between your application code and your JavaScript code.

### Enabling JavaScript

JavaScript is disabled in a `WebView` by default. You can enable it through the `WebSettings` attached to your `WebView`. You can retrieve `WebSettings` with `getSettings()`, then enable JavaScript with `setJavaScriptEnabled()`.

For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

`WebSettings` provides access to a variety of other settings that you might find useful. For example, if you're developing a web application that's designed specifically for the `WebView` in your Android application, then you can define a custom user agent string with `setUserAgentString()`, then query the custom user agent in your web page to verify that the client requesting your web page is actually your Android application.

### Binding JavaScript code to Android code

When developing a web application that's designed specifically for the `WebView` in your Android application, you can create interfaces between your JavaScript code and client-side Android code. For example, your JavaScript code can call a method in your Android code to display a `Dialog`, instead of using JavaScript's `alert()` function.

To bind a new interface between your JavaScript and Android code, call `addJavascriptInterface()`, passing it a class instance to bind to your JavaScript and an interface name that your JavaScript can call to access the class.

For example, you can include the following class in your Android application:

```
public class WebAppInterface {
 Context mContext;

 /** Instantiate the interface and set the context */
 WebAppInterface(Context c) {
 mContext = c;
 }

 /** Show a toast from the web page */
 @JavascriptInterface
 public void showToast(String toast) {
 Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
 }
}
```

**Caution:** If you've set your `targetSdkVersion` to 17 or higher, **you must add the `@JavascriptInterface` annotation** to any method that you want available to your JavaScript (the method must also be public). If you do not provide the annotation, the method is not accessible by your web page when running on Android 4.2 or higher.

In this example, the `WebAppInterface` class allows the web page to create a `Toast` message, using the `showToast()` method.

You can bind this class to the JavaScript that runs in your `WebView` with `addJavascriptInterface()` and name the interface `Android`. For example:

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new WebAppInterface(this), "android");
```

This creates an interface called `Android` for JavaScript running in the `WebView`. At this point, your web application has access to the `WebAppInterface` class. For example, here's some HTML and JavaScript that creates a toast message using the new interface when the user clicks a button:

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')"/>
<script type="text/javascript">
 function showAndroidToast(toast) {
 Android.showToast(toast);
 }
</script>
```

There's no need to initialize the `Android` interface from JavaScript. The `WebView` automatically makes it available to your web page. So, at the click of the button, the `showAndroidToast()` function uses the `Android` interface to call the `WebAppInterface.showToast()` method.

**Note:** The object that is bound to your JavaScript runs in another thread and not in the thread in which it was constructed.

**Caution:** Using `addJavascriptInterface()` allows JavaScript to control your Android application. This can be a very useful feature or a dangerous security issue. When the HTML in the `WebView` is untrustworthy (for example, part or all of the HTML is provided by an unknown person or process), then an attacker can include HTML that executes your client-side code and possibly any code of the attacker's choosing. As such, you should not use `addJavascriptInterface()` unless you wrote all of the HTML and JavaScript that appears in your `WebView`. You should also not allow the user to navigate to other web pages that are not your own, within your `WebView` (instead, allow the user's default browser application to open foreign links—by default, the user's web browser opens all URL links, so be careful only if you handle page navigation as described in the following section).

## Handling Page Navigation

When the user clicks a link from a web page in your `WebView`, the default behavior is for Android to launch an application that handles URLs. Usually, the default web browser opens and loads the destination URL. However, you can override this behavior for your `WebView`, so links open within your `WebView`. You can then allow the user to navigate backward and forward through their web page history that's maintained by your `WebView`.

To open links clicked by the user, simply provide a `WebViewClient` for your `WebView`, using `setWebViewClient()`. For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());
```

That's it. Now all links the user clicks load in your `WebView`.

If you want more control over where a clicked link load, create your own `WebViewClient` that overrides the `shouldOverrideUrlLoading()` method. For example:

```
private class MyWebViewClient extends WebViewClient {
 @Override
 public boolean shouldOverrideUrlLoading(WebView view, String url) {
 if (Uri.parse(url).getHost().equals("www.example.com")) {
 // This is my web site, so do not override; let my WebView load the page
 return false;
 }
 // Otherwise, the link is not for a page on my site, so launch another Activity that handles URLs
 Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
 startActivity(intent);
 return true;
 }
}
```

Then create an instance of this new `WebViewClient` for the `WebView`:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new MyWebViewClient());
```

Now when the user clicks a link, the system calls `shouldOverrideUrlLoading()`, which checks whether the URL host matches a specific domain (as defined above). If it does match, then the method returns false in order to *not* override the URL loading (it allows the `WebView` to load the URL as usual). If the URL host does not match, then an `Intent` is created to launch the default Activity for handling URLs (which resolves to the user's default web browser).

## Navigating web page history

When your `WebView` overrides URL loading, it automatically accumulates a history of visited web pages. You can navigate backward and forward through the history with `goBack()` and `goForward()`.

For example, here's how your `Activity` can use the device *Back* button to navigate backward:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
 // Check if the key event was the Back button and if there's history
 if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack()) {
 myWebView.goBack();
 return true;
 }
 // If it wasn't the Back key or there's no web page history, bubble up to the default
 // system behavior (probably exit the activity)
 return super.onKeyDown(keyCode, event);
}
```

The `canGoBack()` method returns true if there is actually web page history for the user to visit. Likewise, you can use `canGoForward()` to check whether there is a forward history. If you don't perform this check, then once the user reaches the end of the history, `goBack()` or `goForward()` does nothing.

# Migrating to WebView in Android 4.4

In this document

- › [User Agent Changes](#)
- › [Multi-threading and Thread Blocking](#)
- › [Custom URL Handling](#)
- › [Viewport Changes](#)
  - › [Viewport target-densitydpi no longer supported](#)
  - › [Viewport zooms in when small](#)
  - › [Multiple viewport tags not supported](#)
  - › [Default zoom is deprecated](#)
- › [Styling Changes](#)
  - › [The background CSS shorthand overrides background-size](#)
  - › [Sizes are in CSS pixels instead of screen pixels](#)
  - › [NARROW\\_COLUMNS and SINGLE\\_COLUMN no longer supported](#)
- › [Handling Touch Events in JavaScript](#)

Android 4.4 (API level 19) introduces a new version of [WebView](#) that is based on [Chromium](#). This change upgrades [WebView](#) performance and standards support for HTML5, CSS3, and JavaScript to match the latest web browsers. Any apps using [WebView](#) will inherit these upgrades when running on Android 4.4 and higher.

This document describes additional changes to [WebView](#) that you should be aware of if you set your `targetSdkVersion` to "19" or higher.

**Note:** If your `targetSdkVersion` is set to "18" or lower, [WebView](#) operates in "quirks mode" in order to avoid some of the behavior changes described below, as closely as possible—while still providing your app the performance and web standards upgrades. Beware, though, that [single and narrow column layouts](#) and [default zoom levels](#) are **not supported at all** on Android 4.4, and there may be other behavioral differences that have not been identified, so be sure to test your app on Android 4.4 or higher even if you keep your `targetSdkVersion` set to "18" or lower.

To help you work through any issues you may encounter when migrating your app to [WebView](#) in Android 4.4, you can enable remote debugging through Chrome on your desktop by calling `setWebContentsDebuggingEnabled()`. This new feature in [WebView](#) allows you to inspect and analyze your web content, scripts, and network activity while running in a [WebView](#). For more information, see [Remote Debugging on Android](#).

## User Agent Changes

If you serve content to your [WebView](#) based on the user agent, you should be aware of the user agent string has changed slightly and now includes the Chrome version:

```
Mozilla/5.0 (Linux; Android 4.4; Nexus 4 Build/KRT16H) AppleWebKit/537.36
(KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Mobile Safari/537.36
```

If you need to retrieve the user agent but don't need to store it for your app or do not want to instantiate [WebView](#), you should use the static method, `getDefaultUserAgent()`. However, if you intend to override the user agent string in your [WebView](#), you may instead want to use `getUserAgentString()`.

## Multi-threading and Thread Blocking

If you call methods on `WebView` from any thread other than your app's UI thread, it can cause unexpected results. For example, if your app uses multiple threads, you can use the `runOnUiThread()` method to ensure your code executes on the UI thread:

```
runOnUiThread(new Runnable() {
 @Override
 public void run() {
 // Code for WebView goes here
 }
});
```

Also be sure that you [never block the UI thread](#). A situation in which some apps make this mistake is while waiting for a JavaScript callback. For example, **do not** use code like this:

```
// This code is BAD and will block the UI thread
webView.loadUrl("javascript:fn()");
while(result == null) {
 Thread.sleep(100);
}
```

You can instead use a new method, `evaluateJavascript()`, to run JavaScript asynchronously.

## Custom URL Handling

The new `WebView` applies additional restrictions when requesting resources and resolving links that use a custom URL scheme. For example, if you implement callbacks such as `shouldOverrideUrlLoading()` or `shouldInterceptRequest()`, then `WebView` invokes them only for valid URLs.

If you are using a custom URL scheme or a base URL and notice that your app is receiving fewer calls to these callbacks or failing to load resources on Android 4.4, ensure that the requests specify valid URLs that conform to [RFC 3986](#).

For example, the new `WebView` may not call your `shouldOverrideUrlLoading()` method for links like this:

```
Show Profile
```

The result of the user clicking such a link can vary:

- If you loaded the page by calling  `loadData()` or `loadDataWithBaseUrl()` with an invalid or null base URL, then you will not receive the `shouldOverrideUrlLoading()` callback for this type of link on the page.

**Note:** When you use `loadDataWithBaseUrl()` and the base URL is invalid or set null, all links in the content you are loading must be absolute.

- If you loaded the page by calling `loadUrl()` or provided a valid base URL with `loadDataWithBaseUrl()`, then you will receive the `shouldOverrideUrlLoading()` callback for this type of link on the page, but the URL you receive will be absolute, relative to the current page. For example, the URL you receive will be "`http://www.example.com/showProfile`" instead of just "`showProfile`".

Instead of using a simple string in a link as shown above, you can use a custom scheme such as the following:

```
Show Profile
```

You can then handle this URL in your `shouldOverrideUrlLoading()` method like this:

```
// The URL scheme should be non-hierarchical (no trailing slashes)
private static final String APP_SCHEME = "example-app:";

@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
 if (url.startsWith(APP_SCHEME)) {
 urlData = URLDecoder.decode(url.substring(APP_SCHEME.length()), "UTF-8");
 respondToData(urlData);
 return true;
 }
 return false;
}
```

If you can't alter the HTML then you may be able to use `loadDataWithBaseUrl()` and set a base URL consisting of a custom scheme and a valid host, such as "`example-app://<valid_host_name>/`". For example:

```
webView.loadDataWithBaseUrl("example-app://example.co.uk/", HTML_DATA,
 null, "UTF-8", null);
```

The valid host name should conform to [RFC 3986](#) and it's important to include the trailing slash at the end, otherwise, any requests from the loaded page may be dropped.

## Viewport Changes

### Viewport target-densitydpi no longer supported

Previously, `WebView` supported a viewport property called `target-densitydpi` to help web pages specify their intended screen density. This property is no longer supported and you should migrate to using standard solutions with images and CSS as discussed in [Pixel-Perfect UI in the WebView](#).

### Viewport zooms in when small

Previously, if you set your viewport width to a value less than or equal to "320" it would be set to "device-width", and if you set the viewport height to a value less than or equal to the `WebView` height, it would be set to "device-height". However, when running in the new `WebView`, the width or height value is adhered and the `WebView` zooms in to fill the screen width.

### Multiple viewport tags not supported

Previously, if you included multiple viewport tags in a web page, `WebView` would merge the properties from all the tags. In the new `WebView`, only the last viewport is used and all others are ignored.

### Default zoom is deprecated

The methods `getDefaultZoom()` and `setDefaultZoom()` for getting and setting the initial zoom level on a page have are no longer supported and you should instead define the appropriate viewport in the web page.

**Caution:** These APIs are not supported on Android 4.4 and higher at all. Even if your `targetSdkVersion` is set to "18" or lower, these APIs have no effect.

For information about how to define the viewport properties in your HTML, read [Pixel-Perfect UI in the WebView](#).

If you cannot set the width of the viewport in the HTML, then you should call `setUseWideViewPort(true)` to ensure the page is given a larger viewport. For example:

```
WebSettings settings = webView.getSettings();
settings.setUseWideViewPort(true);
settings.setLoadWithOverviewMode(true);
```

## Styling Changes

## The background CSS shorthand overrides background-size

Chrome and other browser have behaved this way for a while, but now `WebView` will also override a CSS setting for `background-size` if you also specify the `background` style. For example, the size here will be reset to a default value:

```
.some-class {
 background-size: contain;
 background: url('images/image.png') no-repeat;
}
```

The fix is to simply switch the two properties around.

```
.some-class {
 background: url('images/image.png') no-repeat;
 background-size: contain;
}
```

## Sizes are in CSS pixels instead of screen pixels

Previously, size parameters such as `window.outerWidth` and `window.outerHeight` returned a value in actual screen pixels. In the new `WebView`, these return a value based on CSS pixels.

It's generally bad practice to try and calculate the physical size in pixels for sizing elements or other calculations. However, if you've disabled zooming and the initial-scale is set to 1.0, you can use `window.devicePixelRatio` to get the scale, then multiply the CSS pixel value by that. Instead, you can also [create a JavaScript binding](#) to query the pixel size from the `WebView` itself.

For more information, see [quirksmode.org](#).

## NARROW\_COLUMNS and SINGLE\_COLUMN no longer supported

The `NARROW_COLUMNS` value for `WebSettings.LayoutAlgorithm` is not be supported in the new `WebView`.

**Caution:** These APIs are not supported on Android 4.4 and higher at all. Even if your `targetSdkVersion` is set to "18" or lower, these APIs have no effect.

You can handle this change in the following ways:

- Alter the styles of your application:

If you have control of the HTML and CSS on the page, you may find that altering the design of your content may be the most reliable approach. For example, for screens where you cite licenses, you may want wrap text inside of a `<pre>` tag, which you could do with the following styles:

```
<pre style="word-wrap: break-word; white-space: pre-wrap;">
```

This may be especially helpful if you have not defined the viewport properties for your page.

- Use the new `TEXT_AUTOSIZING` layout algorithm:

If you were using narrow columns as a way to make a broad spectrum of desktop sites more readable on mobile devices and you aren't able to change the HTML content, the new `TEXT_AUTOSIZING` algorithm may be a suitable alternative to `NARROW_COLUMNS`.

Additionally, the `SINGLE_COLUMN` value—which was previously deprecated—is also not supported in the new `WebView`.

## Handling Touch Events in JavaScript

If your web page is directly handling touch events in a `WebView`, be sure you are also handling the `touchcancel` event. There are a few scenarios where `touchcancel` will be called, which can cause problems if not received:

- An element is touched (so `touchstart` and `touchmove` are called) and the page is scrolled, causing a `touchcancel` to be thrown.

- An element is touched (`touchstart` is called) but `event.preventDefault()` is not called, resulting earlier enough that `touchcancel` is thrown (so `WebView` assumes you don't want to consume the touch events).

# Debugging Web Apps

## Quickview

- You can debug your web app using console methods in JavaScript
- If debugging in a custom WebView, you need to implement a callback method to handle debug messages

## In this document

- [Using Console APIs in the Android Browser](#)
- [Using Console APIs in WebView](#)

## See also

- [Remote Debugging on Android](#)
- [Debugging](#)

If you are testing your web app with a device running Android 4.4 or higher, you can remotely debug your web pages in [WebView](#) with Chrome Developer Tools, while continuing to support older versions of Android. For more information, see [Remote Debugging on Android](#).

If you don't have a device running Android 4.4 or higher, you can debug your JavaScript using the `console` JavaScript APIs and view the output messages to logcat. If you're familiar with debugging web pages with Firebug or Web Inspector, then you're probably familiar with using `console` (such as `console.log()`). Android's WebKit framework supports most of the same APIs, so you can receive logs from your web page when debugging in Android's Browser or in your own [WebView](#). This document describes how to use the console APIs for debugging.

## Using Console APIs in the Android Browser

When you call a `console` function (in the DOM's `window.console` object), the output appears in logcat. For example, if your web page executes the following JavaScript:

```
console.log("Hello World");
```

Then the logcat message looks something like this:

```
Console: Hello World http://www.example.com/hello.html :82
```

The format of the message might appear different depending on which version of Android you're using. On Android 2.1 and higher, console messages from the Android Browser are tagged with the name "browser". On Android 1.6 and lower, Android Browser messages are tagged with the name "WebCore".

Android's WebKit does not implement all of the console APIs available in other desktop browsers. You can, however, use the basic text logging functions:

- `console.log(String)`
- `console.info(String)`
- `console.warn(String)`
- `console.error(String)`

## Logcat

Logcat is a tool that dumps a log of system messages. The messages include a stack trace when the device throws an error, as well as log messages written from your application and those written using JavaScript `console` APIs.

To run logcat and view messages, execute `adb logcat` from your Android SDK `tools/` directory, or, from DDMS, select **Device > Run logcat**.

See [Debugging](#) for more information about .

Other console functions don't raise errors, but might not behave the same as what you expect from other web browsers.

## Using Console APIs in WebView

All the console APIs shown above are also supported when debugging in [WebView](#). If you're targeting Android 2.1 (API level 7) and higher, you must provide a [WebChromeClient](#) that implements the [onConsoleMessage\(\)](#) method in order for console messages to appear in logcat. Then, apply the [WebChromeClient](#) to your [WebView](#) with [setWebChromeClient\(\)](#).

For example, to support API level 7, this is how your code for [onConsoleMessage\(String, int, String\)](#) might look:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebChromeClient(new WebChromeClient() {
 public void onConsoleMessage(String message, int lineNumber, String sourceID) {
 Log.d("MyApplication", message + " -- From line "
 + lineNumber + " of "
 + sourceID);
 }
});
```

However, if your lowest supported version is API level 8 or higher, you should instead implement [onConsoleMessage\(ConsoleMessage\)](#). For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebChromeClient(new WebChromeClient() {
 public boolean onConsoleMessage(ConsoleMessage cm) {
 Log.d("MyApplication", cm.message() + " -- From line "
 + cm.lineNumber() + " of "
 + cm.sourceId());
 return true;
 }
});
```

The [ConsoleMessage](#) also includes a [MessageLevel](#) object to indicate the type of console message being delivered. You can query the message level with [messageLevel\(\)](#) to determine the severity of the message, then use the appropriate [Log](#) method or take other appropriate actions.

Whether you're using [onConsoleMessage\(String, int, String\)](#) or [onConsoleMessage\(ConsoleMessage\)](#), when you execute a console method in your web page, Android calls the appropriate [onConsoleMessage\(\)](#) method so you can report the error. For example, with the example code above, a logcat message is printed that looks like this:

```
Hello World -- From line 82 of http://www.example.com/hello.html
```



# Best Practices for Web Apps

## See also

- [Pixel-Perfect UI in the WebView](#)
- [Creating a Mobile-First Responsive Web Design](#)
- [High DPI Images for Variable Pixel Densities](#)

Developing web pages and web applications for mobile devices presents a different set of challenges compared to developing a web page for the typical desktop web browser. To help you get started, the following is a list of practices you should follow in order to provide the most effective web application for Android and other mobile devices.

### 1. Redirect mobile devices to a dedicated mobile version of your web site

There are several ways you can redirect requests to the mobile version of your web site, using server-side redirects. Most often, this is done by "sniffing" the User Agent string provided by the web browser. To determine whether to serve a mobile version of your site, you should simply look for the "mobile" string in the User Agent, which matches a wide variety of mobile devices. If necessary, you can also identify the specific operating system in the User Agent string (such as "Android 2.1").

Note: Large screen Android-powered devices that should be served full-size web sites (such as tablets) do *not* include the "mobile" string in the user agent, while the rest of the user agent string is mostly the same. As such, it's important that you deliver the mobile version of your web site based on whether the "mobile" string exists in the user agent.

### 2. Use a valid markup DOCTYPE that's appropriate for mobile devices

The most common markup language used for mobile web sites is [XHTML Basic](#). This standard ensures specific markup for your web site that works best on mobile devices. For instance, it does not allow HTML frames or nested tables, which perform poorly on mobile devices. Along with the DOCTYPE, be sure to declare the appropriate character encoding for the document (such as UTF-8).

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
 "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

Also be sure that your web page markup is valid against the declared DOCTYPE. Use a validator, such as the one available at <http://validator.w3.org>.

### 3. Use viewport meta data to properly resize your web page

In your document `<head>`, you should provide meta data that specifies how you want the browser's viewport to render your web page. For example, your viewport meta data can specify the height and width for the browser's viewport, the initial web page scale and even the target screen density.

For example:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
```

For more information about how to use viewport meta data for Android-powered devices, read [Targeting Screens from Web Apps](#).

### 4. Avoid multiple file requests

Because mobile devices typically have a connection speed far slower than a desktop computer, you should make your web pages load as fast as possible. One way to speed it up is to avoid loading extra files such as stylesheets and script files in the `<head>`. Instead, provide your CSS and JavaScript directly in the `<head>` (or at the end of the `<body>`, for scripts that you don't need until the page is loaded).

Alternatively, you should optimize the size and speed of your files by compressing them with tools like [Minify](#).

## 5. Use a vertical linear layout

Avoid the need for the user to scroll left and right while navigating your web page. Scrolling up and down is easier for the user and makes your web page simpler.

For a more thorough guide to creating great mobile web applications, see the W3C's [Mobile Web Best Practices](#). For other guidance on improving the speed of your web site (for mobile and desktop), see Yahoo!'s guide to [Exceptional Performance](#) and Google's speed tutorials in [Let's make the web faster](#).



# Best Practices

Design and build apps the right way. Learn how to create apps that look great and perform well on as many devices as possible, from phones to tablets and more.

## BLOG ARTICLES

---

### Improving App Quality

One way of improving your app's visibility in the ecosystem is by deploying well-targeted mobile advertising campaigns and cross-app promotions. However, there's another time-tested method of fueling the impression-install-ranking cycle: improve the product!

### Say Goodbye to the Menu Button

As Ice Cream Sandwich rolls out to more devices, it's important that you begin to migrate your designs to the action bar in order to promote a consistent Android user experience.

### New Tools For Managing Screen Sizes

Android 3.2 includes new tools for supporting devices with a wide range of screen sizes. One important result is better support for a new size of screen; what is typically called a "7-inch" tablet. This release also offers several new APIs to simplify developers' work in adjusting to different screen sizes.

### Identifying App Installations

It is very common, and perfectly reasonable, for a developer to want to track individual installations of their apps. It sounds plausible just to call `TelephonyManager.getDeviceId()` and use that value to identify the installation. There are problems with this

### Making Android Games that Play Nice

Making a game on Android is easy. Making a *great* game for a mobile, multitasking, often multi-core, multi-purpose system like Android is trickier. Even the best developers frequently make mistakes in the way they interact with the Android system and with other applications

# Supporting Tablets and Handsets

In this document

- [Basic Guidelines](#)
- [Creating Single-pane and Multi-pane Layouts](#)
- [Using the Action Bar](#)
  - [Using split action bar](#)
  - [Using "up" navigation](#)
- [Other Design Tips](#)

The Android platform runs on a variety of screen sizes and the system gracefully resizes your application's UI to fit each one. Typically, all you need to do is design your UI to be flexible and optimize some elements for different sizes by providing [alternative resources](#) (such as alternative layouts that reposition some views or alternative dimension values for views). However, sometimes you might want to go a step further to optimize the overall user experience for different screen sizes. For example, tablets offer more space in which your application can present multiple sets of information at once, while a handset device usually requires that you split those sets apart and display them separately. So even though a UI designed for handsets will properly resize to fit a tablet, it does not fully leverage the potential of the tablet's screen to enhance the user experience.

With Android 3.0 (API level 11), Android introduced a new set of framework APIs that allow you to more effectively design activities that take advantage of large screens: the [Fragment](#) APIs. Fragments allow you to separate distinct behavioral components of your UI into separate parts, which you can then combine to create multi-pane layouts when running on a tablet or place in separate activities when running on a handset. Android 3.0 also introduced [ActionBar](#), which provides a dedicated UI at the top of the screen to identify the app and provide user actions and navigation.

This document provides guidance that can help you create an application that offers a unique and optimized user experience on both handsets and tablets, using fragments and the action bar.

Before you continue with this guide, it's important that you first read the guide to [Supporting Multiple Screens](#). That document describes the fundamental design principles for developing a UI that supports different screen sizes and densities with flexible layouts and alternative bitmaps, respectively.

## Basic Guidelines

Here are a few guidelines that will help you create an app that provides an optimized user experience on both tablets and handsets:

- **Build your activity designs based on fragments** that you can reuse in different combinations—in multi-pane layouts on tablets and single-pane layouts on handsets.

A [Fragment](#) represents a behavior or a portion of user interface in an activity. You can think of a fragment as a modular section of an activity (a "fragment" of an activity), which has its own lifecycle and which you can add or remove while the activity is running.

If you haven't used fragments yet, start by reading the [Fragments](#) developer guide.

- **Use the action bar**, but follow best practices and ensure your design is flexible enough for the system to adjust the action bar layout based on the screen size.

The [ActionBar](#) is a UI component for activities that replaces the traditional title bar at the top of the screen. By default, the action bar includes the application logo on the left side, followed by the activity title, and access to items from the options menu on the right side.

You can enable items from the options menu to appear directly in the action bar as "action items". You can also add navigation features to the action bar, such as tabs or a drop-down list, and use the application icon to supplement the system's *Back* button behavior with the

option to navigate to your application's "home" activity or "up" the application's structural hierarchy.

This guide provides some tips for using the action bar in ways that support both tablets and handsets. For a detailed discussion of the action bar APIs, read the [Action Bar](#) developer guide.

- **Implement flexible layouts**, as discussed in the [Best Practices](#) for supporting multiple screens and the blog post, [Thinking Like a Web Designer](#).

A flexible layout design allows your application to adapt to variations in screen sizes. Not all tablets are the same size, nor are all handsets the same size. While you might provide different fragment combinations for "tablets" and "handsets", it's still necessary that each design be flexible to resize to variations in size and aspect ratio.

The following sections discuss the first two recommendations in more detail. For more information about creating flexible layouts, refer to the links provided above.

**Note:** Aside from one feature in the action bar, all the APIs needed to accomplish the recommendations in this document are available in Android 3.0. Additionally, you can even implement the fragment design patterns and remain backward-compatible with Android 1.6, by using the support library—discussed in the side bar below.

## Creating Single-pane and Multi-pane Layouts

The most effective way to create a distinct user experience for tablets and handsets is to create layouts with different combinations of fragments, such that you can design "multi-pane" layouts for tablets and "single-pane" layouts for handsets. For example, a news application on a tablet might show a list of articles on the left side and a full article on the right side—selecting an article on the left updates the article view on the right. On a handset, however, these two components should appear on separate screens—selecting an article from a list changes the entire screen to show that article. There are two techniques to accomplish this design with fragments:

- *Multiple fragments, one activity*: Use one activity regardless of the device size, but decide at runtime whether to combine fragments in the layout (to create a multiple-pane design) or swap fragments (to create a single-pane design). Or...
- *Multiple fragments, multiple activities*: On a tablet, place multiple fragments in one activity; on a handset, use separate activities to host each fragment. For example, when the tablet design uses two fragments in an activity, use the same activity for handsets, but supply an alternative layout that includes just the first fragment. When running on a handset and you need to switch fragments (such as when the user selects an item), start another activity that hosts the second fragment.

The approach you choose depends on your design and personal preferences. The first option (one activity; swapping fragments) requires that you determine the screen size at runtime and dynamically add each fragment as appropriate—rather than declare the fragments in your activity's XML layout—because you *cannot* remove a fragment from an activity if it's been declared in the XML layout. When using the first technique, you might also need to update the action bar each time the fragments change, depending on what actions or navigation modes are available for each fragment. In some cases, these factors might not affect your design, so using one activity and swapping fragments might work well (especially if your tablet design requires that you add fragments dynamically anyway). Other times, however, dynamically swapping fragments for your handset design can make your code more complicated, because you must manage all the fragment combinations in the activity's code (rather than use alternative layout resources to define fragment combinations) and manage the back stack of fragments yourself (rather than allow the normal activity stack to handle back-navigation).

This guide focuses on the second option, in which you display each fragment in a separate activity when on a smaller screen. Using this technique means that you can use alternative layout files that define different fragment combinations for different screen sizes, keep fragment code modular, simplify action bar management, and let the system handle all the back stack work on handsets.

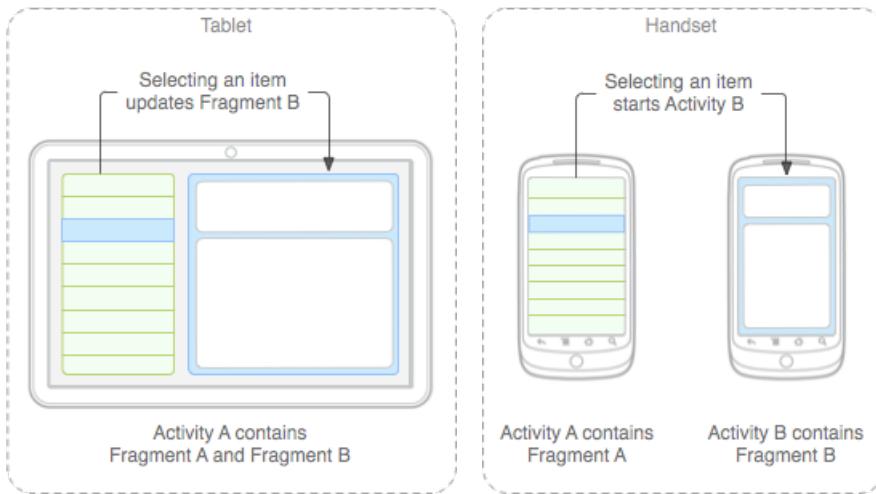
Figure 1 illustrates how an application with two fragments might be arranged for both handsets and tablets when using separate activities for the handset design:

### Remaining backward-compatible

If you want to use fragments in your application *and* remain compatible with versions of Android older than 3.0, you can do so by using the [Android Support Library](#) (downloadable from the SDK Manager).

The support library includes APIs for [fragments](#), [loaders](#), and other APIs added in newer versions of Android. By simply adding this library to your Android project, you can use backward-compatible versions of these APIs in your application and remain compatible with Android 1.6 (your `android:minSdkVersion` value can be as low as "`4`"). For information about how to get the library and start using it, see the [Support Library](#) document.

The support library *does not* provide APIs for the action bar, but you can use code from the sample app, [Action Bar Compatibility](#), to create an action bar that supports all devices.



**Figure 1.** Different design patterns for tablets and handsets when selecting an item to view its details.

In the application shown in figure 1, Activity A is the "main activity" and uses different layouts to display either one or two fragments at a time, depending on the size of the screen:

- On a tablet-sized screen, the Activity A layout contains both Fragment A and Fragment B.
- On a handset-sized screen, the Activity A layout contains only Fragment A (the list view). In order to show the details in Fragment B, Activity B must open.

**Note:** Activity B is never used on a tablet. It is simply a container to present Fragment B, so is only used on handset devices when the two fragments must be displayed separately.

Depending on the screen size, the system applies a different `main.xml` layout file:

`res/layout/main.xml` for handsets:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <!-- "Fragment A" -->
 <fragment class="com.example.android.TitlesFragment"
 android:id="@+id/list_frag"
 android:layout_width="match_parent"
 android:layout_height="match_parent"/>
</FrameLayout>
```

`res/layout-large/main.xml` for tablets:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:id="@+id/frags">
 <!-- "Fragment A" -->
 <fragment class="com.example.android.TitlesFragment"
 android:id="@+id/list_frag"
 android:layout_width="@dimen/titles_size"
 android:layout_height="match_parent"/>
 <!-- "Fragment B" -->
 <fragment class="com.example.android.DetailsFragment"
 android:id="@+id/details_frag"
 android:layout_width="match_parent"
 android:layout_height="match_parent" />
</LinearLayout>
```

**Note:** Although the above sample layout for tablets is based on the "large" screen

**Supporting sizes based on screen width**

configuration qualifier, you should also use the new "minimum width" size qualifiers in order to more precisely control the screen size at which the system applies your handset or tablet layout. See the sidebar for more information.

How the application responds when a user selects an item from the list depends on whether Fragment B is available in the layout:

- If Fragment B is in the layout, Activity A notifies Fragment B to update itself.
- If Fragment B is *not* in the layout, Activity A starts Activity B (which hosts Fragment B).

To implement this pattern for your application, it's important that you develop your fragments to be highly compartmentalized. Specifically, you should follow two guidelines:

- Do not manipulate one fragment directly from another.
- Keep all code that concerns content in a fragment inside that fragment, rather than putting it in the host activity's code.

To avoid directly calling one fragment from another, **define a callback interface in each fragment** class that it can use to deliver events to its host activity, which implements the callback interface. When the activity receives a callback due to an event (such as the user selecting a list item), the activity responds appropriately based on the current fragment configuration.

For example, Activity A from above can handle item selections depending on whether it's using the tablet or handset layout like this:

```
public class MainActivity extends Activity implements TitlesFragment.OnItemSelectedListener {
 ...

 /** This is a callback that the list fragment (Fragment A)
 * calls when a list item is selected */
 public void onItemSelected(int position) {
 DisplayFragment displayFrag = (DisplayFragment) getSupportFragmentManager()
 .findFragmentById(R.id.display_frag);
 if (displayFrag == null) {
 // DisplayFragment (Fragment B) is not in the layout (handset layout),
 // so start DisplayActivity (Activity B)
 // and pass it the info about the selected item
 Intent intent = new Intent(this, DisplayActivity.class);
 intent.putExtra("position", position);
 startActivity(intent);
 } else {
 // DisplayFragment (Fragment B) is in the layout (tablet layout),
 // so tell the fragment to update
 displayFrag.updateContent(position);
 }
 }
}
```

When `DisplayActivity` (Activity B) starts, it reads the data delivered by the `Intent` and passes it to the `DisplayFragment` (Fragment B).

If Fragment B needs to deliver a result back to Fragment A (because Activity B was started with `startActivityForResult()`), then the process works similarly with a callback interface between Fragment B and Activity B. That is, Activity B implements a different callback interface defined by Fragment B. When Activity B receives the callback with a result from the fragment, it sets the result for the activity (with `setResult()`) and finishes itself. Activity A then receives the result and delivers it to Fragment A.

For a demonstration of this technique for creating different fragment combinations for tablets and handsets, see the updated version of the [Honeycomb Gallery](#) sample.

## Using the Action Bar

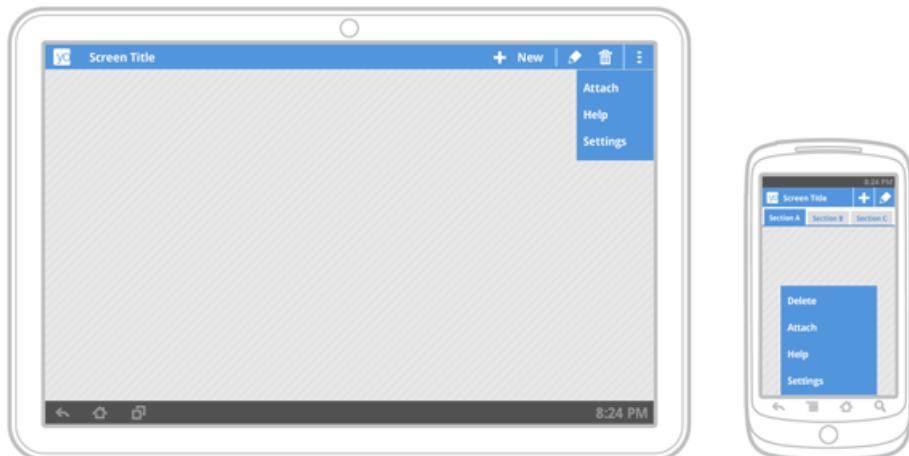
The [Action Bar](#) is an important UI component for Android apps on both tablets and handsets. To ensure that the action bar behaves appropriately on all screen sizes, it's important that you use the [ActionBar](#) APIs without adding complex customizations. By using the standard [ActionBar](#) APIs to design your action bar, the Android system does all the work to gracefully adapt the action bar for different screen sizes. Here are some important tips to follow when creating your action bar:

Android 3.2 (API level 13) adds new APIs that provide more fine-grain control over what screen sizes your app supports and what resources it uses, by declaring screen sizes based on the minimum width your layouts require. For example, both a 5" and 7" device qualify as a "large" screen, so your "large" layout resources are used on both devices. With API level 13, you can distinguish between these two sizes based on the screen width, as measured in density-independent pixels.

For details, read the blog post about [New Tools for Managing Screen Sizes](#).

- When setting a menu item to be an action item, **avoid using the "always" value**. In your [menu resource](#), use "`ifRoom`" for the `android:showAsAction` attribute if you'd like the menu item to appear in the action bar. However, you might need "`always`" when an action view does not provide a default action for the overflow menu (that is, it must appear as an action view). However, you should not use "`always`" more than once or twice. In almost all other cases, use "`ifRoom`" as the value for "`android:showAsAction`" when you want the item to appear as an action item. Forcing too many action items into the action bar can create a cluttered UI and action items may overlap with other action bar elements such as the title or navigation items.
- When adding action items to the action bar with a text title, also **provide an icon**, when appropriate, and declare `showAsAction="ifRoom|withText"`. This way, if there's not enough room for the title, but there is enough room for the icon, then only the icon may be used.
- Always **provide a title** for your action items, even if you don't enable "`withText`", because users can view the title as a "tool-tip" by performing a "long click" on the item—the title text appears momentarily in a toast message. Providing a title is also critical for accessibility, because screen readers read aloud the item title even when not visible.
- Avoid using custom navigation modes when possible**. Use the built-in tab and drop-down navigation modes when possible—they're designed so the system can adapt their presentation to different screen sizes. For example, when the width is too narrow for both tabs and other action items (such as a handset in portrait orientation), the tabs appear below the action bar (this is known as the "stacked action bar"). If you must build a custom navigation mode or other custom views in the action bar, thoroughly test them on smaller screens and make any necessary adjustments to support a narrow action bar.

For example, the mock-ups below demonstrate how the system may adapt an action bar based on the available screen space. On the handset, only two action items fit, so the remaining menu items appear in the overflow menu (because `android:showAsAction` was set to "`ifRoom`") and the tabs appear in a separate row (the stacked action bar). On the tablet, more action items can fit in the action bar and so do the tabs.



**Figure 2.** Mock-up showing how the system re-configures action bar components based on the available screen space.

## Using split action bar

When your application is running on Android 4.0 (API level 14) and higher, there's an extra mode available for the action bar called "split action bar." When you enable split action bar, a separate bar appears at the bottom of the screen to display all action items when the activity is running on a narrow screen (such as a portrait handset). Splitting the action bar ensures that a reasonable amount of space is available to display action items on a narrow screen and also leave room for navigation and title elements at the top.

To enable split action bar, simply add `uiOptions="splitActionBarWhenNarrow"` to your `<activity>` or `<application>` manifest element.



Figure 3. Split action bar with navigation tabs on the left; with the app icon and title disabled on the right.

If you'd like to hide the main action bar at the top, because you're using the built-in navigation tabs along with the split action bar, call `setDisplayHomeAsUpEnabled(false)` to disable the application icon in the action bar. In this case, there's now nothing left in the main action bar, so it disappears and all that's left are the navigation tabs at the top and the action items at the bottom, as shown by the second device in figure 3.

**Note:** Although the `uiOptions` attribute was added in Android 4.0 (API level 14), you can safely include it in your application even if your `minSdkVersion` is set to a value lower than "14" to remain compatible with older versions of Android. When running on older versions, the system simply ignores the attribute because it doesn't understand it. The only condition to adding it to your manifest is that you must compile your application against a platform version that supports API level 14 or higher. Just be sure that you don't openly use other APIs in your application code that aren't supported by the version declared by your `minSdkVersion` attribute.

## Using "up" navigation

As discussed in the [Action Bar](#) developer guide, you can use the application icon in the action bar to facilitate user navigation when appropriate—either as a method to get back to the "home" activity (similar to clicking the logo on a web site) or as a way to navigate up the application's structural hierarchy. Although it might seem similar to the standard *Back* navigation in some cases, the up navigation option provides a more predictable navigation method for situations in which the user may have entered from an external location, such as a notification, app widget, or a different application.

When using fragments in different combinations for different devices, it's important to give extra consideration to how your up navigation behaves in each configuration. For example, when on a handset and your application shows just one fragment at a time, it might be appropriate to enable up navigation to go up to the parent screen, whereas it's not necessary when showing the same fragment in a multi-pane configuration.

For more information about enabling up navigation, see the [Action Bar](#) developer guide.

## Other Design Tips

- When working with a [ListView](#), consider how you might provide more or less information in each list item based on the available space. That is, you can create alternative layouts to be used by the items in your list adapter such that a large screen might display more detail for each item.
- Create [alternative resource files](#) for values such as integers, dimensions, and even booleans. Using size qualifiers for these resources, you can easily apply different layout sizes, font sizes, or enable/disable features based on the current screen size.



# 支持多种屏幕

## 内容快览

- › Android 可在具有不同屏幕尺寸和密度的设备上运行。
- › 显示应用的屏幕可影响其用户界面。
- › 系统会处理大多数工作，使您的应用适应当前屏幕。
- › 您应该创建屏幕特定的资源，以精确控制 UI。

## 本文内容

- › 屏幕支持概览
  - › 术语和概念
  - › 支持的屏幕范围
  - › 密度独立性
- › 如何支持多种屏幕
  - › 使用配置限定符
  - › 设计替代布局和可绘制对象
  - › 声明适用于 Android 3.2 的平板电脑布局
  - › 使用新尺寸限定符
  - › 配置示例
  - › 声明屏幕尺寸支持
- › 最佳做法
- › 其他密度注意事项
  - › 缩放运行时创建的位图对象
  - › 将 dp 单位转换为像素单位
- › 如何在多个屏幕上测试您的应用

## 相关示例

- › 多种分辨率

## 另请参阅

- › 视同 Web Designer
- › 提供备用资源
- › 图标设计指南
- › 管理虚拟设备

Android 可在各种具有不同屏幕尺寸和密度的设备上运行。对于应用，Android 系统在不同设备中提供一致的开发环境，可以处理大多数工作，将每个应用的用户界面调整为适应其显示的屏幕。同时，系统提供 API，可用于控制应用适用于特定屏幕尺寸和密度的 UI，以针对不同屏幕配置优化 UI 设计。例如，您可能想要不同于手机 UI 的平板电脑 UI。

虽然系统为使您的应用适用于不同的屏幕，会进行缩放和大小调整，但您应针对不同的屏幕尺寸和密度优化应用。这样可以最大程度优化所有设备上的用户体验，用户会认为您的应用实际上是专为他们的设备而设计，而不是简单地拉伸以适应其设备屏幕。

按照本文档所述的做法，您可以创建正常显示的应用，然后使用一个 `.apk` 文件在所有支持的屏幕配置中提供优化的用户体验。

**注：**本文档中的信息假设您的应用设计用于 Android 1.6（API 级别 4）或更高级别。如果您的应用只支持 Android 1.5 或更低版本，请先阅读[适用于 Android 1.5 的策略](#)。

另请注意，Android 3.2 引入了新的 API，可用于更精确地控制应用用于不同屏幕尺寸的布局资源。如果您要开发针对平板电脑优化的应

用，这些新功能特别重要。有关详情，请参阅[声明 Android 3.2 的平板电脑布局](#)相关章节。

# 屏幕支持概览

本节概述 Android 对多种屏幕的支持，包括：本文档和 API 中所用术语和概述的简介、系统支持的屏幕配置摘要，以及 API 和基本屏幕兼容性功能的概述。

## 术语和概念

### 屏幕尺寸

按屏幕对角测量的实际物理尺寸。

为简便起见，Android 将所有实际屏幕尺寸分组为四种通用尺寸：小、正常、大和超大。

### 屏幕密度

屏幕物理区域中的像素量；通常称为 dpi（每英寸点数）。例如，与“正常”或“高”密度屏幕相比，“低”密度屏幕在给定物理区域的像素较少。

为简便起见，Android 将所有屏幕密度分组为六种通用密度：低、中、高、超高、超超高和超超超高。

### 方向

从用户视角看屏幕的方向，即横屏还是竖屏，分别表示屏幕的纵横比是宽还是高。请注意，不仅不同的设备默认以不同的方向操作，而且方向在运行时可随着用户旋转设备而改变。

### 分辨率

屏幕上物理像素的总数。添加对多种屏幕的支持时，应用不会直接使用分辨率；而只应关注通用尺寸和密度组指定的屏幕尺寸及密度。

### 密度无关像素 (dp)

在定义 UI 布局时应使用的虚拟像素单位，用于以密度无关方式表示布局维度或位置。

密度无关像素等于 160 dpi 屏幕上的一个物理像素，这是系统为“中”密度屏幕假设的基线密度。在运行时，系统根据使用中屏幕的实际密度按需要以透明方式处理 dp 单位的任何缩放。dp 单位转换为屏幕像素很简单： $px = dp * (dpi / 160)$ 。例如，在 240 dpi 屏幕上，1 dp 等于 1.5 物理像素。在定义应用的 UI 时应始终使用 dp 单位，以确保在不同密度的屏幕上正常显示 UI。

## 支持的屏幕范围

从 Android 1.6 (API 级别 4) 开始，Android 支持多种屏幕尺寸和密度，反映设备可能具有的多种不同屏幕配置。您可以使用 Android 系统的功能优化应用在各种屏幕配置下的用户界面，确保应用不仅正常渲染，而且在每个屏幕上提供最佳的用户体验。

为简化您为多种屏幕设计用户界面的方式，Android 将实际屏幕尺寸和密度的范围分为：

- 四种通用尺寸：小、正常、大和超大

**注：**从 Android 3.2 (API 级别 13) 开始，这些尺寸组已弃用，而采用根据可用屏幕宽度管理屏幕尺寸的新技术。如果为 Android 3.2 和更高版本开发，请参阅[声明适用于 Android 3.2 的平板电脑布局](#)以了解更多信息。

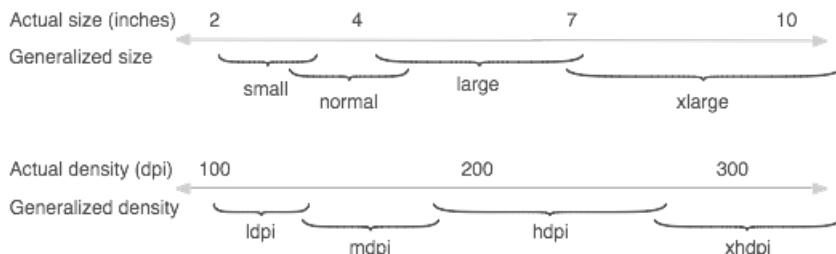
### • 六种通用的密度：

- *ldpi* (低) ~120dpi
- *mdpi* (中) ~160dpi
- *hdpi* (高) ~240dpi
- *xhdpi* (超高) ~320dpi
- *xxhdpi* (超超高) ~480dpi

- *xxxhdpi* (超超超高) ~640dpi

通用的尺寸和密度按照基线配置（即正常尺寸和 *mdpi* (中) 密度）排列。此基线基于第一代 Android 设备 (T-Mobile G1) 的屏幕配置，该设备采用 HVGA 屏幕（在 Android 1.6 之前，这是 Android 支持的唯一屏幕配置）。

每种通用的尺寸和密度都涵盖一个实际屏幕尺寸和密度范围。例如，两部都报告正常屏幕尺寸的设备在手动测量时，实际屏幕尺寸和高宽比可能略有不同。类似地，对于两台报告 *hdpi* 屏幕密度的设备，其实际像素密度可能略有不同。Android 将这些差异抽象概括到应用，使您可以提供为通用尺寸和密度设计的 UI，让系统按需要处理任何最终调整。图 1 说明不同的尺寸和密度如何粗略归类为不同的尺寸 和密度组。



**图 1.** 说明 Android 如何将实际尺寸和密度粗略地 对应到通用的尺寸和密度（数据并不精确）。

在为不同的屏幕尺寸设计 UI 时，您会发现每种设计都需要 最小空间。因此，上述每种通用的屏幕尺寸都关联了系统定义的最低 分辨率。这些最小尺寸以“dp”单位表示 — 在定义布局时应使用相同的单位 — 这样系统无需担心屏幕密度的变化。

- 超大屏幕至少为 960dp x 720dp
- 大屏幕至少为 640dp x 480dp
- 正常屏幕至少为 470dp x 320dp
- 小屏幕至少为 426dp x 320dp

**注：**这些最小屏幕尺寸在 Android 3.0 之前未正确定义，因此某些设备在正常屏幕与大屏幕之间变换时可能会出现分类错误的情况。这些尺寸还基于屏幕的物理分辨率，因此设备之间可能不同 — 例如，具有系统状态栏的 1024x720 平板电脑因系统状态栏要占用空间，所以可供应用使用的空间要小一点。

要针对不同的屏幕尺寸和密度优化应用的 UI，可为任何通用的尺寸和密度提供 [备用资源](#)。通常，应为某些不同的屏幕尺寸提供替代布局，为不同的屏幕密度提供替代位图图像。在运行时，系统会根据当前设备屏幕的通用 尺寸或密度对应用使用适当的资源。

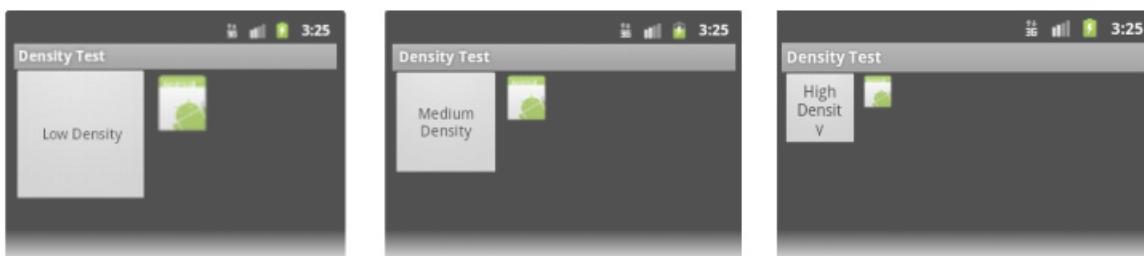
无需为屏幕尺寸和 密度的每个组合提供备用资源。系统提供强大的兼容性功能，可处理在任何设备屏幕上 渲染应用的大多数工作，前提是您已经使用 可以适当调整大小的技术实现 UI（如下面的[最佳做法](#)所述）。

**注：**定义设备通用屏幕 尺寸和密度的特性相互独立。例如，WVGA 高密度屏幕 被视为正常尺寸屏幕，因为其物理尺寸与 T-Mobile G1 (Android 的第一代设备和基线屏幕配置) 大约相同。另一方面，WVGA 中密度 屏幕被视为大尺寸屏幕。虽然它提供相同的分辨率（相同的 像素数），但 WVGA 中密度屏幕的屏幕密度更低，意味着每个像素 实际上更大，因此整个屏幕大于基线（正常尺寸）屏幕。

## 密度独立性

应用显示在密度不同的屏幕上时，如果它保持用户界面元素的物理尺寸（从 用户的视角），便可实现“密度独立性”。

保持密度独立性很重要，因为如果没有此功能，UI 元素（例如 按钮）在低密度屏幕上看起来较大，在高密度屏幕上看起来较小。这些 密度相关的大小变化可能给应用布局和易用性带来问题。图 2 和 3 分别显示了应用不提供密度独立性和 提供密度独立性时的差异。



**图 2.** 不支持不同密度的示例应用在低、中、高密度屏幕上的显示情况。

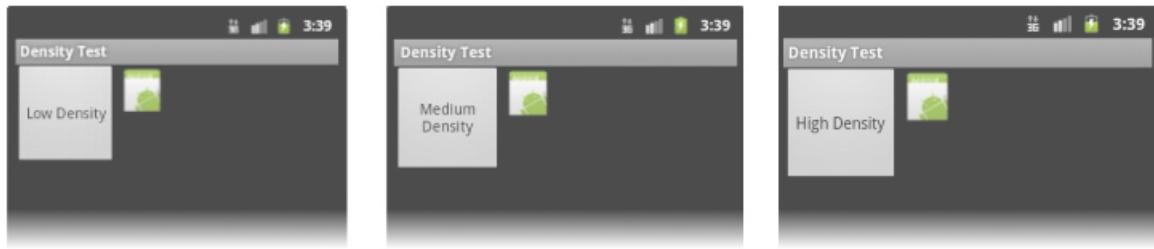


图 3. 良好支持不同密度（密度独立）的示例应用在低、中、高密度屏幕上的显示情况。

Android 系统可帮助您的应用以两种方式实现密度独立性：

- 系统根据当前屏幕密度扩展 dp 单位数
- 系统在必要时可根据当前屏幕密度将可绘制对象资源扩展到适当的大小

在图 2 中，文本视图和位图可绘制对象具有以像素（px 单位）指定的尺寸，因此视图的物理尺寸在低密度屏幕上更大，在高密度屏幕上更小。这是因为，虽然实际屏幕尺寸可能相同，但高密度屏幕的每英寸像素更多（同样多的像素在一个更小的区域内）。在图 3 中，布局尺寸以密度独立的像素（dp 单位）指定。由于密度独立像素的基线是中密度屏幕，因此具有中密度屏幕的设备看起来与图 2 一样。但对于低密度和高密度屏幕，系统将分别增加和减少密度独立像素值，以适应屏幕。

大多数情况下，确保应用中的屏幕独立性很简单，只需以适当的密度独立像素（dp 单位）或 "wrap\_content" 指定所有布局尺寸值。系统然后根据适用于当前屏幕密度的缩放比例适当地缩放位图可绘制对象，以适当的大小显示。

但位图缩放可能导致模糊或像素化位图，您或许已经在上面的屏幕截图中发现了这些问题。为避免这些伪影，应为不同的密度提供替代的位图资源。例如，应为高密度屏幕提供分辨率较高的位图，然后系统对中密度屏幕将使用这些位图，而无需调整位图大小。下一节详细说明如何为不同的屏幕配置提供备用资源。

## 如何支持多种屏幕

Android 支持多种屏幕的基础是它能够管理针对当前屏幕配置以适当方式渲染应用的布局和位图可绘制对象。系统可处理大多数工作，通过适当地缩放布局以适应屏幕尺寸/密度和根据屏幕密度缩放位图可绘制对象，在每种屏幕配置中渲染您的应用。但是，为了更适当地处理不同的屏幕配置，还应该：

- **在清单中显式声明您的应用 支持哪些屏幕尺寸**

通过声明您的应用支持哪些屏幕尺寸，可确保只有其屏幕受支持的设备才能下载您的应用。声明对不同屏幕尺寸的支持也可影响系统如何在较大屏幕上绘制您的应用—特别是，您的应用是否在[屏幕兼容模式](#)中运行。

要声明应用支持的屏幕尺寸，应在清单文件中包含 `<supports-screens>` 元素。

- **为不同屏幕尺寸提供不同的布局**

默认情况下，Android 会调整应用布局的大小以适应当前设备屏幕。大多数情况下效果很好。但有时 UI 可能看起来不太好，需要针对不同的屏幕尺寸进行调整。例如，在较大屏幕上，您可能要调整某些元素的位置和大小，以利用其他屏幕空间，或者在较小屏幕上，可能需要调整大小以使所有内容纳入屏幕。

可用于提供尺寸特定资源的配置限定符包括 `small`、`normal`、`large` 和 `xlarge`。例如，超大屏幕的布局应使用 `layout-xlarge/`。

从 Android 3.2 (API 级别 13) 开始，以上尺寸组已弃用，您应改为使用 `sw<N>dp` 配置限定符来定义布局资源可用的最小宽度。例如，如果多窗格平板电脑布局需要至少 600dp 的屏幕宽度，应将其放在 `layout-sw600dp/` 中。[声明适用于 Android 3.2 的平板电脑布局](#)一节将进一步讨论如何使用新技术声明布局资源。

- **为不同屏幕密度提供不同的位图可绘制对象**

默认情况下，Android 会缩放位图可绘制对象（.png、.jpg 和 .gif 文件）和九宫格可绘制对象（.9.png 文件），使它们以适当的物理尺寸显示在每部设备上。例如，如果您的应用只为基线中密度屏幕 (mdpi) 提供位图可绘制对象，则在高密度屏幕上会增大位图，在低密度屏幕上会缩小位图。这种缩放可能在位图中造成伪影。为确保位图的最佳显示效果，应针对不同屏幕密度加入不同分辨率的替代版本。

可用于密度特定资源的[配置限定符](#)（在下面详述）包括 `ldpi`（低）、`mdpi`（中）、`hdpi`（高）、`xhdpi`（超高）、`xxhdpi`（超超高）和 `xxxhdpi`（超超超高）。例如，高密度屏幕的位图应使用 `drawable-hdpi/`。

**注：**仅当要在 xxhdpi 设备上提供比正常位图大的启动器图标时才需要提供 `mipmap-xxxhdpi` 限定符。无需为所有应用的图像提供 xxxhdpi 资源。

有些设备会将启动器图标增大 25%。例如，如果您的最高密度启动器图标已是超超高密度，缩放处理会降低其清晰度。因此应在 `mipmap-xxxhdpi` 目录中提供更高密度的启动器图标，系统将改为增大较小的图标。

请参阅[提供 xxx-高密度启动器图标](#)以了解详细信息。对启动程序图标以外的 UI 元素不应使用 `xxxhdpi` 限定符。

**注：**将您的所有启动器图标放在 `res/mipmap-[density]/` 文件夹中，而非 `res/drawable-[density]/` 文件夹中。无论安装应用的设备屏幕分辨率如何，Android 系统都会将资源保留在这些密度特定的文件夹中，例如 `mipmap-xxxhdpi`。此行为可让启动器应用为您的应用选择显示在主屏幕上的最佳分辨率图标。如需了解有关使用 `mipmap` 文件夹的详细信息，请参阅[管理项目概览](#)。

尺寸和密度配置限定符对应于前面[支持的屏幕范围](#)中所述的通用尺寸和密度。

**注：**如果不熟悉配置限定符以及系统如何使用它们来应用备用资源，请参阅[提供备用资源](#)了解详细信息。

在运行时，系统通过以下程序确保任何给定资源在当前屏幕上都能保持尽可能最佳的显示效果：

## 1. 系统使用适当的备用资源

根据当前屏幕的尺寸和密度，系统将使用您的应用中提供的任何尺寸和密度特定资源。例如，如果设备有高密度屏幕，并且应用请求可绘制对象资源，系统将查找与设备配置最匹配的可绘制对象资源目录。根据可用的其他备用资源，包含 `hdpi` 限定符（例如 `drawable-hdpi/`）的资源目录可能是最佳匹配项，因此系统将使用此目录中的可绘制对象资源。

## 2. 如果没有匹配的资源，系统将使用默认资源，并按需要向上或向下扩展，以匹配当前的屏幕尺寸和密度。

“默认”资源是指未标记配置限定符的资源。例如，`drawable/` 中的资源是默认可绘制资源。系统假设默认资源设计用于基线屏幕尺寸和密度，即正常屏幕尺寸和中密度。因此，系统对于高密度屏幕向上扩展默认密度资源，对于低密度屏幕向下扩展。

当系统查找密度特定的资源但在密度特定目录中未找到时，不一定会使用默认资源。系统在缩放时可能改用其他密度特定资源提供更好的效果。例如，查找低密度资源但该资源不可用时，系统会缩小资源的高密度版本，因为系统可轻松以 0.5 为系数将高密度资源缩小至低密度资源，与以 0.75 为系数缩小中密度资源相比，伪影更少。

如需有关 Android 如何通过使配置限定符与设备配置匹配来选择备用资源的更多信息，请参阅[Android 如何查找最佳匹配资源](#)。

## 使用配置限定符

Android 支持多种配置限定符，可让您控制系统如何根据当前设备屏幕的特性选择备用资源。配置限定符是可以附加到 Android 项目中资源目录的字符串，用于指定在其中设计资源的配置。

要使用配置限定符：

### 1. 在项目的 `res/` 目录中新建一个目录，并使用以下格式命名：`<resources_name>-<qualifier>`

- `<resources_name>` 是标准资源名称（例如 `drawable` 或 `layout`）。
- `<qualifier>` 是下表 1 中的配置限定符，用于指定要使用这些资源的屏幕配置（例如 `hdpi` 或 `xlarge`）。

您可以一次使用多个 `<qualifier>` — 只需使用短划线分隔每个限定符。

### 2. 将适当的配置特定资源保存在此新目录下。这些资源文件的名称必须与默认资源文件完全一样。

例如，`xlarge` 是超大屏幕的配置限定符。将此字符串附加到资源目录名称（例如 `layout-xlarge`）时，它指向要在具有超大屏幕的设备上使用这些资源的系统。

**表 1.** 可用于为不同屏幕配置提供特殊资源的配置限定符。

屏幕特性	限定符	说明
尺寸	<code>small</code>	适用于小尺寸屏幕的资源。
	<code>normal</code>	适用于正常尺寸屏幕的资源。（这是基线尺寸。）
	<code>large</code>	适用于大尺寸屏幕的资源。
	<code>xlarge</code>	适用于超大尺寸屏幕的资源。
密	<code>ldpi</code>	适用于低密度 ( <code>ldpi</code> ) 屏幕 (~120dpi) 的资源。

度	<code>mdpi</code>	适用于中密度 ( <code>mdpi</code> ) 屏幕 (~160dpi) 的资源。 (这是基线 密度。)
	<code>hdpi</code>	适用于高密度 ( <code>hdpi</code> ) 屏幕 (~240dpi) 的资源。
	<code>xhdpi</code>	适用于超高密度 ( <code>xhdpi</code> ) 屏幕 (~320dpi) 的资源。
	<code>xxhdpi</code>	适用于超超高密度 ( <code>xxhdpi</code> ) 屏幕 (~480dpi) 的资源。
	<code>xxxhdpi</code>	适用于超超超高密度 ( <code>xxxhdpi</code> ) 屏幕 (~640dpi) 的资源。此限定符仅适用于 启动器图标，请参阅上面的 <a href="#">注</a> 。
	<code>nodpi</code>	适用于所有密度的资源。这些是密度独立的资源。不管当前屏幕的密度如何，系统都不会 缩放以此限定符标记的资源。
	<code>tvdpi</code>	适用于密度介于 <code>mdpi</code> 和 <code>hdpi</code> 之间屏幕 (约为 213dpi) 的资源。它并不是“主要”密度组，主要用于电视，而大多数应用都不 需要它 — 对于大多数应用而言，提供 <code>mdpi</code> 和 <code>hdpi</code> 资源便已足够，系统将根据需要对其进行 缩放。如果发现必须提供 <code>tvdpi</code> 资源，应以 1.33* <code>mdpi</code> 的系数 调整其大小。例如， <code>mdpi</code> 屏幕的 100px x 100px 图像应该相当于 <code>tvdpi</code> 的 133px x 133px。
方向	<code>land</code>	适用于横屏 (长宽比) 的资源。
	<code>port</code>	适用于竖屏 (高宽比) 的资源。
纵横比	<code>long</code>	适用于纵横比明显高于或宽于 (分别在竖屏 或横屏时) 基线屏幕配置的屏幕的资源。
	<code>notlong</code>	适用于使用纵横比类似于基线屏幕 配置的屏幕的资源。

**注：**如果是为 Android 3.2 和 更高版本开发应用，请参阅有关[声明适用于 Android 3.2 的平板电脑布局](#)的章节，了解 在为特定屏幕尺寸声明布局资源时应使用的 新配置限定符（而不是使用表 1 中的尺寸限定符）。

如需了解有关这些限定符如何粗略地对应于实际屏幕 尺寸和密度的更多信息，请参阅本文档前面的[支持的屏幕范围](#)。

例如，以下应用资源目录 为不同屏幕尺寸和不同可绘制对象提供不同的布局设计。使用 `mipmap/` 文件夹放置 启动器图标。

```
res/layout/my_layout.xml // layout for normal screen size ("default")
res/layout-large/my_layout.xml // layout for large screen size
res/layout-xlarge/my_layout.xml // layout for extra-large screen size
res/layout-xlarge-land/my_layout.xml // layout for extra-large in landscape orientation

res/drawable-mdpi/graphic.png // bitmap for medium-density
res/drawable-hdpi/graphic.png // bitmap for high-density
res/drawable-xhdpi/graphic.png // bitmap for extra-high-density
res/drawable-xxhdpi/graphic.png // bitmap for extra-extra-high-density

res/mipmap-mdpi/my_icon.png // launcher icon for medium-density
res/mipmap-hdpi/my_icon.png // launcher icon for high-density
res/mipmap-xhdpi/my_icon.png // launcher icon for extra-high-density
res/mipmap-xxhdpi/my_icon.png // launcher icon for extra-extra-high-density
res/mipmap-xxxhdpi/my_icon.png // launcher icon for extra-extra-extra-high-density
```

如需了解如何使用备用资源的更多信息以及 配置限定符的完整列表（不只是屏幕配置），请参阅[提供备用资源](#)。

请注意，当 Android 系统在运行时选择使用哪些资源时，它会使用 特定逻辑确定“最佳匹配”资源。也就是说，您使用的限定符无 需在所有情况下精确匹配当前屏幕配置，系统也可 使用它们。特别是，根据屏幕尺寸限定符选择资源时，如果没有更好的匹配资源，则系统将 使用专为小于当前屏幕的屏幕而设计的 资源（例如，如有必要，大尺寸屏幕将使用标准尺寸的屏幕 资源）。但是，如果唯一可用的资源 大于当前屏幕，则 系统不会使用这些资源，并且如果没有其他资源与设备 配置匹配，应用将会崩溃（例如，如果所有布局资源均用 `xlarge` 限定符标记，但设备是标准尺寸的屏幕）。如需有关系统如何选择资源的更多信息，请参阅[Android 如何查找最佳匹配资源](#)。

**提示：**如果您有一些系统 应该永远不会缩放（或许是因为您在 运行时亲自对图像做一些调整）的可绘制对象资源，则应将它们放在有 `nodpi` 配置限定符的目录中。使用此限定符的资源被视为与密度无关，系统不会缩放 它们。

## 设计替代布局和可绘制对象

您应该创建的备用资源类型取决于应用的需求。通常，您应该使用尺寸和方向限定符提供替代布局资源，并且使用密度限定符提供替代位图可绘制对象资源。

以下各节摘要说明您可能要如何使用尺寸和密度限定符 来分别提供替代布局和可绘制对象。

## 替代布局

一般而言，在不同的屏幕配置上测试应用后，您会知道是否需要用于不同屏幕尺寸的替代布局。例如：

- 在小屏幕上测试时，可能会发现您的布局不太适合屏幕。例如，小屏幕设备的屏幕宽度可能无法容纳一排按钮。在此情况下，您应该为小屏幕提供调整按钮大小或位置的替代布局。
- 在超大屏幕上测试时，可能会发现您的布局无法有效地利用大屏幕，并且明显拉伸填满屏幕。在此情况下，您应该为超大屏幕提供替代布局，以提供针对大屏幕（例如平板电脑）优化、重新设计的UI。

虽然您的应用不使用替代布局也能在大屏幕上正常运行，但必须让用户感觉您的应用看起来像是专为其设备而设计。如果UI明显拉伸，用户很可能对应用体验不满意。

- 而且，对比横屏测试和竖屏测试时可能会发现，竖屏时置于底部的UI在横屏时应位于屏幕右侧。

简而言之，您应确保应用布局：

- 适应小屏幕（让用户能实际使用您的应用）
- 已针对大屏幕优化，可以利用其他屏幕空间
- 已同时针对横屏和竖屏方向优化

如果UI使用的位图即使在系统缩放布局后也需要适应视图大小（例如按钮的背景图片），则应使用九宫格位图文件。九宫格文件基本上是一个指定可拉伸的二维区域的PNG文件。当系统需要缩放使用位图的视图时，系统会拉伸九宫格位图，但只拉伸指定的区域。因此，您无需为不同的屏幕尺寸提供不同的可绘制对象，因为九宫格位图可调整至任何大小。但您应该为不同的屏幕密度提供九宫格文件的替代版本。

## 替代可绘制对象

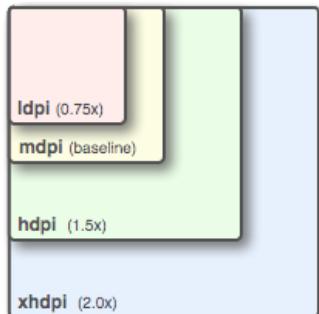


图4. 支持每种密度的位图可绘制对象的相对大小。

基本上每个应用都应该具有不同密度的替代可绘制对象资源，因为基本上每个应用都有启动器图标，而且该图标应该在所有屏幕密度中看起来都很好。同样，如果您的应用中包含其他位图可绘制对象（例如应用中的菜单图标或其他图形），则应该为不同密度提供替代版本或每种密度一个版本。

**注：**您只需要为位图文件（.png、.jpg或.gif）和九宫格文件（.9.png）提供密度特定的可绘制对象。如果您使用XML文件定义形状、颜色或其他可绘制对象资源，应该将一个副本放在默认可绘制对象目录中（drawable/）。

要为不同的密度创建替代位图可绘制对象，应遵循六种通用密度之间的3:4:6:8:12:16缩放比率。例如，如果您的位图可绘制对象是对中密度屏幕使用48x48像素，则所有不同的尺寸应为：

- 36x36 (0.75x) 用于低密度
- 48x48 (1.0x 基线) 用于中密度
- 72x72 (1.5x) 用于高密度
- 96x96 (2.0x) 用于超高密度
- 144x144 (3.0x) 用于超超高密度
- 192x192 (4.0x) 用于超超超高密度（仅限启动器图标；请参阅上面的注）

如需了解有关设计图标的信息，请参阅[图标设计指南](#)，其中包含各种位图可绘制对象（例如启动器图标、菜单图标、状态栏图标、选项卡

图标等) 的大小信息。

## 声明适用于 Android 3.2 的平板电脑布局

对于第一代运行 Android 3.0 的平板电脑，声明平板电脑 的正确方式是将它们放在有 `xlarge` 配置限定符的目录（例如 `res/layout-xlarge/`）中。为适应其他类型的平板电脑和屏幕 尺寸 — 特别是 7 英寸平板电脑 — Android 3.2 引入了为更具体的屏幕尺寸指定资源 的新方式。新技术基于布局需要的空间量（例如 600dp 宽），而不是尝试让您的布局容纳通用化的尺寸组（例如大或超大）。

使用通用化的尺寸组时，为 7 英寸平板电脑设计很棘手的原因在于，7 英寸平板电脑在技术上与 5 英寸手机属于同一个组（大组）。虽然这两种设备在尺寸上似乎很接近，但用于 应用 UI 的空间量明显不同，用户交互的方式也是如此。因此，7 英寸和 5 英寸 屏幕不一定使用相同的布局。为便于您为这两种屏幕提供不同的 布局，Android 现在允许您 根据实际适用于应用布局的宽度和/或高度指定布局资源（以 dp 单位数指定）。

例如，在设计要用于平板电脑样式设备的布局之后，您可能 发现该布局在屏幕宽度小于 600dp 时不适用。此阈值 于是变成平板电脑布局需要的最小尺寸。因此，您现在可以指定应仅当至少有 600dp 宽度供应用的 UI 使用时才使用这些布局资源。

应选择一个宽度并将其设计为最小尺寸，或者在布局设计完成后测试 其支持的最小宽度。

**注：**请记住，这些新尺寸 API 使用的所有数据是密度独立的像素 (dp) 值，您的布局尺寸也应始终 使用 dp 单位定义，因为您关注的是系统 考虑屏幕密度后可用的屏幕空间量（与使用原始像素分辨率相反）。如需了解 密度独立像素的更多信息，请参阅本文档前面的[术语和概念](#)。

## 使用新尺寸限定符

表 2 摘要列出了您可以根据 布局可用空间指定的不同资源配置。与传统的屏幕尺寸组（小、正常、大和超大）相比，这些新的限定符可用于 更多地控制 应用支持的屏幕尺寸。

**注：**您使用这些限定符指定的尺寸 **不是实际屏幕尺寸**。更确切地说，尺寸是 可用于 Activity 窗口的宽度或高度（dp 单位）。Android 系统 可能将某些屏幕用于系统 UI（例如屏幕底部的系统栏或 顶部的状态栏），因此有些屏幕可能不适用于您的布局。因此，您声明的尺寸应与 Activity 需要的尺寸具体相关 — 系统 在声明向您的布局提供的空间量时会计算系统 UI 使用的任何空间。另请注意，[操作栏](#)被视为 应用的 窗口空间的一部分，但您的布局未声明此事，因此会减少 您的布局可用的空间，您在设计时必须考虑进去。

**表 2.** 屏幕尺寸的新配置限定符（在 Android 3.2 中引入）。

屏幕配置	限定符 值	说明
smallestWidth	<code>sw&lt;N&gt;dp</code>  示例： <code>sw600dp</code> <code>sw720dp</code>	<p>屏幕的基本尺寸，由可用屏幕区域的最小尺寸指定。具体来说，设备的 <code>smallestWidth</code> 是屏幕可用高度和 宽度的最小尺寸（您也可以将其视为屏幕的“最小可能宽度”）。无论屏幕的当前方向如何，您均可使用此限定符确保应用 UI 的可用宽度至少为 <code>&lt;N&gt;dp</code>。</p> <p>例如，如果布局要求屏幕区域的最小尺寸始终至少为 600 dp，则可使用此限定符创建布局资源 <code>res/layout-sw600dp/</code>。仅当可用屏幕的最小尺寸至少为 600dp 时，系统才会使用这些资源，而不考虑 600dp 所代表的边是用户所认为的高度还是宽度。<code>smallestWidth</code> 是设备的固定屏幕尺寸特性；<b>设备的 <code>smallestWidth</code> 不会随屏幕方向的变化而改变</b>。</p> <p>设备的 <code>smallestWidth</code> 将屏幕装饰元素和系统 UI 考虑在内。例如，如果设备的屏幕上有一些永久性 UI 元素 占据沿 <code>smallestWidth</code> 轴的空间，则系统会声明 <code>smallestWidth</code> 小于实际屏幕尺寸，因为这些屏幕像素不适用于您的 UI。</p> <p>这可替代通用化的屏幕尺寸限定符（小、正常、大、超大），可让您为 UI 可用的有效尺寸定义不连续的数值。使用 <code>smallestWidth</code> 定义一般屏幕尺寸很有用，因为宽度 通常是设计布局时的驱动因素。UI 经常会 垂直滚动，但 对其水平需要的最小空间具有非常硬性的限制。可用的宽度也是 确定是否对手机使用单窗格布局或是对平板电脑使用多窗格布局 的关键因素。因此，您可能最关注每部 设备上的最小可能宽度。</p>
可用屏幕宽度	<code>w&lt;N&gt;dp</code>  示例： <code>w720dp</code> <code>w1024dp</code>	<p>指定资源应该使用的最小可用宽度（dp 单位） — 由 <code>&lt;N&gt;</code> 值定义。当屏幕的方向在横屏与竖屏之间切换 时，系统对应的 宽度值将会变化，以 反映 UI 可用的当前实际宽度。</p> <p>这对于确定是否使用多窗格布局往往很有用，因为即使是在 平板电脑设备上，您也通常不希望竖屏像横屏 一样 使用多窗格布局。因此，您可以使用此功能指定布局需要的最小宽度，而 无需同时使用屏幕尺寸和方</p>

		向限定符。
可用屏幕高度	<b>h&lt;N&gt;dp</b>  示例： <b>h720dp</b> <b>h1024dp</b> 等等	<p>指定资源应该使用的最小屏幕高度 (dp 单位) — 由 <b>&lt;N&gt;</b> 值定义。当屏幕的方向在横屏与竖屏之间切换时，系统 对应的高度值将会变化，以 反映 UI 可用的当前实际高度。</p> <p>使用此方式定义 布局需要的高度很有用，它与使用 <b>w&lt;N&gt;dp</b> 定义 所需宽度的方式相同，无需同时使用屏幕尺寸和方向限定符。但大多数应用不需要此限定符，考虑到 UI 经常垂直滚动，因此高度更弹性，而宽度更刚性。</p>

虽然使用这些限定符似乎比使用屏幕尺寸组更复杂，但 当您确定 UI 的要求后，它实际上应该更简单。在设计 UI 时，您主要关注的可能是应用在 手机样式 UI 与使用多窗格的平板电脑样式 UI 之间切换时的实际尺寸。此确切的精确时间 取决于特定设计 — 可能平板电脑布局需要 720dp 宽度，但 600dp、480dp 或这两者之间的某个值就够了。使用表 2 中的这些限定符 可以控制布局切换时的精确尺寸。

如需有关这些尺寸配置限定符的更多讨论，请参阅 [提供资源](#) 文档。

## 配置示例

为帮助您针对不同的设备类型确定某些设计，下面提供了一些 常见的屏幕宽度值：

- 320dp：常见手机屏幕 (240x320 ldpi、320x480 mdpi、480x800 hdpi 等)。
- 480dp：中间平板电脑，例如 Streak (480x800 mdpi)。
- 600dp：7 英寸平板电脑 (600x1024 mdpi)。
- 720dp：10 英寸平板电脑 (720x1280 mdpi、800x1280 mdpi 等)。

利用表 2 中的尺寸限定符，您的应用可以使用要用于宽度和/或高度的 的任何值，在用于手机和平板电脑的不同布局资源之间切换。例如，如果 600dp 是平板电脑布局支持的最小可用宽度，您可以提供以下两 组布局：

```
res/layout/main_activity.xml # For handsets
res/layout-sw600dp/main_activity.xml # For tablets
```

在此情况下，可用屏幕空间的最小宽度必须是 600dp，才可 应用平板电脑布局。

对于要进一步自定义 UI 以区分不同尺寸（例如 7 英寸和 10 英寸平板电脑）的其他情况，您可以定义其他最小宽度布局：

```
res/layout/main_activity.xml # For handsets (smaller than 600dp available width)
res/layout-sw600dp/main_activity.xml # For 7" tablets (600dp wide and bigger)
res/layout-sw720dp/main_activity.xml # For 10" tablets (720dp wide and bigger)
```

请注意，上面两组示例资源使用“最小宽度”限定符 **sw<N>dp**，用于指定屏幕两边的最小值，而不管设备 当前的方向如何。因此，使用 **sw<N>dp** 是指定 布局可用于整体屏幕尺寸的简便方法，它会忽略屏幕的方向。

但在某些情况下，可能 必须确定布局 **当前**可用的精确宽度或高度。例 如，如果是并排显示两个片段的双窗格布局，则只要 屏幕提供至少 600dp 的宽度（无论设备是横屏还是竖屏），您可能就要使用该布局。在此情况下，您的资源可能与以下所示类似：

```
res/layout/main_activity.xml # For handsets (smaller than 600dp available width)
res/layout-w600dp/main_activity.xml # Multi-pane (any screen with 600dp available width or more)
```

请注意，第二组使用“可用宽度”限定符 **w<N>dp**。这样，一部设备可能实际使用两种布局，具体取决于屏幕的方向（如果 可用的宽度在一个方向上至少为 600dp，而在另一个方向上小于 600dp）。

如果您关注可用高度，便可使用 **h<N>dp** 限定符执行同样的操作。或者，如果您需要很具体，甚至可以结合 **w<N>dp** 与 **h<N>dp** 限定符。

## 声明屏幕尺寸支持

在对不同的屏幕尺寸实现您的布局后，在 清单文件中声明您的应用支持哪些屏幕相当重要。

与用于屏幕尺寸的新配置限定符一起，Android 3.2 为 [`<supports-screens>`](#) 清单元素引入了新的属性：

```
android:requiresSmallestWidthDp
```

指定需要的最小 **smallestWidth**。**smallestWidth** 是必须为您的应用 UI 提供的 屏幕空间的最短尺寸 (dp 单位) —即 可用屏幕的两个尺寸中的最短者。因此，为使设备 与您的应用兼容，设备的 **smallestWidth** 必须等于或大于此 值。（通常，无论屏幕的当前方向如何，此值 都是布局支持的“最小宽度”。）

例如，如果您的应用只用于最小可用宽度为 600dp 的平板电脑样式设备：

```
<manifest ... >
 <supports-screens android:requiresSmallestWidthDp="600" />
 ...
</manifest>
```

但是，如果您的应用支持 Android 支持的所有屏幕尺寸（小至 426dp x 320dp），则无需声明此属性，因为应用需要的最小宽度就是任何设备上可以实现的最小宽度。

**注意：**Android 系统不关注此属性，因为它不影响应用在运行时的行为，而是被用于在服务（例如 Google Play）上过滤您的应用。

但是，Google Play 目前不支持此属性用于过滤（在 Android 3.2 上），因此如果您的应用不支持小屏幕，您应继续使用其他尺寸属性。

#### android:compatibleWidthLimitDp

此属性可让您指定用户支持的最大“最小宽度”，将屏幕兼容性模式用作用户可选的功能。如果设备可用屏幕的最小边大于您在这里的值，用户仍可安装您的应用，但提议在屏幕兼容性模式下运行。默认情况下，屏幕兼容性模式会停用，并且您的布局照例会调整大小以适应屏幕，但按钮会显示在系统栏中，可让用户打开和关闭屏幕兼容性模式。

**注：**如果您的应用可针对大屏幕正确调整大小，则无需使用此属性。建议不要使用此属性，而是按照本文档的建议，确保您的布局针对较大屏幕调整大小。

#### android:largestWidthLimitDp

此属性可让您指定应用支持的最大“最小宽度”来强制启用屏幕兼容性模式。如果设备可用屏幕的最小边大于您在这里的值，应用将在屏幕兼容性模式下运行，且用户无法停用该模式。

**注：**如果您的应用可针对大屏幕正确调整大小，则无需使用此属性。建议不要使用此属性，而是按照本文档的建议，确保您的布局针对较大屏幕调整大小。

**注意：**针对 Android 3.2 及更高版本开发时，您应改为将旧屏幕尺寸属性与上列属性结合使用。同时使用新属性和旧尺寸属性可能导致非预期的行为。

如需了解每个属性的更多信息，请跟随上面各自的链接。

## 最佳做法

支持多种屏幕的目标是创建一款在 Android 系统支持的通用屏幕尺寸上都可以正常运行且显示良好的应用。本文档前面各节内容介绍了 Android 系统如何使您的应用适应屏幕配置，以及如何在不同的屏幕配置上自定义应用的外观。本节提供另外一些提示以及有助于确保应用针对不同屏幕配置正确缩放的技巧概览。

下面是有关如何确保应用在不同屏幕上正常显示的快速检查清单：

1. 在 XML 布局文件中指定尺寸时使用 `wrap_content`、`match_parent` 或 `dp` 单位。
2. 不要在应用代码中使用硬编码的像素值
3. 不要使用 `AbsoluteLayout`（已弃用）
4. 为不同屏幕密度提供替代位图可绘制对象

下文将提供更详细的信息。

### 1. 对布局尺寸使用 `wrap_content`、`match_parent` 或 `dp` 单位

为 XML 布局文件中的视图定义 `android:layout_width` 和 `android:layout_height` 时，使用 `"wrap_content"`、`"match_parent"` 或 `dp` 单位可确保在当前设备屏幕上为视图提供适当的尺寸。

例如，`layout_width="100dp"` 的视图在中密度屏幕上测出宽度为 100 像素，在高密度屏幕上系统会将其扩展至 150 像素宽，因此视图在屏幕上占用的物理空间大约相同。

类似地，您应选择 `sp`（缩放独立的像素）来定义文本大小。`sp` 缩放系数取决于用户设置，系统会像处理 `dp` 一样缩放大小。

## 2. 不要在应用代码中使用硬编码的像素值

由于性能的原因和简化代码的需要，Android 系统使用像素作为表示尺寸或坐标值的标准单位。这意味着，视图的尺寸在代码中始终以像素表示，但始终基于当前的屏幕密度。例如，如果 `myView.getWidth()` 返回 10，则表示视图在当前屏幕上为 10 像素宽，但在更高密度的屏幕上，返回的值可能是 15。如果在应用代码中使用像素值来处理预先未针对当前屏幕密度缩放的位图，您可能需要缩放代码中使用的像素值，以与未缩放的位图来源匹配。

如果应用在运行时操作位图或处理像素值，请参阅下面有关[其他密度注意事项](#)的一节。

## 3. 不要使用 AbsoluteLayout

与其他布局小工具不同，`AbsoluteLayout` 会强制使用固定位置放置其子视图，很容易导致在不同显示屏上显示效果不好的用户界面。因此，`AbsoluteLayout` 在 Android 1.5 (API 级别 3) 上便已弃用。

您应改用 `RelativeLayout`，它会使用相对定位来放置其子视图。例如，您可以指定按钮小部件显示在文本小工具的“右边”。

## 4. 使用尺寸和密度特定资源

虽然系统会根据当前屏幕配置扩展布局，但您在不同的屏幕尺寸上可能要调整 UI，以及提供针对不同密度优化的可绘制对象。这基本上是重申本文档前面的信息。

如果需要精确控制应用在不同屏幕配置上的外观，请在配置特定的资源目录中调整您的布局和位图可绘制对象。例如，考虑要显示在中密度和高密度屏幕上的图标。只需创建两种不同大小的图标（例如中密度使用 100x100，高密度使用 150x150），然后使用适当的限定符以适当的方向放置两个变体：

```
res/drawable-mdpi/icon.png //for medium-density screens
res/drawable-hdpi/icon.png //for high-density screens
```

**注：**如果密度限定符在目录名称中未定义，系统会假设该目录中的资源是针对基线中密度而设计，对于其他密度将会适当地缩放。

如需了解有效配置限定符的更多信息，请参阅本文档前面的[使用配置限定符](#)。

## 其他密度注意事项

本节详细说明 Android 如何在不同屏幕密度上对位图可绘制对象执行缩放，以及如何进一步控制在不同密度屏幕上位图的绘制。本节信息对大多数应用应该不怎么重要，除非您的应用在不同屏幕密度上运行或操控图形时遇到了问题。

为更好地了解在运行时操控图形时如何支持多种密度，您应该先了解，系统通过以下方式帮助确保正确缩放位图：

### 1. 资源（例如位图可绘制对象）的预缩放

根据当前屏幕的密度，系统将使用您的应用中提供的任何尺寸或密度特定资源，并且不加缩放而显示它们。如果没有可用于正确密度的资源，系统将加载默认资源，并按需要向上或向下扩展，以匹配当前屏幕的密度。系统假设默认资源（没有配置限定符的目录中的资源）针对基线屏幕密度 (mdpi) 而设计，除非它们加载自密度特定的资源目录。因此，系统会执行预缩放，以将位图调整至适应当前屏幕密度的大小。

如果您请求预缩放的资源的尺寸，系统将返回代表缩放后尺寸的值。例如，针对 mdpi 屏幕以 50x50 像素设计的位图在 hdpi 屏幕上将扩展至 75x75 像素（如果没有用于 hdpi 的备用资源），并且系统会这样报告大小。

有时您可能不希望 Android 预缩放资源。避免预缩放最简单的方法是将资源放在有 `nodpi` 配置限定符的资源目录中。例如：

```
res/drawable-nodpi/icon.png
```

当系统使用此文件夹中的 `icon.png` 位图时，不会根据当前设备密度缩放。

### 2. 像素尺寸和坐标的自动缩放

应用可通过在清单中将 `android:anyDensity` 设置为 "false" 或者通过将 `inScaled` 设置为 "false" 对 `Bitmap` 编程来停用预缩放。在此情况下，系统在绘制时会自动缩放任何绝对的像素坐标和像素尺寸值。缩放的目的是确保像素定义的屏幕元素仍以它们在基线屏幕密度 (mdpi) 下的大致相同物理尺寸显示。系统会对应用透明地处理此缩放，并且向应用报告缩放后的像素尺寸，而不是物理像素尺寸。

例如，假设设备具有 480x800 的 WVGA 高密度屏幕，大约与传统 HVGA 屏幕的尺寸一样，但它运行的应用停用了预缩放。在此情况下，系统在查询屏幕尺寸时会对应用“撒谎”，报告 320x533（屏幕密度的近似 mdpi 转换值）。然后，当应用执行绘制操作时，例如作废从

(10,10) 到 (100, 100) 的矩形，系统会将它们缩放适当的量以转变坐标，并且实际作废区域 (15,15) 到 (150, 150)。如果应用直接操控缩放的位图，此差异可能会导致非预期的行为，但这被视为 确保应用最佳性能所需的合理权衡。如果遇到此情况，请参阅[将 dp 单位转换为像素单位](#)一节。

通常，**不应停用预缩放**。支持多种屏幕的最佳方法是采用上面[如何支持多种屏幕](#)中所述的基本技术。

如果您的应用操控位图或以某种其他方式直接与屏幕上的像素交互，您可能需要采取其他步骤支持不同的屏幕密度。例如，如果您通过计算手指滑过的像素数来响应触控手势，则需使用适当的密度独立像素值，而不是实际像素。

## 缩放运行时创建的位图对象



图 5. 预缩放的位图与自动缩放的位图比较。

如果您的应用创建内存中位图 (`Bitmap` 对象)，系统在默认情况下假设位图是针对基线中密度屏幕而设计，然后在绘制时自动缩放位图。当位图具有不明的密度属性时，系统会对 `Bitmap` 应用“自动缩放”。如果未正确考虑当前设备的屏幕密度和指定位图的密度属性，自动缩放可能导致缩放伪影，就像未提供备用资源一样。

要控制是否缩放运行时创建的 `Bitmap`，可以使用 `setDensity()` 指定位图的密度，从 `DisplayMetrics` 传递密度常量，例如 `DENSITY_HIGH` 或 `DENSITY_LOW`。

如果使用 `BitmapFactory` 创建 `Bitmap`，例如从文件或流创建，可以使用 `BitmapFactory.Options` 定义位图的属性（因为它已经存在），确定系统是否要缩放或者如何缩放。例如，您可以使用 `inDensity` 字段定义位图设计时的密度，使用 `inScaled` 字段指定位图是否应缩放以匹配当前设备的屏幕密度。

如果将 `inScaled` 字段设置为 `false`，然后停用系统可能应用到位图的预缩放，则系统在绘制时将自动缩放它。使用自动缩放代替预缩放可能耗用的 CPU 更多，但耗用的内存更少。

图 5 所示为在高密度屏幕上加载低 (120)、中 (160) 和高 (240) 密度位图时预缩放和自动缩放机制产生的效果。差异很小，因为所有位图都针对当前屏幕密度而缩放，但根据在绘制时是预缩放还是自动缩放，缩放后位图的外观略有不同。

**注：**在 Android 3.0 的更高版本中，由于图形框架的改进，应该觉察不出预缩放的位图与自动缩放的位图之间的差异。

## 将 dp 单位转换为像素单位

在某些情况下，您需要以 `dp` 表示尺寸，然后将它们转换为像素。设想一个在用户手指移动至少 16 像素之后可以识别滚动或滑动手势的应用。在基线屏幕上，用户必须移动 `16 pixels / 160 dpi`（等于一英寸的 1/10 或 2.5 毫米），然后才会识别该手势。在具有高密度显示屏

(240dpi) 的设备上，用户必须移动 16 pixels / 240 dpi (等于一英寸的 1/15 或 1.7 毫米)。此距离更短，应用因此似乎对用户更灵敏。

要修复此问题，手势阈值必须在代码中以 dp 表示，然后转换为实际像素。例如：

```
// The gesture threshold expressed in dp
private static final float GESTURE_THRESHOLD_DP = 16.0f;

// Get the screen's density scale
final float scale = getResources().getDisplayMetrics().density;
// Convert the dps to pixels, based on density scale
mGestureThreshold = (int) (GESTURE_THRESHOLD_DP * scale + 0.5f);

// Use mGestureThreshold as a distance in pixels...
```

`DisplayMetrics.density` 字段根据当前屏幕密度指定将 dp 单位转换为像素必须使用的缩放系数。在中密度屏幕上，`DisplayMetrics.density` 等于 1.0；在高密度屏幕上，它等于 1.5；在超高密度屏幕上，等于 2.0；在低密度屏幕上，等于 0.75。此数字是一个系数，应用其乘以 dp 单位以获取用于当前屏幕的实际像素数。（然后在转换时加上 `0.5f`，将该数字四舍五入到最接近的整数。）如需了解详细信息，请参阅 `DisplayMetrics` 类。

但是，不能为此类事件定义任意阈值，而应使用 `ViewConfiguration` 中的预缩放配置值。.

## 使用预缩放的配置值

您可以使用 `ViewConfiguration` 类访问 Android 系统使用的通常距离、速度和时间。例如，使用 `getScaledTouchSlop()` 可获取框架用作滚动阈值的距离（像素）：

```
private static final int GESTURE_THRESHOLD_DP = ViewConfiguration.get(myContext).getScaledTouchSlop();
```

`ViewConfiguration` 中以 `getScaled` 前缀开头的方法确定会返回不管当前屏幕密度为何都会正常显示的像素值。

## 如何在多个屏幕上测试您的应用



图 6. 一组用于测试屏幕支持的 AVD。

在发布应用之前，应在所有支持的屏幕尺寸和密度中全面测试。Android SDK 包含可以使用的模拟器皮肤，它们会复制您的应用可能要在其中运行的常见屏幕配置的尺寸和密度。也可修改模拟器皮肤的默认尺寸、密度和分辨率，以复制任何特定屏幕的特性。使用模拟器皮肤和其他自定义配置可测试任何可能的屏幕配置，因此您无需仅仅为了测试应用的屏幕支持而购买不同的设备。

要设置环境以测试应用的屏幕支持，应使用能模拟您希望应用支持的屏幕尺寸和密度的模拟器皮肤和屏幕配置创建一系列 AVD (Android Virtual Devices)。要执行此操作，可以使用 AVD Manager 创建 AVD 并使用图形界面启动它们。

要启动 Android SDK Manager，从您的 Android SDK 目录执行 `SDK Manager.exe` (仅在 Windows 上)，或者从 `<sdk>/tools/` 目录执行 `android` (在所有平台上)。图 6 所示为用于测试不同屏幕配置的 AVD Manager (选择了 AVD)。

表 3 所示为 Android SDK 中可用的各种模拟器皮肤，可用以模拟某些最常见的屏幕配置。

如需了解有关创建和使用 AVD 测试应用的详细信息，请参阅[使用 AVD Manager 管理 AVD](#)。

表 3. Android SDK (粗体表示) 及其他代表性解决方案中模拟器皮肤提供的各种屏幕配置

	低密度 (120), <i>ldpi</i>	中密度 (160), <i>mdpi</i>	高密度 (240), <i>hdpi</i>	超高密度 (320), <i>xhdpi</i>
小屏幕	QVGA (240x320)		480x640	
正常屏幕	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x960
大屏幕	WVGA800** (480x800) WVGA854** (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024		
超大屏幕	1024x600	WXGA (1280x800)† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

\* 要模拟此配置，在创建使用 WVGA800 或 WVGA854 皮肤的 AVD 时请指定自定义密度 160。

\*\* 要模拟此配置，在创建 使用 WVGA800 或 WVGA854 皮肤的 AVD 时请指定自定义密度 120。

† 此皮肤可用于 Android 3.0 平台

要查看支持任何指定屏幕配置的活动设备的相对数量，请参阅 [屏幕尺寸和密度](#) 仪表板。

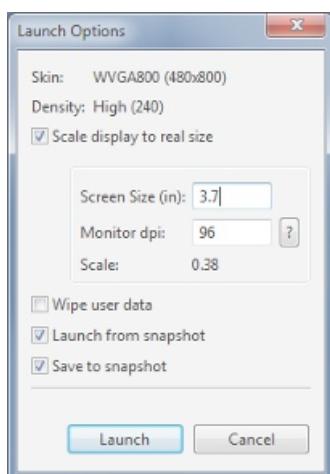


图 7. 从 AVD Manager 启动 AVD 时可以设置的尺寸和密度选项。

我们还建议您在 设置为以接近实际设备的物理尺寸运行的模拟器中测试应用。这样 更容易比较不同尺寸和密度时的结果。要 完成此操作，需 要知道计算机显示器的大约密度 (dpi)， 例 如 30 英寸 Dell 显示器的密度约为 96 dpi。从 AVD Manager 启动 AVD 时，可在 Launch Options 中 指定用于模拟器和您的 显示器的屏幕尺寸 (dpi)，如图 7 所示。

如果要在使用内置皮肤 不支持的分辨率或密度的屏幕上测试应用，可以创建使用自定义分辨率或密度的 AVD。从 AVD Manager 创建 AVD 时，指 定 Resolution，而不要选择 Built-in Skin。

从命令行启动 AVD 时，可以使用 `-scale` 选项指定用于 模拟器的缩放比例。例如：

```
emulator -avd <avd_name> -scale 96dpi
```

要调整模拟器的大小，可使用 `-scale` 选项指定代表所需缩放系数的 0.1 至 3。

如需了解从命令行创建 AVD 的更多信息，请参阅[从命令行管理 AVD](#)。



# Distributing to Specific Screens

In this document

- [Declaring an App is Only for Handsets](#)
- [Declaring an App is Only for Tablets](#)
- [Publishing Multiple APKs for Different Screens](#)

Although we recommend that you design your application to function properly on multiple configurations of screen size and density, you can instead choose to limit the distribution of your application to certain types of screens, such as only tablets and other large devices or only handsets and similar-sized devices. To do so, you can enable filtering by external services such as Google Play by adding elements to your manifest file that specify the screen configurations your application supports.

However, before you decide to restrict your application to certain screen configurations, you should understand the techniques for [supporting multiple screens](#) and implement them to the best of your ability. By supporting multiple screens, your application can be made available to the greatest number of users with different devices, using a single APK.

## Declaring an App is Only for Handsets

Because the system generally scales applications to fit larger screens well, you shouldn't need to filter your application from larger screens. As long as you follow the [Best Practices for Screen Independence](#), your application should work well on larger screens such as tablets. However, you might discover that your application can't scale up well or perhaps you've decided to publish two versions of your application for different screen configurations. In such a case, you can use the `<compatible-screens>` element to manage the distribution of your application based on combinations of screen size and density. External services such as Google Play use this information to apply filtering to your application, so that only devices that have a screen configuration with which you declare compatibility can download your application.

The `<compatible-screens>` element must contain one or more `<screen>` elements. Each `<screen>` element specifies a screen configuration with which your application is compatible, using both the `android:screenSize` and `android:screenDensity` attributes. Each `<screen>` element **must include both attributes** to specify an individual screen configuration—if either attribute is missing, then the element is invalid (external services such as Google Play will ignore it).

For example, if your application is compatible with only small and normal size screens, regardless of screen density, you must specify eight different `<screen>` elements, because each screen size has four density configurations. You must declare each one of these; any combination of size and density that you do *not* specify is considered a screen configuration with which your application is *not* compatible. Here's what the manifest entry looks like if your application is compatible with only small and normal screen sizes:

```

<manifest ... >
 <compatible-screens>
 <!-- all small size screens -->
 <screen android:screenSize="small" android:screenDensity="ldpi" />
 <screen android:screenSize="small" android:screenDensity="mdpi" />
 <screen android:screenSize="small" android:screenDensity="hdpi" />
 <screen android:screenSize="small" android:screenDensity="xhdpi" />
 <!-- all normal size screens -->
 <screen android:screenSize="normal" android:screenDensity="ldpi" />
 <screen android:screenSize="normal" android:screenDensity="mdpi" />
 <screen android:screenSize="normal" android:screenDensity="hdpi" />
 <screen android:screenSize="normal" android:screenDensity="xhdpi" />
 </compatible-screens>
 ...
 <application ... >
 ...
 <application>
 </manifest>

```

**Note:** Although you can also use the `<compatible-screens>` element for the reverse scenario (when your application is not compatible with smaller screens), it's easier if you instead use the `<supports-screens>` as discussed in the next section, because it doesn't require you to specify each screen density your application supports.

## Declaring an App is Only for Tablets

If you don't want your app to be used on handsets (perhaps your app truly makes sense only on a large screen) or you need time to optimize it for smaller screens, you can prevent small-screen devices from downloading your app by using the `<supports-screens>` manifest element.

For example, if you want your application to be available only to tablet devices, you can declare the element in your manifest like this:

```

<manifest ... >
 <supports-screens android:smallScreens="false"
 android:normalScreens="false"
 android:largeScreens="true"
 android:xlargeScreens="true"
 android:requiresSmallestWidthDp="600" />
 ...
 <application ... >
 ...
 </application>
</manifest>

```

This describes your app's screen-size support in two different ways:

- It declares that the app does *not* support the screen sizes "small" and "normal", which are traditionally not tablets.
- It declares that the app requires a screen size with a minimum usable area that is at least 600dp wide.

The first technique is for devices that are running Android 3.1 or older, because those devices declare their size based on generalized screen sizes. The `requiresSmallestWidthDp` attribute is for devices running Android 3.2 and newer, which includes the capability for apps to specify size requirements based on a minimum number of density-independent pixels available. In this example, the app declares a minimum width requirement of 600dp, which generally implies a 7"-or-greater screen.

Your size choice might be different, of course, based on how well your design works on different screen sizes; for example, if your design works well only on screens that are 9" or larger, you might require a minimum width of 720dp.

The catch is that you must compile your application against Android 3.2 or higher in order to use the `requiresSmallestWidthDp` attribute. Older versions don't understand this attribute and will raise a compile-time error. The safest thing to do is develop your app against the platform that matches the API level you've set for `minSdkVersion`. When you're making final preparations to build your release candidate, change the build target to Android 3.2 and add the `requiresSmallestWidthDp` attribute. Android versions older than 3.2 simply ignore that XML attribute, so there's no risk of a runtime failure.

For more information about why the "smallest width" screen size is important for supporting different screen sizes, read [New Tools for Managing Screen Sizes](#).

**Caution:** If you use the `<supports-screens>` element for the reverse scenario (when your application is not compatible with *larger* screens) and set the larger screen size attributes to "`false`", then external services such as Google Play **do not** apply filtering. Your application will still be available to larger screens, but when it runs, it will not resize to fit the screen. Instead, the system will emulate a handset screen size (about 320dp x 480dp; see [Screen Compatibility Mode](#) for more information). If you want to prevent your application from being downloaded on larger screens, use `<compatible-screens>`, as discussed in the previous section about [Declaring an App is Only for Handsets](#).

Remember, you should strive to make your application available to as many devices as possible by applying all necessary techniques for [supporting multiple screens](#). You should use `<compatible-screens>` or `<supports-screens>` only when you cannot provide compatibility on all screen configurations or you have decided to provide different versions of your application for different sets of screen configurations.

## Publishing Multiple APKs for Different Screens

---

Although we recommend that you publish one APK for your application, Google Play allows you to publish multiple APKs for the same application when each APK supports a different set of screen configurations (as declared in the manifest file). For example, if you want to publish both a handset version and a tablet version of your application, but you're unable to make the same APK work for both screen sizes, you can actually publish two APKs for the same application listing. Depending on each device's screen configuration, Google Play will deliver it the APK that you've declared to support that device's screen.

Beware, however, that publishing multiple APKs for the same application is considered an advanced feature and **most applications should publish only one APK that can support a wide range of device configurations**. Supporting multiple screen sizes, especially, is within reason using a single APK, as long as you follow the guide to [Supporting Multiple Screens](#).

If you need more information about how to publish multiple APKs on Google Play, read [Multiple APK Support](#).

# Screen Compatibility Mode

In this document

- » [Disabling Screen Compatibility Mode](#)
- » [Enabling Screen Compatibility Mode](#)

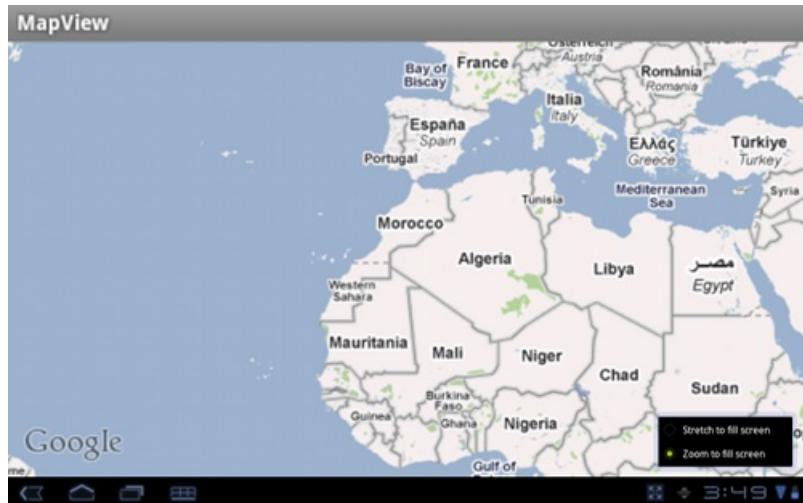


Figure 1. An application running in compatibility mode on an Android 3.2 tablet.

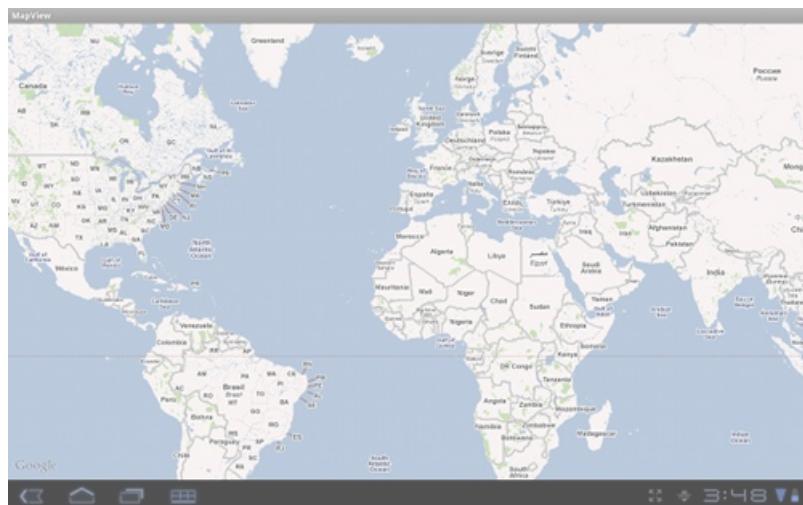


Figure 2. The same application from figure 1, with compatibility mode disabled.

**Notice:** If you've developed an application for a version of Android lower than Android 3.0, but it does resize properly for larger screens such as tablets, you should disable screen compatibility mode in order to maintain the best user experience. To learn how to quickly disable the user option, jump to [Disabling Screen Compatibility Mode](#).

Screen compatibility mode is an escape hatch for applications that are not properly designed to resize for larger screens such as tablets. Since Android 1.6, Android has supported a variety of screen sizes and does most of the work to resize application layouts so that they properly fit each screen. However, if your application does not successfully follow the guide to [Supporting Multiple Screens](#), then it might encounter some rendering issues on larger screens. For applications with this problem, screen compatibility mode can make the application a little more usable on larger screens.

There are two versions of screen compatibility mode with slightly different behaviors:

## Version 1 (Android 1.6 - 3.1)

The system draws the application's UI in a "postage stamp" window. That is, the system draws the application's layout the same as it would on a normal size handset (emulating a 320dp x 480dp screen), with a black border that fills the remaining area of the screen.

This was introduced with Android 1.6 to handle apps that were designed only for the original screen size of 320dp x 480dp. Because there are so few active devices remaining that run Android 1.5, almost all applications should be developed against Android 1.6 or greater and should not have version 1 of screen compatibility mode enabled for larger screens. This version is considered obsolete.

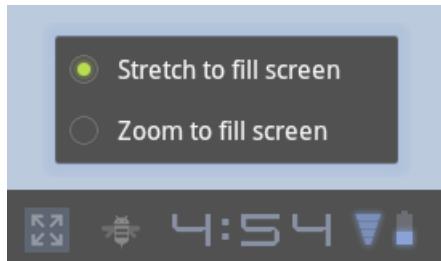
To disable this version of screen compatibility mode, you simply need to set `android:minSdkVersion` or `android:targetSdkVersion` to "4" or higher, or set `android:resizeable` to "true".

## Version 2 (Android 3.2 and greater)

The system draws the application's layout the same as it would on a normal size handset (approximately emulating a 320dp x 480dp screen), then scales it up to fill the screen. This essentially "zooms" in on your layout to make it bigger, which will usually cause artifacts such as blurring and pixelation in your UI.

This was introduced with Android 3.2 to further assist applications on the latest tablet devices when the applications have not yet implemented techniques for [Supporting Multiple Screens](#).

In general, large screen devices running Android 3.2 or higher allow users to enable screen compatibility mode when the application does not **explicitly declare that it supports large screens** in the manifest file. When this is the case, an icon (with outward-pointing arrows) appears next to the clock in the system bar, which allows the user to toggle screen compatibility mode on and off (figure 3). An application can also explicitly declare that it *does not* support large screens such that screen compatibility mode is always enabled and the user cannot disable it. (How to declare your application's support for large screens is discussed in the following sections.)



**Figure 3.** The pop up menu to toggle screen compatibility mode (currently disabled, so normal resizing occurs).

As a developer, you have control over when your application uses screen compatibility mode. The following sections describe how you can choose to disable or enable screen compatibility mode for larger screens when running Android 3.2 or higher.

## Disabling Screen Compatibility Mode

If you've developed your application primarily for versions of Android lower than 3.0, but **your application does resize properly** for larger screens such as tablets, **you should disable screen compatibility mode** in order to maintain the best user experience. Otherwise, users may enable screen compatibility mode and experience your application in a less-than-ideal format.

By default, screen compatibility mode for devices running Android 3.2 and higher is offered to users as an optional feature when one of the following is true:

- Your application has set both `android:minSdkVersion` and `android:targetSdkVersion` to "10" or lower and **does not explicitly declare support** for large screens using the `<supports-screens>` element.
- Your application has set either `android:minSdkVersion` or `android:targetSdkVersion` to "11" or higher and **explicitly declares that it does not support** large screens, using the `<supports-screens>` element.

To completely disable the user option for screen compatibility mode and remove the icon in the system bar, you can do one of the following:

- **Easiest:**

In your manifest file, add the `<supports-screens>` element and specify the `android:xlargeScreens` attribute to "true":

```
<supports-screens android:xlargeScreens="true" />
```

That's it. This declares that your application supports all larger screen sizes, so the system will always resize your layout to fit the screen. This works regardless of what values you've set in the `<uses-sdk>` attributes.

- **Easy but has other effects:**

In your manifest's `<uses-sdk>` element, set `android:targetSdkVersion` to "11" or higher:

```
<uses-sdk android:minSdkVersion="4" android:targetSdkVersion="11" />
```

This declares that your application supports Android 3.0 and, thus, is designed to work on larger screens such as tablets.

**Caution:** When running on Android 3.0 and greater, this also has the effect of enabling the Holographic theme for your UI, adding the [Action Bar](#) to your activities, and removing the Options Menu button in the system bar.

If screen compatibility mode is still enabled after you change this, check your manifest's `<supports-screens>` and be sure that there are no attributes set `"false"`. The best practice is to always explicitly declare your support for different screen sizes using the `<supports-screens>` element, so you should use this element anyway.

For more information about updating your application to target Android 3.0 devices, read [Optimizing Apps for Android 3.0](#).

## Enabling Screen Compatibility Mode

When your application is targeting Android 3.2 (API level 13) or higher, you can affect whether compatibility mode is enabled for certain screens by using the `<supports-screens>` element.

**Note:** Screen compatibility mode is **not** a mode in which you should want your application to run—it causes pixelation and blurring in your UI, due to zooming. The proper way to make your application work well on large screens is to follow the guide to [Supporting Multiple Screens](#) and provide alternative layouts for different screen sizes.

By default, when you've set either `android:minSdkVersion` or `android:targetSdkVersion` to "11" or higher, screen compatibility mode is **not** available to users. If either of these are true and your application does not resize properly for larger screens, you can choose to enable screen compatibility mode in one of the following ways:

- In your manifest file, add the `<supports-screens>` element and specify the `android:compatibleWidthLimitDp` attribute to "320":

```
<supports-screens android:compatibleWidthLimitDp="320" />
```

This indicates that the maximum "smallest screen width" for which your application is designed is 320dp. This way, any devices with their smallest side being larger than this value will offer screen compatibility mode as a user-optimal feature.

**Note:** Currently, screen compatibility mode only emulates handset screens with a 320dp width, so screen compatibility mode is not applied to any device if your value for `android:compatibleWidthLimitDp` is larger than 320.

- If your application is functionally broken when resized for large screens and you want to force users into screen compatibility mode (rather than simply providing the option), you can use the `android:largestWidthLimitDp` attribute:

```
<supports-screens android:largestWidthLimitDp="320" />
```

This works the same as `android:compatibleWidthLimitDp` except it force-enables screen compatibility mode and does not allow users to disable it.