Save a file on internal storage

Your app's internal storage directory is specified by your app's package name in a special location of the Android file system that can be accessed with the following APIs.

**Note:** Unlike the [external storage directories](https://developer.android.com/training/data-storage/files" \l "WriteExternalStorage), your app does not require any system permissions to read and write to the internal directories returned by these methods.

Write a file

When saving a file to internal storage, you can acquire the appropriate directory as a [File](https://developer.android.com/reference/java/io/File.html) by calling one of two methods:

[getFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getFilesDir())

Returns a [File](https://developer.android.com/reference/java/io/File.html) representing an internal directory for your app.

[getCacheDir()](https://developer.android.com/reference/android/content/Context.html#getCacheDir())

Returns a [File](https://developer.android.com/reference/java/io/File.html) representing an internal directory for your app's temporary cache files. Be sure to delete each file once it is no longer needed and implement a reasonable size limit for the amount of memory you use at any given time, such as 1MB.

**Caution:** If the system runs low on storage, it may delete your cache files without warning.

To create a new file in one of these directories, you can use the [File()](https://developer.android.com/reference/java/io/File.html#File(java.io.File,%20java.lang.String)) constructor, passing the [File](https://developer.android.com/reference/java/io/File.html) provided by one of the above methods that specifies your internal storage directory. For example:

KOTLIN

JAVA

val file = File(context.filesDir, filename)

Alternatively, you can call [openFileOutput()](https://developer.android.com/reference/android/content/Context.html" \l "openFileOutput(java.lang.String,%20int)) to get a [FileOutputStream](https://developer.android.com/reference/java/io/FileOutputStream.html) that writes to a file in your internal directory. For example, here's how to write some text to a file:

KOTLIN

JAVA

val filename = "myfile"  
val fileContents = "Hello world!"  
context.openFileOutput(filename, Context.MODE\_PRIVATE).use {  
        it.write(fileContents.toByteArray())  
}

Notice that the [openFileOutput()](https://developer.android.com/reference/android/content/Context.html" \l "openFileOutput(java.lang.String,%20int)) method requires a file mode parameter. Passing [MODE\_PRIVATE](https://developer.android.com/reference/android/content/Context.html#MODE_PRIVATE) makes it private to your app. The other mode options, [MODE\_WORLD\_READABLE](https://developer.android.com/reference/android/content/Context.html#MODE_WORLD_READABLE) and [MODE\_WORLD\_WRITEABLE](https://developer.android.com/reference/android/content/Context.html#MODE_WORLD_WRITEABLE), have been deprecated since API level 17. Starting with Android 7.0 (API level 24), Android throws a [SecurityException](https://developer.android.com/reference/java/lang/SecurityException.html) if you use them. If your app needs to share private files with other apps, you should instead use a [FileProvider](https://developer.android.com/reference/android/support/v4/content/FileProvider.html) with the [FLAG\_GRANT\_READ\_URI\_PERMISSION](https://developer.android.com/reference/android/content/Intent.html#FLAG_GRANT_READ_URI_PERMISSION). For more information, see [Sharing Files](https://developer.android.com/training/secure-file-sharing/index.html).

On Android 6.0 (API level 23) and lower, other apps can read your internal files if you set the file mode to be world readable. However, the other app must know your app package name and file names. Other apps cannot browse your internal directories and do not have read or write access unless you explicitly set the files to be readable or writable. So as long as you use [MODE\_PRIVATE](https://developer.android.com/reference/android/content/Context.html#MODE_PRIVATE) for your files on the internal storage, they are never accessible to other apps.

Write a cache file

If you instead need to cache some files, you should use [createTempFile()](https://developer.android.com/reference/java/io/File.html" \l "createTempFile(java.lang.String,%20java.lang.String)). For example, the following method extracts the file name from a [URL](https://developer.android.com/reference/java/net/URL.html) and creates a file with that name in your app's internal cache directory:

KOTLIN

JAVA

private fun getTempFile(context: Context, url: String): File? =  
        Uri.parse(url)?.lastPathSegment?.let { filename ->  
            File.createTempFile(filename, null, context.cacheDir)  
        }

Files created with [createTempFile()](https://developer.android.com/reference/java/io/File.html" \l "createTempFile(java.lang.String,%20java.lang.String)) are placed in a cache directory that's private to your app. You should regularly [delete the files](https://developer.android.com/training/data-storage/files#DeleteFile) you no longer need.

**Caution:** If the system runs low on storage, it may delete your cache files without warning, so make sure you check for the existence of your cache files before reading them.

Open an existing file

To read an existing file, call [openFileInput(name)](https://developer.android.com/reference/android/content/Context.html" \l "openFileInput(java.lang.String)), passing the name of the file.

You can get an array of all your app's file names by calling [fileList()](https://developer.android.com/reference/android/content/Context.html" \l "fileList()).

**Tip:** If you need to package a file in your app that is accessible at install time, save the file in your project's **res/raw/**directory. You can open these files with **[openRawResource()](https://developer.android.com/reference/android/content/res/Resources.html" \l "openRawResource(int))**, passing the **R.raw.*filename*** resource ID. This method returns an **[InputStream](https://developer.android.com/reference/java/io/InputStream.html)** that you can use to read the file. You cannot write to the original file.

Open a directory

You can open a directory on the internal file system with the following methods:

[getFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getFilesDir())

Returns a [File](https://developer.android.com/reference/java/io/File.html) representing the directory on the file system that's uniquely associated with your app.

[getDir(name, mode)](https://developer.android.com/reference/android/content/Context.html#getDir(java.lang.String,%20int))

Creates a new directory (or opens an existing directory) within your app's unique file system directory. This new directory appears inside the directory provided by [getFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getFilesDir()).

[getCacheDir()](https://developer.android.com/reference/android/content/Context.html#getCacheDir())

Returns a [File](https://developer.android.com/reference/java/io/File.html) representing the cache directory on the file system that's uniquely associated with your app. This directory is meant for temporary files, and it should be cleaned up regularly. The system may delete files there if it runs low on disk space, so make sure you check for the existence of your cache files before reading them.

To create a new file in one of these directories, you can use the [File()](https://developer.android.com/reference/java/io/File.html#File(java.io.File,%20java.lang.String)) constructor, passing the [File](https://developer.android.com/reference/java/io/File.html) object provided by one of the above methods that specifies your internal storage directory. For example:

KOTLIN

JAVA

val directory = context.filesDir  
val file = File(directory, filename)

Save a file on external storage

Using the external storage is great for files that you want to share with other apps or allow the user to access with a computer.

After you [request storage permissions](https://developer.android.com/training/data-storage/files" \l "ExternalStoragePermissions) and [verify that storage is available](https://developer.android.com/training/data-storage/files#CheckExternalAvail), you can save two different types of files:

* [Public files](https://developer.android.com/training/data-storage/files#PublicFiles): Files that should be freely available to other apps and to the user. When the user uninstalls your app, these files should remain available to the user. For example, photos captured by your app or other downloaded files should be saved as public files.
* [Private files](https://developer.android.com/training/data-storage/files" \l "PrivateFiles): Files that rightfully belong to your app and will be deleted when the user uninstalls your app. Although these files are technically accessible by the user and other apps because they are on the external storage, they don't provide value to the user outside of your app.

**Caution:** The external storage might become unavailable if the user removes the SD card or connects the device to a computer. And the files are still visible to the user and other apps that have the [**READ\_EXTERNAL\_STORAGE**](https://developer.android.com/reference/android/Manifest.permission.html#READ_EXTERNAL_STORAGE) permission. So if your app's functionality depends on these files or you need to completely restrict access, you should instead write your files to the [internal storage](https://developer.android.com/training/data-storage/files#WriteInternalStorage).

Request external storage permissions

To write to the public external storage, you must request the [WRITE\_EXTERNAL\_STORAGE](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE) permission in your [manifest file](https://developer.android.com/guide/topics/manifest/manifest-intro.html):

<manifest ...>  
    <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
    ...  
</manifest>

**Note:** If your app uses the [**WRITE\_EXTERNAL\_STORAGE**](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE) permission, then it implicitly has permission to read the external storage as well.

If your app only needs to read the external storage (but not write to it), then you need to declare the[READ\_EXTERNAL\_STORAGE](https://developer.android.com/reference/android/Manifest.permission.html" \l "READ_EXTERNAL_STORAGE) permission:

<manifest ...>  
    <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  
    ...  
</manifest>

Beginning with Android 4.4 (API level 19), reading or writing files in your app's private external storage directory—accessed using [getExternalFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String))—does not require the [READ\_EXTERNAL\_STORAGE](https://developer.android.com/reference/android/Manifest.permission.html#READ_EXTERNAL_STORAGE) or [WRITE\_EXTERNAL\_STORAGE](https://developer.android.com/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE) permissions. So if your app supports Android 4.3 (API level 18) and lower, and you want to access only the private external storage directory, you should declare that the permission be requested only on the lower versions of Android by adding the [maxSdkVersion](https://developer.android.com/guide/topics/manifest/uses-permission-element.html" \l "maxSdk) attribute:

<manifest ...>  
    <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"  
                     android:maxSdkVersion="18" />  
    ...  
</manifest>

Verify that external storage is available

Because the external storage might be unavailable—such as when the user has mounted the storage to a PC or has removed the SD card that provides the external storage—you should always verify that the volume is available before accessing it. You can query the external storage state by calling [getExternalStorageState()](https://developer.android.com/reference/android/os/Environment.html" \l "getExternalStorageState()). If the returned state is [MEDIA\_MOUNTED](https://developer.android.com/reference/android/os/Environment.html#MEDIA_MOUNTED), then you can read and write your files. If it's [MEDIA\_MOUNTED\_READ\_ONLY](https://developer.android.com/reference/android/os/Environment.html#MEDIA_MOUNTED_READ_ONLY), you can only read the files.

For example, the following methods are useful to determine the storage availability:

KOTLIN

JAVA

/\* Checks if external storage is available for read and write \*/  
fun isExternalStorageWritable(): Boolean {  
    return Environment.getExternalStorageState() == Environment.MEDIA\_MOUNTED  
}  
  
/\* Checks if external storage is available to at least read \*/  
fun isExternalStorageReadable(): Boolean {  
     return Environment.getExternalStorageState() in  
        setOf(Environment.MEDIA\_MOUNTED, Environment.MEDIA\_MOUNTED\_READ\_ONLY)  
}

Save to a public directory

If you want to save public files on the external storage, use the [getExternalStoragePublicDirectory()](https://developer.android.com/reference/android/os/Environment.html" \l "getExternalStoragePublicDirectory(java.lang.String)) method to get a [File](https://developer.android.com/reference/java/io/File.html) representing the appropriate directory on the external storage. The method takes an argument specifying the type of file you want to save so that they can be logically organized with other public files, such as [DIRECTORY\_MUSIC](https://developer.android.com/reference/android/os/Environment.html#DIRECTORY_MUSIC) or[DIRECTORY\_PICTURES](https://developer.android.com/reference/android/os/Environment.html#DIRECTORY_PICTURES). For example:

KOTLIN

JAVA

fun getPublicAlbumStorageDir(albumName: String): File? {  
    // Get the directory for the user's public pictures directory.  
    val file = File(Environment.getExternalStoragePublicDirectory(  
            Environment.DIRECTORY\_PICTURES), albumName)  
    if (!file?.mkdirs()) {  
        Log.e(LOG\_TAG, "Directory not created")  
    }  
    return file  
}

If you want to hide your files from the Media Scanner, include an empty file named .nomedia in your external files directory (note the dot prefix in the filename). This prevents media scanner from reading your media files and providing them to other apps through the [MediaStore](https://developer.android.com/reference/android/provider/MediaStore.html) content provider.

Save to a private directory

If you want to save files on external storage that are private to your app and not accessible by the [MediaStore](https://developer.android.com/reference/android/provider/MediaStore.html) content provider, you can acquire a directory that's used by only your app by calling [getExternalFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String)) and passing it a name indicating the type of directory you'd like. Each directory created this way is added to a parent directory that encapsulates all your app's external storage files, which the system deletes when the user uninstalls your app.

**Caution: Files on external storage are not always accessible**, because users can mount the external storage to a computer for use as a storage device. So if you need to store files that are critical to your app's functionality, you should instead store them on [internal storage](https://developer.android.com/training/data-storage/files" \l "WriteInternalStorage).

For example, here's a method you can use to create a directory for an individual photo album:

KOTLIN

JAVA

fun getPrivateAlbumStorageDir(context: Context, albumName: String): File? {  
    // Get the directory for the app's private pictures directory.  
    val file = File(context.getExternalFilesDir(  
            Environment.DIRECTORY\_PICTURES), albumName)  
    if (!file?.mkdirs()) {  
        Log.e(LOG\_TAG, "Directory not created")  
    }  
    return file  
}

If none of the pre-defined sub-directory names suit your files, you can instead call [getExternalFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String)) and passnull. This returns the root directory for your app's private directory on the external storage.

Remember that [getExternalFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String)) creates a directory that is deleted when the user uninstalls your app. If the files you're saving should remain available after the user uninstalls your app—such as when your app captures photos and the user should keep those photos—you should instead [save the files to a public directory](https://developer.android.com/training/data-storage/files" \l "PublicFiles).

Regardless of whether you use [getExternalStoragePublicDirectory()](https://developer.android.com/reference/android/os/Environment.html" \l "getExternalStoragePublicDirectory(java.lang.String)) for files that are shared or[getExternalFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String)) for files that are private to your app, it's important that you use directory names provided by API constants like [DIRECTORY\_PICTURES](https://developer.android.com/reference/android/os/Environment.html" \l "DIRECTORY_PICTURES). These directory names ensure that the files are treated properly by the system. For instance, files saved in [DIRECTORY\_RINGTONES](https://developer.android.com/reference/android/os/Environment.html" \l "DIRECTORY_RINGTONES) are categorized by the system media scanner as ringtones instead of music.

Select between multiple storage locations

Sometimes, a device that allocates a partition of the internal memory for use as the external storage *also* provides an SD card slot. This means that the device has two different external storage directories, so you need to select which one to use when writing "private" files to the external storage.

Beginning with Android 4.4 (API level 19), you can access both locations by calling [getExternalFilesDirs()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDirs(java.lang.String)), which returns a [File](https://developer.android.com/reference/java/io/File.html) array with entries for each storage location. The first entry in the array is considered the primary external storage, and you should use that location unless it's full or unavailable.

If your app supports Android 4.3 and lower, you should use the support library's static method, [ContextCompat.getExternalFilesDirs()](https://developer.android.com/reference/android/support/v4/content/ContextCompat.html" \l "getExternalFilesDirs(android.content.Context,%20java.lang.String)). This always returns a [File](https://developer.android.com/reference/java/io/File.html) array, but if the device is running Android 4.3 and lower, then it contains just one entry for the primary external storage (if there's a second storage location, you cannot access it on Android 4.3 and lower).

Query free space

If you know ahead of time how much data you're saving, you can find out whether sufficient space is available without causing an [IOException](https://developer.android.com/reference/java/io/IOException.html) by calling [getFreeSpace()](https://developer.android.com/reference/java/io/File.html" \l "getFreeSpace()) or [getTotalSpace()](https://developer.android.com/reference/java/io/File.html" \l "getTotalSpace()). These methods provide the current available space and the total space in the storage volume, respectively. This information is also useful to avoid filling the storage volume above a certain threshold.

However, the system does not guarantee that you can write as many bytes as are indicated by [getFreeSpace()](https://developer.android.com/reference/java/io/File.html" \l "getFreeSpace()). If the number returned is a few MB more than the size of the data you want to save, or if the file system is less than 90% full, then it's okay to proceed. Otherwise, you probably shouldn't write to storage.

**Note:** You aren't required to check the amount of available space before you save your file. You can instead try writing the file right away, then catch an **[IOException](https://developer.android.com/reference/java/io/IOException.html)** if one occurs. You may need to do this if you don't know exactly how much space you need. For example, if you change the file's encoding before you save it by converting a PNG image to JPEG, you won't know the file's size beforehand.

Delete a file

You should always delete files that your app no longer need. The most straightforward way to delete a file is to call [delete()](https://developer.android.com/reference/java/io/File.html#delete()) on the [File](https://developer.android.com/reference/java/io/File.html) object.

KOTLIN

JAVA

myFile.delete()

If the file is saved on internal storage, you can also ask the [Context](https://developer.android.com/reference/android/content/Context.html) to locate and delete a file by calling [deleteFile()](https://developer.android.com/reference/android/content/Context.html" \l "deleteFile(java.lang.String)):

KOTLIN

JAVA

myContext.deleteFile(fileName)

**Note:** When the user uninstalls your app, the Android system deletes the following:

* All files you saved on internal storage.
* All files you saved external storage using **[getExternalFilesDir()](https://developer.android.com/reference/android/content/Context.html" \l "getExternalFilesDir(java.lang.String))**.

However, you should manually delete all cached files created with **[getCacheDir()](https://developer.android.com/reference/android/content/Context.html" \l "getCacheDir())** on a regular basis and also regularly delete other files you no longer need.