Data and file storage overview

Android uses a file system that's similar to disk-based file systems on other platforms. The system provides several options for you to save your app data:

- **App-specific storage:** Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
- **Shared storage:** Store files that your app intends to share with other apps, including media, documents, and other files.
- **Preferences:** Store private, primitive data in key-value pairs.
- **Databases:** Store structured data in a private database using the Room persistence library.

The characteristics of these options are summarized in the following table:

	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files (/training/data-storage/app-specific)	Files meant for your app's use only	From internal storage, `getFilesDir()` or `getCacheDir()` From external storage, `getExternalFilesDir()` or `getExternalCacheDir()`	Never needed for internal storage Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	No	Yes
Media (/training/data-storage/shared/media)	Shareable media files (images, audio files, videos)	`MediaStore` API	`READ_EXTERNAL_STORAGE` when accessing other apps' files on Android 11 (API level 30) or higher `READ_EXTERNAL_STORAGE` or `WRITE_EXTERNAL_STORAGE` when accessing other apps' files on Android 10 (API level 29) Permissions are required for **all** files on Android 9 (API level 28) or lower	Yes, though the other app needs the `READ_EXTERNAL_STORAGE` permission	No
Documents and other files (/training/data-storage/shared/documents-files)	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker	No
App preferences (/training/data-storage/shared-preferences)	Key-value pairs	Jetpack Preferences (/guide/topics/ui/settings/use-saved-values) library	None	No	Yes
Database	Structured data	Room (/training/data-storage/room) persistence library	None	No	Yes

The solution you choose depends on your specific needs:

How much space does your data require?

Internal storage has limited space for app-specific data. Use other types of storage if you need to save a substantial amount of data.

How reliable does data access need to be?

If your app's basic functionality requires certain data, such as when your app is starting up, place the data within internal storage directory or a database. App-specific files that are stored in external storage aren't always accessible because some devices allow users to remove a physical device that corresponds to external storage.

What kind of data do you need to store?

If you have data that's only meaningful for your app, use app-specific storage. For shareable media content, use shared storage so that other apps can access the content. For structured data, use either preferences (for key-value data) or a database (for data that contains more than 2 columns).

Should the data be private to your app?

When storing sensitive data—data that shouldn't be accessible from any other app—use internal storage, preferences, or a database. Internal storage has the added benefit of the data being hidden from users.

Categories of storage locations

Android provides two types of physical storage locations: *internal storage* and *external storage*. On most devices, internal storage is smaller than external storage. However, internal storage is always available on all devices, making it a more reliable place to put data on which your app depends.

Removable volumes, such as an SD card, appear in the file system as part of external storage. Android represents these devices using a path, such as `/sdcard`.

Caution: The exact location of where your files can be saved might vary across devices. For this reason, don't use hard-coded file paths.

Apps themselves are stored within internal storage by default. If your APK size is very large, however, you can indicate a preference within your app's manifest file to install your app on external storage instead:

```
<manifest ...
  android:installLocation="preferExternal">
  ...
</manifest>
```

Permissions and access to external storage

Android defines the following storage-related permissions: `READ_EXTERNAL_STORAGE` (/reference/android/Manifest.permission#READ_EXTERNAL_STORAGE), `WRITE_EXTERNAL_STORAGE` (/reference/android/Manifest.permission#WRITE_EXTERNAL_STORAGE), and `MANAGE_EXTERNAL_STORAGE` (/reference/android/Manifest.permission#MANAGE_EXTERNAL_STORAGE).

On earlier versions of Android, apps needed to declare the `READ_EXTERNAL_STORAGE` permission to access any file outside the app-specific directories (/training/data-storage/app-specific#external) on external storage. Also, apps needed to declare the `WRITE_EXTERNAL_STORAGE` permission to write to any file outside the app-specific directory.

More recent versions of Android rely more on a file's purpose than its location for determining an app's ability to access, and write to, a given file. In particular, if your app targets Android 11 (API level 30) or higher, the `WRITE_EXTERNAL_STORAGE` permission doesn't have any effect on your app's access to storage. This purpose-based storage model improves user privacy because apps are given access only to the areas of the device's file system that they actually use.

Android 11 introduces the `MANAGE_EXTERNAL_STORAGE` permission, which provides write access to files outside the app-specific directory and `MediaStore`. To learn more about this permission, and why most apps don't need to declare it to fulfill their use cases, see the guide on how to manage all files (/training/data-storage/manage-all-files) on a storage device.

Scoped storage

To give users more control over their files and to limit file clutter, apps that target Android 10 (API level 29) and higher are given scoped access into external storage, or *scoped storage*, by default. Such apps have access only to the app-specific directory on external storage, as well as specific types of media that the app has created.

Note: If your app requests a storage-related permission at runtime, the user-facing dialog indicates that your app is requesting broad access to external storage, even when scoped storage is enabled.

Use scoped storage unless your app needs access to a file that's stored outside of an app-specific directory (/training/data-storage/app-specific) and outside of a directory that the `MediaStore` (/reference/android/provider/MediaStore) APIs can access. If you store app-specific files on external storage, you can make it easier to adopt scoped storage by placing these files in an app-specific directory on external storage (/training/data-storage/app-specific#external) . That way, your app maintains access to these files when scoped storage is enabled.

To prepare your app for scoped storage, view the storage use cases and best practices (/training/data-storage/use-cases) guide. If your app has another use case that isn't covered by scoped storage, file a feature request (https://source.android.com/setup/contribute/report-bugs). You can temporarily opt-out of using scoped storage (/training/data-storage/use-cases#opt-out-scoped-storage).

View files on a device

To view the files stored on a device, use Android Studio's Device File Explorer (/studio/debug/device-file-explorer) .

Additional resources

For more information about data storage, consult the following resources.

Videos

- Preparing for Scoped Storage (Android Dev Summit '19) (https://www.youtube.com/watch?v=UnJ3amzJM94)