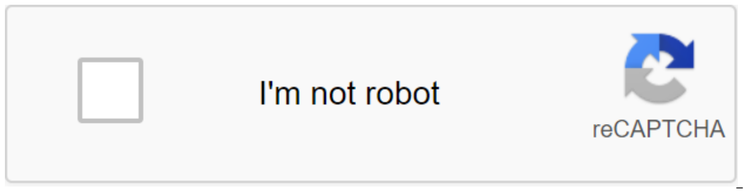
Moto mods manager apk



Continue

This document is designed to be viewed using the frame function. If you see this message, you're using a web client that's inecest of a frame. A reference to the non-framed version. FOLLOW USA ModManagerService is a major component of the Moto Mod platform. The ModManager class allows third-party apps to detect and communicate with Moto Mods. Apps must use ModManager to detect Moto Mod ATTACH or DETACH events. Each Moto Mod attached to your smartphone has the appropriate ModDevice. When Moto Mod is attached, the app can get a copy attached to ModDevice through ModManagerService. Once the app has received ModDevice, it can receive information about Moto Mod, including supplier ID, product ID, UID, and ModProtocol, which it supports. Preconditions of development You must first install the Moto Mod Android SDK library, customize your Android project to use the SDK library and make the necessary changes to your app's Android manifest. See how to set up the development environment. Equipment Preconditions In order to be able to test the features of the Moto Mod SDK, you will also need a Moto Mod HDK or a protocol-compatible Moto Mod. Protocol-specific functions that will be performed in the ModDevice class will only be available when Moto Mod is attached to the protocol. Determining whether moto Mods supports your app If your app has to interact with shared Android devices in addition to Moto, Moto Mod SDK allows your app to check while running whether ModServices is available on the target device. This time check can be done with ModManager.isModServicesAvailable. This feature will check whether ModServices is available on your device (for example, whether your device supports the Moto Mod interface), and whether the Moto Mod version required by your app is compatible, as you have in the manifest, with the Moto Mod version currently supported by your device. Before you use isModServicesAvailable, your app must be associated with ModManagerService. If (ModManager.isModServicesAvailable (MainActivity.this) - ModManager.SUCCESS) !) This device supports Moto Mods/ in addition ... In addition, this device does not support Moto Mods - isModServicesAvailable () API performs two checks: whether ModManagerService works on your device. This is used to check the running time whether the device running the app supports the Moto Mod interface or not. Is this device supported by the minimum version of Moto Mod Platform required by your app? This is a compatibility check based on the minimum version of the Moto Mod platform listed in your app's manifest. Moto Mod Definition Each Moto Mod has a set of hardware and the hardware manifesto. They're programmed in Moto Mod Mod Production. The ID contains a moto Mod Product ID and a vendor ID, while the manifesto declares the functionality supported by the hardware. Moto Mod will provide Moto with identifying attributes including supplier ID, product ID, product name, and vector name. The app can get these values with ModManager and ModDevice Java classes. The identified Id 32 Supplier Details format has a value of 32 bits of value, unique to the vendor, and assigned to Motorola. For example, '296' is Motorola. Product ID 32 bit value A 32 bit value, corresponding to the product ID, and assigned by the supplier. The vendor name is the vendor name that corresponds to the vendor ID as a line ('Motorola'). Product name Line Product Name is like a string. Moto Mod Unique ID String Unique UID device, use instead of serial number for most mods. UID is read from the Moto Mod processor. For current design, the example of UID 00000000000000-2030-3538-5436-500c0059004d For certified Moto Mods, the supplier ID is a unique 32 bits of INT. Moto Mod id provider is appointed by Motorola and is guaranteed to be unique to all suppliers. The product ID is assigned by the Moto Mod manufacturer and should be great for each individual product. (0xFFFFFFFFFFFF and 0x00000000 are reserved product ID values). The product name and supplier's name are lines programmed in the Moto Mod hardware manifest from the Moto Mod supplier. They may not match the name under which the product is on the market. When developing Moto Mod with Moto Mod HDK, we recommend developers who do not have Motorola assigned Vendor ID to self-assign unique developer VID and PID, and names. Certification information about each Moto Mod attachment, Moto will also check whether the attached device is certified or not. Devices identified as 'Moto Mods' will be immediately available for use when attached without additional user consent. Devices that cannot be tested as certified when attached will require additional user consent after initial affiliation before they are available for use in Moto. The platform automatically encourages the user to give consent. Please refer to the PARTNER section to review certification requirements and understand how to work with Motorola to certify your Moto Mod support protocols, which Moto Mod can announce support for one or more greybus protocols through its hardware manifest. Each Moto Mod protocol declares support the hardware manifest must have the appropriate equipment and firmware on the Moto Mod. The ModProtocol class has a protocol record for each supported greybus protocol, with the exception of the basic control protocol, which is common. Supported protocols correspond to the logical interfaces defined in the first version of the version Mod platform. The protocol supported in the initial release of the Moto Mod platform as their respective ModProtocol values are: Logical Interface Appropriate ModProtocol identified permanent Used for raw RAW Direct Communication Channel between device and Moto Mod Transmission Energy Battery Transmission Protocol and Power-Supplies Battery BATTERY display-ext MODS_DISPLAY External audio-ext displays MODS_AUDIO Audio devices hid announced by ModDevice when it is attached. Moto Mod SDK Elementary Classes Basic Classes provided by Moto Mod SDK are: Class Details ModManager Central Infrastructure Component Moto Mod. It notifies Moto Mod event apps and gives access to ModDevice. ModManager also allows apps to register ModListener, which will be called on government changes in Moto Mod connection status. ModDevice A ModDevice presents individual Moto Mod, product information and stated interfaces and protocol. ModDevice is listed through ModManager and corresponds to the moto fashion attached. Once the app has received the Application ModDevice, it can determine what protocols the device provides. ModProtocol ModProtocol Class defines protocols or interfaces advertised by Moto Mod. ModConnection The ModConnection contains detailed information about the current connection to the attached ModDevice. It should be used when the app needs to request a state of connection or receive connection error codes. The working mod manager's intentions to attach and separate the life cycle of Moto Mods can be attached and separated from

the device at any time. Android apps can receive notifications when Moto Mod is attached to your device. When Moto Mod is attached to the device, ModManagerService will handle all the following tasks: ModManagerService will detect the attachment of the accessory. ModManagerService will perform a certification check as described above. Your device must be unlocked (the last secure lockscreen) when the Moto mod is attached for the first time. If Moto Mod is certified, you don't need to enter. If the device is not certified, ModManagerService will ask the user for consent before joining and listing the device. None of its functionality will be available until consent is obtained and ACTION_MOD_ATTACH broadcast until consent is obtained. ModManagerService will broadcast ACTION_MOD_ATTACH notify the app that the Moto Mod has been attached. This intention announces that the Moto Mod is attached, but does not guarantee that the functions of the device are fully initiated and in operation When all the interfaces advertised by Moto Mod have been successfully brought up, ModManagerService will broadcast ACTION_MOD_ENUMERATION_DONE intentions. The app must listen to the data ACTION_MOD_ENUMERATION_DONE if it needs confirmation that all Moto Mod-supported interfaces are available. Any error will result in ACTION_MOD_ERROR intentions to be broadcast. The app must track ERROR's intention to detect any failure in the Moto Mod attachment. When the user disconnects Moto Mod from Moto, ModManagerService will demolish all active interfaces and broadcast ACTION_MOD_DETACH to indicate that the accessory has been removed. Additional information available in ACTION_MOD_ATTACH intentions, ModManager conveys com.motorola.mod.ModManager.ACTION_MOD_ATTACH intent when the Moto Mod was attached to the device. The purpose of the ACTION_MOD_ATTACH only indicates the Moto Mod was attached, it does not guarantee its functionality is available for use. The app that is registered to receive ACTION_MOD_ATTACH can get more information about the exact Moto Mod attached from 6 additional intentions. To sell these intentions additionally safely, your app must include the Moto Mod SDK library. Additional EXTRA_VENDOR_ID the identifier of the Fashion Supplier, the 32bit value assigned by Motorola to the mod supplier and unique to the manufacturer. EXTRA_PRODUCT_ID product ID Fashion, 32 bits value unique on the product. EXTRA_VENDOR the vendor's name as a line ('StanCo'). EXTRA_PRODUCT product name as a line ('Pitchfork'). EXTRA_UNIQUE_ID UID devices, use instead of a serial number for most mods. UID is read from the Moto Mod processor. This is usually in the format of 00000000002039345452325 00C00250041. EXTRA_MOD_DEVICE ModDevice, which corresponds to this mod, which can be used to request classes supported by Moto Mod. The code example below shows how to disassemble the vendor ID, product ID, unique identifier, vendor name, and product name Moto Mod. String Actions - intent.getAction if (action ! - null) if (action.equals (ModManager.ACTION_MOD_ATTACH)))) - int vid - intent.getIntExtra (ModManager.EXTRA_VENDOR_ID, INVALID_ID); int pid - intent.getIntExtra (ModManager.EXTRA_PRODUCT_ID, INVALID_ID); ParcelUuid p (ParcelUuid)intent.getParcelableExtra (ModManager.EXTRA_UNIQUE_ID); UUID wedi et rh th zero? INVALID_UID: p.getUuid(); Line Supplier - intent.getStringExtra (ModManager.EXTRA_VENDOR); String product - intent.getStringExtra (ModManager.EXTRA_PRODUCT); The app can also announce a filter of intent for filtering for com.motorola.mod.ModManager.ACTION_MOD_ATTACHintent. The following example shows that the action is zlt;activity. </intent-filter> </activity> как объявить фильтр намерения: Наличие Наличие MOD After Moto Mod is attached to the device and all its announced interfaces are available for use, ModManager will broadcast com.motorola.mod.ModManager.ACTION_MOD_ENUMERATION_DONE intent. This intention will only be sent after ACTION_MOD_ATTACH, and when all advertised Moto Mod features are available for use. If (ModManager.ACTION_MOD_ENUMERATION_DONE.equals (action) - String Supplier and intent.getStringExtra (ModManager.EXTRA_VENDOR); String product - intent.getStringExtra (ModManager.EXTRA_PRODUCT); Line s - Finished listing - product by supplier; Detection of application bug failures If the user tries to attach Moto Mod, but the attachment fails, the system will broadcast com.motorola.mod.ModManager.MOD_ATTACH_FAILED intent. The error code can be extracted from the EXTRA_RESULT_CODE. The possible value of these error codes is determined in the OsConstants class from Android. The app can also check the status of ModConnection to learn more about moto Mod's current state of readiness and availability. As an example, this intention will be abandoned when the Moto Mod has manifesto vices that cannot be disassembled. After receiving the MOD_ATTACH_FAILED result - intent.getIntExtra (ModManager.EXTRA_RESULT_CODE, -1); Failure to bring up some interfaces After the broadcast ACTION_MOD_ATTACH, ModManagerService will only be broadcast ACTION_MOD_ENUMERATION_DONE when all stated Moto Mod protocols are available for use. If the app gets in ACTION_MOD_ATTACH but no further ACTION_MOD_ENUMERATION intention is broadcast, then one or more interfaces announced by Moto Mod have not been brought up successfully. The app must then examine the state of ModConnection for the protocols announced by this ModDevice. The Moto Mod Modager disconnection conveys com.motorola.mod.ModManager.ACTION_MOD_DETACH intent when the Moto Mod is separated from the device. The app can get more information about a separate fashion from additional services DETACHintent. Working with mod Manager The ModManager allows the app to get a list of Moto Mods ModDevice attached, access this ModDevice, register a listener to track changes in Moto Mod status, and determine if your app's device is working on a device that supports Moto Mods. The ModManager app must have PERMISSION_MOD_ACCESS_INFO permission to access ModManager. Applications must call bindService () with the intention of ACTION_BIND_MANAGER to link to ModManagerservice, as described in the code example below: Intent intention and new intentions (ModManager.ACTION_BIND_MANAGER); intent.setComponent (ModManager.MOD_SERVICE_NAME); Context.bindService @Override public void onServiceConnected (ComponentName className, IBinder binder) - IModManager mMgrSrvc - IModManager.Stub.asInterface (Binder); mManager new ModManager (Context, mMgrSrvc); ..... , Context.BIND_AUTO_CREATE); Determine whether the Moto Mods app supports the application can perform a time check with ModManager.isModServicesAvailable to determine whether the moto Mods device supports it. This feature will check whether ModServices is available on your device (for example, whether your device supports the Moto Mod interface), and whether the Moto Mod version required by your app (as in your manifest) is compatible with the Moto Mod version currently supported by your device. GetModModPlatformSDKVersion () and getModSdkVersion () API allows the app to receive a version of the Moto Mod platform that is supported by Moto. An app that requires certain APIs or hardware features only available in the new release of the Moto Mod and SDK platform should check PlatformSDK and ModSDKversion before trying to use these APIs or features. The ModPlatform SDK version represents the smartphone-supported version of SDK. The SDK version will only change with the new release of Moto software. The Platform SDK version matches the final set of features, APIs, and device-supported device protocols. Its purpose is comparable to the use and function of the Android platform API levels. Checking the ModPlatform SDK version will allow the app to determine whether the Moto s platform supports the required ModProtocol (grey classes) ModSDK version of the Moto Mod Service SDK. As ModManager is updated through the Play Store, new Moto Mod API programs and functionality will be presented through Play Store updates. The Mod Platform SDK version may be higher than the SDK version. Using the listener to track the changing state of Moto Mod If your app must be notified of a status change in ModManager, for example, Moto Mod is attached or separated from the device, it can register ModListener through com.motorola.mod.ModManager.registerModListener () API. This listener will receive a notification about moto Mod's attachment and deletion, as well as connection status changes and asynchronous messages from Moto Mod. mManager.registerModListener (MainActivity.this, new ModProtocol.PROTOCOL_ALL int); ModManager will notify the ModListener instance of state, connection, and features changes on the Moto Mod attached. When Moto Mod is attached or disconnected, ModManager will send a notification to each listener through the onDeviceHotplug method () When all Moto Mod interfaces on Mod are ready, ModManager will be notify each listener using onEnumerationDone. When the link between Moto Mod and Moto q changes, ModManager will notify each listener through onConnectionStatusChange () When Moto Mod reports an event changing asynchronous capabilities, ModManager will send a notification to each listener using onCapabilityChanged.) The app can learn more about the type of feature change by calling getCapabilityVendor, getCapabilityReason() and getCapabilityLevel ()methods. The Moto Mod developer can identify their own messages to transmit to the app through the feature-changing interface. Receiving the Moto Mod package by default Moto Mod can indicate in your manifest the default name of the package (apps). When Moto Mod is first initiated, it uses that name to encourage the user to download the corresponding package. The app can use ModManager.getDefaultModPackage (ModDevice mod) to get the name of the package from Fashion for informational purposes. Working with ModDevice A ModDevice is an actual Moto Mod attached to Moto. Once the device is attached and available, the app can use the ModDevice class to obtain information about that particular Moto Mod and access its functionality. Getting the ModDevice attached with getModList () getModList method allows the app to list all modDevices that are currently attached to you smartphone, and determine whether they are fully functional or not. Once linked to ModManager, the app should trigger ModManager.getModList. Mods - mManager.getModList (admittedly); Return Moto Mod, which is currently attached to the Note device: In the current release of the Moto Mod platform, only one ModDevice can be attached to your Moto s. ModManager.getModList will return one list of items. If the app only needs to determine whether Moto Mod supports a specific protocol (such as BATTERY), getModList also allows you to filter returned Moto Mods by interface type. The app can also use getModInterfaceDelegationsByProtocol () to get the interface to the attached ModDevice. The filter_array used should be an array of protocols. Protocols List of Protocol.BATTERY, Protocol.AUDIO filter_array list; The mods are mManager.getModList (filter_array, false); This will only bring back Moto Mods, which include battery and audio interfaces to obtain information about the attached Moto Mod Once the app has received ModDevice, it can use features exposed by the ModDevice class to obtain information about Moto Mod. ModDevice.getVendorId will return the Moto Mod'lt/ModDevice vendor ID; </ModDevice> </ModDevice> return the Supplier's name. ModDevice.getProductId () and ModDevice.getProductName will return the product name. / - getModList (admittedly) will return the list currently attached by ModDevices (/ List of ModDevice'gt; attachedMods and mManager.getModList (admittedly); If (attachedMods!' null and attachedMods.size)- ModDevice modDevice - attachedMods.get (0); mVendor - modDevice.getVendorId (); mProduct - modDevice.getProductId(); Moto Mod can specify in its hardware manifest the preferred name of the Android app package. When moto Mod is initialized for the first time, it will direct the user to download the app from the device's app store. The app can also be named after this related app's package through a getDefaultModPackage call. This is an optional item in the hardware manifest. After receiving a list of protocols supported by this device After receiving ModDevice currently attached from ModManager, the app can use the features put forward by ModDevice to obtain information about Moto Mod. ModDevice.get AnnProtocols () to receive a list of greybus protocols supported by Moto Mod, which is currently attached to a physical device. The app can also use has AnnouncProtocol () function to determine whether ModDevice raw has announced an interface. if (device.has AnoProtocol (ModProtocol.Protocol.RAW) - / Open raw interfaces. If (device.hasProtocol.PROTOCOL.BATTERY)) / We know that this mod has a battery and access to a specific API protocol After receiving a check to see if ModDevice supports a specific protocol, your application can use mManager.getClassManager () API to get a specific class object to access a specific API protocol. If (device.hasProprotocol (ModProtocol.Protocol.MODS_DISPLAY) - display ModDisplay (ModDisplay)mManager.getClassManager (ModProtocol.Protocol.MODS_DISPLAY); Your app can work with the ModDisplay API - The following Java classes are defined by SDK and provide interfaces to the relevant Moto Mod: ModBattery ModDisplay protocols