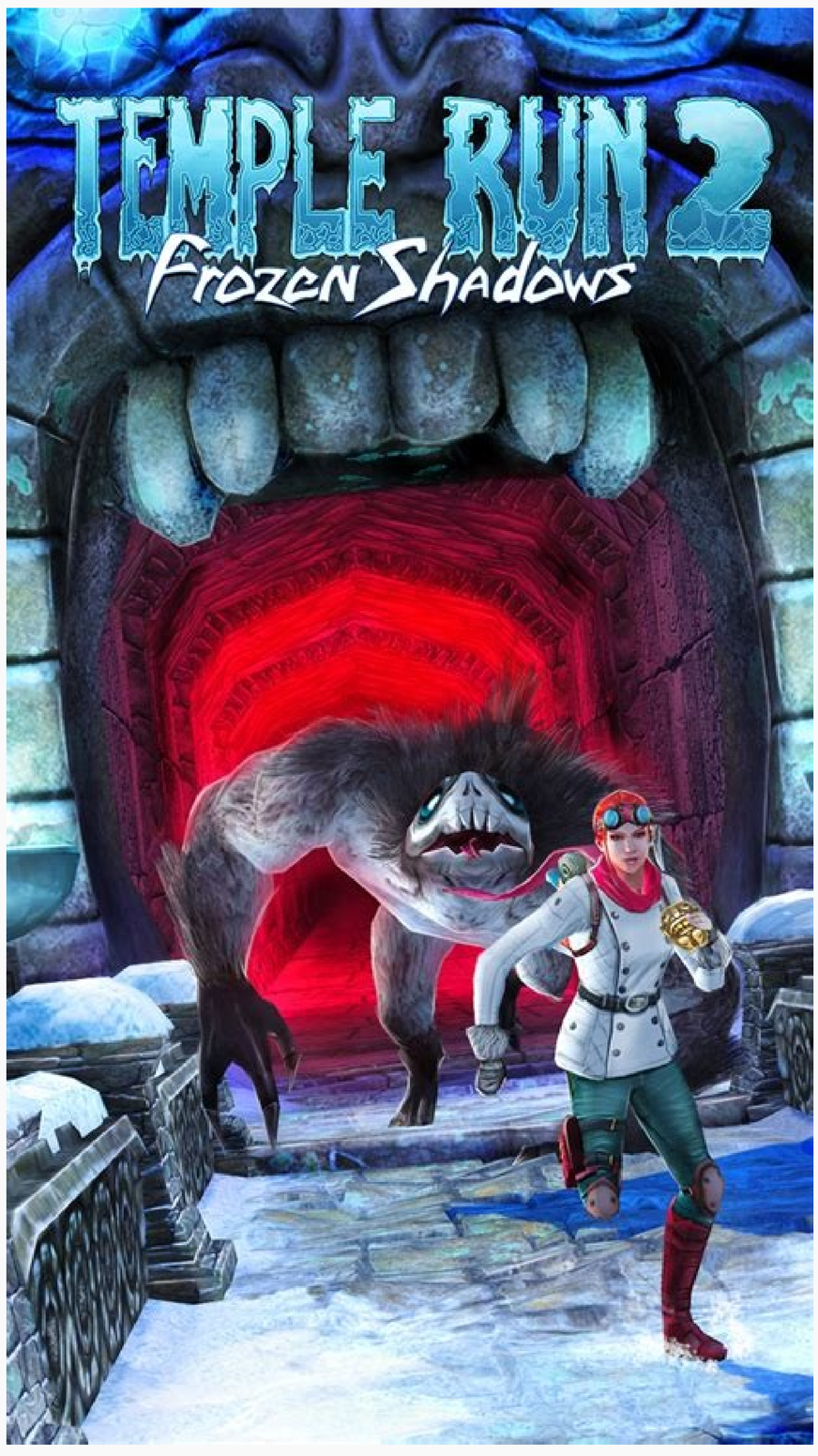
How to run android app on localhost

Continue



Today I will show you how you can convert your existing React js app to an android or ios app using ionic capacitor.So what exactly is Capacitor? According to documentationCapacitor is a cross-platform app runtime that makes it easy to build web apps that run natively on iOS, Android, Electron, and the web. We call these apps “Native Progressive Web Apps” and they represent the next evolution beyond Hybrid apps.So today we will build our android or ios app using the existing react app.RequiremensExisting React AppIonicAndroid StudioXcodeFirst, go to the root of your existing react app and create a file capacitor.config.json and inside that put the below code.2. Now create another file name ionic.config.json and inside that insert the below code.Note: replace nameofyourapp in both files with the name of your app.3. Now we need to build our react project. To build your react app open your terminal to the root of the project and run the below-mentioned commandNote: this will create the build folder in your root porject and the name of the folder should match the webDir name inside capacitor.config.json file4. Now we will install ionic globally in our machine. To install ionic globally in your machine open your terminal and run the below command.5. Now install the capacitor core in our project.Android6. After that, we will first create an android app with our existing react app. Open your terminal and typeThis will create the android folder in your root project and install all the required dependencies.7. Now run the below command to open your android project in android studio.Wait some time and then it will ask you to update the Gradle. Just update the Gradle to the latest version and run the project in the emulator. You can also connect your mobile to run the project live on your mobile phone.8. Now open the build menu from the android studio and build your apk file.iOS9. To create ios app run the below commandThis will install all the required dependencies and ios folder to your project.10. Now run the below command to open your ios project in Xcode.11. Now open the app in emulator or physical device and build the app.Visit the capacitor homepage for detailed information and documentation.Here is the git repository for reference Arthur_Shevtsov/Shutterstock.com You may be surprised to learn that Android is the most popular operating system on the planet—even bigger than Windows. That makes it a target for malicious attacks. But do you actually need antivirus apps on your Android phone? That’s a fair question to ask. Antivirus software has been recommended for Windows users for many years. Thankfully, Microsoft has built-in better tools to combat malware—Windows has come with a built-in antivirus since Windows 8—but what about Android? Let’s talk about some of the ways Android is protecting you. Google Play Protect Android’s biggest built-in defense against malware is Google Play Protect. There are a few different components to Play Protect—including the Find My Device tools—but a big part of it is malware scanning. Every Android device that included the Google Play Store has Play Protect. You may have noticed the “No Harmful Apps Found” message at the top of the apps and games management section in the Play Store. Play Protect makes sure the apps you download from the Play Store are safe. Play Protect doesn’t just work in the Play Store, though. It keeps an eye on everything outside of the store as well. Even apps that have been sideloaded from outside the Play Store are scanned by Play Protect. Sideloading is still inherently riskier, but it’s nice to know Play Protect is watching. In addition to scanning apps, Play Protect can protect you when browsing with Google Chrome too. Just like on Chrome for desktop, if you visit a site with malicious code, Chrome will warn you and take you back to safety. RELATED: What is Google Play Protect and How Does it Keep Android Secure? Monthly Security Updates Another big thing that protects your Android device is monthly security updates. These are smaller updates that don’t typically have shiny new features, but they’re very important. New vulnerabilities and exploits are popping up all the time. If your Android phone was only updated once a year, these things would pile up and become dangerous. It’s critically important to routinely squash these things as they come up. That’s why monthly security updates are necessary. Sadly, not all Android devices receive these updates in a timely manner or at all. Google releases the security updates every month and it’s up to its partners (Samsung, OnePlus, etc.) to approve the fixes, add any of their own, and release them to devices. If you want the most secure Android phone, your best bet is a Google Pixel or Samsung Galaxy device. Both Google (unsurprisingly) and Samsung are the most

reliable when it comes to keeping devices up to date with the latest patches. RELATED: What Are Android Security Updates, and Why Are They Important? So, Does My Phone Need Antivirus? These are the tools Android uses to keep your phone safe but are they enough on their own? They are for the vast majority of people. There’s no reason to download an antivirus app on your Android phone. The same basic rules for using any device safely apply to Android devices: Get your apps from the official source, the Google Play Store. Don’t go to shady-looking websites. Don’t click suspicious links in emails. Along with Play Protect, this is more than enough protection. On the other hand, you may need to download an antivirus app if you use Android devices without the Play Store, sideload APKs from untrustworthy sources, or have an old, outdated version of Android. If that sounds like you, an app such as Bitdefender or Norton 360 may be useful. The moral of the story is you don’t need an antivirus app on your Android phone if you use it responsibly. If your device has the Play Store, you’re in good shape. RELATED: How Antivirus Software Works Local Testing lets you test work-in-progress web and mobile apps at scale without hosting them on public staging environments. To learn how to use Local Testing for your development or staging environment, pick a BrowserStack product you test with: Did this page help you? Testing your car app ensures that users don't encounter unexpected results or a poor experience. How you test your app depends on whether your app will be running on Android Auto or Android Automotive OS. See one of the following sections to get started: Test your app for Android Auto. Test your app for Android Automotive OS. Important: In addition to testing your app's user experience, make sure that your app meets all of the criteria listed on the Android app quality for cars page before you submit your app to Google Play for review. Test your app for Android Auto The Desktop Head Unit (DHU) enables your development machine to emulate an Android Auto head unit, so that you can run and test Android Auto apps. The DHU runs on Windows, macOS, and Linux systems. This section teaches you how to install and run the DHU on your development machine to test your apps. After you've installed the DHU, you can test your Android Auto apps by connecting your mobile device to it via one of the supported connection mechanisms described in Run the DHU. Note: If you are testing on a mobile device running Android 10 (API level 29) or above, you need to sign in to the Google Play Store and update the Android Auto app before running the DHU for the first time. If you are building a media app, you can also test your media session implementation using the Media Controller Test app. To report bugs or request features related to the DHU, use this issue tracker. Install the DHU The latest available version of the DHU is 2.0. This version includes features that are not available in the previous version (1.1) and is the recommended version to install and use. If you are experiencing issues with version 2.0, you can install version 1.1 instead. Install DHU version 2.0 Follow these steps to install DHU 2.0 on your development machine: Enable developer mode on a mobile device running Android 6.0 (API level 23) or higher, as described in Enabling on-device developer options. Compile and install your app on the device. Install Android Auto on the device. If Android Auto is already installed, make sure that you are using the latest version. Open Android Studio and switch Android Studio to use the Beta channel. If you are prompted to update to a newer version of Android Studio, you can ignore the update. Open the SDK Manager and navigate to the SDK tools tab, then download version 2.0 of the Android Auto Desktop Head Unit Emulator package. The DHU is installed in the SDK_LOCATION/extras/google/auto/ directory. After the download has finished, you can switch Android Studio back to the Stable channel if you don't want to use the Beta channel version of other tools. On Linux or macOS systems, run the following command to ensure the DHU binary is executable: chmod +x ./desktop-head-unit Note: DHU 2.0 includes a libusb library, which is downloaded to the same directory. This library should stay in the same location as the DHU executable for it to run properly. Additional steps for Linux only If you are running version 2.0 on a Linux system, you need to install some additional libraries. The DHU binary depends on GLIBC version 2.32 or above. You can check the GLIBC version of your system by running this command: ldd --version If the version is lower than 2.32, you must either update GLIBC to 2.32 or above, or upgrade the OS to a version that is compatible with GLIBC 2.32 or above. Note: Version 2.32 is not currently available on the Debian stable release yet. If this applies to you, please reach out to us. You must also install the libc++1 and libc++abi1 libraries. The installation procedure varies depending on your Linux distribution. For example, on Debian-derived Linux distributions, you can install the libraries with this command: sudo apt-get install libc++1 libc++abi1 Install DHU version 1.1 If you are encountering issues with version 2.0, follow these steps to install DHU 1.1 on your development machine: Enable developer mode on a mobile device running Android 6.0 (API level 23) or higher, as described in Enabling on-device developer options. Compile and install your app on the device. Install Android Auto on the device. If Android Auto is already installed, make sure that you are using the latest version. Open Android Studio and check that Android Studio is using the Stable channel. Open the SDK Manager and navigate to the SDK tools tab, then download version 1.1 of the Android Auto Desktop Head Unit Emulator package. The DHU is installed in the SDK_LOCATION/extras/google/auto/ directory. After the download has finished, you can switch Android Studio back to another update channel if you don't want to use the Stable channel version of other tools. On Linux or macOS systems, run the following command to ensure the DHU binary is executable: chmod +x ./desktop-head-unit Additional steps for Linux only If you are running version 1.1 on a Linux system, you need to install some additional libraries. Install the portaudio, libpng, sdl2, and sdl2_ttf libraries. The installation procedure varies depending on your Linux distribution. For example, on Debian-derived Linux distributions, you can install the libraries with this command: sudo apt-get install libsdl2-2.0-0 libsdl2-ttf-2.0-0 libportaudio2 libpng12-0 Run the DHU After you install the DHU, you can test your Android Auto apps by connecting your mobile device and workstation over USB. Run the DHU by connecting your mobile device to a development machine using either the Accessory Mode (recommended for DHU 2.0), or ADB Tunneling. Connecting using Accessory Mode (DHU 2.0 only) Android Auto supports connecting to the DHU with the Android Open Accessory (AOA) protocol, using the following command: ./desktop-head-unit --usb By default, the DHU scans through the list of available USB devices and attempts to connect to the first compatible one. To target a specific device, include the device ID as part of the --usb command, as follows: ./desktop-head-unit --usb=[DEVICE_ID] Note: On Windows, this feature may require installation of the WinUSB driver against your mobile device, and may interfere with the ADB connection. If the DHU fails to discover and connect to the mobile device, try running adb kill-server before starting the DHU, or use the alternative connection method below if needed. Connecting using ADB Tunneling This method requires setting up a connection to the Android Auto head unit server over Android Debug Bridge (ADB). Follow these steps to set up tunneling and run the DHU: Do one of the following to open the Android Auto settings, depending on the version of Android running on the device: Android 10 or higher: On the device tap Settings > Apps & notifications > See all apps > Android Auto > Advanced > Additional settings in the app. Android 9 or lower: In the Android Auto app, tap the Menu, then tap Settings. Scroll to the About section near the bottom and tap Version to display all of the version and permission information. Tap the Version and permission info section 10 times. The Allow development settings dialog appears. Tap OK. Developer mode is now enabled and you can access developer options in the overflow menu. You only need to enable developer mode the first time you run the Android Auto app. If the head unit server is not already running, tap the three dots in the top right of the screen to open the overflow menu and select Start head unit server (see figure 1). On the device, a foreground service appears in the notification area that indicates that the server is running (see figure 2). Figure 1. Context menu with developer options. Figure 2. Notification that the head unit server is running. In the Android Auto app, tap Previously connected cars near the top of the Settings, and ensure Add new cars to Android Auto is enabled. Connect the mobile device to the development machine via USB. Make sure the mobile device has its screen unlocked; otherwise it cannot launch the DHU. On the development machine, run the following adb command to forward socket connections from the development machine's port 5277 to the same port number on the Android device. This configuration enables the DHU to connect to the head unit server running on your phone over a TCP socket. adb forward tcp:5277 tcp:5277 Start the DHU by running the command desktop-head-unit.exe (on Windows) or ./desktop-head-unit (on macOS or Linux) from the SDK_LOCATION/extras/google/auto/ directory. cd SDK_LOCATION/extras/google/auto desktop-head-unit.exe # Windows ./desktop-head-unit # macOS or Linux If this is the first time you are connecting the mobile device to the DHU, check the mobile device's screen to accept the terms of service and adjust permission settings as needed. After the DHU has started, you can use DHU console commands to run and test your app. Command line options By default, the head unit server connects over port 5277. To override the host or port (for example, to forward over SSH), use the --adb= flag, as in the following example: ./desktop-head-unit --adb=5999 By default, the DHU emulates a Android Auto-compatible head unit with a touch screen user interface. You can simulate user touches by clicking the DHU with a mouse. To emulate head units that use a rotary controller for input, you can use the -i controller flag, as in this example: ./desktop-head-unit -i controller When the DHU is in rotary-controller mode, you can use keyboard shortcuts to simulate controller operations. See the Rotary Controller for more details. In rotary controller mode, the DHU ignores mouse clicks; you must operate Android Auto with the simulated rotary-controller operations. The full set of command line options that the DHU supports vary depending on whether you are using version 2.0 or 1.1. Version 2.0 Option Description c, --config=FILE Use the specified configuration .ini file. Refer to the Configure the DHU section for more details. i, --input=INPUT Use specified input mode, one of: touch, rotary, or hybrid. -a, --adb=HOSTPORT Use ADB transport (optional host:port or port). Defaults to port 5277. -a, --usb=DEVICE_ID Use USB (AOA) transport (optional DEVICE_ID). v, --version Show version information. l, --licenses Show open source licenses. h, --headless Run in headless mode (no UI). ?, --help Show this help listing. Version 1.1 Option Description i, --input=INPUT Use specified input mode, one of: touch or rotary. -a, --adb=HOSTPORT Use ADB transport (optional host:port or port). Defaults to port 5277. v, --version Show version information. l, --licenses Show open source licenses. h, --headless Run in headless mode (no UI). DHU User Guide (2.0) This section describes the features supported by version 2.0 of the DHU and how to use them. Note: DHU 2.0 is still in the beta channel, and there might be instability. Please report bugs on the issue tracker if you encounter issues. Commands are entered in the terminal window where you started DHU. Multiple commands can be run on the same line by separating them with a ;. System Command Key Description help [command] Shows full command set. If a command name is specified (for example, help day), shows help for that command. quit exit Alt+Q Quits the head unit. sleep [seconds] Sleeps for one second. If a delay is specified, sleeps that many seconds. This can be used for scripting the head unit (./desktop-head-unit < script.txt), for example in a CI environment. screenshot filename.png Saves a screenshot to filename.png. licenses Display licenses of libraries used in DHU keycode keycode Send keycode, one of the names listed in the Keycodes section. Touch and Touchpad When touch or touchpad is enabled, clicking in either the display window (for touch) or touchpad window (for touchpad) will be registered as touch events. For touch, you can simulate multi-touch by right clicking on the first finger location, then click-and-hold the right button for the second finger. The centerpoint of the two fingers remains fixed and moving the mouse will allow you to either rotate them around the center or pinch in-and-out (or both). For touchpad, click-and-hold the left button to scroll between different UI elements on the screen. Clicking the right button will select the element that currently has focus. Command Key Description tap x y Simulate a touch event at the specified coordinates Rotary Controller The DHU has rotary controller support. When it is enabled, the following actions are supported: Up, down, left, and right on the Dpad Rotating clockwise and counterclockwise Performing a rotation flick (5 steps at once) Clicking down on the controller and back on the controller The commands and key bindings are listed below. A scroll wheel on a mouse will send dpad rotate commands, and the middle mouse button click (usually the scroll wheel) will send dpad click commands. Note that most cars have touch screens. Some cars have rotary controller-only, and some have a hybrid touch and controller. Cars can also support a touchpad which can be used for map panning and text entry. Bear these different configurations in mind when simulating different vehicle head units. Command Key Description dpad {up|down|left|right} Arrow keys Movement of the rotary controller. dpad {ur|dl|ul|dr} Movement of the rotary controller. dpad soft {left|right} Shift + Arrow keys Side buttons available on some rotary controllers. dpad click Return Pressing the rotary controller. dpad back Backspace Back button available below some rotary controllers. dpad rotate left 1 Rotary controller left (counter-clockwise) movement. dpad rotate right 2 Rotary controller right (clockwise) movement. dpad flick left Shift+1 Fast counter-clockwise spin of the rotary controller. dpad flick right Shift+2 Fast clockwise spin of the rotary controller. dpad 0-9*#+ Num pad Microphone The DHU supports using a microphone for voice input or playing a pre-recorded voice track. For your convenience, we have provided the following sound files for common voice commands. These sound files are installed in the SDK_LOCATION/extras/google/auto/voice/ directory. File name Text navhome.wav Navigate to home. navwork.wav Navigate to work. navsoh.wav Navigate to Sydney Opera House. navgoogle.wav Navigate to 1600 Amphitheatre Parkway, California, USA. exitnav.wav Exit navigation. howlong.wav How long until I get there? showtraffic.wav Show traffic. showalternateroute.wav Show alternate routes. pause.wav Pause music nextturn.wav When is my next turn? To run an individual .wav file in the DHU prompt: mic play /path/to/filename.wav Command Key Description mic begin M Activates the microphone (equivalent to clicking the steering-wheel button) and waits for input from the computer microphone. mic play filename.wav Activates the microphone and plays the specified WAV file recording. Note: You will not hear the WAV file being played, but you will hear the response from Android Auto. mic repeat Repeats the last recording used with mic play. mic reject {on|off} Enables or disables rejection of microphone requests. When on, all microphone requests will be rejected. Sensors The DHU supports simulating changes in the vehicle's sensor data with the following commands. To mock sensor data and send them to Android Auto, the corresponding sensors have to be enabled with the configuration .ini file when starting the DHU. Command Description fuel [percentage] Set the fuel level to the specified percent, or do not pass in a value to disable fuel level. range [km] Set the range to the specified kilometers, or do not pass in a value to disable range data. lowfuel [{on|off}] Set the low fuel warning sensor, or do not pass in a value to disable the sensor. accel [x] [y] [z] Set the accelerometer to the specified x, y, and z values (m/s^2), or do not pass in parameters to unset the accelerometer data. Pass in NAN to skip optional parameters if needed. compass bearing [pitch] [roll] Set the compass to the specified bearing, pitch, and roll values (degrees). Pass in NAN to skip optional parameters if needed. gyro [x] [y] [z] Set the gyroscope to the specified x, y, and z rotation speed (rad/s), or do not pass in parameters to unset the gyroscope data. Pass in NAN to skip optional parameters if needed. location lat long [accuracy] [altitude] [speed] [bearing] Set the location to the specified lat and long values along with the optional accuracy (m), altitude (m), speed (m/s), and bearing (degrees). Pass in NAN to skip optional parameters if needed. odometer km [current_trip_km] Set the odometer to the specified kilometers along with an optional current trip km value. speed [speed] Set the vehicle speed to the specified value (m/s), or do not pass in a value to disable the sensor. tollcard {insert|remove} Indicates whether a toll card was inserted or removed. Day/Night Mode The DHU supports simulating changing between day and night mode with the following commands and key bindings. Command Key Description day Shift+N Activate day mode (high-brightness, full color). night Ctrl+N Activate night mode (low-brightness, high-contrast). daynight nightday N Toggle current day or night mode. Focus Management The DHU supports simulating whether or not Android Auto has focus on the head unit with the following commands. Command Description focus video {on|off|toggle} Enables or disables whether Android Auto has video focus on the head unit. Turning off video focus simulates the head unit going into native mode. focus audio {on|off|toggle} Enables or disables whether Android Auto has audio focus on the head unit. Turning off audio focus simulates the head unit playing its own audio source. focus nav {on|off|toggle} Enables or disables whether Android Auto has navigation focus on the head unit. Turning off navigation focus simulates the head unit running its own navigation system. Restrictions/Driving Status The DHU supports simulating certain restrictions when the vehicle is moving, such as disabling the keyboard and disallowing phone configurations. Command Key Description restrict none U Disables all restrictions. restrict all Shift+U Enables all restrictions (for example, to simulate driving). Instrument Cluster The DHU supports emulating an instrument cluster (usually located behind the steering wheel) which shows basic info during navigation about the next turn (such as the next road name, distance/time, or a turn arrow) and information about an ongoing phone call. This can be enabled using the instrumentcluster entry in the configuration file, or by the navcluster or phonecluster entries to control each feature separately. The DHU supports an additional information display to show media playback status, for example, what track is playing. This can be enabled using the playbackstatus entry in the configuration file. Keycodes The DHU provides a set of default keycodes that simulates the set of shortcut buttons that are commonly available in different vehicles. For example, the home keycode can be triggered in the DHU prompt as follows: keycode home Keycode Description home Go to the home screen. back Go back. call endcall Make or end a call. search Trigger search. media_play_pause media_play media_pause Play or pause a media. media_next media_previous Go to the next or previous media track. media Go to the default media app. navigation Go to the default navigation app. tel Go to the default telephone app. Configure the DHU The DHU supports a configuration .ini file to change the input modes available (touch, controller), the frame rate, resolution and DPI the head unit requests from the phone. The default location for the configuration file is ~/.android/headunit.ini. If you wish to change the configuration of the head unit on your system, you can modify this file. You can also specify a configuration file to load using the -c flag. ./desktop-head-unit -c /path/to/config.ini Sample configuration [general] touch = true touchpad = false controller = false instrumentcluster = false resolution = 800x480 dpi = 160 framerate = 30 fueltypes = unleaded,electric,hydrogen evconnectors = supercharger Sample configuration files can be found under the SDK_LOCATION/extras/google/auto/config/ folder to demonstrate different head unit settings you can test against. For more advanced use cases, please refer to the supported settings below. Input Configuration Name Default Type Description inputmode default String Defines the input mode. The options touch, rotary, hybrid enable and disable touchscreen and rotary support and set default keycodes as appropriate, while default defers to the touch and controller options below. controller false Boolean Enable rotary controller input. Ignored unless inputmode is default. touch true Boolean Enables touchscreen. Ignored unless inputmode is default. touchpad false Boolean Enables touchpad. touchpadnavigation false Boolean Enables touchpad to be used for UI navigation. touchpadtapasselect false Boolean When true, a tap on the touchpad is a select event. touchpaduiabsolute false Boolean When touchpadnavigation is true, this sets whether the touches should be treated as absolute or as gestures. Video Configuration Android Auto supports three video resolutions: 480p (800x480 -- default) 720p (1280x720) 1080p (1920x1080) To support different aspect ratios other than the ones above, the head unit is able to specify a margin to letterbox/pillarbox as necessary. For example, if you wanted a 1000x600 screen, you would set the resolution to 720p (1280x720) and a marginwidth of 280 and marginheight of 120. This has the effect of adding a 140px margin on both the left and right edges, and 60px top and bottom. Example configuration to emulate a typical 6 inch screen (750x450): [general] ... resolution = 800x480 marginwidth = 50 marginheight = 30 And a very-wide screen: [general] ... resolution = 1280x720 marginwidth = 0 marginheight = 220 Name Default Type Description resolution 800x480 Strings One of: 800x480, 1280x720, 1920x1080. dpi 160 Integer normalizedpi false Boolean When true, reduces DHU window size to account for larger DPI values. When false, larger DPI values result in larger windows (which does not mimic real displays, but does allow easier inspecting of visual content). realdpi 160 Integer Used in video configuration. framerate 30 Integer Used in video Configuration. marginheight 0 Integer Used in video configuration. marginwidth 0 Integer Used in video configuration. margins 0,0,0,0 String Used in video configuration, overrides marginwidth and marginheight if present. Format is top, bottom, left, right. contentinsets 0,0,0,0 String Used in video configuration. Format is top, bottom, left, right. stablecontentinsets 0,0,0,0 String Used in video configuration. Defaults to the value of contentinsets. Format is top, bottom, left, right. cropmargins false Boolean If margins, marginheight, or marginwidth is specified, a true value for this setting removes those margins from the displayed video. This reflects more closely what the end user sees. pixelaspectratio 1.0 Float Used in video configuration. Sensors Configuration To mock sensor data and send them to Android Auto, the corresponding sensors have to be enabled using the options below. If a sensor is disabled, any data sent for that sensor using the DHU commands is ignored. Name Default Type Description accelerometer false Boolean Enable accelerometer sensor data. compass false Boolean Enable compass sensor data. driving_status false Boolean Enable driving status sensor data. fuel false Boolean Enable fuel sensor data. gyroscope false Boolean Enable gyroscope sensor data. location false Boolean Enable vehicle location sensor data. night_mode false Boolean Enable night mode sensor data. odometer false Boolean Enable odometer sensor data. speed false Boolean Enable speed sensor data. toll_card false Boolean Enable toll card sensor data. Miscellaneous Name Default Type Description instrumentcluster false Boolean Enable instrument cluster. Adds a window to the UI to display navigation and phone status. navcluster false Boolean Enable instrument cluster. Adds a window to the UI to display navigation status. phonecluster false Boolean Enable instrument cluster. Adds a window to the UI to display phone status. playbackstatus false Boolean Enable playback status. Adds a window to the UI to display playback status messages. driverposition left String One of: left, center, or right. windowleft Integer Set the left position of the main window. windowtop Integer Set the top position of the main window. fueltypes unleaded Strings One or more of (separated by ,): unleaded leaded diesel-1 diesel-2 biodiesel e85 lpg cng lng hydrogen electric other unknown evconnectors Strings Zero or more of (separated by ,): j1772 mennekes chademo combo-1 combo-2 roadster hpwc gbt supercharger other unknown Should only be set if fueltypes contains electric. DHU User Guide (1.1) DHU commands allow you to test your app with Android Auto features, such as playing voice input or switching between night and day display mode. You can issue commands to the DHU from the terminal window where you launched the DHU. You can also issue commands by selecting the DHU window and using keyboard shortcuts. The DHU commands and key bindings for all controls are listed in DHU commands and key bindings. Switch between day and night mode Android Auto supports different color schemes for day and night. You should test your app in both day and night mode. You can switch between night and day mode in either of the following ways: Run the command daynight in the terminal where you launched the DHU. Select the DHU window and press the N key. Microphone testing The DHU supports using a microphone for voice input. You can also instruct the DHU to treat a prerecorded voice track as input, as if the DHU had heard the track through the microphone. To use a prerecorded sound file as input, enter this command: mic play /.wav For your convenience, we have provided the following sound files for common voice commands. These sound files are installed in the /extras/google/auto/voice/ directory. File name Text navhome.wav Navigate to home. navwork.wav Navigate to work. navsoh.wav Navigate to Sydney Opera House. navgoogle.wav Navigate to 1600 Amphitheatre Parkway, California, USA. exitnav.wav Exit navigation. howlong.wav How long until I get there? showtraffic.wav Show traffic. showalternateroute.wav Show alternate routes. pause.wav Pause music nextturn.wav When is my next turn? DHU commands and key bindings The DHU supports the following commands. Table 1. Commands and key bindings Category Command Subcommand Argument(s) Keyboard Shortcut(s) Description System help [command] Shows the full command set. Specifying a command name (for example, help day) causes the system to show help for that command. quit Alt+q Quits the head unit. sleep [seconds] Sleeps for one second. Specifying an argument (for example, sleep 30) causes the system to sleep the specified number of seconds. This command is useful if you are writing scripts for the DHU. (You can run a script by using I/O redirection from the command line: ./desktop-head-unit < script.txt loads commands from the file script.txt.) screenshot filename.png Saves a screenshot to filename.png. Microphone mic begin m Activates the microphone (equivalent to pressing the microphone button on the steering wheel) and waits for input from the computer microphone. play filename.wav Causes the DHU to treat filename.wav as voice input, as if that sound were relayed through the microphone. You do not hear the sound file being played, but you do hear the response from Android Auto. repeat Repeats the last mic play command, as if you had called mic play again with the same sound file parameter. Input dpad up down left right Arrow keys Simulates moving the rotary controller. soft left soft right Shift+Arrow keys Simulates pressing the side buttons available on some rotary controllers. click Return Simulates pressing the rotary controller. back Backspace Simulates pressing the back button available below some rotary controllers. rotate left rotate right 1 2 Simulates rotating the rotary controller left (counter-clockwise) or right (clockwise). flick left flick right Shift+1 Shift+2 Simulates a fast spin of the rotary controller to the left (counter-clockwise) or right (clockwise). tap x y Simulates a touch event at the specified coordinates. For example, tap 50 100 Day/Night day Shift+N Activates day mode (high brightness, full color). night Control+N Activates night mode (low brightness, high contrast). daynight n Toggles current day/night mode. Troubleshooting DHU showing a blank screen when first connected This is a known issue we are working on. You can work around this by doing the following: Close the DHU. Follow Step 5 described in Run the DHU to stop and restart the Head Unit Server. Start the DHU again. There may be additional permission granting steps on the phone screen. Complete those steps and the DHU might shut down once more. Restart the DHU. Test your app for Android Automotive OS You can use the Android Emulator to test how your driver-optimized app runs on an Android Automotive OS vehicle display. This section describes how to set up an Android Virtual Device (AVD) to test your app. Note: If you are building a media app, you can also test your media session implementation using the Media Controller Test app. Edit your run configurations for media apps Media apps on Automotive OS apps are different than other Android apps. Android Automotive OS interacts with your media app using explicit intents and by sending calls to your media browser service. In order to test your app, verify that your app has no launch activity in its manifest, and then prevent your automotive module from launching with an activity by following these steps: In Android Studio, select Run > Edit Configurations. Select your automotive module from the list of modules in your app. Under Launch Options > Launch, select Nothing. Click Apply, and then click OK. Add system images Before you can create AVDs that match specific manufacturer hardware, you need to add system images for these devices through the Android Studio SDK Manager. Then, when you're creating an AVD, you can download these system images to use with the AVD. Note: The emulator system images for Automotive should be considered as beta, since the emulator doesn't support Bluetooth and the system image has a GNSS related CTS test failure Add system images from car manufacturers Currently, you can add OEM-specific system image for the Polestar 2 & Volvo. Follow these steps to add a system image: In Android Studio, select Tools > SDK Manager. Click the SDK Update Sites tab. Click Add . Enter the following Name and URL, then click OK: Name: Polestar 2 System Image URL: Name: Volvo System Image URL: Click Apply, then click OK. Install generic system images Android Studio also includes a generic system image for Android Automotive OS that you can also use to test your app. This generic system image includes the Google Assistant so that you can test your app's integration. Follow these steps to install the generic system image: In Android Studio, select Tools > SDK Manager. Click the SDK Platforms tab. Click Show Package Details. Under Android 9.0 (Pie), select Automotive Intel x86 Atom System Image. Click Apply, then click OK. Create a car AVD and run the emulator Follow these steps to create an Android Virtual Device (AVD) that represents an Android Automotive OS vehicle and then use that AVD to run the emulator: In Android Studio, select Tools > AVD Manager. Click Create Virtual Device. From the Select Hardware dialog, select Automotive, and then select a device and click Next. Select a system image that targets Automotive, such as Android 9.0 (Automotive), and click Next. Name your AVD and select any other options that you want to customize and then click Finish. From tool window bar, select your Android Automotive OS AVD as your deployment target. Click Run . Note: System images from Google and OEMs contain Google apps including Google Maps, Google Assistant, and the Google Play Store. Use these apps to test your app's behavior, Google Assistant integration, and your app's download and install experience. Test your app for phone screens Follow the steps below to sideload your app onto a phone and test. Step 1. In the Android Auto app, enable Developer mode and unknown sources Install the Android Auto app, available on Google Play, on your phone. If Android Auto is already installed, make sure that you are using the latest version. In the Android Auto app, tap the Hamburger menu, then tap Settings. Scroll to the About section near the bottom and tap Version to display all of the version and permission information. Tap the Version and permission info section 10 times. The Allow development settings dialog appears. Tap OK. Tap the three dots in the top right of the screen to open the overflow menu, then tap Developer Settings. Tap Application Mode and select Developer. The app returns to the list of developer settings. Scroll down until you see Unknown sources and tap the checkbox. Restart Android Auto. Step 2. On your phone, enable USB debugging In Settings > About phone, tap Build number seven times to enable the Developer Options. Go back to Settings > Developer Options, enable USB debugging. Step 3. Install your app onto the phone and test Connect the phone to your machine through USB, so you can install apps directly to it as you develop. Use the adb devices command to ensure that your development computer can detect your device when connected via USB. Once your device is set up and connected via USB, navigate to your SDK's platform-tools/ directory and install the .apk on the device by running the following command: adb -d install path/to/your/app.apk You can now test your app and verify that everything works. Additional testing requirements for media apps If you are testing a media app, then you should also test for the scenarios that are covered in this section (in addition to testing your app on Android Auto, Android Automotive OS, or both). To help keep drivers and passengers safe, users have additional restrictions on how they can interact with apps while driving. For this reason, Android Auto and Android Automotive OS have some MediaBrowserService startup scenarios that your app must be able to handle so that users can continue to enjoy your content while they're on the road. Test your app to make sure it can handle each of the following scenarios: The MediaBrowserService is run before any Activity is opened. The MediaBrowserService is run when no Activity can be shown. The MediaBrowserService is run when the user is not signed in. While testing for these sceanrios, be sure to try the following methods: Force stop the media app, then launch Android Auto or Android Automotive OS. Clear the media app data, then launch Android Auto or Android Automotive OS. Also make sure to set an appropriate error message when necessary. The Media Controller Test app allows you to test the intricacies of media playback on Android and helps verify your media session implementation. To get started with this tool, see Using the media controller test app.