Turning to Mobile Device Emulators

Whether you are developing a website or a web application, considering the way it will display and handle on different devices is an issue that you must deal with.

If you’ve never used an emulator before, you can think of it as a virtual device or program that performs similarly to the actual device. Many emulators allow you to experience how the device will behave from start-up to shut down. Other emulators are application wrappers that allow you to use certain applications of the mobile device. When dealing with a mobile site, emulators give you an idea of how the site will be displayed and how the user will interact with what they are presented.

Using an emulator for mobile devices is especially important as mobile devices translate some events differently than desktop browsers. Another reason that you should use an emulator is to understand the wide array of screen resolutions available to mobile devices. While it’s always good to test the theory of responsive design and site layout, until you have used either an emulator or an actual device it may be difficult to fully appreciate how your design and site layout handles on a tablet device instead of on a smartphone.

From a cost perspective, device emulators offer a great starting point for small shops and individual developers.

Emulators are generally free or only require a short developer application to be filled out prior to download. This makes it easy for you to get started testing and creating websites and web applications that run as smoothly as possible on the actual device.

While using an emulator may sound like a developer’s dream come true, there are actually several potential problems that you need to keep in mind. The biggest issue stems from emulators not giving a completely accurate portrayal of how the device works.

Think about how most mobile devices work. Most tablets or smartphones use a touchscreen. When you use an emulator you are presented with a screen or image of the device or application. The problem then becomes manipulating the device. While the mouse does a fair job at acting out taps, some gestures or swipes are impossible to perform. Many emulators have found some assisted ways around this, adding some features that allow you to hold down a key on your keyboard while click-and-dragging to create gestures like pinching. Other emulators include special keyboard and mouse modes to help you navigate around them.

Another problem you run into when using emulators is speed. The Android Emulator for example is based on QEMU (the Quick EMUlator) which emulates an ARM processor (so it can run the actual code of your application) on your computer which has a x86 processor. I surely don’t need to explain why emulating a processor by a software isn’t a very good idea if you want something reactive and usable. Furthermore, other hardware features are also emulated by the software. All those things combined plus the fact that smartphones have some very powerful chipsets lead to a very slow emulator.

Another problem is application support. Many emulators come with a limited feature set and limited ability to add new programs. Since most emulators provide a browser this will not be too much of an inconvenience for testing websites or web applications, but it can be a little bit of a shock if you are not expecting it.

Following is a quick summary of the pros and cons to using device emulators:

Pros of using device emulators:

* Low cost alternative for development
* Properly displays device resolution
* Can be used to test responsive layouts between devices
* Easily test websites and applications inside native device browsers

Cons of using device emulators:

* Device input is emulated and may not be completely accurate.
* Device operating speed may be incorrect.
* Some emulators may not run on your current OS.
* Lack of full device feature support.

Now that you know why you should use emulators, and some of the common pitfalls that come with them, let’s look at the process of acquiring them and running them on your system.

Finding Emulators for Testing

Getting your hands on an emulator is usually a straightforward process. Visiting the developer section of almost any mobile OS website usually results in a link to sign up for development tools or the Software Development Kit (SDK). Some sites make getting the emulator easy, while others require you to sign up as a developer. Once you have jumped through the correct hoops and filled out the required forms you are given the link to the files that contain the emulator.

Android

The Android emulator lets you prototype, develop and test Android applications without using a physical device. The Android emulator mimics all of the hardware and software features of a typical mobile device, except that it cannot place actual phone calls. It provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application. It also provides a screen in which your application is displayed, together with any other active Android applications.

The emulator for Android is part of the Android SDK, which is available for download at http://developer.android.com/sdk/index.html. The download does not require any signup or registration and is available for Windows, OSX, and Linux systems. Once you have downloaded and either unpacked or installed the Android SDK, you can use the Android Virtual Device (AVD) Manager to create emulators for different Android devices.

To use the emulator, you first must create one or more AVDs. An AVD is a device configuration for the Android emulator that allows you to model different configurations of Android-powered devices.

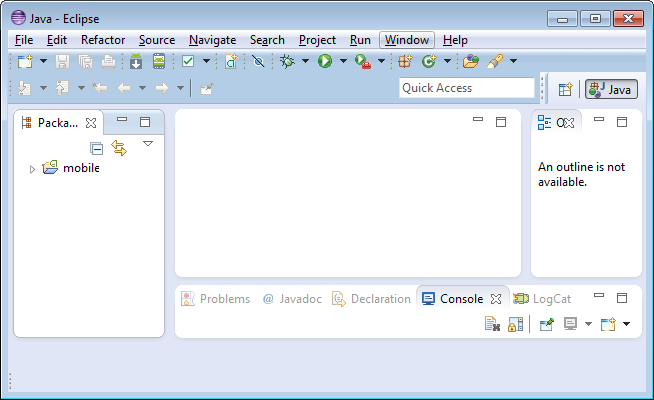
In each configuration, you specify an Android platform to run in the emulator and the set of hardware options and emulator skin you want to use. Then, when you launch the emulator, you specify the AVD configuration that you want to load.

You can launch the emulator as a standalone application from a command line, or you can run it from within your Eclipse development environment. In either case, you specify the AVD configuration to load and any start-up options you want to use.

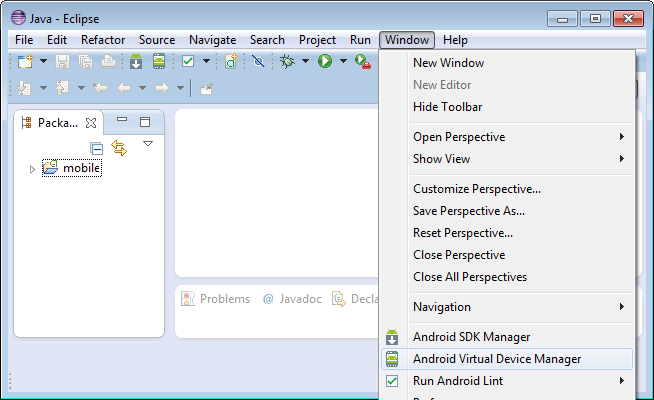
You can run your application on a single instance of the emulator or, depending on your needs, you can start multiple emulator instances and run your application in more than one emulated device.

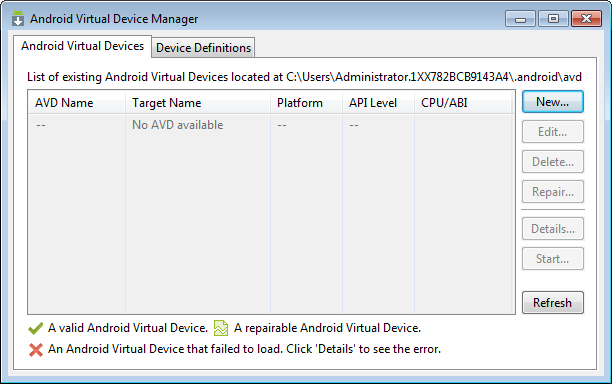
Lets get started…..

1. Start **eclipse**, if prompted click **Run**.

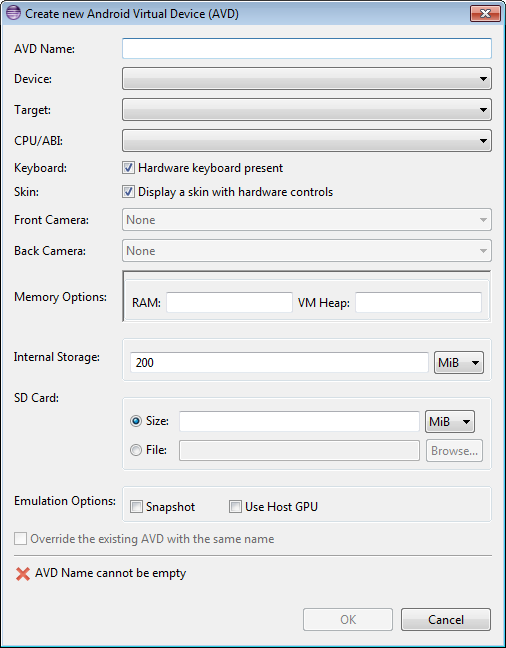


1. Click on the **Window** menu.
2. Click the **Android** **Virtual** **Device** **Manager** (AVD Manager) link.



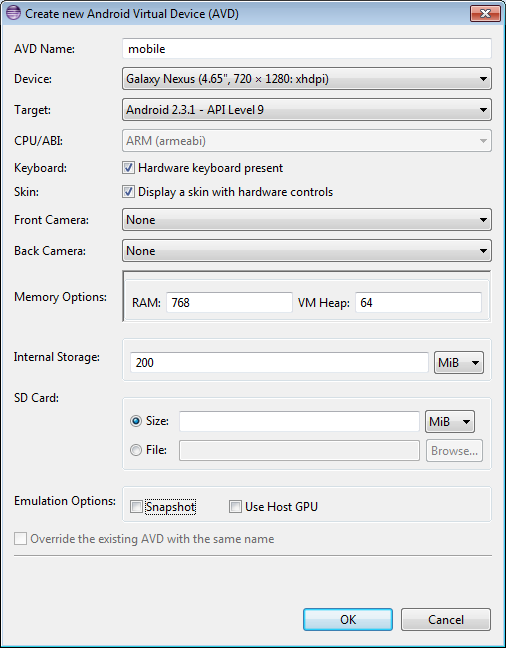


1. Click on the **New** button. From there you are presented with a virtual device creation screen.

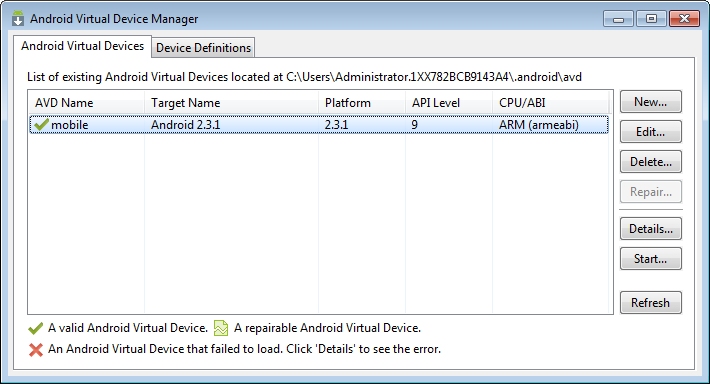


1. To create the **AVD** you need:

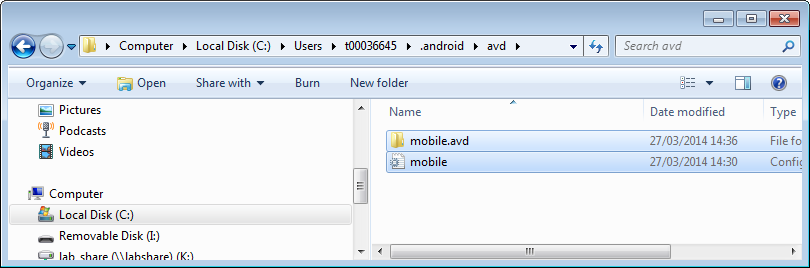
* name it
* select a device,
* select a version or target version of the OS,
* select the amount of memory available,



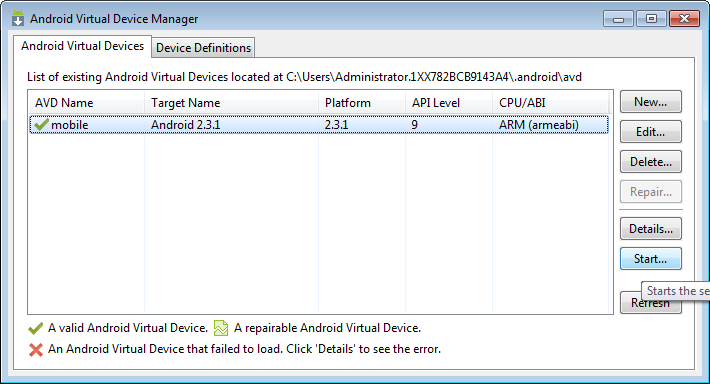
1. Once you have filled out these options you can click “**OK**” to finish creating your AVD.

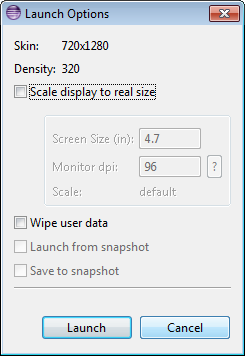


1. For the **labs** **only**, the location of the AVD is wrong. As you can see above it is created in the Admin folder, this may be different on your machine. We need to move the contents of the AVD folder to:

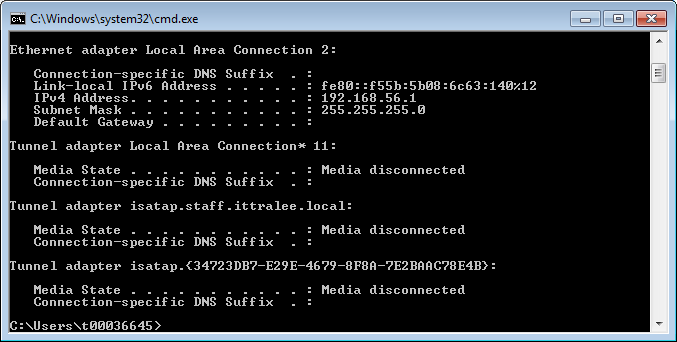


1. Again the t number will be different for each user.
2. After you have created your AVD and moved it to the correct location, you can then start or boot the device by selecting it and clicking the **Start** button back in the AVD Manager.



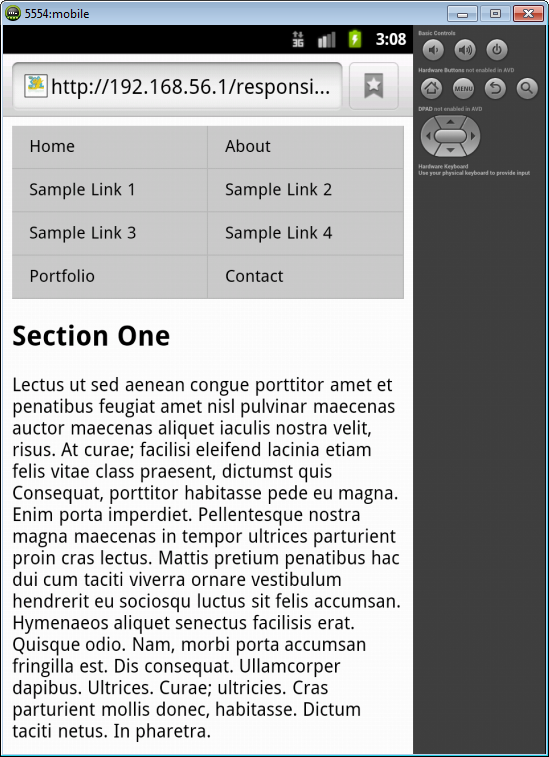


1. Click **Launch**.
2. When you do a new window appears, and the device begins the boot sequence. This may take a while.
3. While you are waiting for the emulator to open, launch Apache and also locate the ip address of your machine.
   * Type **cmd** from the start prompt to launch the command prompt
   * Type **ipconfig** at the dos prompt



* Locate the IP address and note it.
* Close this command window.

1. Back to the emulator… When finished you are presented with the lock screen of whatever version of Android you picked to boot.
2. Once you have unlocked the device you are presented with the home screen. From there you have an icon that gets you the built-in web browser.
3. Once you have opened the browser, you type the address of the file you want to open. If you are trying to connect to a server running on the computer that you are running the AVD from, then you need to reference the site by **IP** of the computer instead of by using **localhost**.



1. When the site loads you can see how it responds just as it would on an actual device. When you want to interact with the site you have to use your mouse and use a click and drag to simulate swiping. By clicking your mouse you initiate a tap event.

Other options

There are so many emulators out there we couldn’t possibly look at them all. You might want to just look at 1 other for a comparison. You can download the mobile emulator from the Opera website. It is a very light simple emulator that will suit your needs for this project perfectly (30 seconds to get up and going) [**http://www.opera.com/developer/mobile-emulator**](http://www.opera.com/developer/mobile-emulator). As with the last emulator you need to use the IP address as opposed to the localhost method.

Some Terms you may come across

An **IDE** is a development environment - a supped-up text editor with additional support for developing (such as forms designers, resource editors, etc.), compiling and debugging applications. e.g. Eclipse, Visual Studio.

A **Library** is a chunk of code that you can call from your own code, to help you do things more quickly/easily. Basically code created by other folks so that you don't have to reinvent the wheel. For example, a Bitmap Processing library will provide facilities for loading and manipulating bitmap images, saving you having to write all that code for yourself. A library is a collection of functions, often grouped into multiple program files but packaged into a single archive file. A library generally has a key role but does all of its work behind the scenes, it doesn't have a GUI.

An **API** (application programming interface) is a term meaning the functions/methods in a library that you can call to ask it to do things for you - the interface to the library. Put another way it’s the library publisher's documentation - this is how you should use my library.

An **SDK** (software development kit) is a library (often with extra tool applications, data files and sample code) that aid you in developing code that uses a particular system. Think of like a McDonald's Happy Meal. You have everything you need (and don't need) boxed neatly: main course, drink, dessert and a bonus toy. An SDK is a bunch of different software components assembled into a package, such that they're "ready-for-action" right out of the box. It often includes multiple libraries and can but may not necessarily include plugins, API documentation, even an IDE itself. (e.g. iOS Development Kit).

A **framework** is a big library that provides many services (rather than perhaps only one focussed ability as most libraries do). For example, .NET provides an application framework - it provides most (if not all) of the services you need to write a vast range of applications - so one "library" provides support for pretty much everything you need to do.