Check camera permission android programmatically

I'm not robot

reCAPTCHA

Continue

activity_main.xml?xml version?1.0 encoding'utf-8?'lt;lt;LinearLayout xmlns:android' xmlns:app' xmlns /root_layout android:layout_width'match_parent android:layout_height'match_parent android:padding'16dp android:orientation'vertical android:background'#82a474'gt; zlt'button android:id'id/btn_do_task android:layout_width'wrap_content android:layout_height'wrap_content'android:text'check'gt; (LinearLayout; MainActivity.java com.cfsuman.me.androidcodesnippets; imported android. Manifesto; Import android.app.Activity; import android.content.Context; import android.content.pm.PackageManager; Import android.os.Build; import android.support.v4.content.ContextCompat; import android.support.v7.app.AppCompatActivity; Import android.os.Bundle; Import android.view.View; import android.widget.Button; import android.widget.LinearLayout; import android.widget.Toast; MainActivity public class expands AppCompatActivity - private context mContext; mActivity's private activities; Private LinearLayout mRootLayout; Private button mBtnDoTask; @Override protected void onCreate (preserved in the InstanceState) - super.onCreate (savedInstanceState); setContentView (R.layout.activity_main); Get the context of the mContext app - getApplicationContext (); mActivity - MainActivity.this; Get a link to the widget from the xml layout mRootLayout - findViewById (R.id.root_layout); mBtnDoTask - findViewById (R.id.btn_do_task); Install the listener button for the mBtnDoTask.setOnlickCListener.ru button - @Override public space on Click (View) - if (Build.VERSION.SDK_INT qgt;) Build.VERSION_CODES. M) int checkSelfPermission (Context, Line Resolution) Determine whether you have been granted a certain permit. Settings Context: Context Resolution Line: The name of the verified resolution. Returns int : PERMISSION_GRANTED if you have a permit, or PERMISSION_DENIED if not. You can find this class through getPackageManager. The result of the PERMISSION_DENIED int int resolution check: this is returned to checkPermission (String, String) if permission has not been granted to this package. Permanent value : -1 (0xffffffffff) - - Int PERMISSION_GRANTED Resolution Check Result: this is returned to checkPermission (String, String) if permission has been granted to this package. Permanent value: 0 (0x0000000000) if (ContextCompat.checkSelfPermission (mActivity,Manifest.permission.CAMERA) - PackageManager.PERMISSION_GRANTED) / Camera resolution provided by Toast.makeText,Resolutionhe-e-e- / Camera permission not granted by Toast.makeText (mContext, permission not granted.,Toast.LENGTH_SHORT).); } }); AndroidManifest.xml (Resolution) Every zlt;uses-permission android:name'android.permission.camera'lt;uses-permission-gt; Android app works in a sandbox with limited access. If the app needs to use resources or information outside its own sandbox, the app must request permission. You state that your app needs permission by listing the resolution in the app's manifest and then requesting that the user approve each resolution during the run (on Android 6.0 and above). The basic principles are: ask permission in context when the user begins to interact with the function that requires it. Don't block the user. Always allow you to cancel the user interface's educational stream associated with permissions. If a user denies or cancels the permission that the feature needs, gracefully demeans your app so that the user can continue to use your app, perhaps by disabling a feature that requires permission. Don't come up with any systemic behavior. This page goes through the step-by-step process of adding permissions to the app and requesting those permissions at the right time. Add permissions to the manifest on all versions of Android to declare that your app needs permission, put the item in the app's manifesto as a child of the upper-level element. Warning: Think carefully about what permissions you declare in the application's manifesto. Only add the permissions your app needs. For every permission your app asks, make sure that it gives the user clear benefits and that the request is executed in a way that is obvious to them. For example, an app that should access the Internet will have this line in the manifest: zlt;manifest xmlns:android/ package/com.example.snazzyapp use-permission android:name'android.permission.!-- INTERNET The behavior of the system after the resolution is announced depends on how confidential the permit is. Some permissions are considered normal, so the system immediately provides them when installed. Other permissions are considered dangerous so the user must explicitly provide access to the app. For more information on different types of permits, see Check for permissions. that requires permission. On Android 6.0 (API level 23) and above, users can revoke dangerous permissions from any app at any time. Note: Don't check or ask for permission when the user opens your app. Instead, wait until the user selects or opens a feature that requires a specific resolution. Identify, use-permission. The app has already received permission to check if the user has already given your app a certain permission to pass this permission to contextCompat.checkSelfPermission (the method). This method returns either PERMISSION_GRANTED PERMISSION_DENIED, depending on whether your app has permission. Explain why your app needs permission If ContextCompat.checkSelfPermission () PERMISSION_DENIED, call shouldShowRequestPermissionRationale(). If this method returns correctly, show an educational user interface for the user. In this user interface, describe why the feature the user wants to include needs some resolution. Request Permissions After the user views the educational user interface, or the return value shouldShowRequestPermissionRationale () indicates that you do not need to show an educational user interface this time, request permission. Users see a system permission dialogue where they can choose whether to grant a specific permission to your app. Traditionally, you control the request code yourself as part of a resolution request and include that request code in the resolution callback logic. Another option is to use the RequestPermission contract, which is included in the AndroidX library, where you allow the system to control the permission request code for you. Because using The RequestPermission simplifies your logic, it's a good idea to use it as much as possible. Allow the system to control the permission request code to allow the system to control the request code associated with the permission request, add dependence on the androidx.activity library to your module's build.gradle file. Use version 1.2.0 or later library. Note: Version 1.2.0 of the library is in the alpha development stage. You can then use one of the following classes: the following steps show how to use the RequestPermission contract. The process is almost the same for the RequestMultiplePermissions contract. In the logic of initializing an action or piece, go to ActivityResultCallback in a forActivityResult registration call. ActivityResultCallback determines how your app handles a user's response to a permission request. Keep a link to the Refund Value of the ForActivityResult register, which is the ActivityResultLauncher type. To display the system's dialog permissions, if necessary, call the start-up method on the ActivityResultLauncher instance that you saved in the previous phase. After launch, the system permissions dialogue is triggered. When a user makes a choice, the asynchronous system triggers your ActivityResultCallback implementation, which you identified in the previous phase. Note: Your app is not set up the conversation that appears when you start the call. To provide the user with additional information or context, change the app's user interface to make it easier for users to understand why they need a feature in the app app Resolution. For example, you can change the text in the button that allows the feature. In addition, the text in the system permission dialogue refers to a resolution group related to the requested permit. This permission grouping is designed to make the system easy to use, and your application should not rely on permissions that are in a specific permission group or beyond. The following piece of code shows how to handle the resolution response: / Register a permission callback that handles the user's response to the system permissions dialogue / Save the return value, instance / ActivityResultLauncher. You can use either the shaft, as shown in this snippet, / or lateinit var in your onAttach () or onCreate () method. val requestPermissionLauncher - RegistrationForActivityResult (RequestPermission)- does not have permission: Boolean - If (isGranted) // Permission is granted. Private activitiesResultLauuncher'lt;String'gt; requestPermissionLauncher - registrationForActivityResult (new RequestPermission (), isGranted - if (isGranted) -/ Permission is granted. process in your / app. otherwise / Explain to the user that the feature is not available because / features require permission, which the user refused. And this piece of code demonstrates the recommended resolution verification process, and by requesting permission from the user when necessary: if necessary: using an API that requires permission, Manifest.permission.REQUESTED_PERMISSION (PackageManager.PERMISSION_GRANTED - zgt; - you can use an API that requires permission to qgt.) In this user interface, / turn on the undo button or no gratitude that allows the user // continue to use your app without granting permission. showInContextUI (...) still - // You can directly request permission. // Registered ActivityResultCallback receives the result of this request. requestPermissionLauncher.launch (Manifest.permission.REQUESTED_PERMISSION) - if (Context- PackageManager.PERMISSION_GRANTED) / You can use an API that requires permission. performAction (...); - even if (shouldShowRequestPermissionRationale (...)) / In the educational user interface, explain to the user why your application requires this / permission for a particular function to behave as expected. In this user interface, / turn on the undo or no thank you button, which allows the user // to continue using your app without giving permission. showInContextUI(...); - more than that/ You can directly request permission. An ActivityResultCallback registered receives the result of this request. requestPermissionLauncher.launch (Manifest.permission.REQUESTED_PERMISSION); Manage the permission request code yourself As an alternative to allowing the system to control the permission request code, you can control the permission request code yourself. To do this, include the request code in the call for the Mission Request. Note: Your app can't set up the conversation that appears when you call the Formissions request. The text in the system permissions dialogue refers to a resolution group, but this permission grouping is designed to make the system easy to use. Your app should not rely on permissions that they are in or outside a particular group of permissions. The following piece of code demonstrates how to request permission: when Manifest.permission.REQUESTED_PERMISSION) - PackageManager.PERMISSION_GRANTED - / You can use an API that requires permission. performAction (...) - shouldShowRequestPermissionRationale (...) - / In an educational user interface, explain to the user why your application requires this/ permission for a specific function to behave as expected. In this user interface, / turn on the undo or no thank you button, which allows the user // to continue using your app without giving permission. showInContextUI (...) still - you can directly ask for permission. requestPermissions (CONTEXT, arrayOf (Manifest.permission.REQUESTED_PERMISSION), REQUEST_CODE) - if (ContextCompat.checkSelfPermission (CONTEXT, Manifest.permission.REQUESTED_PERMISSION) - PackageManager.PERMISSION_GRANTED) / You can use an API that requires permission. performAction (...); In the educational user interface, explain to the user why your app requires this / permission for a particular function to behave as expected. In this user interface, / turn on the undo or no thank you button, which allows the user // to continue using your app without giving permission. showInContextUI(...); - more - / You directly request permission. requestPermissions (CONTEXT, new line) - Manifest.permission.REQUESTED_PERMISSION, REQUEST_CODE); Once the user responds to the system permissions dialogue, the system calls for the application to be implemented onRequestPermissionsResult. System System in the user's response to the resolution dialogue, as well as the query code that you've identified, as shown in the following piece of code: override the fun on RequestPermissionsResult (requestCode: Int, permissions: Array'lt;String'gt; grantResults: IntArray) - when (requestCode) - PERMISSION_REQUEST_CODE -PackageManager.PERMISSION_GRANTED qgt; Continue the action or workflow / in the app. otherwise / Explain to the user that the feature is not available because the functions require permission, which the user refused. At the same time, respect the user's decision. Don't refer to the system settings in an attempt to convince the user to change their solution. Return - / Add other lines when to check other ! permissions that this app can request. still - - / Ignore all other requests. @Override public void onRequestPermissionsResults (int requestCode, Resolution String, int' grantResults) - Switch (requestCode) zgt; - case PERMISSION_REQUEST_CODE: / PackageManager.PERMISSION_GRANTED If the request is cancelled, the arrays of results are empty. Continue the action or workflow / in the app. otherwise / Explain to the user that the feature is not available because the functions require permission, which the user refused. At the same time, respect the user's decision. Don't refer to the system settings in an attempt to convince the user to change their solution. Return; Other case lines to check other ! permissions that this app can request. Denial of processing permission If a user refuses to request permission, your app should help users understand the consequences of a denial of permission. In particular, the app should communicate features that do not work due to lack of permission to users. When you do this, keep in mind the following best practices: A user's attention guide. Highlight a specific part of the app's user interface where there is limited functionality because your app doesn't have the permission you need. A few examples of what you could do include: Show a message where the results or these features would appear. Display another button that contains an error icon and color. Be specific. Don't show the overall message. instead, mention which features aren't available because your app doesn't have the permission you need. Don't block the user interface. In other words, don't display a full-screen warning that prevents users from continuing to use your app at all. At the same time, your app must respect the user's decision not to allow it. Starting with Android 11 (API level 30) if a user clicks Deny for a specific more than once during your app, the zlt;string installing on the device, the user does not see the dialogue of the system permissions if your application asks for this permission again. The user's actions imply not to ask again. In previous versions, users will see the system permissions dialogue every time your app asks for permission, if the user hasn't previously chosen the checkbox or don't ask again. In some situations, permission can be denied automatically, without any user action. (Similarly, permission can be granted automatically as well.) It is important not to assume anything about automatic behavior. Every time an app needs access to a resolution-required functionality, you should check that your app is still granted that permission. To provide the best user experience when requesting app permissions, also see The Request to Become the Default Handler, if necessary, some applications depend on access to sensitive user information related to call logs and SMS messages. If you want to request permissions specific to calling magazines and SMS messages and posting an app in the Play Store, you should encourage the user to customize your app as the default handler for the system's core function before requesting those permissions while running. For more information about default handlers, including recommendations for showing the default handler's hint to users, see a guide to permissions used only in default handlers. Announcing permissions at the API level To declare permission only on devices that support launch permissions, i.e. devices running Android 6.0 (API level 23) or above, include the use-permission-sdk-23 tag instead of the use permission tag. With any of these tags, you can install the maxSVersiondk attribute to indicate that you don't need a specific resolution on devices that run with a higher version. More resources For more information about permissions, read these articles: Permits review of App Solutions Best Practices to learn more about requesting permissions, download the following sample apps: Android RuntimePermissionsBasic Java Sample Kotlin Kotlin