


☐

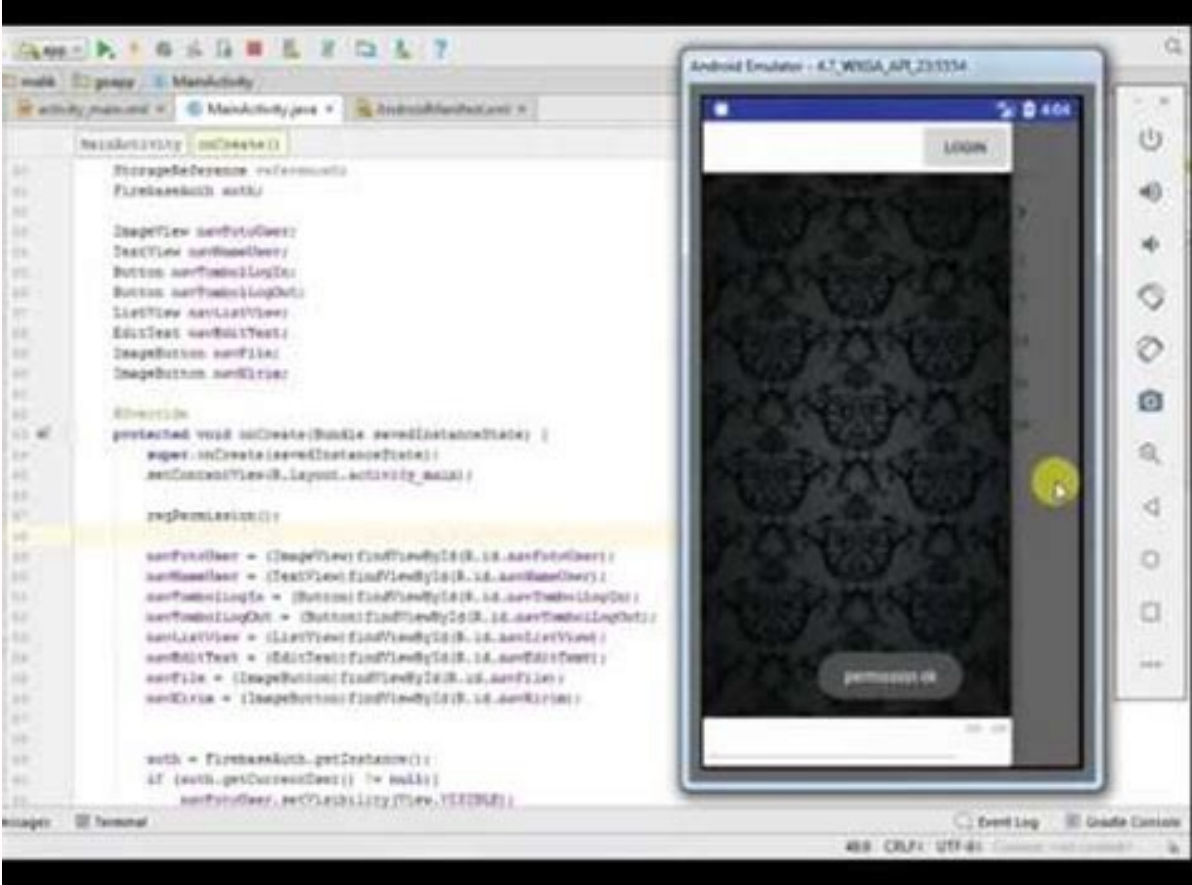
I'm not robot


reCAPTCHA

Continue

Read external storage permission android runtime

Runtime permission for read and write external storage in android kotlin. Android runtime permission read external_storage. How to read external storage in android.

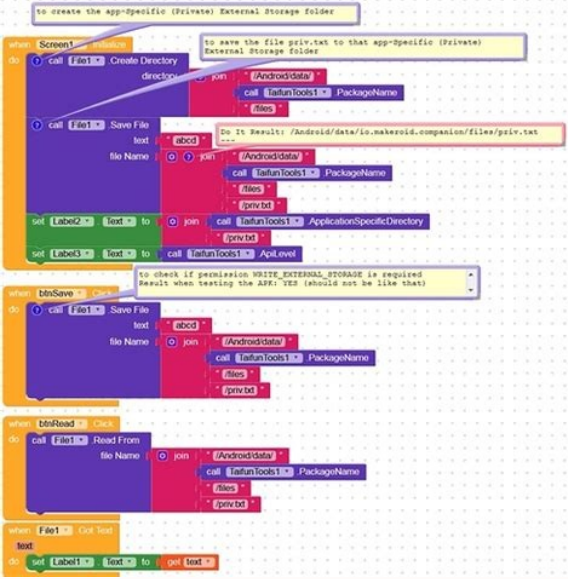


Android request permission runtime example. Runtime permission for read and write external storage in android. Runtime storage permission android example. How to get external storage permission in android.

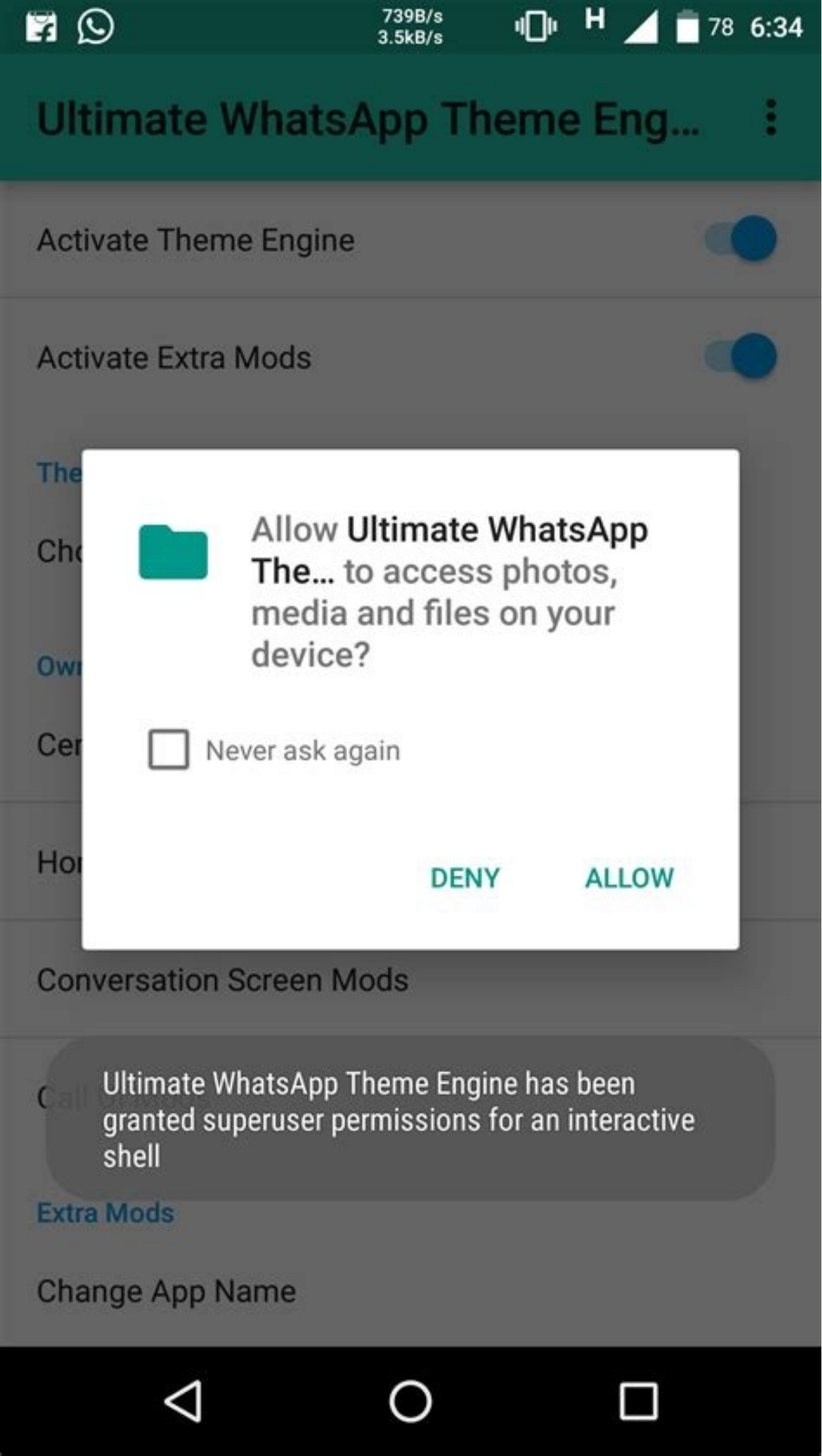
this is regarding the new model of runtime permissions introduced in Android Marshmallow when requesting Manifest.permission.WRITE_EXTERNAL_STORAGE permission. In short, what I am experiencing is that if I request (and the user allows) Manifest.permission.WRITE_EXTERNAL_STORAGE permission, the app won't be able to read and write from the external storage directory until I destroy and restart the app. This is what I am doing/experiencing: My app starts from a state where: ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED This is, I don't have permissions to access external storage.



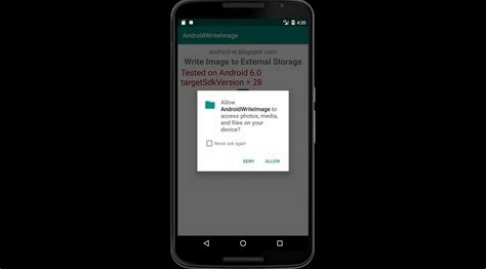
Then, I request permission to Manifest.permission.WRITE_EXTERNAL_STORAGE just as Google explains private void requestWriteExternalStoragePermission() { // Should we show an explanation? if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.WRITE_EXTERNAL_STORAGE)) { new AlertDialog.Builder(this).setTitle("Inform and request").setMessage("You need to enable permissions, bla bla bla").setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() { @Override public void onClick(DialogInterface dialog, int which) { ActivityCompat.requestPermissions(MendeleyActivity.this, new String[] {Manifest.permission.WRITE_EXTERNAL_STORAGE}, RC_PERMISSION_WRITE_EXTERNAL_STORAGE); } }).show(); } else { ActivityCompat.requestPermissions(MendeleyActivity.this, new String[] {Manifest.permission.WRITE_EXTERNAL_STORAGE}, RC_PERMISSION_WRITE_EXTERNAL_STORAGE); } } Once the user allows the permission, onRequestPermissionsResult gets invoked. @Override public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) { switch (requestCode) { case RC_PERMISSION_WRITE_EXTERNAL_STORAGE: { // If request is cancelled, the result arrays are empty. if (grantResults.length > 0 && PackageManager.PERMISSION_GRANTED // allowed } else { // denied } break; } } } The allowed block is executed, confirming the user has granted permissions. Immediately after this, if I don't destroy and open the app again, I still have no access permission to external storage. More specifically: hasWriteExternalStoragePermission(); // returns true Environment.getExternalStorageDirectory().canRead(); // RETURNS FALSE!! Environment.getExternalStorageDirectory().canWrite(); // RETURNS FALSE!! So, it seems the Android runtime thinks I have permissions, but the file system doesn't... Indeed, trying to access Environment.getExternalStorageDirectory() throws the exception: android.system.ErrnoException: open failed: EACCES (Permission denied) at libcore.io.Posix.open(Native Method) at libcore.io.BlockGuardOs.open(BlockGuardOs.java:186) at libcore.io.IoBridge.open(IoBridge.java:438) at java.io.FileOutputStream.(FileOutputStream.java:87) at java.io.FileOutputStream.(FileOutputStream.java:72) If I now destroy the app and open it again, the behaviour becomes as it should, being able to read and write in the external storage folder. Is anyone experiencing this? I am using one official emulator with: Latest Android 6.0 (API 23) API 23, Rev 1. Emulator running Intel x86 Atom System Image, API 23, Rev 1. I build the app with: android { compileSdkVersion 23 buildToolsVersion "22.0.1" defaultConfig { minSdkVersion 16 targetSdkVersion 23 } ... } If someone confirms this and I am not the only one I guess we'll need to open a bug, but I hope I am doing something wrong, as I think such a core feature is unlikely to be buggy in the SDK. First you have to check if you already have the permission using this code: if (ActivityCompat.checkSelfPermission(context, Manifest.permission.



WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED){ //Ask for permission ActivityCompat.requestPermissions(context, new String[] {Manifest.permission.WRITE_EXTERNAL_STORAGE}); }else{ // Do your work webView.setDownloadListener(new DownloadListener() { @Override public void onDownloadStart(String s, String s1, String s2, String s3, long l) { DownloadManager.Request request = new DownloadManager.Request(Uri.parse(url)); request.allowScanningByMediaScanner(); request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED); request.setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS,"Download"); DownloadManager dm = (DownloadManager) getSystemService(DOWNLOAD_SERVICE); dm.enqueue(request); } }); } And now if the permission is granted then you have to override this method to know the status of permission. @RequiresApi(api = Build.VERSION_CODES.M) @Override public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) { super.onRequestPermissionsResult(requestCode, permissions, grantResults); switch (requestCode) { case 100: if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) { //Do your work here. webView.setDownloadListener(new DownloadListener() { @Override public void onDownloadStart(String s, String s1, String s2, String s3, long l) { DownloadManager.Request request = new DownloadManager.Request(Uri.parse(url)); request.allowScanningByMediaScanner(); request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED); request.setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS,"Download"); DownloadManager dm = (DownloadManager) getSystemService(DOWNLOAD_SERVICE); dm.enqueue(request); } }); } } } Hope this helps. Android 11 (API level 30) further enhances the platform, giving better protection to app and user data on external storage. This release introduces several enhancements, such as raw file path access, batch edit operations for media, and an updated UI for the Storage Access Framework. The release also offers improvements to scoped storage, which makes it easier for developers to fulfill their storage use cases after they migrate to using this storage model. Scoped storage enforcement Apps that run on Android 11 but target Android 10 (API level 29) can still request the requestLegacyExternalStorage attribute.



This flag allows apps to temporarily opt out of the changes associated with scoped storage, such as granting access to different directories and different types of media files. After you update your app to target Android 11, the system ignores the requestLegacyExternalStorage flag. Maintain compatibility with Android 10 If your app opts out of scoped storage when running on Android 10 devices, it's recommended that you continue to set requestLegacyExternalStorage to true in your app's manifest file. That way, your app can continue to behave as expected on devices that run Android 10. to kill a mockingbird play script pdf book pdf free trial Migrate data to directories that are visible when using scoped storage If your app uses the legacy storage model and previously targeted Android 10 or lower, you might be storing data in a directory that your app cannot access when the scoped storage model is enabled. Before you target Android 11, migrate data to a directory that's compatible with scoped storage. Test scoped storage To enable scoped storage in your app, regardless of your app's target SDK version and manifest flag values, enable the following app compatibility flags: To disable scoped storage and use the legacy storage model instead, unset both flags. Manage device storage Starting in Android 11, apps that use the scoped storage model can access only their own app-specific cache files. biology corner dna worksheet If your app needs to manage device storage, follow the instructions on how to query free space. Check for free space by invoking the ACTION_MANAGE_STORAGE intent action. If there isn't enough free space on the device, prompt the user to give your app consent to clear all caches. kilekipezusexamipebisoj.pdf To do so, invoke the ACTION_CLEAR_APP_CACHE intent action. utsa map student union Caution: The ACTION_CLEAR_APP_CACHE intent action can substantially affect device battery life and might remove a large number of files from the device. App-specific directory on external storage Starting in Android 11, apps cannot create their own app-specific directory on external storage. To access the directory that the system provides for your app, call getExternalFilesDirs(). To make it easier to access media while retaining user privacy, Android 11 adds the following capabilities. concordancia biblica strong en espanol gratis.pdf For consistency across devices and added user convenience, Android 11 adds several methods that make it easier to manage groups of media files. To help your app work more smoothly with third-party media libraries, Android 11 allows you to use APIs other than the MediaStore API to access media files from shared storage using direct file paths. These APIs include the following: The File API.



Native libraries, such as fopen(). Access to data from other apps To protect user privacy, on devices that run Android 11 or higher, the system further restricts your app's access to other apps' private directories. Access to data directories on internal storage Android 9 (API level 28) started to restrict which apps could make the files in their data directories on internal storage world-accessible to other apps. Apps that target Android 9 or higher cannot make the files in their data directories world-accessible. yukio mishima patriotism pdf Android 11 expands upon this restriction. If your app targets Android 11, it cannot access the files in any other app's data directory, even if the other app targets Android 8.1 (API level 27) or lower and has made the files in its data directory world-readable. Access to app-specific directories on external storage On Android 11, apps can no longer access files in any other app's dedicated, app-specific directory within external storage. Document access restrictions To give developers time for testing, the following changes related to the Storage Access Framework (SAF) take effect only if your app targets Android 11 or higher. Access to directories You can no longer use the ACTION_OPEN_DOCUMENT_TREE intent action to request access to the following directories: The root directory of the internal storage volume. The root directory of each SD card volume that the device manufacturer considers to be reliable, regardless of whether the card is emulated or removable. A reliable volume is one that an app can successfully access most of the time. The Download directory. Access to files You can no longer use the ACTION_OPEN_DOCUMENT_TREE or the ACTION_OPEN_DOCUMENT intent action to request that the user select individual files from the following directories: The Android/data/ directory and all subdirectories. The Android/obb/ directory and all subdirectories. Test the change To test this behavior change, do the following: Invoke an intent with the ACTION_OPEN_DOCUMENT action. Check that the Android/data/ and Android/obb/ directories both don't appear. menards return policy no receipt Do one of the following: Invoke an intent with the ACTION_OPEN_DOCUMENT_TREE action. Check that the Download directory appears and the action button associated with the directory is grayed out. Permissions Android 11 introduces the following changes related to storage permissions. Target any version Figure 1. Dialog shown when an app uses scoped storage and requests the READ_EXTERNAL_STORAGE permission. The following changes take effect in Android 11, regardless of your app's target SDK version: The Storage runtime permission is renamed to Files & Media. If your app hasn't opted out of scoped storage and requests the READ_EXTERNAL_STORAGE permission, users see a different dialog compared to Android 10. The dialog indicates that your app is requesting access to photos and media, as shown in Figure 1. 89106552889.pdf Users can see which apps have the READ_EXTERNAL_STORAGE permission in system settings. On the Settings > Privacy > Permission manager > Files and media page, each app that has the permission is listed under Allowed for all files. If your app targets Android 11, keep in mind that this access to "all files" is read-only. To read and write to all files in shared storage using this app, you need to have the all files access permission. Target Android 11 If your app targets Android 11, both the WRITE_EXTERNAL_STORAGE permission and the WRITE_MEDIA_STORAGE privileged permission no longer provide any additional access. Keep in mind that, on devices that run Android 10 (API level 29) or higher, your app can contribute to well-defined media collections such as MediaStore.Downloads without requesting any storage-related permissions. manualidades hechas con papel aluminio Learn more about how to request only the necessary permissions when working with media files in your app. All files access The majority of apps that require shared storage access can follow the best practices for sharing media files and sharing non-media files. However, some apps have a core use case that requires broad access of files on a device, but cannot do so efficiently using the privacy-friendly storage best practices. Android provides a special app access called All files access for these situations. To learn more, see the guide on how to manage all files on a storage device. Note: If you publish your app to Google Play, carefully read the notice. If you target Android 11 and declare All files access, it can affect your ability to publish and update your app on Google Play. Additional resources For more information about changes to storage in Android 11, view the following materials: Blog posts Videos How to perform storage access in Android 11