Getting Started with NativeScript

Getting Started with NativeScript concisely shows you NativeScript’s built-in framework that allows you to rapidly develop a fully functional compiled cross-platform application in just a few chapters. It starts by laying the foundations of NativeScript and working through the fundamentals to create a basic shell of the application. Moving on, you’ll see how to build a fully-fledged application step by step. We’ll show you how to use plugins and how to communicate with the native OS libraries easily so that you can customize your application as if your app was created in Java or Objective C. We then deal with the issues that arise from being cross platform and compensate for the different screen sizes, screen resolutions, and device abilities. Finally, we progress to testing and deploying your app.

So, whether you have already developed multiple applications or none, this book will help you to develop your next application in a cross-platform framework quickly, saving you a massive amount of time and money.

Who this book is written for
If you are a JavaScript developer and want to build cross-platform applications, then this book is the right one for you!

What you will learn from this book
- Install and compile your application in NativeScript
- Get important information on the NativeScript project structure
- Develop and style your screens for multiple platforms
- Create a full-fledged cross-platform communication application
- Import and use several third-party components
- Simplify and deal with device resolution and cross-platform issues
- Test and deploy your application

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Introduction to NativeScript'
- A synopsis of the book’s content
- More information on Getting Started with NativeScript
Nathanael J. Anderson has been developing software for over 20 years in a wide range of industries, including areas of games, time management, imaging, service, printing, accounting, land management, security, web, and even (believe it or not) some successful government projects. He is currently a contract developer for master technology and can create a solution for several types of applications (native, web, mobile, and hybrid) running on any operating system.

As a senior developer engineer, he can work, tune, and secure everything from your backend servers to the final destination of the data on your desktop or mobile devices. By understanding the entire infrastructure, including the real and virtualized hardware, he can completely eliminate different types of issues in all parts of a framework.

Currently, he has multiple highly rated cross-platform plugins for NativeScript, and he works heavily in the NativeScript community by providing things such as bleeding edge build servers to build knightly code. He has also provided multiple patches and features to the main NativeScript project.
Welcome to Getting Started with NativeScript. In this book, we are going to go on an awesome journey of building cross-platform applications in JavaScript. We will cover everything from how NativeScript works, to how to test, debug, and finally deploy your application. Over the course of this book, we are going to explore how to build a full-featured, cross-platform messaging platform. The application will work the same on all NativeScript-supported platforms. With your ability to develop in JavaScript and the insights provided in this book, you will be releasing your own cool applications in no time.

What this book covers

Chapter 1, Introduction to NativeScript, will teach you about NativeScript and how to install and build your first NativeScript application.

Chapter 2, The Project Structure, provides an overview of what all the different files and folders are used for, and we will build and switch to a second screen for our application here.

Chapter 3, Declarative UI, Styling, and Events, works through how to create screens using the Declarative UI, style them and then how to create and respond to events.

Chapter 4, Building a Featured Application, helps you to actually sit down and build a full-featured, cross-device messaging application using just the standard NativeScript components.

Chapter 5, Installing Third-Party Components, delves into how to install several different types of third-party components to enhance our cool communication application.
Preface

Chapter 6, *Platform Differences*, looks at how to deal with the differences between iOS and Android and the differences in the actual physical characteristics of the devices even on the same platform.

Chapter 7, *Testing and Deploying Your App*, looks at how to use several different types of testing frameworks, how to debug your application, and finally, how to actually deploy your application.
In this chapter, we are going to introduce you to Telerik's **NativeScript**, and discuss how NativeScript is totally unique in the cross-platform mobile device market, and how it is radically revolutionary for mobile **JavaScript** development. We will also walk you through the prerequisites of how to install the NativeScript command-line tool. Once the NativeScript tool is installed, we will walk you through the basic usage of the tool and briefly describe the most common parameters. Finally, we wrap up the chapter by creating and running our first NativeScript application.

In this chapter, we will be covering the following topics:

- What is NativeScript?
- NativeScript and TypeScript
- Common modules
- Installing NativeScript
- The NativeScript command line
- Creating your first application

**NativeScript**

If you are looking at this book, maybe you want to know why you should use NativeScript and what sets it apart from the crowded competition. Why shouldn't you use any of the other cross-platform tools? Let us dig in, and I'll explain why NativeScript is the answer to the best way of executing cross-platform mobile development.
Telerik's NativeScript

Telerik's NativeScript is a fairly new open source development system for creating cross-platform mobile applications almost entirely in JavaScript, with some optional CSS and XML to simplify developing the display layout. You can find the main location of each of the different projects that make up NativeScript at https://github.com/NativeScript. Even though it is new to the market, it is already fully compatible on Apple's iOS and Google's Android. In addition, Microsoft's Windows 10 Mobile is currently in development by Telerik, and others are working on Firefox Mobile. NativeScript uses the V8 engine (as used by Google Chrome and node.js) on Android and Apple's JavaScriptCore engine on iOS devices.

Other competitors

Now, there are several other competing JavaScript development systems for mobile devices. Some of these competitors have been established for a while. Other development systems may have large companies working on them. But neither of those will make any of the other tools the best choice. What makes NativeScript stand out from the crowd of other JavaScript environments is its unique design. Every other JavaScript environment requires a special bridge, or a compiled extension of some sort, which basically exposes some part of the native functionality of the host operating system to your JavaScript code. A lot of them are actually just web browsers wrapped in an application shell, so all the work you do is actually in a browser. If you decide you want Bluetooth on your iOS phone in one of the other products, you have to find someone who has made the iOS Bluetooth bridge or extension module in some other non-JavaScript language. In a lot of cases, you will even have to compile the module, and then you will still be hoping that the module has all the functionality you need.

NativeScript uniqueness

NativeScript is unique because it allows you to access the native elements of the host platform via JavaScript. In NativeScript, you can still see if someone has written a convenient JavaScript library to access the Bluetooth API. If so, since you understand JavaScript, you can easily make any changes you need. If not, then you can make your own JavaScript module to access all the host platforms of Bluetooth API. NativeScript is not a wrapper around a web view; it literally allows your JavaScript to work directly with the host platform APIs.
For example, to find out if a file exists, we can just call the native Android method in JavaScript:

```javascript
var javaFile = new java.io.File('/some/file/name.ext');
var exists = javaFile.exists();
```

Or the native iOS Objective C code in JavaScript:

```javascript
var fileManager = NSFileManager.defaultManager();
var exists = fileManager.fileExistsAtPath('/some/file/name.ext');
```

Since NativeScript allows you access to the full operating system libraries and third-party libraries from your JavaScript code, you do not need to wait for someone else to create a wrapper or bridge to talk to any part of any iOS or Android API. You can now fully use any of the APIs as a first-class citizen, which even includes using any new APIs when they are first released.

NativeScript allows you to be a fully first-class citizen; you have FULL access to the devices' entire released API from JavaScript. So anything that you can do in Android Java or iOS Objective C, you can now do directly in JavaScript.

NativeScript is easy

Now, before you get worried about having to know both iOS and Android to make your application, NativeScript has that covered. To simplify things, NativeScript already has a wide number of components, or modules, that wrap the most common things a developer will need, which are called the NativeScript common core modules. So, instead of having to write any Android or iOS specific code like I did above to see if a file exists, you can just write the following code:

```javascript
var fs = require('file-system');
var exists = fs.File.exists(path);
```

The NativeScript filesystem module has each of the native platforms’ API wrapped up so all you have to do is write to a common interface. But when you need to do something outside of the built-in modules and components, NativeScript is the only environment that allows you to easily have full access to everything the device offers right from JavaScript.
NativeScript and TypeScript

We are going to explain how parts of NativeScript are developed in TypeScript and what that means for you in terms of developing your awesome application using NativeScript.

What is TypeScript?

In 2012, Microsoft released an interesting language called TypeScript. This language is fully open sourced because the company felt it was something the JavaScript community needed. It is, in a nutshell, a superset of JavaScript with types and several other additional language features. If you write any code in TypeScript, you then have to use the TypeScript transpiler to convert the code from TypeScript back into JavaScript. One of the primary reasons people use TypeScript over regular JavaScript is that TypeScript offers the ability to do static type checking at the point it converts the code to JavaScript. So, you don't have a runtime hit, and you don't have to do a lot of runtime parameter checks if the code is all in TypeScript. This feature alone eliminates a type of bug that is very easy to access in JavaScript. In addition to static typing, it has several class/object type features that make inheritance and class definition considerably simpler and safer.

Types possess the ability to add markup to code denoting the type expected:

```javascript
private validateMe(name: string, password: string): boolean {
};
```

The string and boolean are declarations telling what the exact parameter types and expected return type are. This allows the transpiler to verify that the code matches the static types during the building stage. Transpiler is a shortened term from translation compiler used to mean the code is converted from one language to another language. So, in this case, we are translating the code from TypeScript into JavaScript.

TypeScript's use in NativeScript

The NativeScript command-line utility, common modules, and components are all written in TypeScript. TypeScript is then transpiled to JavaScript before it is distributed for all us developers to download, install, and use. So, unless you are actually pulling the open source code from the NativeScript repositories, then all the code you will see is in JavaScript.
Fortunately for us, the majority of the differences between TypeScript and JavaScript are fairly minor, so the code transpiled to JavaScript in almost all cases is very close to the original TypeScript, which still makes it very readable.

Telerik just released a brand new module in v1.5.0 that will allow TypeScript to now be a first-class citizen in the development of your application. If you don't use this module, then you have to manually transpile all your TypeScript code each time before you build an application. After you execute a `nativescript install typescript` command, when the NativeScript command does anything with your code, it will automatically transpile all your TypeScript code first. This makes your development a much more streamlined process.

Choosing a development language

Since the final output of all the code must be JavaScript, you are able to write any of your applications or modules in TypeScript, CoffeeScript, or any other language that can be transpiled into JavaScript. This book is going to focus on doing everything in JavaScript as this is what the final code output must be for all the devices, and it is the common language that binds everything together.

Common modules

Common modules were created to solve the issue of JavaScript files polluting the global namespace with variables and functions that another JavaScript file could easily overwrite accidentally. JavaScript allows you to redeclare or monkey patch your functions on a whim, which is part of what makes it so powerful. However, with that much power comes the ability to very easily shoot yourself in both feet simultaneously. Then, you are left scratching your head why you just lost both feet. To attempt to solve the issue of one included file function or variable being overwritten by another include file, developers came up with several techniques that evolved into the common module formats we use today. There are three standards available for you to use: the CommonJS module format, which is what node.js popularized; the AMD module format, which was designed for the asynchronous resolution of JavaScript files in a browser environment; and the brand new ECMAScript 6 module format, which, when finally released, should become the de facto module format. All three of them wrap the source code so that none of the code in a module by default can interfere with the global namespace. NativeScript follows the CommonJS module format where you use `exports` and `module.exports` to tell what parts of the code in the module you want to expose to outside parties. When you see `var coolModule = require('cool-module');`, this is the syntax that the CommonJS module format uses to load a module.
Installing NativeScript

We are going to cover what you'll need for installation and development. Then, we will discuss how to install the NativeScript command, which you will use for anything relating to your NativeScript project.

Prerequisites

To get started on our journey of using NativeScript, we first must have several tools installed. The primary tool you will use for virtually everything is the nativescript command, or you can alternatively use the shorter alias of tns (short for Telerik NativeScript).

node.js

For the nativescript command to be installed and to make it work, you must first install node.js, which can be downloaded and installed from https://nodejs.org/. The nativescript command is also written in TypeScript and already pre-converted to JavaScript before you even download it. It uses node.js and several other common modules to perform all its work.

iOS

If you are planning on distributing your app on any iOS devices, you will need to have access to an Apple Macintosh platform. This is a requirement by Apple as they only allow you to compile and upload the app to the app store from a Macintosh. The Macintosh must have the Xcode and Xcode command-line tools installed. It is also highly recommended that you have an iPhone 4s or better to test at least the final application before you distribute it. The iOS emulator is not very accurate and because it gives you an idea of how your app will look, an actual device will accurately show you what your customers will see. The specific install instructions for iOS can be found at http://docs.nativescript.org/setup/ns-cli-setup/ns-setup-os-x.html.
Android

Android is a lot easier because you can execute development for Android on Linux, OSX, and Windows. For Android, you need to install Java JDK and then the Android SDK and tools. I also highly recommend that you install an optional third-party Android emulator rather than use the stock Android emulator. The third-party emulators are considerably faster than both the stock Android emulator and the add-on Intel HAXM emulator. The stock emulator and Intel HAXM can be downloaded via the Android SDK Manager. The instructions on the NativeScript site can walk you through installing all the tools that you will need for building and testing your application. The specific instructions for each of the platforms are located at http://docs.nativescript.org/setup/quick-setup#the-nativescript-cli.

Installation

Once you have node.js installed, you just need to do a simple npm install –g nativescript at a terminal shell (or known on Windows as a command prompt window). And then npm will download and install the required NativeScript code and command-line tool. It will install it globally so that it can be used from any project you are working on.

With the command path on Linux and Macintosh, the global commands are normally placed in /usr/local/bin, which is typically already in your path, so the nativescript command should work right away. On Windows, unfortunately, you might have to add the path where npm installs the program globally. Type nativescript and see if it runs. If it fails to run, then you need to add npm’s global directory to your path. Type npm config get prefix to get the current global directory. Type systempropertiesadvanced, then click the Environment Variables button, then click on PATH, and finally, click the Edit button. Then, you can type a semicolon and add in the directory path to the all the npm global commands like the NativeScript command. So, your path might look something like this: C:\windows;C:\windows\system;C:\program files (x86)\nodejs.

Once the NativeScript command and required support tools are installed, you are all ready to begin developing NativeScript applications.
Installation help

Sometimes, getting NativeScript installed properly can be difficult because of all the differences between computers. Here are a couple places you can go to get help if you need it: https://groups.google.com/forum/#!forum/nativescript and https://plus.google.com/communities/117408587889337015711.

The NativeScript command line

Now, before we get to the creating a project part, let us have an overview of the commands available from the new nativescript command you just installed. If you forget any of these, you can easily type nativescript alone without any parameters to see a help screen at a console window, or type nativescript /? for a help screen in your browser.

NativeScript commands

These are just some of the most commonly used valid commands for the nativescript command. Several of these we will be covering as we progress in the book.

<table>
<thead>
<tr>
<th>Command line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nativescript --version</td>
<td>This returns the version of the nativescript command. If you are running an older version, then you can use npm to upgrade your nativescript command like this: npm install -g nativescript.</td>
</tr>
<tr>
<td>nativescript create &lt;your project name&gt;</td>
<td>This creates a brand new project.</td>
</tr>
<tr>
<td>nativescript platform add &lt;platform&gt;</td>
<td>This adds a target platform to your project.</td>
</tr>
<tr>
<td>nativescript platform list</td>
<td>This shows you what platforms you are currently targeting.</td>
</tr>
<tr>
<td>Command line</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nativescript platform remove &lt;platform&gt;</td>
<td>This command is normally not needed, but if you are messing with the platform directory and totally mess up your platform, you can remove and then add it back. Please note this deletes the entire platform directory, so if you have any specific customizations to your Android manifest or iOS Xcode project file, you should back it up before running the remove command.</td>
</tr>
<tr>
<td>nativescript platform update &lt;platform&gt;</td>
<td>This is actually a pretty important command. NativeScript is still a very active project under a lot of development. This command upgrades your platform code to the latest version, which typically eliminates bugs and adds lots of new features. Please note this should also be done with an upgrade of the common JavaScript libraries as most of the time, they also are typically in sync with each other.</td>
</tr>
<tr>
<td>nativescript build &lt;platform&gt;</td>
<td>This builds the application for that platform.</td>
</tr>
<tr>
<td>nativescript deploy &lt;platform&gt;</td>
<td>This builds and deploys the application to a physical or virtual device for that platform.</td>
</tr>
<tr>
<td>nativescript emulate &lt;platform&gt;</td>
<td>This builds and deploys the application to an emulator.</td>
</tr>
<tr>
<td>nativescript run &lt;platform&gt;</td>
<td>This builds, deploys, and starts the application on a physical device or an emulator. This is the command you will use the majority of the time to run your application and check out the changes.</td>
</tr>
<tr>
<td>nativescript debug &lt;platform&gt;</td>
<td>This builds, deploys, and then starts the application on a physical device or an emulator in debug mode. This is probably the second most used command.</td>
</tr>
<tr>
<td>nativescript plugin add &lt;plugin&gt;</td>
<td>This allows you to add a third-party plugin or component. These plugins typically include JavaScript based code, but occasionally, they might also contain an actual compiled Java or ObjectiveC library.</td>
</tr>
<tr>
<td>Command line</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nativescript livesync - -watch</td>
<td>This allows you to have the nativescript command watch for changes and automatically push them to the device. This is probably the third most used command if you are not using a better third-party LiveSync system. We call it LiveSync because it automatically syncs all your changes directly to the device in real time, without you having to rebuild the application. This speeds up your development of an application drastically.</td>
</tr>
<tr>
<td>nativescript doctor</td>
<td>This allows you to run some diagnostic checks on your environment if the nativescript command does not appear to be working.</td>
</tr>
<tr>
<td>nativescript install</td>
<td>This will (re)install any dependencies listed in your package.json file. The package.json file is typically modified by the plugin or library add function, so this is used typically to reinstall the plugins or libraries in the event you add a new platform or reinstall one.</td>
</tr>
<tr>
<td>nativescript test [ init</td>
<td>&lt;platform&gt; ]</td>
</tr>
</tbody>
</table>

Now that we have described some of the commands, let's use them to create your first mobile app via the nativescript command tool.

**Creating your first application**

I am going to walk you through how to create an application as it currently requires a couple steps that you need to do from a terminal shell. Then, we will show you how it looks.
Creating the application in easy steps

Creating a project is actually a fairly simple process; we are going to start by creating the application we will develop throughout this book.

1. Make sure you are at a terminal shell and then type:
   ```bash
   nativescript create crossCommunicator
   ```

2. This will create `crossCommunicator` as a subdirectory of the current directory you are running the `nativescript` command in. This new project directory will have all the required files for developing your project.

3. Next, you will need to switch to that new `crossCommunicator` directory that it created for you by typing:
   ```bash
   cd crossCommunicator
   ```

4. Then, type:
   ```bash
   nativescript platform add android
   ```
   And/or:
   ```bash
   nativescript platform add ios
   ```

5. To add each of the target environment(s), you will be compiling your app from this machine:

   ![Image of command output]

   If everything worked properly, you should see something like the preceding image results.
Now, I personally do all my development on the Windows platform using JetBrain's wonderful cross-platform PHPStorm development environment. You are free to use any JavaScript editor you are comfortable with. Since I am using the Windows platform, I do the majority of my testing and debugging on the Android platform and emulator. The Android emulator is basically a full Android in your computer, so it behaves like an actual device. However, since I am also targeting iOS, every so often I also copy or sync the code from my Windows app folder to a Macintosh and then verify that everything looks good and works as expected on the iOS emulator and eventually an iOS device. The very first time when I copy my code over to a Macintosh, I use the nativescript platform add ios command as I need to add the iOS target on the Macintosh.

First time requirements
The very first time you create a project or add a platform, you do need an active Internet connection so the nativescript tool can download the current version of the runtime and platform libraries that it uses. In any future executions, it will first try to download the newest version, but if it fails, it will then use the currently cached version.

Running the app
With those simple steps, NativeScript creates a framework for an application that you can already run on your mobile device. Now, to run the project, we execute:

```bash
nativescript run android --emulator
```

Or:

```bash
nativescript run ios --emulator
```

NativeScript will then copy your application to the emulator, automatically start it up, and then you should see your first application running on the emulator screen like this:
Summary

We covered a lot in this chapter. You learned what NativeScript is and what makes NativeScript unique among all the different JavaScript mobile development systems. You also learned how to install NativeScript and explored a large number of different command-line parameters you will use. And finally, you actually created and ran a new project.

Now you have the tools you need installed, and most importantly, we now have a running application. Let's dive into what makes the whole project tick. We will explore all the different files and folders for your project and gain an understanding of how each file is used, and finally, we will enhance the default application in Chapter 2, The Project Structure.
Where to buy this book

You can buy Getting Started with NativeScript from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.