

Chapter V: Working with background services

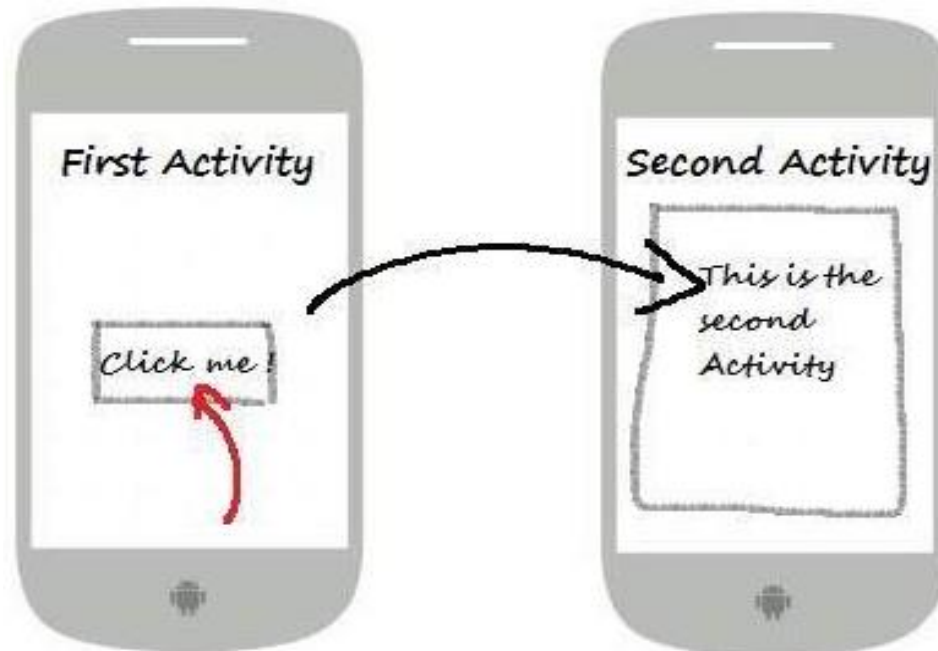


Introduction to Intents

- An Intent is a simple message object that is used to communicate between android components such as activities, content providers, broadcast receivers and services.
- Intents are also used to transfer data between activities.
- Intents are used generally for starting a new activity using `startActivity()`.

Use of Intent

- For Launching an Activity
- To start a New Service
- For Broadcasting Messages
- To Display a list of contacts in ListView



Types of Intent in Android

- Two types of Intent:

- Implicit Intent
- Explicit Intent

1. Implicit Intent

- specifies an action that can invoke any app on the device to be able to perform an action.
- is useful when your app cannot perform the action but other apps probably can and you'd like the user to pick which app to use.

```
Intent i=new Intent();  
i.setAction(Intent.ACTION_SEND);
```

2. Explicit Intent

- is an Intent where you explicitly define the component that needs to be called by the Android System.
- is one that you can **use to launch a specific app component**, such as a particular activity or service in your app.



```
Intent I = new Intent(getApplicationContext(),NextActivity.class);  
I.putExtra("value1" , "This value for Next Activity");  
I.putExtra("value2" , "This value for Next Activity");
```

- **The Different Methods Used in Intent**
 - **Action_Main:** Adds an action to an intent Filter.
 - **Action_Pick:** It is using for picking the image from CAMERA or GALLERY
 - **Action_Chooser:** It is used for choosing the image from the gallery
 - **Action_Dial:** Display the phone dialer with the given number filled in.
 - **Action_Call:** Placing an immediate phone call
 - **Action_Send:** Sending Text content from one activity to other.
 - **Action_SendTo:** Preparing an SMS. The text is supplied as an Extra element. The intent expects such as values to be called "sms_body"

```
Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);  
  
    photoPickerIntent.setType("image/*");  
  
    startActivityForResult(photoPickerIntent, SELECT_PHOTO);
```

Select profile Photo!

Take Photo

Choose from Library

Cancel

```
Intent select = new Intent(Intent.ACTION_CAMERA);  
    send.setData(uri);  
    startActivity(Intent.createChooser(  
        select, "Select an Image from Camera"));
```

Choose an action



Gallery



Camera



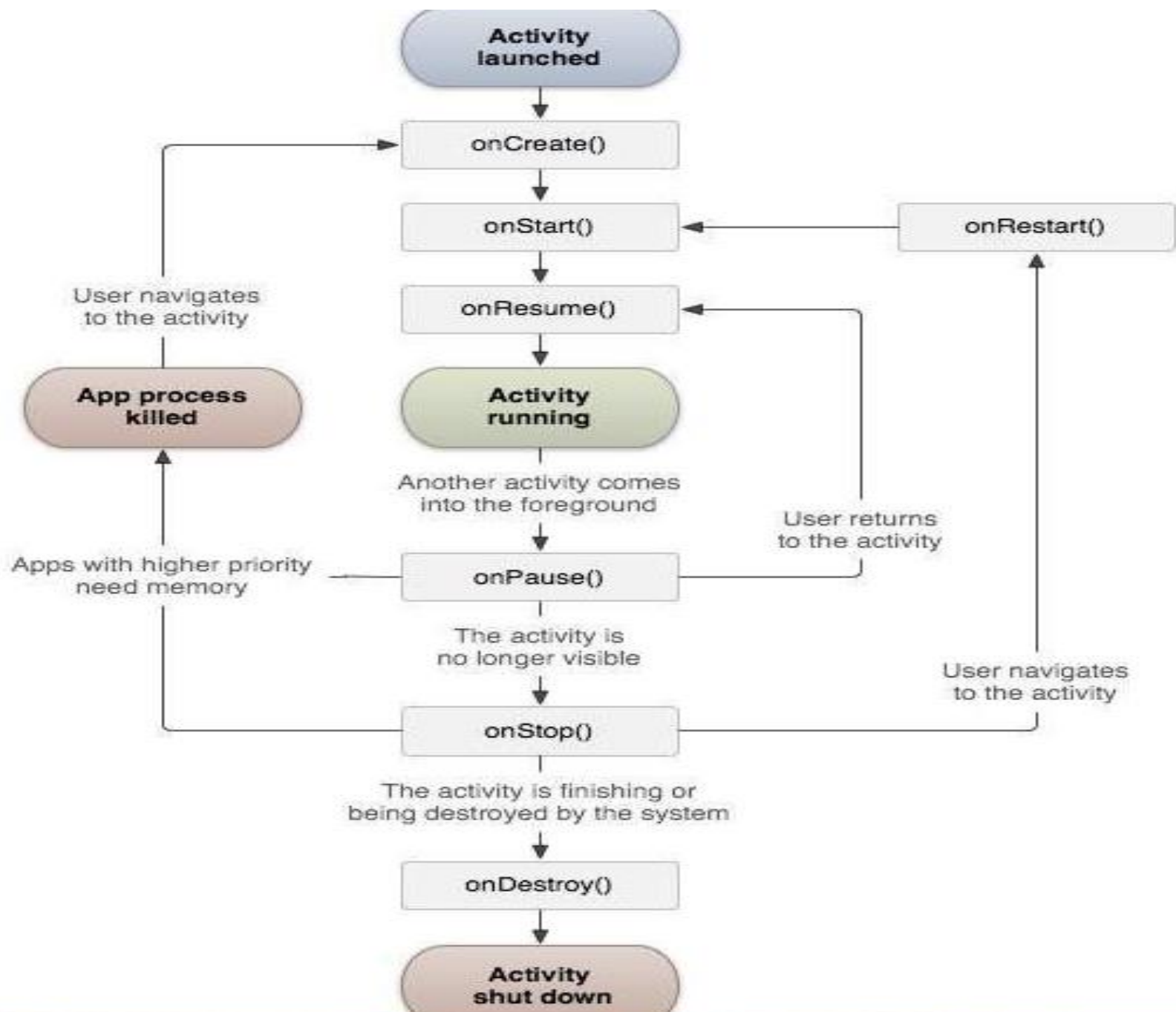
Photos

```
Intent sendIntent = new Intent();  
sendIntent.setAction(Intent.ACTION_SEND);  
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to Send");  
sendIntent.setType("text/plain");  
startActivity(sendIntent);
```



Launching Activities and Sub Activities

- An activity represents a single screen with a user interface just like window or frame of Java.
- Android system initiates its program with in an Activity starting with a call on *onCreate()* callback method.
- There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram.



Sr.No	Callback & Description
1	onCreate() This is the first callback and called when the activity is first created.
2	onStart() This callback is called when the activity becomes visible to the user.
3	onResume() This is called when the user starts interacting with the application.
4	onPause() The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	onStop() This callback is called when the activity is no longer visible.
6	onDestroy() This callback is called before the activity is destroyed by the system.
7	onRestart() This callback is called when the activity restarts after stopping it.

WebView

- WebView is a view that **display web pages inside your application.**
- In order to add WebView to your application, you have to add `<WebView>` element to your xml layout file. Its syntax is as follows –

```
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

-
- In order to **use** it, you have to **get a reference** of this view in Java file.
 - To get a reference, create an object of the class `WebView`. Its syntax is:

```
WebView browser = (WebView) findViewById(R.id.webview);
```
 - In order to **load a web url into the WebView**, you need to **call a method `loadUrl(String url)`** of the `WebView` class, specifying the required url.
 - Its syntax is:

```
browser.loadUrl("http://www.google.com/");
```

Sr.No	Method & Description
1	canGoBack() This method specifies the WebView has a back history item.
2	canGoForward() This method specifies the WebView has a forward history item.
3	clearHistory() This method will clear the WebView forward and backward history.
4	destroy() : This method destroy the internal state of WebView.
5	findAllAsync(String find) This method find all instances of string and highlight them.
6	getProgress() : This method gets the progress of the current page.
7	getTitle() : This method return the title of the current page.
8	getUrl() : This method return the url of the current page.

-
- If you click on any link inside the webpage of the WebView, that page will not be loaded inside your WebView.
 - In order to do that you need to extend your class from **WebViewClient** and override its method

```
browser = (WebView) findViewById(R.id.webView);  
WebSettings webSettings= browser.getSettings();  
webSettings.setJavaScriptEnabled(true);  
browser.loadUrl("http://www.google.com/");  
browser.setWebViewClient(new WebViewClient());
```

Creating and Invoking Services

- The primary mechanism for background work in Android is the *service*.
- An Android service is a component that is **designed to do some work without a user interface**.
- A service might **download a file, play music, or apply a filter to an image**.
- Services can also be used for interprocess communication (*IPC*) between Android applications.
- For example one Android app might use the music player service that is from another app or an app might expose data (such as a person's contact information) to other apps via a service.

Types of Services

- A service can essentially take three states –
- **Foreground**
 - A foreground service performs some operation that is noticeable to the user.
 - For example, an audio app would use a foreground service to play an audio track. Foreground services must display a Notification. Foreground services continue running even when the user isn't interacting with the app.
 - When you use a foreground service, you must display a notification so that users are actively aware that the service is running. This notification cannot be dismissed unless the service is either stopped or removed from the foreground. WorkManager API is preferable to using foreground services directly.

Types of Services

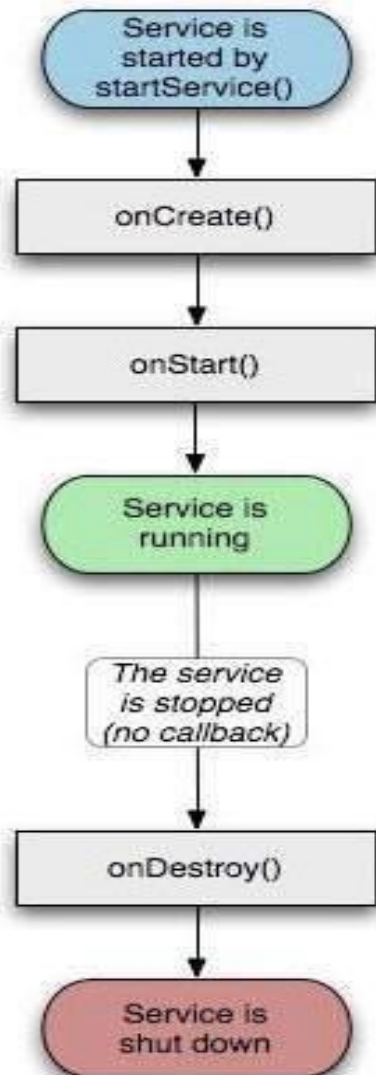
- Background

- A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

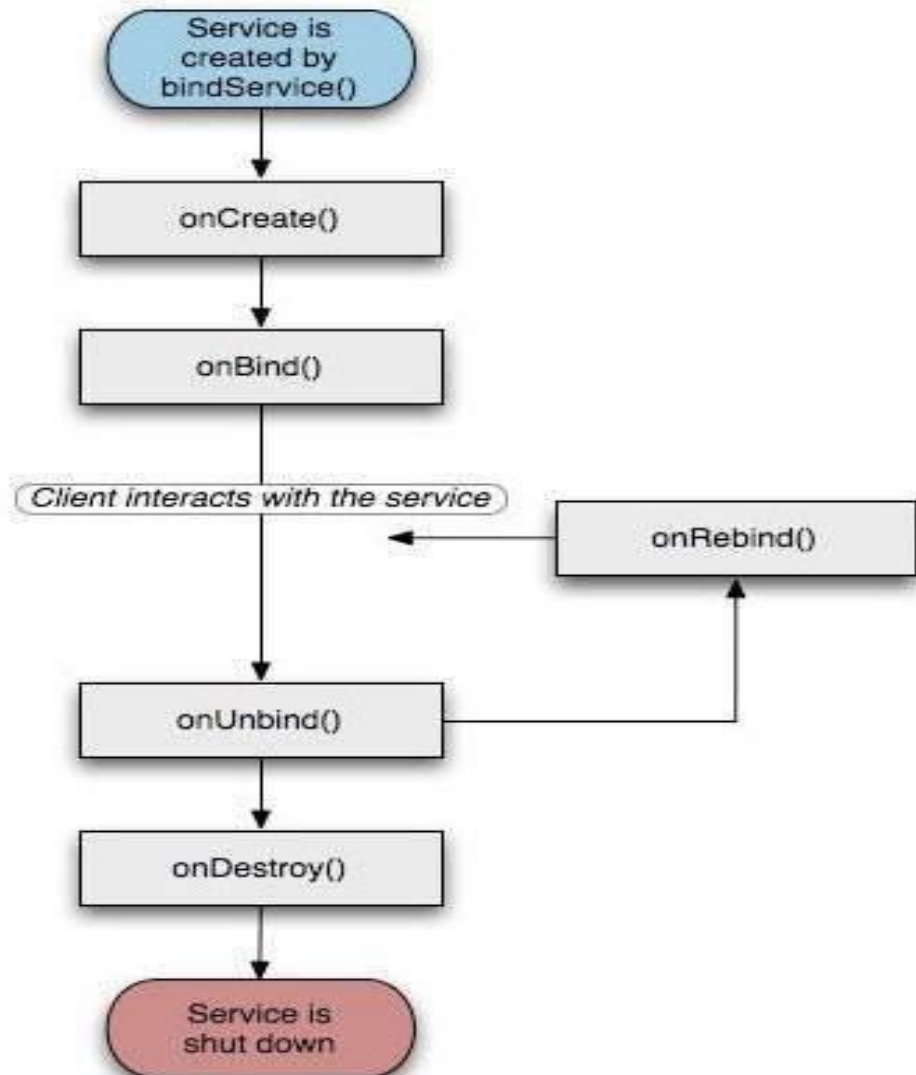
- Bound

- A service is bound when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with inter-process communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

-
- A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage.
 - The following diagram on the left shows the life cycle when the service is created with `startService()` and the diagram on the right shows the life cycle when the service is created with `bindService()`:



UN Bounded Service



Bounded services

-
- To create an service, you create a Java class that extends the Service base class or one of its existing subclasses.
 - The **Service** base class defines various callback methods below.
 - You don't need to implement all the callbacks methods.
 - However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback & Description

onStartCommand()

The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.

onBind()

The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.

onUnbind()

The system calls this method when all clients have disconnected from a particular interface published by the service.

Callback & Description

`onRebind()`

The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its `onUnbind(Intent)`.

`onCreate()`

The system calls this method when the service is first created using `onStartCommand()` or `onBind()`. This call is required to perform one-time set-up.

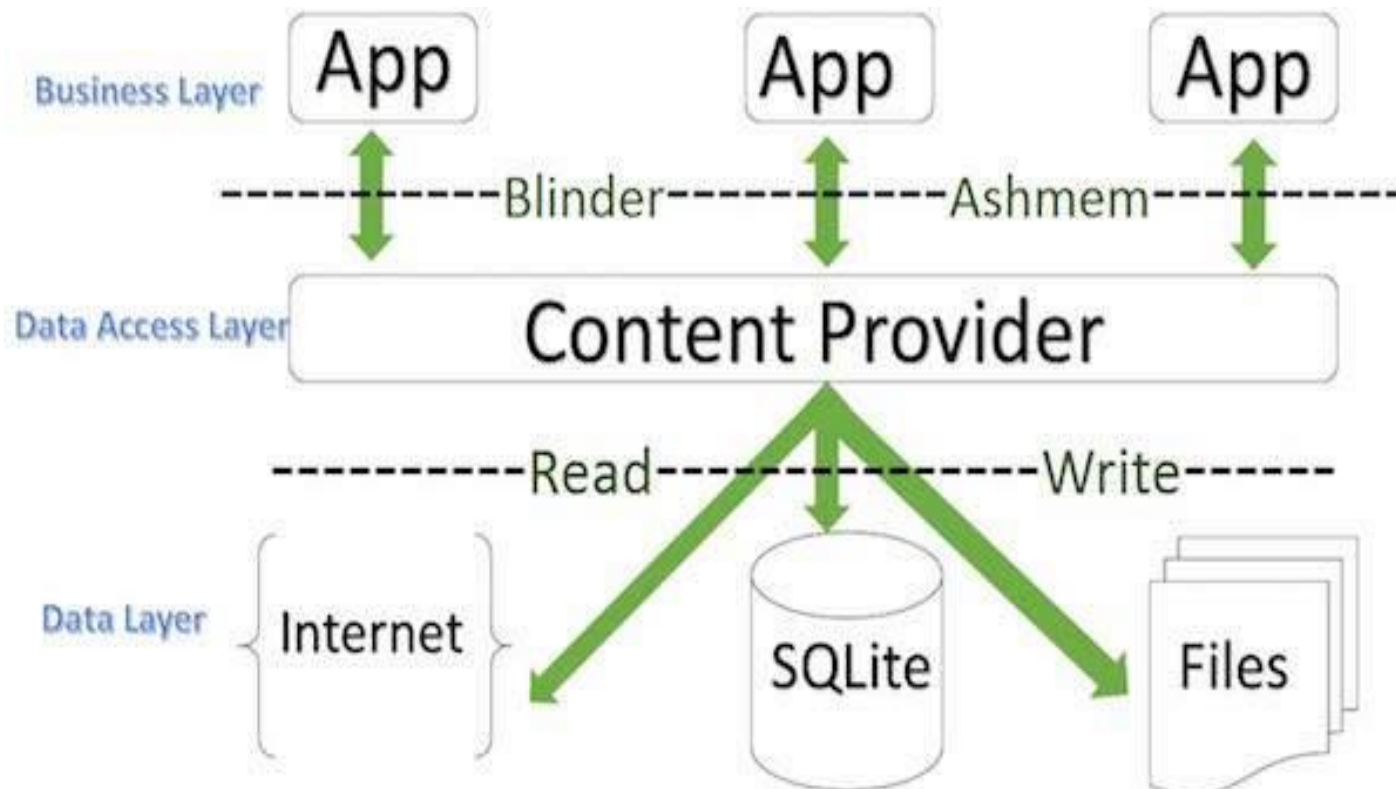
`onDestroy()`

The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

Using a Content Provider

- A content provider manages access to a central repository of data.
- A provider is part of an Android application, which often provides its own UI for working with the data.
- However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object.
- Together, providers and provider clients offer a consistent, standard interface to data that also handles inter-process communication and secure data access.

- A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



-
- A content provider coordinates access to the data storage layer in your application for a number of different APIs and components , these include:
 - Sharing access to your application data with other applications
 - Sending data to a widget
 - Returning custom search suggestions for your application through the search framework using [SearchRecentSuggestionsProvider](#)
 - Synchronizing application data with your server using an implementation of [AbstractThreadedSyncAdapter](#)
 - Loading data in your UI using a [CursorLoader](#)

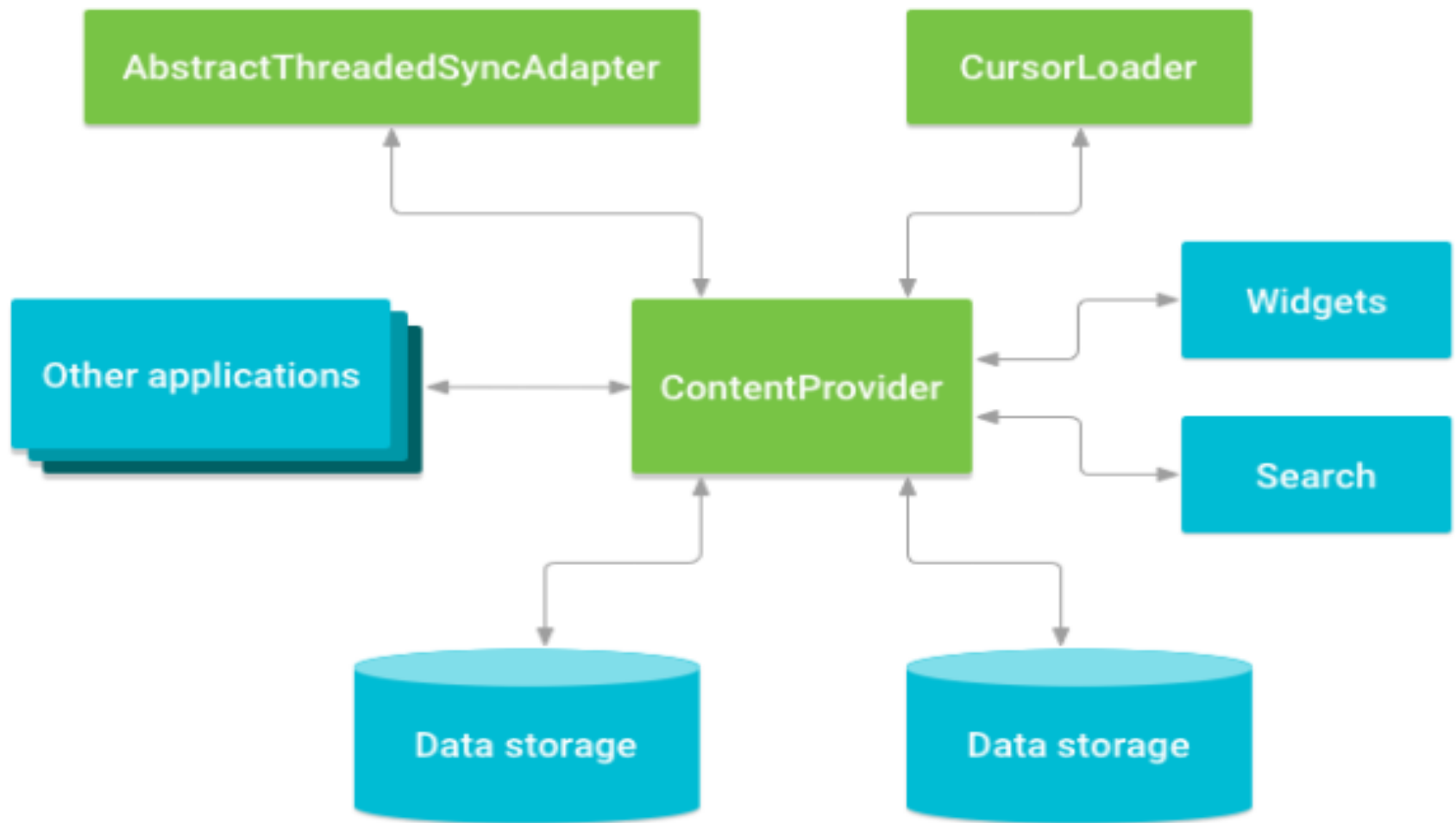


Figure 1. Relationship between content provider and other components.

Requesting and Requiring Permission

- Every Android app runs in a limited-access sandbox.
- If an app needs to use resources or information outside of its own sandbox, the app has to request the appropriate **permission**.
- You declare that in **AndroidManifest.xml**
 - put a **<uses-permission>** element in your app manifest, as a child of the top-level **<manifest>** element.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.INTERNET"/>
    <!-- other permissions go here -->

    <application ...>
        ...
    </application>
</manifest>
```

Packaging an Application(Deployment)

- In Android Studio, select **Build > Build Bundle(s) / APKs > Build APK(s)** in the menu bar.
- The APKs are saved in **AndroidStudioProjects\App-Name\app\build\outputs\apk\debug**

Note: Before you can generate a release version of your app for public distribution, you must [sign your APK](#)

Installing Application on Android Device

- We have already seen how to create .apk file and how to send into mobile device
- Finally, how we can install the .apk file

Publishing on the Android Market

- Publishing is the general process that makes your Android applications available to users.
- When you publish an Android application you perform two main tasks:
 1. You prepare the application for release.
 - During the preparation step you build a release version of your application, which users can download and install on their Android-powered devices.
 2. You release the application to users.
 - During the release step you publicize, sell, and distribute the release version of your application to users.
- For more information visit:
 - <https://developer.android.com/studio/publish/>

Tips: Android Color



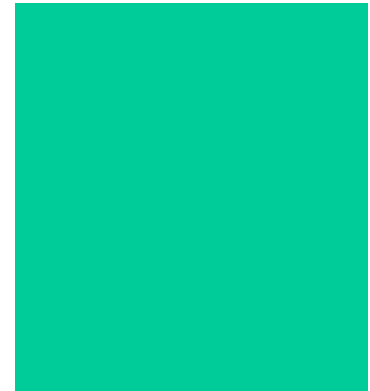
#0066cc



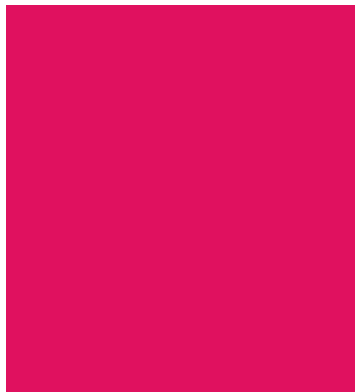
#0f52ba



#11e092



#00cc99



#e0115f



#cc0066



#d1009c



#cc0099

Tips: Android Color



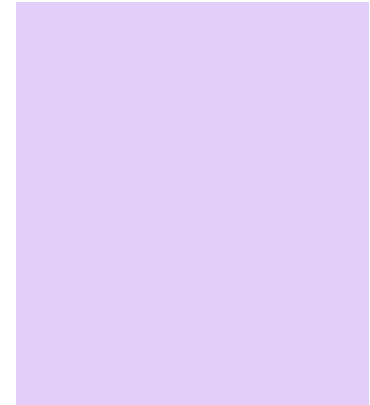
`#c9e99e`



`#861c1b`



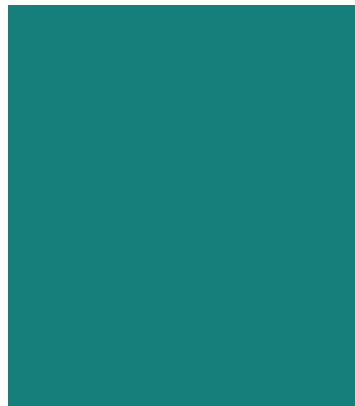
`#9d343f`



`#e3cdf9`



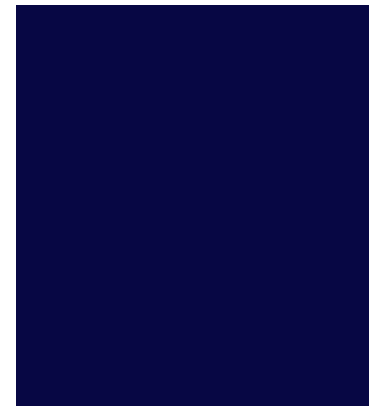
`#eea259`



`#167f7c`



`#b6b220`



`#070744`

Tips: Android Color



#720d0d



#3c3030



#799898



#4c7f18



#4b187f



#fd5e53



#4b181c



#393820



#2f434a



#32c4c1



#ff7256



#d73925

Tips: Android Color



#48a48f



#544cac



#009999



#339999



#48a48f



#ff007f



#f6546a



#ff4444



#dc143c



#fc4312

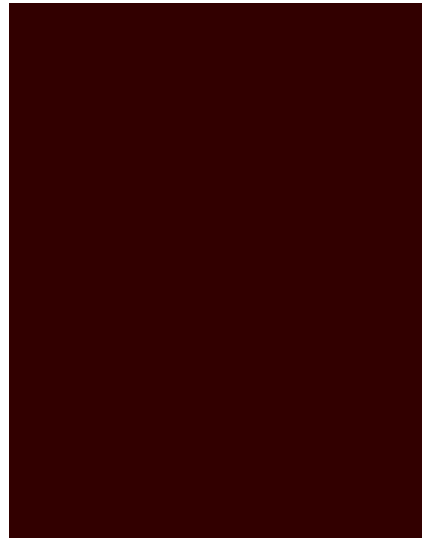


#a11607



#4b150c

Tips: Android Color



#320000



#050000

Thank You

