Convert apk to bar linux

In this article I'll talk about converting Android apps (which are on the APK format) to The Blackberry Playbook format. This article is a bit extensive, but extremely useful. The process is divided into three parts: preparation, transformation, signature. It's worth noting that many Android apps still don't work even after refitting, and can not use certain features such as the camera (already available in beta OS 2.1), Bluetooth, Wi-Fi, etc. See the full list of limitations on: Requirements: Java installed (you can find thousands of installation manuals for your system on the Internet) Android SDK Tools: Download into your system on: Selects Other IDE to download only SDK. Playbook Command Line Tools: the same is used to install .bar files, download: Keys to sign the package: Request your phone: Fill out the form, in the PIN to create a password that will be used in a subscription to the converted app, check the option for Playbook. Once you've sent the data, you'll get a key to sign (two attached files) by email for a maximum of two hours. Prepare Android SDK Open Android SDK Manager after installing it (or retrieving it if you're on Linux), if it's on Windows it's created a shortcut in the launch menu if it's on linux, access the folder/android-sdk-linux/tools and run the android app (give excutable permissions with the chmod x Android team, then run with the team. Once the list is updated, check the following options: Folder Tools, Android SDK Tools and Android SDK Plataform Tools, Folder Android 2.3.3 (API 10), Plataform SDK Clieque option to install packages, wait for installation and readiness. extracted by blackberry.tools.SDK/bin. In it will be where you will do all the work, you will also need to move two signatures of key files received by email to this folder. Open the Terminal (if on linux) or Team (if you just hold the shift in the windows and click the direct button, then choose the Open Command Window here). Setting up a signature (first time) Is the hardest part, because to install .bar files on the Playbook you need to sign them, and the configuration to create a certificate is a bit great (not to say exaggerated). ONLY THESE TEAMS ARE REQUIRED FOR THE FIRST TIME A CONVERTED APP. For linux users, always remember that before running the commands give permission to perform (team chmod x) and run the commands (example: apk2bar commands, run typing ./apk2bar). Let's go to the commands that will run in the same folder blackberry.tools.SDK/bin: #blackberry-signer -csksetup -cskpass SEUPINA'E As expected, replace SEUPINA'E with a PIN code (password) registered in the form of a key query. Now you will do the procedure for each file received by email, note that both are .csj files, but the name changes, one starts with the client-RDK and the other with the client-PBDT, the rest of the name code generated for each key query. #blackberry-signer-register -csjpin SEUPINA'e -cskpass SEUPINA's client-PBDT-CODIGOGERADO.csj Will once again replace SEUPINA with pin pair and CODIGOGENERATED with your client's code-PBDT file (you can simply copy the file name and replace the end of the team). Now again for the client-RDK'.csj file: #blackberry-signer-register -csjpin SEUPINA-Cskpass SEUPINA'e client-RDK-CODIGOGERADO.csj Once again replace SPINAEU'e with PIN and CODE GENERATED code of your client-RDK file (you can just copy the name of the file and replace the end of the team). Note both of the above commands there are any connection errors with the RIM server, so if you submit an error message, try again. Now we will generate .p12 file to be used to subscribe to converted applications: #blackberry-keytool-genkeypair-keystore NX.p12 -storepass SEUPINA'e -dname cn'NAMEEMPRESAA'a -alias author Replace YOURPINA with PIN, and NAMEEMPRESAA'E by name your company or put any name. This will create a NOX.p12 file that will serve to sign converted applications. If you miss this file, you'll have to do it over and over again, so it's good to have a backup of it. Conversion Surprisingly, this is the easiest part. Copy .apk you want to convert to blackberry.tools.SDK/bin folder, then open the Terminal or Command Window in the folder and run the command below: #apk2bar the zlt'ar'arquivo apk' Example: Air Horn free.apk Android SDK Location: C: Files of the program 'Android-sdk #apk2bar Air Horn free.apk C: 'files of the program'Android-sdk If any error message is displayed it is a sign that apk can not be converted. Any questions can be put in the comments. If all goes well, it will generate a .bar file with the same source name. Signed after conversion and already configured, the qnx.p12 file has reached the last step, signature. Assuming the apk file was Air Horn free.apk, the air horn free.bar file was generated in conversion. Move файла .bar в папку&lt;/pasta&gt; &lt;/arquivo&gt; &lt;/comando&gt; &lt;/comando&gt; from the bin folder (you can create it yourself). In my case I created a bar folder inside the bin folder. Now I'm running the team below: #batchbar-signer zlt;Pasta com. NX.p12 SEUPINA'ui In my case Air Horn free.apk that is inside the bar folder and my password was 12345: #batchbar-signed bar NX.p1 2 12345 If it works appears in the mezagame as a result (in addition to other information) AirHorn'free.apk.bar signature You can even put several files inside .bar for direct .bar. If all goes well, the bar file will be ready to install and then just install using different methods such as this: It can also happen to present a connection error on the server. In this case, try again. Another possibility is that someone has already signed the same file, then will let you know that the file has previously been signed and will not sign your .bar. In this case, go to the option below (or look online, because in these cases you tend to find in some forum or a good E-Reader) Re-sign (If the application is already signed) Set on your computer notepad, download on the priemira option below: Apos installed it will automatically open, then in the program click on the menu file qgt;open and look at your .bar file, in my example. After opening it will see at the beginning of the file more or less so: Package-Author: MyPlaybook Package-Author-Id: testEBRaRkVPnpsReKdNs4eKTS4 Package-Name: com.alwaysonpc.com .alwaysonpc.android.v Nc Package-Id: testELxDSuWxUsp7Jx3Uc'mFMF4 Package-Version: 1.0.15.0 Package-Version-Id: test EFSI7KIKI Ed IXDNYN9adxXAU Package-Type: App Package-Architecture: armle-v7 Application-Name: AlwaysOnPC App-Id: testEFh42gFesBuzJgLgh1Da Appendix-version: 1.0.15.0 Version-Id: testEF'7kIEdIXDNYN9adxAU Now change only the package version and version of the app, slightly increasing the version. For example, the current version is 1.0.15.0, so try changing the number on both to 1.0.16.0. Click on the file menu, then close and try to sign again. If it still leads to previously signed, open the .bar file again and further increase the version until it works. Sometimes it's necessary because someone has already done what you're trying to do (increase the version). Usually it works on the first attempt. Well guys, that's it, any doubt or error post post in the comments. Android Studio is the official integrated development environment (IDE) for the development of Applications for Android based on IntelliJ IDEA. In addition to intelliJ's powerful code editor and developer tools, Android Studio offers even more features that improve performance when creating Android apps, such as: A'lt/PastaGradle is based on the Build System Fast and Multi-comment Emulator Unified Environment where you can develop for all Android devices Apply changes to push code and resource changes in the running application without rebooting the application code templates and integrating GitHub to help you build common app features and import sample code Extensive testing tools and Lint framework tools to catch the performance, usability , compatibility versions and other support problems C and NDK Built-in support of Google's cloud platform, which makes it easy to integrate Google Cloud Messaging and App Engine This page provides an introduction to the main functions of Android Studio. For a quick run from the latest changes, see Project Structure Figure 1. Project files in the Android view. Each project in Android Studio contains one or more modules with source files and resource files. Types of modules include: Android modules app Library modules Google App Engine modules by default, Android Studio displays your project files in Android Project Vision, as shown in Figure 1. This view is organized by modules to ensure quick access to the key source files of the project. All build files are visible on the top level under Gradle Scripts, and each module of the app contains the following folders: manifests: Contains AndroidManifest.xml file. Java: Contains java code files, including JUnit test code. res: Contains all non-cococ resources such as XML layouts, UI lines, and bit card images. The structure of the Android project on the disk is different from this flattened view. To see the actual structure of the project file, select Project from the dropdown project (figure 1 shows it as Android). You can also customize the presentation of the project files to focus on specific aspects of app development. For example, when you select a project problem view, you see links to the original files that contain any recognized coding and syntax errors, such as the missing XML closing tag in the layout file. Figure 2. Project files in the Problem view showing the layout file with the problem. For more information, see the Android Studio Main Window's user interface consists of several logical areas identified in Figure 3. Figure 3. The main window of Android Studio. The toolbar allows you to perform a wide range of activities, including launching an app and launching Android tools. The navigation bar will help you navigate the project and open the editing files. It provides a more compact view of the structure visible in the project window. An editor's window is where you create and change code. Depending on the current type of file, the editor may change. When you view the layout file, the editor displays the Layout editor. The tool window panel runs around the outside of the IDE window and contains buttons that allow you to expand or or separate tool windows. Tool windows give you access to specific tasks such as project management, search, version management, and more. You can expand them and roll them up. The state bar shows the state of your project and the IDE itself, as well as any warnings or messages. You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use shortcuts to access most of IDE's features. At any time you can search in the source code, databases, actions, user interface elements, and so on, twice pressing the Shift key or tapping the magnifying glass in the top right corner of the Android Studio window. This can be very useful if, for example, you are trying to find a specific IDE action that you have forgotten how to trigger. The window tool Instead of using pre-installed vistas, Android Studio follows your context and automatically brings up the appropriate tool windows as you work. By default, the most commonly used tool windows are attached to the tool's window rack at the edges of the app window. To extend or roll up the tool window, click the name of the tool in the tool's window rack. You can also drag, pin, unplug, attach and unplug the tool windows. To get back to the current default tool window layout, click Window and Restore Default Layout or set up the default layout by clicking on the default Windows Current Layout store. To show or hide the entire tool window bar, tap the window icon in the bottom left corner of the Android Studio window. To find a specific tool window, hover over the window icon and select the tool box from the menu. You can also use shortcuts to open tool windows. Table 1 lists the labels for the most common windows. Table 1. Keyboard shortcuts for some useful tool windows. Windows Tool Window and Linux Mac Project Alt'1 Team No. 1 Control Versions Alt'9 Command No.9 Launch Shift-F10 Control R Debugging F9 Control F9 Control D Logcat Alt6 Command 6 Return to The Editor's Esc Esc Hide All The Tools Of Windows Control Shift Shift F12 CommandF12 If You Want, To hide all the toolbars, tool windows, and tab editor, click qgt; This allows you to divert free mode. To get out of distraction free mode, click the release of Distraction Free Mode. You can use Speed Search to search and filter in most tool windows in Android Studio. To use Search Speed, select the tool window and then enter the search query. For more tips, look at keyboard shortcuts. Completion of Android Studio code three types of code completion that can be accessed with keyboard shortcuts. Table 2. Key shortcuts to complete the code. A description like Windows and Linux Mac Basic Completion displays basic sentences on variables, types, methods, expressions, and so on. If you call the basic completion twice in a row, you see more results, including private member members unimmitable static members. Control of Space Control and Space Smart Completion Displays appropriate options based on context. Intelligent completion is aware of the expected type and data streams. If you call Smart Completion twice in a row, you see more results, including chains. Completion of space management and management completes the current statement for you by adding missing brackets, brackets, brackets, brackets, formatting, etc. Finding a sample code code Code Code Code In Android Studio will help you find high-quality, Google-provided samples of Android code based on the currently highlighted symbol in your project. For more information, see here are some tips to help you navigate Android Studio. Switch between recently accessing files with the Last Files action. The last available file is selected by default. You can also access any tool window through the left column in this action. View the structure of the current file using the file structure action. Prepare the file structure by clicking on Control-F12 (F12 Command on Mac). Using this action, you can quickly move on to any part of your current file. Search and navigate a particular class in your project with the Go to Class action. Prepare the action by clicking on Control-N. Go to a class supported by complex expressions, including camel humps, paths, go-to lines, head-names matching, and more. If you call it twice in a row, it shows you the results from the project classes. Go to a file or folder with the Go to File action. Bring Go to file by clicking on Control-Shift-N (Command-Shift-O on Mac). To search for folders, not files, add/at the end of the expression. Go to the method or field by name with the Go to the Symbol action. Bring Go to the Symbol by clicking on Control-Shift-Alt-N. Find all code fragments that refer to class, method, field, setting, or approval in the current cursor position by clicking on Alt-F7 (Option-F7 on Mac). When editing Android Studio automatically applies the formatting and styles specified in the code style settings. Code style settings can be customized in programming language, including conventions for tabs and indentations, spaces, packaging, and brackets, and empty rows. To set up code style settings, click On the zgt; file of the code style (Android Studio, the Editor's Code Style on Mac.) formatting while you're working, you can also directly call the action Reformat Code by clicking on Control Control on Mac), or automatically indent all lines by clicking Control AltI (Control Option I on Mac). Figure 4. The code before formatting. Figure 5. Code after formatting. The Android Studio version management framework supports a variety of version management (VCS) systems, including Git, GitHub, CVS, Mercurial, Subversion and Google Cloud Source Repositories. After importing your app into Android Studio, use the Android Studio VCS menu settings to enable VCS support for the desired version management system, create a repository, import new files into version management, and perform other version management operations: From the Android Studio VCS menu, click Enable version control integration. From the drop-off menu, select the version control system to link it to the root of the project, and then click OK. The VCS menu now displays a range of version management options based on the system you choose. Note: You can also use the option to zgt; customize the version controls to customize and change the settings for version management. The Paddle Build Android Studio system uses Gradle as the basis of the assembly system, with more Android-specific features provided by the Android plugin for Gradle. This build system works as an integrated tool from the Android Studio menu, and regardless of the command line. You can use the assembly system to customize, customize, and expand the build process. Create multiple APKs for your app with different features using the same design and modules. Reuse code and resources from different sources. Using Gradle's flexibility, you can achieve all of this without changing the main source files of the application. The Android Studio build files are called build.gradle. These are simple text files that use Groovy syntax to customize the build with elements provided by the Android plug-in for Gradle. Each project has one top-level build file for the entire project and separate assembly files at module level for each module. When you import an existing Project, Android Studio automatically generates the assembly files you need. To learn more about the build system and how to set up, see Create The Build System options can help you create different versions of the same application from the same project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs to different device configurations in Google Play. For more information about setting up build options, see multi-APK Multiple APK support to effectively create multiple APKs based on screen density or ABI. For example, you can create separate APKs apps for hdpi and mdpi screen density, while treating them as a single option and allowing them to APK, javac, dx, and ProGuard test settings. For more information about several support, read Build a few APKs. Reducing resources in Android Studio automatically removes untapped resources from dependencies on packaged apps and libraries. For example, if your app uses Google Play services to access Google Drive functionality and you don't currently use Google Sign-In, you may remove various drawing resources for TheSignInButton buttons. Note: Reducing resources works in conjunction with code-cutting tools such as ProGuard. For more information on code and resource reductions, see Dependency Management for the project by name in the build.gradle file. Gradle cares about finding dependencies and accessing them in the build. In the build.gradle file, you can declare a dependency on modules, deleted binary dependencies and local binary dependencies. Android Studio sets up projects to use the default Maven central repository. (This configuration is included in the top-level build file for the project.) For more information about dependency setting, please read Add Build Dependency. Android Studio helps you debug and improve your code performance, including debugging and performance analysis tools. Debugging inline Using built-in debugging to improve code holes in debugging view with built-in checks of links, expressions, and variables. Inline debugging information includes: Inline Variable Values Citing Objects That Refer to the Selected Object Lambda Return Method and Tooltip Expression Operator figure 6. Inline is a variable value. To enable debugging in the debugging window, click Settings and select a checkbox for Show inline values. The performance of Android Studio profilers provides performance to profilers, so you can more easily track your app's memory and processor usage, find deallocated objects, find memory leaks, optimize graphics performance, and analyze network queries. With an app running on your device or emulator, open the Android Profiler tab. For more information on performance profilers, see a pile of landfills When you're profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and reset the Java pile of snapshots into the Android-specific HPROF Binary File. The HPROF viewer displays classes, instances of each class, and reference tree to track memory usage and find memory leaks. For more information about how to handle heaps, see Memory Profile you can use memory profile to track memory distribution and see where objects are distributed when certain actions are performed. these highlights help optimize your application's performance and memory by setting up the method calls to these actions. For information on tracking and analysis of selections, see Access to data files Android SDK Tools, such as Systrace and logcat, generate performance and debugging data for detailed application analysis. Open the Captures tool window to see the available data files. In the list of generated files, double-click the file to view the data. Click the right button on any .hprof files to convert them into a standard RAM file format. Code Checks Whenever you compile your program, Android Studio automatically launches customized Lint and other IDE checks to help you easily identify and fix structural quality problems in your code. The Lint tool checks the original Android project files for potential errors and improves optimization for correctness, security, performance, usability, availability, and internationalization. Figure 7. The results of Lint's check at Android Studio. In addition to Lint checks, Android Studio also performs IntelliJ code checks and checks annotations to optimize the coding workflow. For more information, see Better Code by checking the pile. Android Studio Android Studio annotations support annotations to variables, parameters, and return values that will help you catch bugs such as zero-pointer exceptions and resource-type conflicts. Android SDK Manager packs an annotation support library into the Android support store for use with Android Studio. Android Studio checks customized annotations while checking the code. For more information about Android annotations, see Log Messages When you create and launch an app using Android Studio, you can view adb output and device log messages in the Logcat window. Productivity Profiling If you want to profile your app's processor, memory and network performance, open the Android Profiler by clicking on The View of the Android Profiler. You can log into your developer account at Android Studio to access additional tools that require authentication, such as cloud tools for Android Studio and the app's action testing tool. When you sign up, you give these tools permission to view and manage data on Google services. Once you've opened the project in Android Studio, you can log in to your developer account or switch your developer accounts as follows: tap the profile icon at the end of the toolbar, as shown in Figure 8. Figure 8. Click the profile icon at the end of the toolbar to enter it. In the window that appears, do one of the following: If have not yet joined in, click login and allow Android Studio to access these services. If you've already signed up, click Add account to log in to another Google account. Alternatively, you can click out and repeat the previous steps to get in Account. Account.