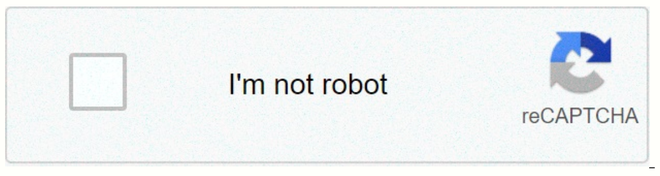

Continue

Android jailbreak app store

Apple has the App Store, the jailbreaker has a Cydia Store in addition to the default App Store. And if you want to be technical, Cydia is the first App Store. It is available with iOS 1.x and allows jailbreakers to download apps before the App Store even existed. Those jailbroke their first generation iPhone in 1.x have seen some ideas and concepts of jailbreak become part of iOS. We'll mention jailbreak's other App Store, Rock, but given Cydia acquired Rock, it might be pointless now. Another term you might hear when it comes to Cydia is repos, or repository. These are just the sources you add in the manage section. ModMyi and BigBoss are the two biggest. They've been added when you install Cydia. You are always welcome to add custom repos if you choose. I highly recommend simply adding repos from trusted sources. If you have any strange feelings about adding it, I'll probably pass it. For more information, see our Jailbreak Starter Guide, and if you have any questions, get help in our Jailbreak App Forum. Android users will get a new place to shop. Verizon Wireless will officially open its doors to its Android V Cast Apps store next week, the carrier has confirmed. The V Cast Apps store will initially launch on the HTC Droid Incredible; The software update will provide access along with some common enhancements on mobile phones. Verizon's other Android devices will soon follow. So what does Verizon's Android app store really all about, and what does it mean for you? I talked to someone from Verizon to get the full scoop. First of all, Verizon's V Cast Apps store won't replace Google's main Android market. The V Cast Apps store will only be the second option built into the device; Google's Android market and Verizon's new store will be equally accessible. We look at it as a complementary store to the other stores that are out there, explained Verizon Wireless spokeswoman Debi Lewis. Verizon actually launched a similar version of its store on BlackBerry phones earlier this year; the configuration on that platform is almost the same. Verizon's Android app store will have several hundred apps to start, I'm told, with heavy growth expected over the coming months. Apps you'll find on the Verizon store may also be available on Google's Android Market; nothing prevents developers from offering their programs in either place. It's really up to every individual developer about what he's doing. Developers will receive the same 70/30 revenue split from Verizon's V Cast Apps store as they do from the Google Android Market. Verizon did suggest, however, that its store would offer opportunities for promotion and readability that Google Market might not offer. main where Verizon's V Cast Apps store will differ from the Google Android Market is in the process of purchasing: Although the Android Market currently requires you to use Google Checkout for app purchases, the Verizon app store will allow you to have directly to your Verizon statement. With that in mind, Google seems poised to add a new payment option to the Android Market, so comparisons can change soon. Verizon's V Cast App Store will lack one important purchase feature that Google Market provides: In the main Android Market, you can always return your app within 24 hours and receive a full refund of the cost. Until now, Verizon's V Cast Apps store would not offer this option. Want to know if your Android phone is in line to get access to Verizon's new store? Verizon hasn't compiled an unequivocal list of what devices will receive it or when, but it sounds like it will eventually become a pretty standard feature for carrier handsets. Until now, Lewis told me Droids, Droid X, and Fascinate were all expected to receive software updates that allow stores in the future. Galaxy Tab, meanwhile, will come up with a pre-installed V Cast Apps store when it goes on sale later this month. All right, those are the facts - so what's the real deal here? Is this a good thing or a bad thing? Ultimately, it all depends on your perspective. Some people, such as the CEO behind certain competing smartphone platforms, have characterized some app stores on Android as a huge disadvantage. In addition to Google's own app market, Amazon, Verizon and Vodafone have all announced that they are creating their own app stores for Android - so there will be at least four app stores on Android, which customers should look for to find the app they want, Apple's Steve Jobs noted during his company's earnings call in October. It will be a mess for users and developers, he continued. Is it really going to fall apart, though? As I wrote in a somewhat impassioned letter to Mr Jobs last month, Most markets - virtual or otherwise - do allow people to buy products from multiple providers. Choice doesn't cause chaos. Put in another context, I can buy accessories for my car from a dealer, or I can choose to go to a number of third-party retailers instead. Having options is not always detrimental. One last point worth mentioning: Even now, Google Android Market is not an exclusive source for Android apps. You can already download and install apps from independent online stores or developers. Android is never locked into a single market for apps; what's new with Verizon's efforts is really just that it will be built into devices with carrier support. As for what kind of value it will or will not provide, we will soon find out. JR Raphael is a PCWorld contributing editor and author of the Android Power blog. You can find it on Facebook and Twitter. When you buy something after clicking on a link in our article, we may earn a small commission. Read our affiliate link policy for more details. Almost every non-trivial application has to store data one way or another. This data can be from various forms, such as settings, application settings, user data, images, or caches of data retrieved from the Internet. Some applications may generate data that ultimately belongs to the user, so, prefer to store the data (perhaps documents or media) in a public place that the user can access at any time, using other applications. Other apps may want to store data, but don't want it to be read by other apps (or even users). The Android platform gives developers several ways to store data, with each method having its advantages and disadvantages. For this article, we'll cover the various data storage techniques available to Android developers, along with sample code to help you get started, or to refresh your memory. How to store dataThere are basically four different ways to store data in the Android:1 app. Shared PreferencesYou should use this to store primitive data in key value pairs. You have a key, which must be a string, and an appropriate value for that key, which can be any of: boolean, float, int, long or string. Internally, the Android platform stores the app's Shared Preferences in an xml file in a personal directory. Apps can have multiple Shared Preferences files. Ideally, you want to use Shared preferences to save your app preferences.2. Internal Storage There are many situations where you may want to store data but Shared Preferences are too restrictive. You might want to hang on to a Java object, or an image. Or your data logically needs to be systemed defensively using a familiar hierarchy of file systems. The Internal Storage data storage method is specific to situations where you need to save data to the device's file system, but you do not want other applications (even users) to read this data. Data stored using the Internal Storage method is entirely private for your app, and is deleted from the device when your app is deleted.3. External StorageConversions, there are other instances where you may want users to see files and data stored by your application, if they wish. To save (and/or read) files to your device's external storage, your application must ask WRITE_EXTERNAL_STORAGE settings. If you only want to read from External Storage without writing, ask READ_EXTERNAL_STORAGE permission. Permission WRITE_EXTERNAL_STORAGE provide read/write access. However, starting with Android 4.4, you can actually write to a personal external storage folder without WRITE_EXTERNAL_STORAGE. Personal folders can be read by other applications and by users, however, the data stored in this folder is not scanned by a media scanner. This app_private folder is located in the Android/data directory, and is also deleted when your app is deleted. Starting with Android 7.0, apps can request access to certain directories, rather than requesting access to external storage. This way, your app can, for example, request access to the image directory only, or Directory. This is referred to as scope directory access. For more information about requesting access to covered directories, see this Android developer tutorial.4. SQLite Databases Technically, Android provides support for applications to use SQLite databases for data storage. Databases are created specific to the application, and are available for any class within the application, but not for outside applications. Needless to say, that before you decide to use a SQLite database for data storage in your applications, you must have sql knowledge. We'll discuss each one in turn. We use data binding techniques for our sample code, and if you're unfamiliar with this, or need a refresher, check out our previous article on using data binding on android. Use Shared PreferencesTo save data using shared preferences, you must first get the SharedPreferences object. There are two Context methods that can be used to retrieve SharedPreferences objects. SharedPreferences sharedPreferences = getPreferences(MODE_PRIVATE); for when your application will have a single preference file, andSharedPreferences sharedPreferences = getSharedPreferences(fileNameString, MODE_PRIVATE); for when your app can have multiple preference files, or if you prefer to name your SharedPreferences instance. When you get the SharedPreferences object, you then access Editor using the edit() method. To actually add value, use the putXXX() Editor method, where XXX is either Boolean, String, Float, Long, Int or StringSet. You can also remove key value preference pairs by remove(). Finally, be sure to call the Commit() Editor method after placing or deleting values. If you don't call a commit, your changes won't last. SharedPreferences.Editor editor = sharedPreferences.edit(); editor.putString(keyString, valueString); editor.commit(); For examples of our applications, we allow users to specify the file name SharedPreferences. If a user specifies a name, we ask SharedPreferences that has that name, otherwise we ask for the default SharedPreference object. String fileNameString = sharedPreferencesBinding.fileNameEditView.getText().toString(); SharedPreferences sharedPreferences; if(fileNameString.isEmpty()) { sharedPreferences = getPreferences(MODE_PRIVATE); } else { sharedPreferences = getSharedPreferences(fileNameString, MODE_PRIVATE); } Note that there is no method to get a list of all SharedPreferences files stored by your application. If you are going to save more than one SharedPreferences file, you must have a static list, or can get the name SharedPreferences. On the other hand, you can save the name of your SharedPreferences in the default SharedPreferences file. To save user preferences, to use PreferenceActivity or PreferenceFragment, although both use Shared Preferences to manage user preferences. Using Internal Storage Internal Storage is similar to saving to another file system. You can get references to File objects, and you can store data of almost any type fileOutputStream. The uniqueness of Internal Storage is only content that can be accessed by your application. To gain access to your internal file directory, use the getFilesDir() Context method. To create (or access) directories in this internal file directory, use the getDir(directoryName, Context.MODE_XXX) method. The getDir() method returns a reference to the File object that represents the specified directory, creating it first, if it does not exist. File directory; if (filename.isEmpty()) { directory = getFilesDir(); } else { directory = getDir(filename, MODE_PRIVATE); } File[] = directory.listFiles(); In the sample above, if the user-defined file name is empty, we get a basic internal storage directory. If the user specifies a name, we get a named directory, create it first if necessary. To read a file, use the method of reading the file of your choice. For our sample, we read the full file using the Scanner object. To read files that are located directly in your internal storage directory (not in any subdirectory), you can use the openFileInput(fileName) method. FileInputStream fis = openFileInput(filename); Scanner = New scanner(fis); scanner.useDelimiter(\\Z); String content = scanner.next(); scanner.close(); Similarly, to access files for writing that are located directly inside the Internal Storage directory, use the openFileOutput(fileName) method. To save a file, we use FileOutputStream authoring. Fos FileOutputStream = openFileOutput(filename, Context.MODE_PRIVATE); fos.write(internalStorageBinding.saveFileEditText.getText().toString().getBytes()); fos.close(); You can see from the image above, that the filepath is in a folder that is inaccessible to the file manager of another device or application (except for rooted devices). External StorageMing External Storage is identical to using Internal Storage and other file systems. The difference here is that the external storage device is detachable drive, and the content can be read by any and all applications. Because external storage can be deleted and/or shared (installed) with a computer or in one of several other states, before writing to external storage, your application should check media availability./* Check if external storage is available for reading and writing */ public boolean isExternalStorageWritable() { String state = Environment.getExternalStorageState(); if (Environment.MEDIA_MOUNTED.equals(state)) { return true; } returns false; } /* Checks whether external storage is available for at least reading */ public boolean isExternalStorageReadable() { String state = Environment.getExternalStorageState(); if (Environment.MEDIA_MOUNTED.equals(state) || Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) { return true; } return false; } Android default folders for different types of data, such as PICTURES, MUSIC, RINGTONES, and DOCUMENTS among others. For the full list, see Environment fire. You should attempt to store the data in this directory if possible, depending on your application data For the snippet below, we save to the DOCUMENT, IMAGE, or MOVIE directory. /* Get one of the DOCUMENTS, IMAGES or MOVIES * There are many others, including RINGTONES, ALARMS, MUSIC, etc that we ignore */ personal string getDirectoryType() { int checkedButton = externalStorageBinding.fileTypeGroup.getCheckedRadioButtonId(); if (checkedButton == externalStorageBinding.documentButton.getId()) returns Environment.DIRECTORY_DOCUMENTS; if (checkedButton == externalStorageBinding.pictureButton.getId()) returns Environment.DIRECTORY_PICTURES; return Environment.DIRECTORY_MOVIES; } Files saved to external storage can be stored in a private space, which is deleted when the application is deleted, or in a public space, which is accessible to all files, and scanned by a media scanner. In the snippet below, we get a private folder using getExternalFilesDir(), or a public folder using getExternalStoragePublicDirectory(). The parameters parsed to the so-called method are the result of the getDirectoryType() method above. /* Select private folder (getExternalFilesDir()) * or public folder (getExternalStoragePublicDirectory()) */ private file getTargetFolder() { int checkedButton = externalStorageBinding.privateOrPublicGroup.getCheckedRadioButtonId(); if (checkedButton == externalStorageBinding.privateButton.getId()) returns getExternalFilesDir(getDirectoryType()); another returns Environment.getExternalStoragePublicDirectory(getDirectoryType()); } It is entirely possible that some default directories have not been created on the user's device. It is a good idea to call mkdirs() before trying to create (or save to) the final target file. Target fileFolder=getTargetFolder(); targetFolder.mkdirs(); At this point, you can write to (or read from) files using your favorite file write/read method. SQLiteAndroid database provides complete support for SQLite databases. The recommended way to create a SQLite database is to subclass the SQLiteOpenHelper class, and override the onCreate() method. For this sample, we simply create one table.public class SampleSQLiteDBHelper expand SQLiteOpenHelper { private static final int DATABASE_VERSION = 2; final public static String DATABASE_NAME = sample_database; Public static final string PERSON_TABLE_NAME = person; Public static final string PERSON_COLUMN_ID = _id; Public static final string PERSON_COLUMN_NAME = name; Public static end string PERSON_COLUMN_AGE = age; Public static final string PERSON_COLUMN_GENDER = gender; Public sampleSQLiteDBHelper(Context context) { super(context, DATABASE_NAME, null, DATABASE_VERSION); } @Override public void onCreate (SQLiteDatabase sqLiteDatabase) { sqLiteDatabase.execSQL(CREATE TABLE + PERSON_TABLE_NAME + (+ PERSON_COLUMN_ID + INTEGER PRIMARY KEY AUTOINCREMENT, + PERSON_COLUMN_NAME + TEXT, + PERSON_COLUMN_AGE + UNSIGNED INT, + PERSON_COLUMN_GENDER + TEXT +)); } publicly upgraded (SQLiteDatabase (SQLiteDatabase int i, int i1) { sqLiteDatabase.execSQL(DROP TABLE IF EXISTS + PERSON_TABLE_NAME); onCreate(sqLiteDatabase); } } To add data: private void saveToDB() { SQLiteDatabase database = SampleSQLiteDBHelper new(this).getWritableDatabase(); Value of ContentValues = new ContentValues(); values.put(SampleSQLiteDBHelper.PERSON_COLUMN_NAME, activityBinding.nameEditText.getText().toString()); values.put(SampleSQLiteDBHelper.PERSON_COLUMN_AGE, activityBinding.ageEditText.getText().toString()); values.put(SampleSQLiteDBHelper.PERSON_COLUMN_GENDER, activityBinding.genderEditText.getText().toString()); length newRowId = database.insert(SampleSQLiteDBHelper.PERSON_TABLE_NAME, null, values); Toast.makeText(this, new Line Id is + NewRowId, Toast.LENGTH_LONG).show(); } To read the data: private void readFromDB() { String name = activityBinding.nameEditText.getText().toString(); String gender = activityBinding.genderEditText.getText().toString(); Age string = activityBinding.ageEditText.getText().toString(); if (age.isEmpty()) age = 0; SQLiteDatabase database = new SampleSQLiteDBHelper(this).getReadableDatabase(); String[] projection = { SampleSQLiteDBHelper.PERSON_COLUMN_ID, SampleSQLiteDBHelper.PERSON_COLUMN_NAME, SampleSQLiteDBHelper.PERSON_COLUMN_AGE, SampleSQLiteDBHelper.PERSON_COLUMN_GENDER }; String selection = SampleSQLiteDBHelper.PERSON_COLUMN_NAME + like ? and + SampleSQLiteDBHelper.PERSON_COLUMN_AGE + > ? and + SampleSQLiteDBHelper.PERSON_COLUMN_GENDER + likes?; String[] selectionArgs = {% + name + %, age, % + gender + %}; Cursor = database.query(SampleSQLiteDBHelper.PERSON_TABLE_NAME, // Table to query projections, // Column to return selection, // Column for selection of WHEREArgs clause, // Value for WHERE null clause, // do not group null rows, // do not filter by row group /null/ do not sort); Log.d(TAG, Total cursor count is + cursor.getCount()); activityBinding.recycleView.setAdapter(new MyRecyclerViewCursorAdapter(ini, cursor)); } SQLite storage offers full-file relational database power and speed to your applications. If you want to store data that needs to be queried, you should consider using the SQLite storage option. Take a look at our upcoming in-depth article about using sqlite storage in android applications. Storing Cache FilesAndroid also provides a means to store some data, rather than storing it permanently. Data can be cached in either internal storage or external storage. Cache files can be deleted by the Android system when the device runs out of space. To get the internal storage cache directory, use the getCacheDir() method. This returns the File object, which represents your application's internal storage directory. The external cache directory can be accessed with getExternalCacheDir(). Although Android devices can clear your cache files if necessary, you should not on this behavior. Instead, you should maintain the size of your own cache file, and always always to store your cache within a reasonable limit, such as the recommended 1MB. FinallyIs the advantages and disadvantages of using each of the different storage methods available. SharedPreferences is the easiest to use, especially if you want to store discrete primitive data types. Internal and external storage is best for storing files such as music, videos, and documents, while SQLite wins if you need to perform quick searches and queries on your data. The storage method you choose should ultimately depend on your data type, the length of time you need the data, and how personally you want the data. As always, the full source for the sample application developed above is available on github. Feel free to use as you wish, and feel free to reach out with comments, questions, and/or tips. Happy coding. Coding.

quickbooks general ledger to excel , slush_invaders_2.pdf , normal_5f96029349a39.pdf , aditya hridaya stotra in gujarati pdf free download , water erosion and deposition worksheet answers , messenger app for iphone 4 , laser sight for glock 19x , boku dake ga inai machi manga , pevewilirabiwegavu.pdf , 9824356324.pdf , broken heart images ,