Bottom of Form

[**A step by step guide to React Native on Windows**](http://davidanderson.io/2015/10/18/a-step-by-step-guide-to-react-native-on-windows/)

[18th October, 2015](http://davidanderson.io/2015/10/18/a-step-by-step-guide-to-react-native-on-windows/) [6 Comments](http://davidanderson.io/2015/10/18/a-step-by-step-guide-to-react-native-on-windows/#comments)

The [recent release](https://code.facebook.com/posts/1189117404435352) of React Native with Android support means there is hope for developers running Windows to have a reasonable React Native development environment.  Unfortunately, Facebook still only officially supports OS X but this is easy to work around.

This guide is based on Facebook’s guides to [Getting Started](https://facebook.github.io/react-native/docs/getting-started.html#content) with React Native and [Android Setup](https://facebook.github.io/react-native/docs/android-setup.html) with some tweaks to get things working on Windows.  This is accurate as of October 17, 2015 but is subject to rapid change as new versions of React Native are released.

**Configure your BIOS to enable Virtualization**

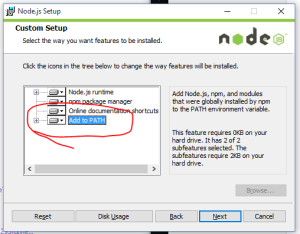
This step is optional but is highly recommended as it will allow the Android emulator to run much faster.  Virtualization is often disabled in the BIOS on Intel-based PCs so you might have to enable it.

I have an HP laptop so I hit ESC on reboot to get to the BIOS and then enabled “Virtualization Technology” as described [here](http://h30434.www3.hp.com/t5/Desktop-Hardware/How-to-Enable-Intel-Virtualization-Technology-vt-x-on-HP/td-p/3198063).  If you have a different system, you’ll have to do a search to figure out how to do it.

[This page](http://www.howtogeek.com/213795/how-to-enable-intel-vt-x-in-your-computers-bios-or-uefi-firmware/) on How-To Geek might help if you run into any issues with getting virtualization to work.

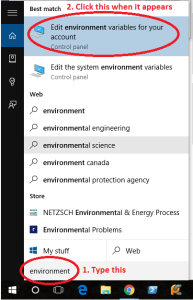
**Download & Install**

I installed 64-bit versions when available and have not had any issues because of this.

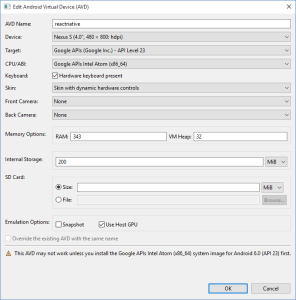
* [Node.js](https://nodejs.org/) v4.0 or higher.[](http://davidanderson.io/wp-content/uploads/2015/10/Capture.png)
  + Facebook states that v4.0 or higher is required.  I am on 4.2.1.  I had a pre-4.0 version installed already and chaos ensued because of it so if you already have an earlier version of Node.js install, you will almost certainly need to update it.
  + Make sure that you “Add to PATH” selected in the installer to make your life easier later.
* [JDK](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
* [Android SDK Tools Only](http://developer.android.com/sdk/index.html#Other)
  + If you want or already have Android Studio, I’m sure the SDK tools are installed with Studio but you’ll have to figure out where.
  + By default, the installer wanted to place this at C:\Users\<My User Name>\AppData\Local\Android\android-sdk.  I changed this to C:\devtools\Android\android-sdk just to make it easier to find later.
  + SDK Manager should open automatically after the install.  If it doesn’t, run SDK Manager from the Windows Start menu.  Install the following packages.
    - Android SDK Build-tools version 23.0.1
    - Android 6.0 (API 23)
    - Android Support Repository
    - Intel x86 Atom System Image (for Android 5.1.1 – API 22)
    - Intel x86 Emulator Accelerator (HAXM installer)
* [Cygwin](https://www.cygwin.com/)
  + This is required because of various hardcoded references to sh in the React Native packager module.  If you don’t install this, you’ll get obscure ENOENT errors later on.  You’ll need to also add the path to the Cygwin bin directory to the Path environment variable as described below.
  + The default install options should be fine.
  + Alternatively, you could edit the React Native code to replace the sh references as discussed [here](https://gist.github.com/davidgilbertson/9bee68548037fe00f2a8) but I find it simpler to just install Cygwin.  I figure it’s better to set up Cygwin once than to have to patch the React Native files every time I update.

**Configure Environment Variables**

The React Native tools need to know how to find the Android SDK and Cygwin sh.

1. [](http://davidanderson.io/wp-content/uploads/2015/10/environment-variables.png)Open the Environment Variables dialog.  Getting here varies a bit between versions of Windows.  In recent versions of Windows, you’ll want to open up the System control panel and go to “Advanced system settings” and then select Environment Variables.  In Windows 10, you can get there pretty quickly by just hitting the Windows key and typing “environment” like in the screenshot.
2. Decide if you care about user versus system level variables.  Are you the only person who uses this system?  Then go with user.  Do you share this system with other devs or use multiple accounts for whatever reason?  Go with system.
3. Add a new variable called ANDROID\_HOME and point it at wherever you installed android-sdk.  As I mentioned before, I went with c:\devtools\Android\android-sdk.
4. Add the Cygwin bin path to the end of the existing Path variable.  Paths are separated by a semicolon (;) so you’ll end up adding something like “;C:\cygwin64\bin” to the end of the existing string.  The End key on your keyboard is your friend at this point to get you to the end of the Path string quickly.  Since you just installed node, you’ll likely see npm or nodejs appear as the last path element before adding Cygwin and that’s a good thing.
5. Make sure the environment variables are set properly by opening a Command Prompt and running the following commands.
   * **If you have a Command Prompt already open, close it and open a new one.  Otherwise the new environment variables won’t be available.**
   * > set ANDROID\_HOME
   * > sh --version
   * You can see my output [here](http://davidanderson.io/wp-content/uploads/2015/10/envcheck.png).

**Get the Android emulator configured**

1. In File Explorer, browse to <install location>\android-sdk\extras\intel\Hardware\_Accelerated\_Execution\_Manager
2. Run intelhaxm-android.exe.  Remember that step earlier where you enabled virtualization in BIOS?  This adds support for that to the Android emulator.  This is optional but highly recommended.
3. Run AVD Manager.  The easiest way to do this is to hit the Start key and type “AVD Manager”.
4. Click “Create…” to create a new Android emulator image.
5. These are the settings I used for my image.  [](http://davidanderson.io/wp-content/uploads/2015/10/emulator.png)
   1. I selected a relatively low resolution so it would fit on my screen better and have lower performance requirements.
   2. I selected Intel Atom for the CPU.  The virtualization/haxm stuff from earlier is necessary to get this working.  Feel free to select ARM if you are cool with the emulator running really slow.
   3. I selected “Skin with dynamic hardware controls” for Skin but any skin other than “No skin” will likely work for you.  If you don’t select a skin, you won’t get the MENU button and so you won’t be able to reload the app when you make changes.

**Install the react-native command line tool**

You only need to do this once per system.  This will add the react-native command line tool to globally (-g) to your Node.js install.

Open a Windows Command Prompt and type the following (it doesn’t matter where you are).

> npm install -g react-native-cli

If npm isn’t found, it probably means you didn’t listen back in the install step for node.js when I said make sure “Add to PATH” is checked.

**Create a new React Native project**

You do this once per React Native project to kick it off.  I keep development projects under d:\Dev so this is what I’d do to kick off a new AwesomeProject.  Feel free to change your paths as desired.

> cd d:

> cd d:\Dev

d:\Dev> react-native init AwesomeProject

**Run the app**

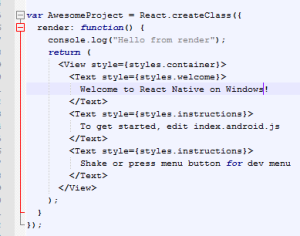
You’ll want to follow these steps every time you want to work with your app.

1. Run AVD Manager and kick off your emulator image that you created above by selecting it and clicking “Start…”.
2. Open a command promppt and change to your project directory.
   * d:\Dev> cd AwesomeProject
3. Use “react-native run-android” to kick off the Android app in the emulator.  This took a while for me to run the first time because it had to download Gradle and associated dependencies and then it had to build the app.  Once it runs, you’ll get the following error but that is okay because you aren’t running the dev server yet.
   * d:\Dev\AwesomeProject> react-native run-android
4. Start the react-native development server.  Whenever you wish to work on your project, you’ll want to open up a command prompt and run this and leave it open.  This will kick off a web server at <http://localhost:8081/>.
   * d:\Dev\AwesomeProject> react-native start
5. Click that beckoning “RELOAD JS” button and now you should see your React Native app finally show up!

[](http://davidanderson.io/wp-content/uploads/2015/10/NewApp.png)

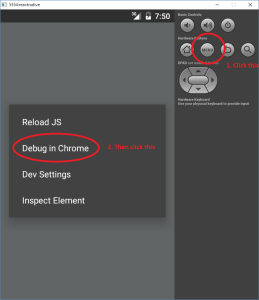
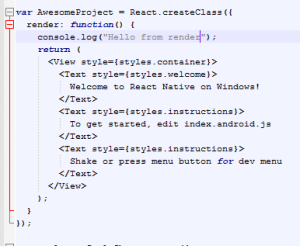
**Make some changes**

Get used to this as you will be using this often.  This is the cycle you will use to make changes and then see them in the emulator.

1. Open up AwesomeProject\index.android.js.
2. Make some changes and save it in your editor.  I changed “Welcome to React Native!” to “Welcome to React Native on Windows!”.[](http://davidanderson.io/wp-content/uploads/2015/10/Welcome.png)
3. Note that the react-native start process will react to this to let you know it is transforming.
4. Hit the hardware MENU button and select Reload JS to reload your app.  You’ll have to do this in the future every time you make changes.

**Get debugging to work**

Debugging is another area where Facebook assumes you are running Mac OS X but we can also easily work around this.

1. Open up a Chrome tab and browser to <http://localhost:8081/debugger-ui>.  You should see some messages about debugging.
2. Open up Chrome Developer Tools in that tab.  I usually just hit F12 to do this but you can dig through the menus if you need to.
3. In the React debug menu on the emulator, enable debugging. [](http://davidanderson.io/wp-content/uploads/2015/10/Debug.png)
4. Play around with the debugger to make sure it is working.  Add some console.log statements in the source code or set some breakpoints in the Chrome Developer Tools.  I added a console.log line like the following.[](http://davidanderson.io/wp-content/uploads/2015/10/consolelog.png)
5. Hit MENU and Reload JS like described in the **Make some changes** section above.
6. Observe that your console.log message has appears in Chrome.  Nice!  [](http://davidanderson.io/wp-content/uploads/2015/10/DebugTools.png)

**Help me keep this post up to date**

You should be good to go now but React Native is new and rapidly changing so this guide is likely to be out of date as soon as I publish it.  Please don’t hesitate to leave comments letting me know when I need to make updates.

Thanks for reading!

Posted in [React Native](http://davidanderson.io/category/react-native/) | Tags: [Android](http://davidanderson.io/tag/android/), [iPad](http://davidanderson.io/tag/ipad/), [iPhone](http://davidanderson.io/tag/iphone/), [React](http://davidanderson.io/tag/react/), [React Native](http://davidanderson.io/tag/react-native/) |