DEVELOPING FOR firefox OS

- QUICK GUIDE -

ANDRÉ GARZIA

Quick Guide For Firefox OS App Development

Creating HTML5 based apps for Firefox OS

Andre Garzia

This book is for sale at http://leanpub.com/quickguidefirefoxosdevelopment

This version was published on 2015-12-28

Tweet This Book!

Please help Andre Garzia by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I am learning to develop for #FirefoxOS

The suggested hashtag for this book is #FirefoxOS.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#FirefoxOS

To Elisangela Mendonça de Andrade Garzia

Contents

Acknowledgments

To my wife Lili, the best wife in the world!

To Mozilla for always believing in us, for keeping the web open and free and for always placing the user first!

To the Brazilian Mozilla Community for receiving me so well and being just awesome!

To my GSoC mentor Marcos Caceres, the Mozilla WebAPI Team, the Mozilla Tech Evangelists and Dev Engagement teams for being more than awesome!

To Google for the Google Summer of Code 2013! This program is wonderful.

I also say thank you from the bottom of my heart to all people who invested their time and effort in sending me pull requests to make this book better: Ryuno-Ki, chisophugis, ghost, dholbert, marcoscaceres.

This book is in perpetual beta

My plan is to update this book often, expanding its contents and revising the text as issues are found by readers. Since some APIs are still being implemented by Firefox OS, you will want to make sure you're reading an up-to-date version of this book.

Also, always check the **version history** section because it lists what changed and has important information about decisions made before releasing each book update.

Me, myself, and I

In this book you will find many parts where I express my personal opinion and make decisions that may be different from what other programmers would do - particularly if it helps explain an idea more easily. I will always try to make it clear and explain my reasoning when I am giving my opinion. Anyway, if there is an error in what I am saying, I will revise the text and update the book. See the Feedback & Pull Requests section for more information.

How this book came to be

Originally, I'd been writing this book in my spare time - but thanks to the good help of my Google Summer of Code (GSoC) mentor, Marcos Caceres, this book became part of my GSoC project - which aimed to create useful developer resources for Firefox OS. So, huge thanks to Google for funding this work and to Mozilla's Web API team for letting me join them over the summer.

Staying up to date

This book is distributed for **free** using Leanpub[1].

You can register your email to receive automatic updates when you download this book from its book page at Leanpub[2]. The plan is to update this book many times per month. If you got this book from a friend or from some other site, you should consider going to the page above to download and register there thus making sure you will receive the update notices.

Donations

Writing a book requires a lot of work and I would like to dedicate more time in my life for this type of activity after the 2013 Google Summer of Code is done. Those that think that this book is useful (or cool) may move the price slider on Leanpub download page from zero to any desired amount and give me some bucks.

Those that would rather donate using PayPal, I can receive donations under the *agarzia@mac.com* account.

I can accept bitcoin donations at:

17TNW3pw9iJUD24k2CB4wv6rJaQkZUsGFU[3]

And dogecoin donations (such generosity, much development) at:

DQ4puwF5uFcdt7WZNMoo7V3Y373VD6aHFx[4]

Regardless of donations, you should fill your email on the download form to make sure that once the book is updated you will receive a notice!

How to contact the author

To send comments and feedback please send an email to fxosquickguide@andregarzia.com[5]. My website is http://andregarzia.com[6]. My Twitter account is @soapdog[7].

If you want to help improve the content of this book, please see the Feedback & Pull Requests section.

[1]http://leanpub.com
[2]http://leanpub.com/quickguidefirefoxosdevelopment
[3]bitcoin:17TNW3pw9iJUD24k2CB4wv6rJaQkZUsGFU
[4]dogecoin:DQ4puwF5uFcdt7WZNMoo7V3Y373VD6aHFx
[5]mailto:fxosquickguide@andregarzia.com
[6]http://andregarzia.com
[7]http://twitter.com/soapdog

Cover Illustration

The cover page was created by Raphael Eckhardt, a designer and illustrator from Brazil. You can check out his work and contact him (he is a freelancer) at http://raphaeleckhardt.com/[8].

Who should read this book

This book is written for readers with an intermediate knowledge of HTML, CSS and JavaScript who wants to build mobile applications for Firefox OS. Teaching HTML, CSS and JavaScript is beyond the scope of this book. I will give you links for good reference books though.

Best Practices vs Beginner Friendliness

Experienced developers will notice that sometimes I don't follow all the good practices in the source code of the examples of this book. Even though I am avoiding anti-patterns in here, I am trying to keep the use of immediate functions and other similar practices to a minimum. The main reason for that is to make the source code beginner friendly as this is an introductory book. Seasoned programmers will know when and how to change things while beginner coders will still be able to understand what is going on. All code here works and as I update this book I may revisit the code and use more and more best practices depending on the readers feedback.

If you want to dive deeper in the world of high quality JavaScript coding here are some good books:

- JavaScript: The Good Parts[9]: The JavaScript Book.
- JavaScript Patterns[10]: Patterns and best practices.
- JavaScript Enlightenment[11]: Advanced JavaScript techniques.
- Maintainable JavaScript[12]: Writing code that is easy to maintain and work with.
- Javascript Allongé[13]: a wonderful book that makes you look to JS with a functional programming lens. You'll learn more about functions, combinators, decorators and all the cool stuff that you can do in our favorite web language.

[8]http://raphaeleckhardt.com/
[9]http://shop.oreilly.com/product/9780596517748.do
[10]http://shop.oreilly.com/product/9780596806767.do
[11]http://shop.oreilly.com/product/0636920027713.do
[12]http://shop.oreilly.com/product/0636920027713.do
[13]https://leanpub.com/javascript-allonge

More books about Firefox OS

I maintain a website listing the current known books about Firefox OS at http://firefoxosbooks.org[14]. I've written another book about Firefox OS focused on Game Development[15], you might want to check it out. Both books are available as a bundle as well[16].

Feedback & Pull Requests

This is a Free and Open book and I am excited to receive all feedback that you people can give me. All the content of the book is at a GitHub repository[17] and is built using Markdown (with some extensions by Leanpub). To give me feedback, bug fixes and improvements just send me a pull request. Thanks in advance for all contributions.

The Git repository for this book is at https://github.com/soapdog/firefoxos-quick-guide[18].

Translations

This book was originally written in Portuguese and translated into English by me. Both versions are available for free on the net at:

- Portuguese Version[19]: Guia Rapido para Desenvolvimendo para Firefox OS.
- English Version[20]: Quick Guide for Firefox OS App Development.
- Chinese Version[21]: Firefox OS App [illegible]
- Italian Version[22]: Guida veloce allo sviluppo per Firefox OS.

I welcome all help to translate this book to even more languages (and to fix my broken English).

Version history

Version 0.4: WIP

- Added bitcoin and dogecoin donation options.

[14]http://firefoxosbooks.org
[15]https://leanpub.com/buildinggamesforfirefoxos
[16]https://leanpub.com/b/firefoxosbooks
[17]https://github.com/soapdog/firefoxos-quick-guide
[18]https://github.com/soapdog/firefoxos-quick-guide
[19]http://leanpub.com/guiarapidofirefoxos
[20]http://leanpub.com/quickguidefirefoxosdevelopment
[21]https://leanpub.com/quickguidefirefoxos
[22]https://leanpub.com/guidavelocesviluppofirefoxos

- Moved the old simulator 1.1 chapter to an appendix.
- Updated content to use WebIDE.

The **App Manager** was replaced by the **WebIDE** and lots of the content of the book had to be rewritten. Also the current work on **Firefox OS Building Blocks** is being done using **Web Components** and will only work on devices running **Firefox OS 2.3+**. Most of the devices on the market are running **Firefox OS 1.x+** but not up to **2.3** which makes it impossible to recommend using the **Web Components** solution at the moment.

This version is a interim release. I just wanted to do a quick update for the WebIDE stuff while I write more content.

Version 0.3

Added new content for **App Manager**. Since most of the current devices still running **Firefox OS 1.1** we're keeping the old simulator stuff.

This version was done as a quick update to include the App Manager section. This section will be improved in the next releases. If you find anything wrong with this new section (or any section) then please report the issues at issue tracker on GitHub[23].

Version 0.2

Book was revised by Marcos Caceres of Mozilla's WebAPI team. The content of each chapter was checked for technical correctness, and many grammatical mistakes and typos were fixed throughout.

Version 0.1

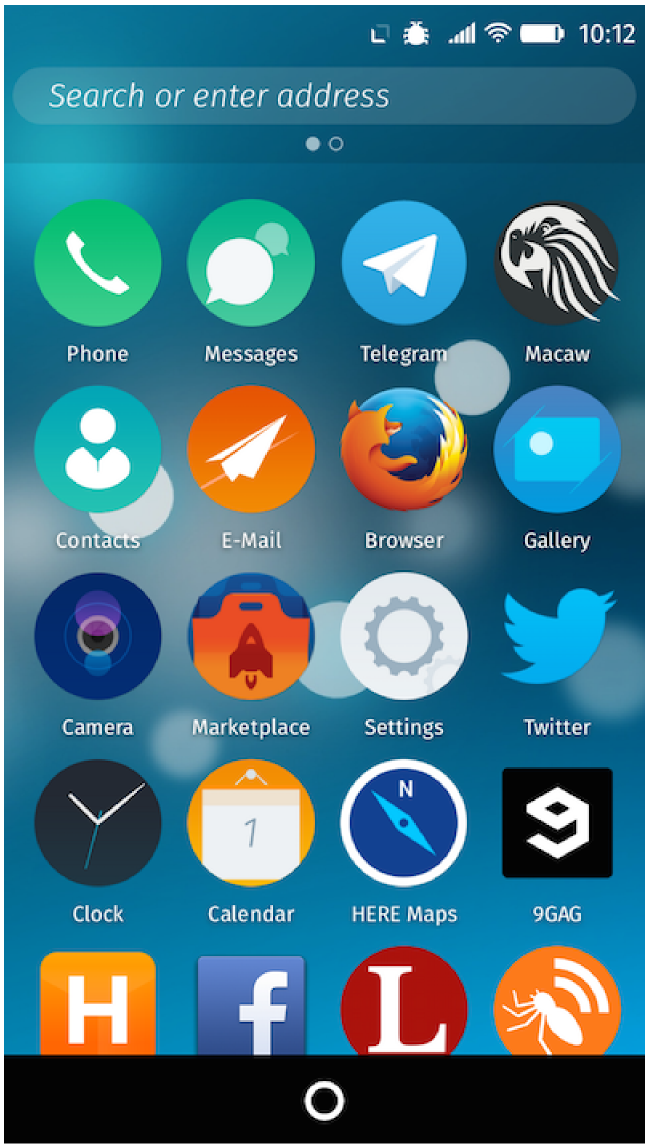
This is the first version of this book. I am yet to run this through an editor and it was not revised for typos, grammar mistakes and general bad things. English is not my first language so please correct me when I am wrong. What you're reading here begun on the 20th of August of 2013 as a quick guide to be distributed at BrazilJS Conference[24] that happened on the 22nd and 23rd. So you're basically reading a quick draft written in two days.

I am using the Leanpub[25] system to write this book. This system allows me to iterate quickly and manage this project while keeping me sane. This version is a quasi-literal translation from the original in Portuguese.

[23]https://github.com/soapdog/firefoxos-quick-guide/issues
[24]http://braziljs.com.br/
[25]http://leanpub.com

Introduction

Firefox OS



Firefox OS

Firefox OS[26] is a new mobile platform developed by Mozilla[27] and its partners. Devices running Firefox OS are already available in many countries and will reach even more places by the end of 2015.

Originally targeted at emerging markets, Firefox OS has the mission of helping to bring the next billion people online. To achieve this, Firefox OS devices are built to serve as a *great first smartphone* along with competitive pricing. Firefox OS devices should not be compared with high-end smartphones such as the Apple iPhone 6 and Samsung Galaxy S5; they are built to be an alternative to feature phones so that people using said devices are able to upgrade to a Firefox OS one at an affordable cost and receive the *full smartphone experience*.

In developing markets such as Brazil and Colombia, smartphones with decent performance are generally too expensive for the average consumer. People are able to buy cheap phones, but the platforms used in these phones are intended for high-end devices - as such, the phone's hardware tends to underperform, which leads to a terrible user experience. Firefox OS is specifically designed to run on limited hardware while providing a decent user experience.

Another differentiating factor of Firefox OS is its openness. Consider that the current mainstream mobile operating systems are proprietary silos, where each vendor has the privilege to force his way on the developers and users regardless of their wishes (remember when Apple tried banning languages other than Objective-C from the iTunes App Store?). In those proprietary ecosystems you can only distribute your apps on authorized channels - and the vendor usually keeps a significant part of the the money from any purchases made on the device.

Besides locking the developers to proprietary distribution channels, these systems lock you to their software development kits (SDKs). If you want to build a native app for both iOS and Android using the official toolkits you will need to code one app using Objective-C and another with Java respectively. This means that, code-wise, a developer will reuse very little between projects (and maybe reuse some media assets). That kind of effort requires that the developer learns two languages and build the same software twice.

Firefox OS differentiates itself by using "HTML5" as the development platform. HTML5 is a marketing term used to mean the ever-evolving collection of Web standards known as HTML, CSS and JavaScript. These royalty free standards are implemented by the major web browsers, and are what make web applications possible. By leveraging the technologies that encompass HTML5, millions of web developers are already able to code for Firefox OS. And apps built for Firefox OS are easy to port to another platform by using wrappers such as Phonegap[28] and Cordova[29].

Due to its open nature and flourishing community the new system has been ported to a number of new platforms and is being used by business in devices beyond the smartphone form-factors. As the new devices become available we plan to create chapters or maybe even new books about them. Just to trigger a spark of curiosity we'll list some of the new types of devices with links if you want to check them out:

[26]http://firefoxos.mozilla.community

[27]http://mozilla.org

[28]http://phonegap.com

[29]http://cordova.io/

Smart TVs

Panasonic is a partner of Mozilla and developed a range of Smart TVs powered by Firefox OS. As of 2015 most of the Smart TVs platforms are converging to HTML5 based app ecosystems. LG is using webOS, Samsung will soon launch Tizen based sets, Sony is rumored to go with Android. All those systems are able to work with HTML5 based apps which makes web technologies a very good investment. You can learn more about Mozilla and Panasonic Smart TVs here[30] and on this debut announcement[31].

IoT and the Web of Things

The internet of things has a little sibling called the web of things which is composed of hardware appliances talking to each other using web technologies. They are fun to build and easy to interface with. A group of enthusiasts from Mozilla employees to volunteers is working to port Firefox OS to the Raspberry Pi[32] and could use more help. Once this is working, you'll be able to experiment with hardware design and construction and keep programming fun by using web stuff.

Runcible

Runcible is a new concept from Monohm[33] which is very disruptive. A brand new class of device to provide you with the benefits of a connected device such as a Smartphone but none of the antisocial problems of a nagging device. Also, its beautiful and round, like a pocket watch. Monohm is also helping with porting Firefox OS to the Raspberry Pi and have a wonderful Web of Things SDK available[34] for us to use.

The Platform That HTML5 Deserves

As we learned from the sections above, **the web is everywhere**. Its on your computer, mobile phone, smart TV, and even in your video game consoles. The programming language of the web, JavaScript, is one of the most popular languages in the world. As already mentioned, when people talk about HTML5 they usually mean the collection of three technologies known as HTML, CSS and JavaScript. Recent advances in HTML have brought in a range of new features - advanced form controls, Web sockets, and more semantic markup - when compared to XHTML 1.0 and HTML 4.01. Advances in CSS have also introduced lots of new features, such as Flexbox and CSS Animations, that make it a lot easier to create beautiful responsive layouts. And recent advances in JavaScript have brought

[30]https://www.mozilla.org/en-US/firefox/os/devices/tv/
[31]https://blog.mozilla.org/blog/2015/05/14/first-panasonic-smart-tvs-powered-by-firefox-os-debut-worldwide/
[32]https://wiki.mozilla.org/Fxos_on_RaspberryPi
[33]http://mono.hm/runcible.html
[34]http://mono.hm/sensible.html

significant performance improvements and new capabilities, all while remaining easy to use for both beginners and seasoned developers alike.

Firefox OS is in essence, an extension of the mobile web. By making HTML5 a first class citizen, Mozilla has opened its platform to millions of web developers. Even if some other browser vendors implement HTML5 in their mobile offerings, Firefox OS goes beyond that by offering a collection of APIs to access the underlying hardware and system using JavaScript. These APIs are collectively known as the WebAPIs.

Accessing The Hardware Using The WebAPI

Some earlier platforms also tried to create operating systems that used web technologies for app creation. For example, when the iPhone was introduced to the world, the only way to create apps was using web technologies. However, those web apps were limited in that they had no hardware or device access - meaning that only a limited range of applications could be built. When Apple then allowed developers to code apps in Objective-C, and also access the device's capabilities, it spurred a huge amount of innovation. Sadly, web apps did not gain access to the device's capabilities, and were thus left as "second-class citizens" - this made them unattractive to both users and developers alike, and unable to compete with native apps in that system.

When we say device capabilities we actually mean accessing hardware and OS level features and services: We're talking about things such as updating the address book, sending SMS, and accessing the camera and media gallery. On Firefox OS, the WebAPI[35]s are the means by which you will access many of those capabilities.

Another earlier platform, WebOS, also offered hardware access via JavaScript but never tried to standardize its APIs. Mozilla is working with the W3C and other stakeholders to make sure that the WebAPIs are an open standard and that other browsers adopt them too. As these APIs are implemented by other browsers, your apps will require less and less changes to work across different platforms.

It's important to emphasize that the WebAPIs are not exclusive to Firefox OS devices. Mozilla is implementing it for the other platforms on which Firefox runs, such as desktop and android. This way, you can use your *open web app* in Firefox OS, Firefox on the desktop and Firefox for Android.

Freedom to Develop and Distribute

Like everything that Mozilla does, Firefox OS is developed in the open and is free. All development can be followed on the Mozilla B2G repository[36] on GitHub. With Firefox OS you have the freedom to follow and contribute with the development of the system and you also have the freedom to

[35]https://developer.mozilla.org/en-US/docs/WebAPI

[36]https://github.com/mozilla-b2g/B2G

distribute your applications on your own channels or on The Firefox Marketplace[37]. What's really awesome is that all the system applications are written in HTML5, so you can check them out and see how the are put together.

The main idea is that you're not locked to Mozilla for anything. If you want to pick the source code for the system and change it for your own needs, so be it. If you need to build apps for internal use on your company, or if you want to distribute your creations only on your own web site, you're free to do it. Usually, in other platforms you're locked into the official app store as the only channel to distribute your apps. Firefox OS also has an official market called Firefox Marketplace which has an approval process but you're free to distribute your app outside this store if you want. Like in the web where you can host your web site anywhere you want, on Firefox OS you can do the same with your applications.

This comes with a small caveat, sadly: some of the WebAPIs are too security sensitive to just allow anyone to use them. To distribute apps that use some of the more "privileged" APIs, you will need to get your applications signed and reviewed by Mozilla's staff.

Summary

HTML5 is here to stay and will only get better. Firefox OS is the new open mobile operating system by Mozilla completely based on web technologies. This system is built on the open and offers a robust HTML5 implementation that goes beyond the other platforms by offering the WebAPI which is a collection of APIs to access *hardware and operating system services using JavaScript*. These new APIs are being standardized through the World Wide Web Consortium (W3C) and will hopefully be adopted by other browsers in the future.

In the next chapter we'll quickly get you set up with everything you need to develop for Firefox OS.

[37]https://marketplace.firefox.com/

Setup For Firefox OS App Development

The Gecko Engine

Browsers use different engines for rendering web pages: Google Chrome and Opera use Blink (a fork of WebKit), Internet Explorer uses Trident, the new browser from Microsoft called Edge uses a new engine, and Safari uses WebKit. Mozilla has its own engine, called Gecko which is used in Firefox desktop, Firefox for Android, and Firefox OS. As these products use the same engine, it is possible to develop for Firefox OS using the Firefox desktop browser (but with some caveats[38]).

What applications do you need?

To develop and test apps made for Firefox OS you will need:

- A recent version of the Firefox Browser[39]. We personally recommend using the new Firefox Developer Edition[40] because it is updated more often and comes with new features for developers.
- A text editor for programming[41] or you can simply use the WebIDE editor that comes bundled with Firefox.

If you're using an old Firefox OS 1.1 device

If you need to connect an old Firefox OS 1.0 or 1.1 device refer to the Appendix 2: The Firefox OS Simulator.

[38]Although the same engine is used for all Mozilla products, the version of the engine in Firefox OS is generally behind that of the desktop browser. This is because the release cycle of Firefox OS is currently slower than that of the Desktop browser. In practice, this will mean that some features may not be available (or work as expected) when you try them out in Firefox OS - so always make sure you test your applications on a device that runs Firefox OS. Also, be mindful that users might also be on different versions of Firefox OS, so they might not have all the bleeding edge features. Be sure to always provide a fallback in case where some feature is unavailable. Also, did you knew that Mozilla is creating a brand new engine called servo? This new engine will provide a safer browser with lots of benefits from its parallel processing engine, it will rock! You can learn more about it on the Mozilla Servo github page.

[39]http://getfirefox.com

[40]https://www.mozilla.org/en-US/firefox/developer/

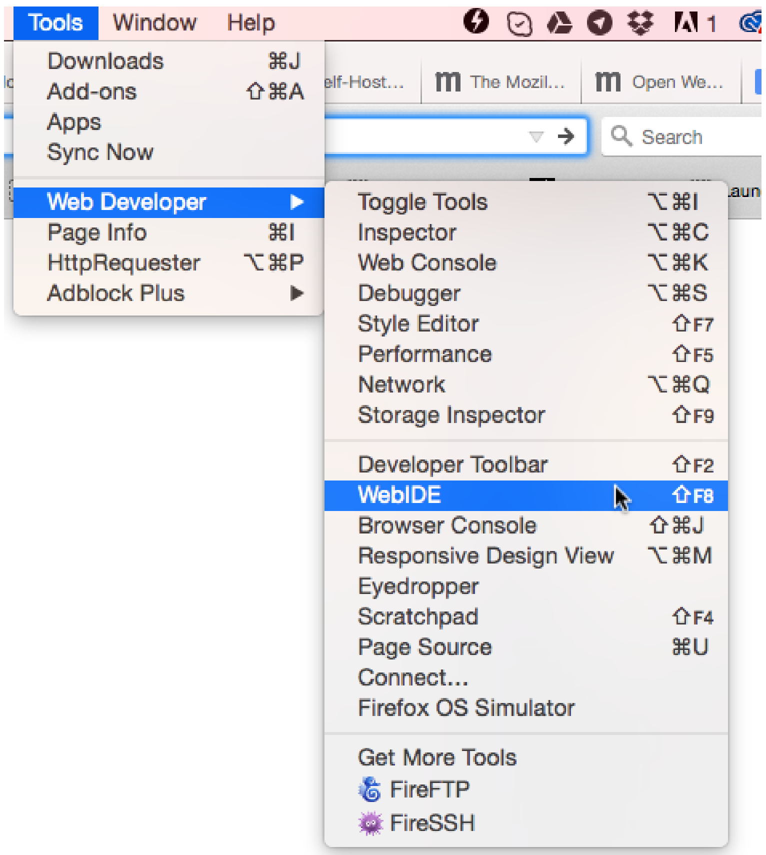
[41]There are many good editors out there with different levels of complexity and features. A very popular one that I recommend for those that don't have a favorite one is Atom. Personally, I use WebStorm which is a complete IDE for web app creation.

WebIDE Setup

If you're running the current version of Firefox then you have the WebIDE available to you. Having the WebIDE is not enough though. You still need to install the Firefox OS simulators on the WebIDE itself to be able to test things without hooking a device to your machine. Mozilla has extensive documentation about the WebIDE[42] so if you want to dive a bit deeper check it out.

The WebIDE is able to manage multiple Firefox OS versions so you can install simulators for version 1.2, up to 3.0, remember that the higher the version number the earlier it is. And by earlier I mean buggy but since we can have multiple versions then we should just install them all so that we're able to test our apps with different Firefox OS versions.

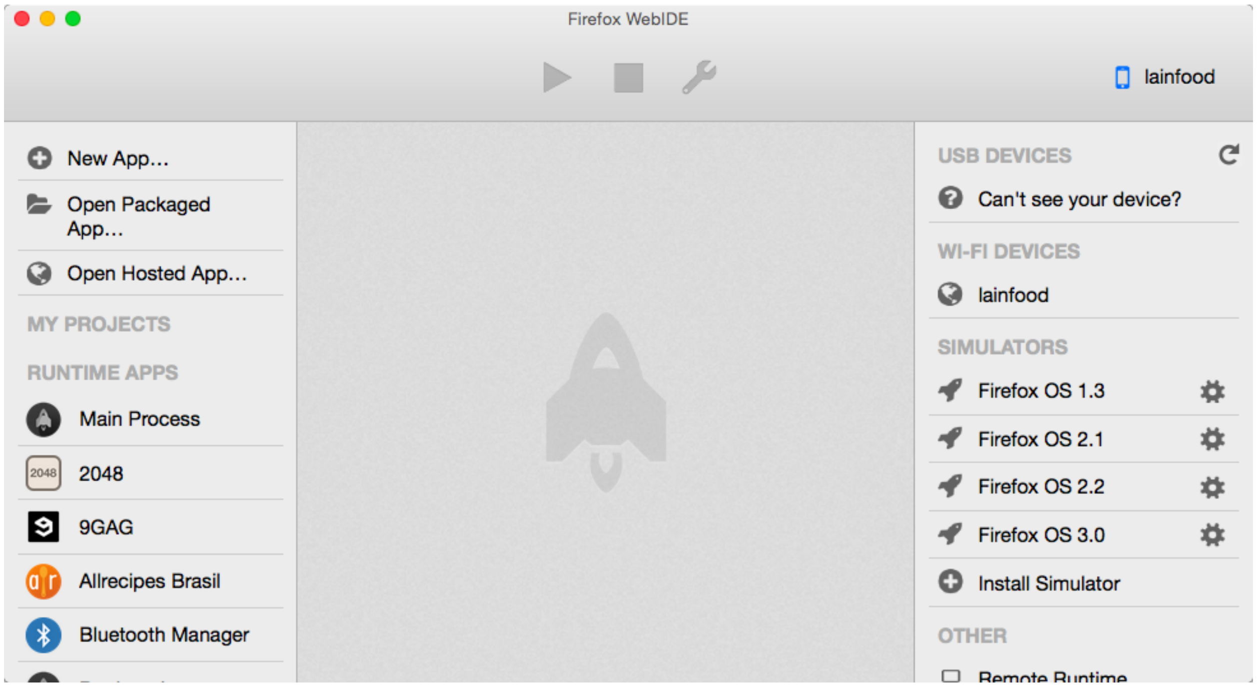
Lets take the new WebIDE for a spin and install the stuff we'll need for later. To launch the it go to the menu **Tools -> Web Developer -> WebIDE**.



Where you can find the Web IDE

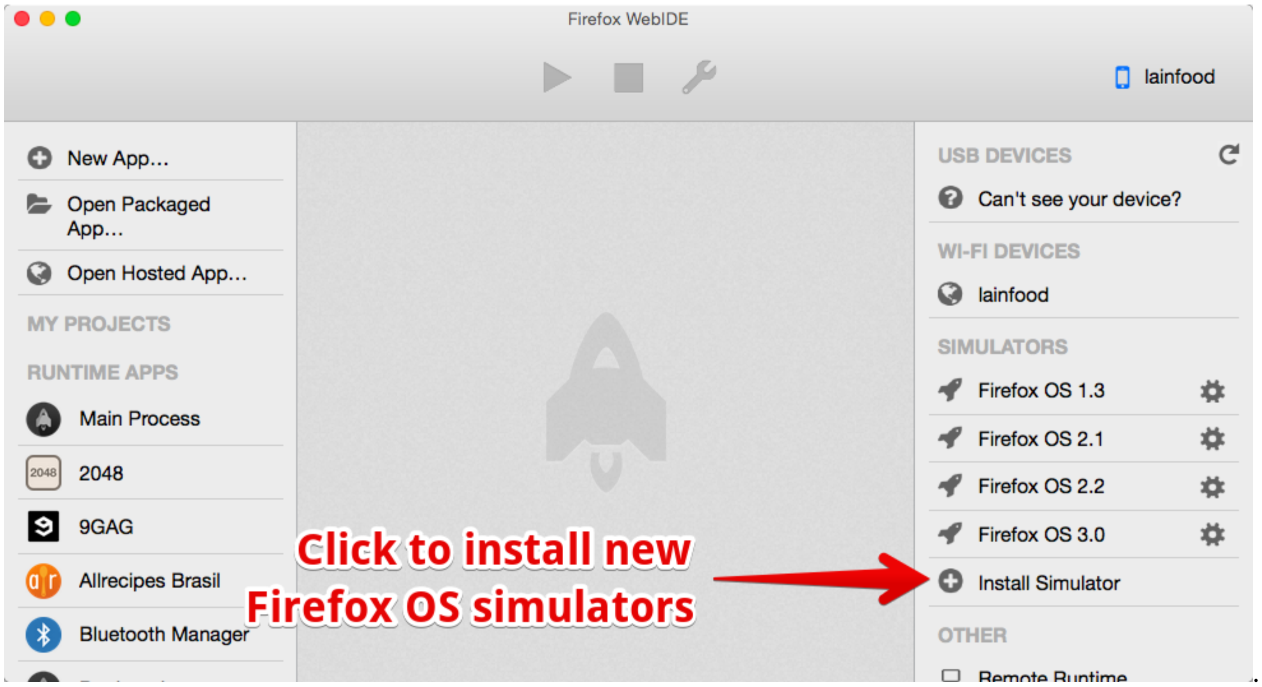
After you launch the WebIDE you will see a screen like this

[42]https://developer.mozilla.org/en-US/docs/Tools/WebIDE



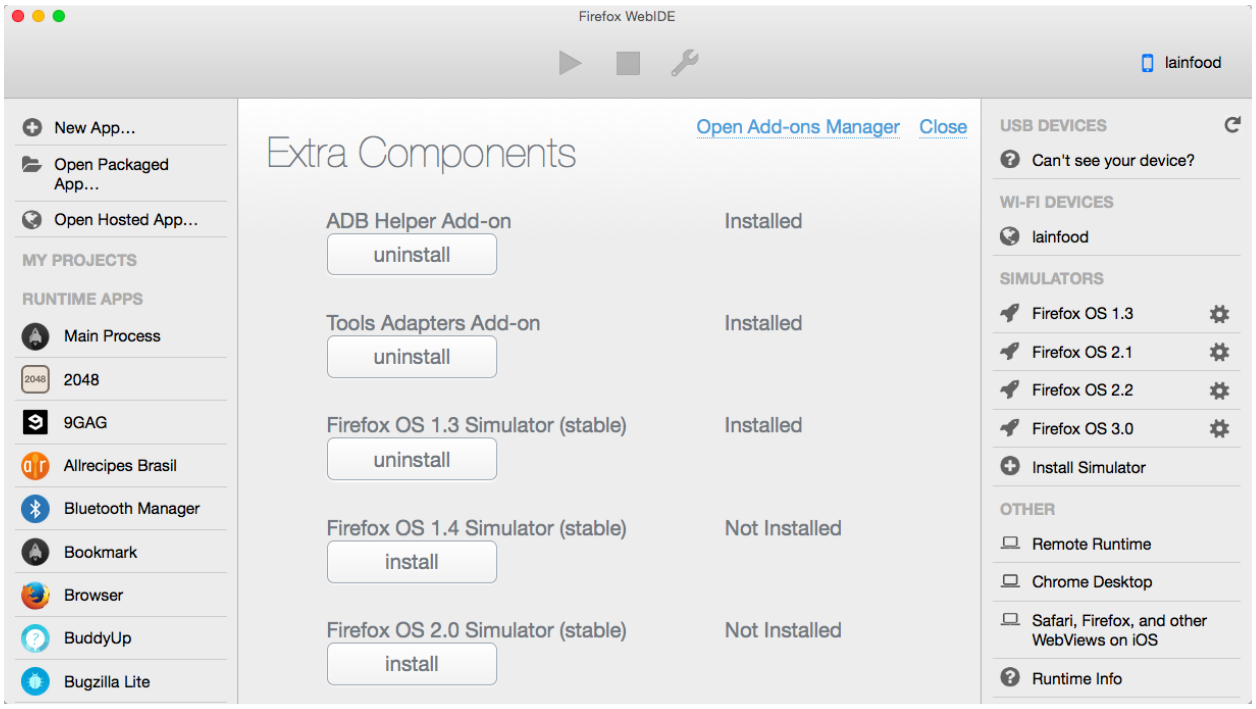
The WebIDE main screen

You need to click the **Select Runtime** button and select the **Install Simulator** option as seen below.



That will lead you to the **Extra Components** screen where you have the options of installing different simulator versions and both the ADB add-on and the tools add-on. The WebIDE uses ADB to communicate with connected devices. It is able to handle all the ADB stuff for you if you install the **ADB Helper Add-on**. The **Tools Add-on** will help you debug other connected devices such as Android devices and iOS devices.

My recommendation is to install everything.



Extra Components

ADB Helper Add-on (uninstall)	Installed
Tools Adapters Add-on (uninstall)	Installed
Firefox OS 1.3 Simulator (stable) (uninstall)	Installed
Firefox OS 1.4 Simulator (stable) (install)	Not Installed
Firefox OS 2.0 Simulator (stable) (install)	Not Installed

The extra components page

Summary

In this chapter we learned that all we need to develop Firefox OS apps is the Firefox browser, the WebIDE and Firefox OS Simulators (and a good text editor is a plus).

Now that we have setup all the tools we need, lets learn some basic concepts before we build our first app.

Basic Concepts

Before we get our hands dirty and build our first app, let's learn some basic concepts about developing for Firefox OS. We learned in the introduction that, just like web pages, apps in Firefox OS are based on HTML5. However, we haven't explained what makes Firefox OS apps different from regular web pages.

If we use our collective knowledge about other mobile platforms we can see that native application generally will have:

- A name and an icon that the user can press to launch the app.
- Access to system services and hardware capabilities.

Looking at the big picture, a Firefox OS app is just a web page that has an icon, a name and is usually able to work offline (depending on how the app is implemented). All the data about an application such as name, icon and more is defined in a *application manifest file* that is the focus of our next section.

The Application Manifest

The manifest[43] is a JSON[44] file that describes aspects of an hosted web app. Usually this file is called **manifest.webapp** and lives next to your main HTML file that is usually called **index.html**.

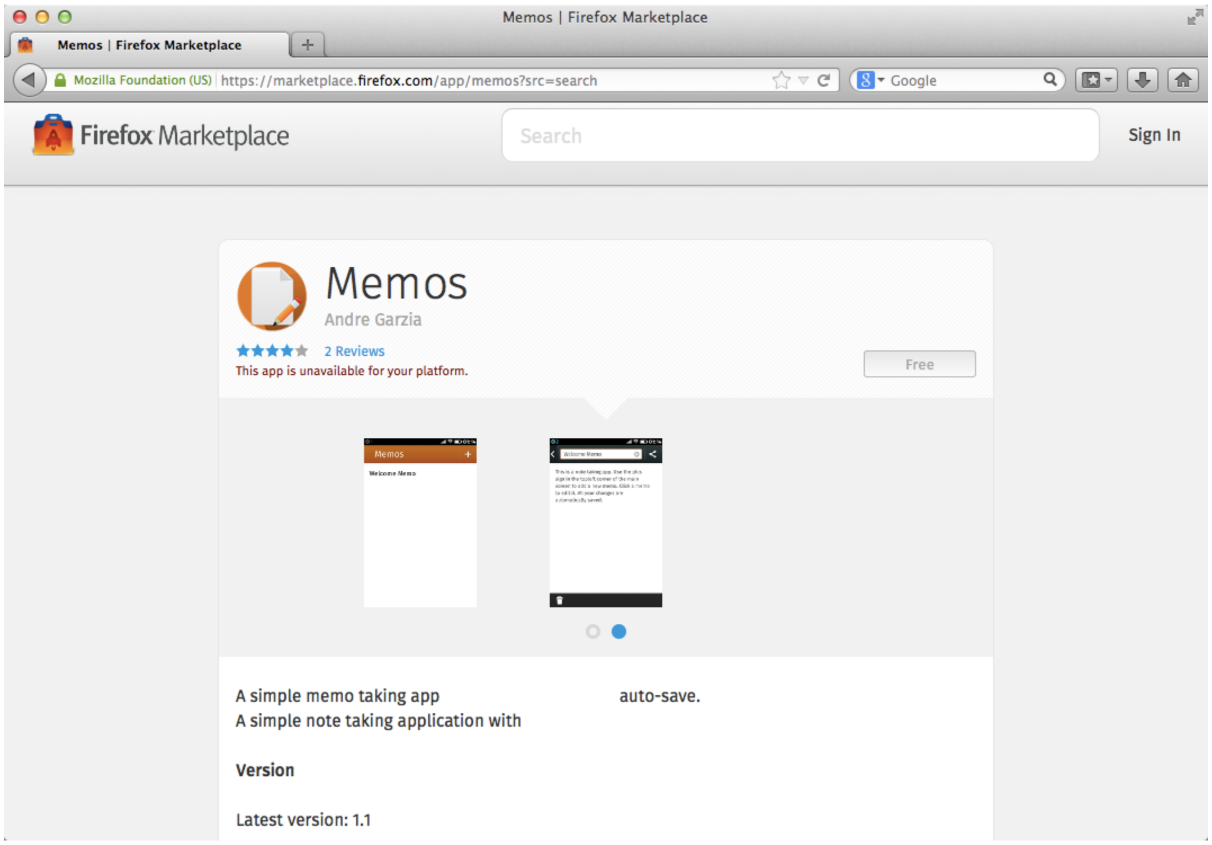
Sample Manifest

```
{
  "name": "Memos",
  "version": "1.1",
  "description": "A simple memo taking app",
  "launch_path": "/index.html",
  "permissions": {
    "storage": {
      "description": "Required for storing and retrieving notes."
      }
  },
  "developer": {
```

[43] https://developer.mozilla.org/docs/Apps/Manifest
[44] http://json.org

```
    "name": "Andre Garzia",
    "url": "http://andregarzia.com"
  },
  "icons": {
    "60": "/style/icons/icon_60.png",
    "128": "/style/icons/icon_128.png"
  }
}
```

Above we can see the manifest for an application called memos[45]. Among other things it describes who created the application, which icons are used, what is the name of the app, what file is used to launch the app (in this case it is *index.html*), what hardware access permissions your app requires, etc. This file is used by Firefox OS to add the application to the device's home screen and by the Firefox Marketplace to display the application on the catalog as we can see in the image below.


Memos | Firefox Marketplace
https://marketplace.firefox.com/app/memos?src=search
Firefox Marketplace
Search
Sign In
Memos
Andre Garzia
2 Reviews
This app is unavailable for your platform.
Free

A simple memo taking app
A simple note taking application with auto-save.

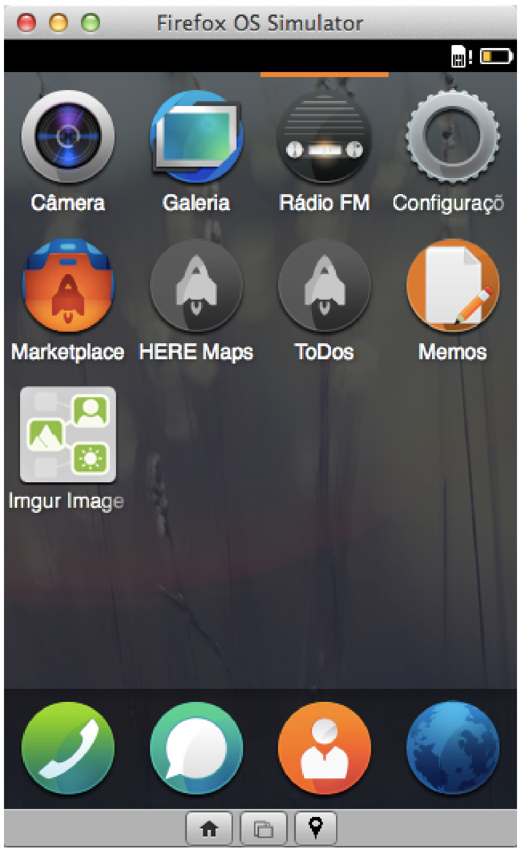
Version

Latest version: 1.1

Memos app shown at the Firefox Marketplace

Note how the information from the manifest is used by the system to add the app to the homescreen, as we can see on the following screenshot.

[45]This is a sample app for Firefox OS as seen on the Firefox Marketplace for which the source code is on GitHub.



Memos on the simulator

By gathering your HTML, CSS, JavaScript, and a manifest file you already have an application ready to run on Firefox OS. Moving on our topic about basic concepts let's learn more about what application types there are.

Types of Application

Firefox OS currently has two types of applications: hosted apps and packaged apps - though more types may become available in the future (e.g. custom keyboards and the ability to create other system services).

- **Hosted Apps**: Are hosted on a web server just like normal websites. This means that when the user launches a hosted app, its content is loaded from the remote server (or from the cache, if available).
- **Packaged Apps**: Are distributed as a zip file and copied to the device when installed. When the user launches a packaged app, its contents are loaded from the zip file instead of a remote server.

There are pros and cons to both types. On the one hand, hosted apps are easier to maintain, as all you need to do is maintain files on your web server. However, it's harder to make them work offline because it requires the use of the much despised **appcache**[46]. Hosted apps are also limited in which WebAPIs they can use, which means they can't do all the things a packaged app can do.

On the other hand, packaged apps have all their content stored on the device - which means they are always available when the user is offline. They also have the ability to access security-sensitive WebAPIs that are not available to hosted apps. Updating them can be a bit painful, because you need to upload any new version to the Firefox Marketplace - which means going through a review process, which can take some time.

When trying to choose which type of application to build, consider: if you require advanced WebAPIs, then you should use a packaged app. However, if your application works fine without needing to access any advanced system services or device capabilities beyond those already available in a web browser, then always choose a hosted app. It is ok to use packaged apps if you don't have a place to host it.

Above I mentioned that appcache can be problematic (which is sometimes required for hosted apps). Don't worry too much as there are tools available to make appcache generation and deployment easier [47].

In this book we're going to build a packaged app, as it will allow us to explore what is possible with the WebAPIs. However, most of what we will learn about manifests applies to hosted apps. If you want to know more about distributing hosted apps, check the hosted applications link at the developer hub[48].

Now that we've covered the two types of applications that Firefox OS supports, let's look at the different levels of system access they can have.

Security Access Levels

There are three security levels on Firefox OS - with each level having more access to APIs than the previous level.

- **Plain (a.k.a. web)**: This is the default level for all applications. This level applies to hosted apps and packaged apps that do not declare a `type` property in their manifest file. These apps have access to the common set of APIs found in browsers - but don't have access to any of Mozilla's WebAPIs.
- **Privileged**: This type of app has access to all common APIs found in the Firefox browser, plus some additional ones, such as contacts, and system alarms. Only **packaged apps can be privileged apps** and the package must be digitally signed by the Firefox OS Marketplace.

[46]https://developer.mozilla.org/pt-BR/docs/HTML/Using_the_application_cache

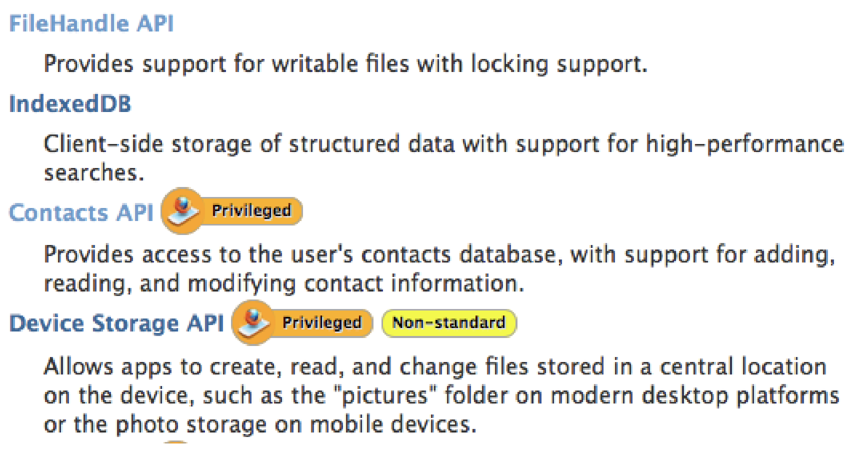
[47]There are many useful tools out there, check out Gulp, Grunt, Yeoman, Bower. There is a lot of overlap among these tools, its a matter of preference which one you use. (I like Gulp more than Grunt mostly because gulpfiles are easier for me to read).

[48]https://marketplace.firefox.com/developers/docs/hosted

- **Certified**: For security reasons, this level is only available to Mozilla and its partners (e.g. phone manufacturers, telecoms, etc.). Certified apps are able to access all the APIs, such as telephony and more. An example of certified app is the Firefox OS dialer application.

During development, it is possible for us to access privileged APIs without needing any special permission from Mozilla. But when we want to distribute a privileged app, it first needs to go to the Firefox Marketplace. There, the code is checked as part of a rigorous approval process, and if it's found to be OK, it will be digitally signed - which tells users of Firefox OS that this application is allowed to access sensitive APIs.

On the page about the WebAPIs on the Mozilla Developer Network[49] we can see what APIs are implemented on what platforms and what access level is needed to use each API.


FileHandle API

Provides support for writable files with locking support.

IndexedDB

Client-side storage of structured data with support for high-performance searches.

Contacts API Privileged

Provides access to the user's contacts database, with support for adding, reading, and modifying contact information.

Device Storage API Privileged Non-standard

Allows apps to create, read, and change files stored in a central location on the device, such as the "pictures" folder on modern desktop platforms or the photo storage on mobile devices.

Access levels for the APIs

As we can see on the image above, any application can access the *IndexedDB API and FileHandle API* but only privileged apps can access the *Contacts API and Device Storage API.*

Mozilla's WebAPIs

Firefox OS provides us with the APIs that enable us to build applications that are just as capable as native apps on other platforms. Access to hardware and services is done through the WebAPIs. To learn more about the list of available APIs for the Firefox OS check out the Web API page on the Mozilla Developers Network[50].

Lets review some code examples to see how easy those APIs are to use. Don't take this example as a full documentation of the WebAPIs, they are just a small sample to make you understand how we can access device features using JavaScript.

[49] https://developer.mozilla.org/en-US/docs/WebAPI
[50] https://developer.mozilla.org/en-US/docs/Web/Reference/API

Example #1: Making calls

Imagine that you have an application that needs to open the dialer with a phone number already filled in. You can just use the following code:

Sending a phone number to the dialer

```
var call = new MozActivity({
  name: "dial",
  data: {
    number: "5555-9999"
  }
});
```

This code makes a request to the dialer app to call a particular number. Note that this doesn't actually place a call - the user will still need to tap the dial button to place the call. Requiring explicit user action before executing some other action is pretty common: it's a good security pattern because it requires user interaction through consent before allowing something to happen. Other APIs that can place calls without user interaction are available for more elevated access levels. Certified apps can place calls without interaction for example. The API used in the code above, called "Web Activities", is available to all apps though.

Check out the the Mozilla Blog for more information about Web Activites[51].

Example #2: Saving a contact

Imagine that you have a company intranet and you want to provide a way to transfer a contact from the online intranet address book to the phone address book. You can do that with the Contacts API.

Saving a contact

```
var contact = new mozContact();
contact.init({name: "Odin"});

var request = navigator.mozContacts.save(contact);
request.onsuccess = function() {
  // contact saved successfully
};
request.onerror = function() {
  // there was an error while trying to save the contact
};
```

[51]https://hacks.mozilla.org/2013/01/introducing-web-activities/

This API creates an object with the contact data and saves it into the phone address book without requiring user interaction. Because access to contacts carries potential privacy implications, this API is only available for *privileged apps*. This pattern where you create an object with a success and an error callback is used in many of the WebAPIs.

To learn more about this API, read the page about the *Contacts API* on the Mozilla Wiki[52].

Example #3: Picking an image from the camera

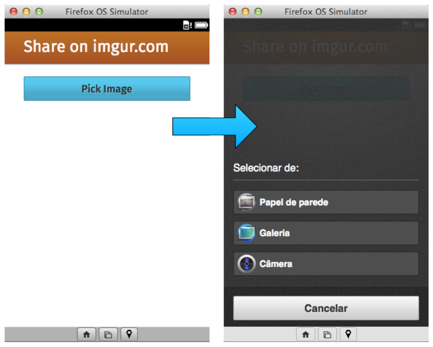
Imagine you are building an application that applies fancy filters to pictures. You want to place a button in your app that allows the user to pick a photo from a photo album or from the camera.

Picking an image

```
var getphoto = new MozActivity({
  name: "pick",
  data: {
    type: ["image/png", "image/jpg", "image/jpeg"]
  }
});

getphoto.onsuccess = function () {
  var img = document.createElement("img");
  if (this.result.blob.type.indexOf("image") != -1) {
    img.src = window.URL.createObjectURL(this.result.blob);
  }
};

getphoto.onerror = function () {
        // error!
};
```

Here we see another example of a WebActivity[53]. These activities are available to all applications. In this specific sample we're using the *pick* activity and passing in the *MIME Types* of the files that we wish to retrieve. When this code is executed, the system shows a screen to the user asking where he or she wants to retrieve the image from (camera, gallery, wallpapers). If the user selects an image, the success callback is triggered. If the user cancels the operation, the error callback is executed. On the image below, we can see the dialog that lets the user pick a photo from the device:

[52]https://wiki.mozilla.org/WebAPI/ContactsAPI
[53]https://hacks.mozilla.org/2013/01/introducing-web-activities/



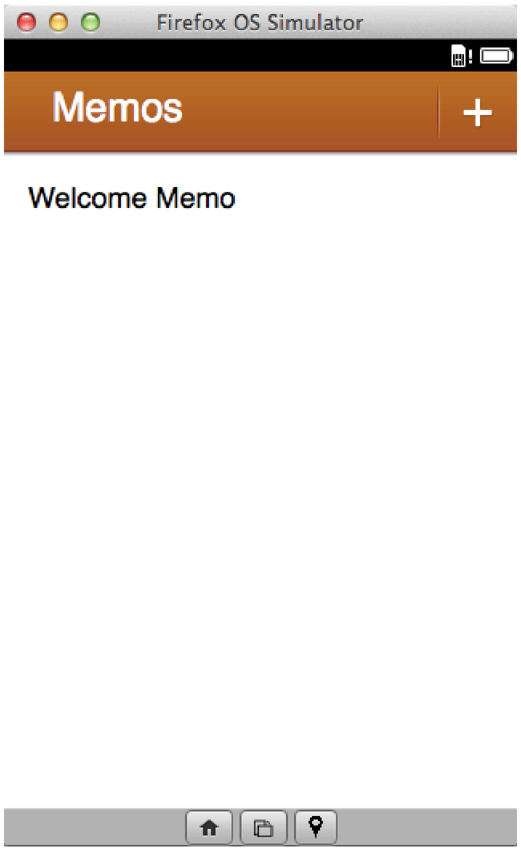
Example of the *pick activity*

Summary

In this chapter we saw that, unlike regular web pages, both Firefox OS' hosted apps and packaged apps rely on a manifest file. We also saw that, from a security perspective, packaged apps can be "privileged" or "certified". Only privileged and certified apps can access Mozilla's powerful set of WebAPIs. The more sensitive WebAPIs are not available to hosted apps or to regular web pages.

Now it's about time we get our hands dirty and create an app!

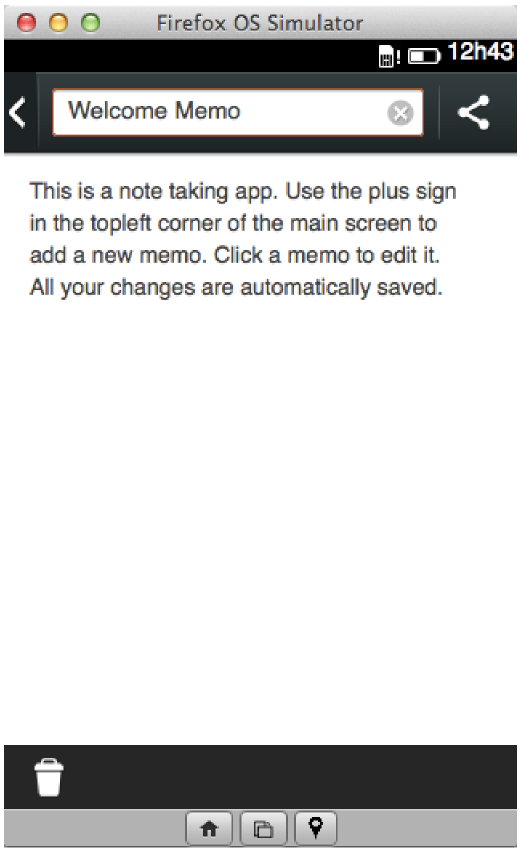
Our First App



Memos, a minimalist notepad app

In this chapter we're going to build a simple **Memos** application, which is an application for taking notes. Before coding, let's review how this app works.

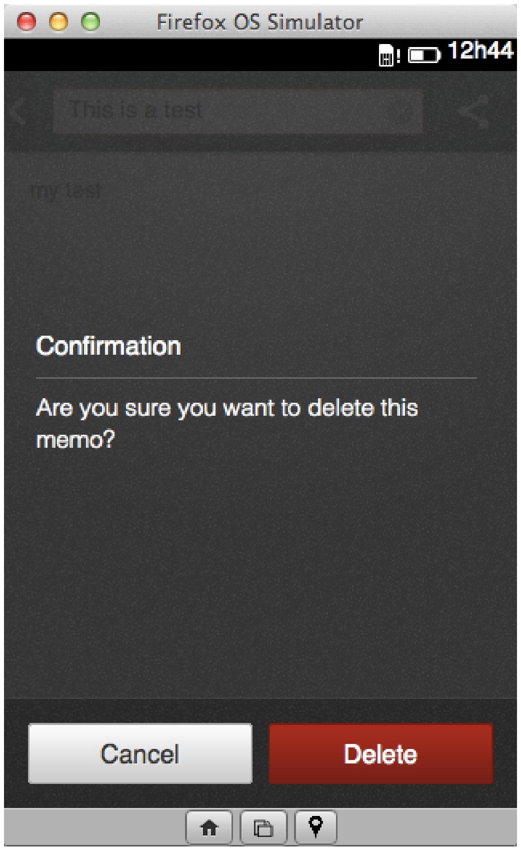
The app has three screens. The first one is the main screen and has a list of your stored notes by title. When you click a note (or add a new one) you're moved to the detail screen that allows you to edit the content and title of the given note. This is shown in the figure below.



This is a note taking app. Use the plus sign in the topleft corner of the main screen to add a new memo. Click a memo to edit it. All your changes are automatically saved.

Memos, editing screen

On the screen shown above the user can choose to delete the selected note by clicking on the trash icon. This will cause a confirmation dialog to be shown.



Confirmation

Are you sure you want to delete this memo?

Memos, note removal confirmation screen

The source code for Memos is available at the Memos Github Repo[54] (also available as a .zip[55] file). I recommend you download the files, so it's easier to follow along. Another copy of the source code is available on the **code** folder inside the github repository for this book[56].

Memos uses IndexedDB[57] to store the notes and the Firefox OS Building Blocks[58] to build the interface. You can use any library and layout that you want. You're not forced into using the Firefox OS look and feel. We'll use it for our demo because it looks awesome.

Cool Frameworks & Libraries

There are many professional frameworks and libraries out there. In this book we're using a minimal source code created by yours truly just to make this demo but on your own apps you probably want to consider using better solutions or even rolling your own. Below we list some popular frameworks and libraries which we think deserve some attention.

[54]https://github.com/soapdog/memos-for-firefoxos
[55]https://github.com/soapdog/memos-for-firefoxos/archive/master.zip
[56]https://github.com/soapdog/firefoxos-quick-guide
[57]https://developer.mozilla.org/en-US/docs/IndexedDB
[58]https://developer.mozilla.org/en-US/Apps/Design/Firefox_OS_building_blocks

- Enyo JS[a]: Complete mobile app development framework. Easy to use, well documented, sane. Its my favorite.
- Ember JS[b]: A framework for creating ambitious web apps.
- React[c]: A framework for creating user interfaces.
- Angular JS[d]: A popular framework for building web applications.
- jQuery Mobile[e]: A touch optimized web framework.

Of course there are hundred others but these are all popular/good/interesting enough and should serve you well depending on your needs.

[a]http://enyojs.com
[b]http://emberjs.com/
[c]https://facebook.github.io/react/
[d]https://angularjs.org/
[e]https://jquerymobile.com/

The first step is to create a folder for the application, let's call this folder **memos**.

Creating the app manifest

Memos manifest is pretty straight forward. Create a file named **manifest.webapp** on the **memos** folder. Manifests are JSON[59] files that describes an application. In this file we place things such as the name of the app, who the developer is, what icons are used, what file is used to launch the app, what privileged APIs it would like to use, and more.

Below we can see the contents of the Memos app manifest. Attention when copying this data because it's very easy to place a comma on the wrong place and create an invalid JSON. There are many tools that you can use to validate JSON files but there is a special one that is built specifically for validating app manifests. You can check out this online tool at http://appmanifest.org/[60]. To learn more about app manifests read this page on MDN about them[61].

[59]http://json.org
[60]http://appmanifest.org/
[61]https://developer.mozilla.org/docs/Apps/Manifest

Memos manifest file (*manifest.webapp*)

```
{
  "name": "Memos",
  "version": "1.1",
  "description": "A simple memo taking app",
  "launch_path": "/index.html",
  "permissions": {
    "storage": {
      "description": "Required for storing and retrieving notes."
      }
  },
  "developer": {
    "name": "Andre Garzia",
    "url": "http://andregarzia.com"
  },
  "icons": {
    "60": "/style/icons/icon_60.png",
    "128": "/style/icons/icon_128.png"
  }
}
```

Let's review the fields from the manifest above.

Field	**Description**
name	This is the name of the application.
version	This is the current version of the app.
launch_path	What file is used to launch your application.
permissions	What API permissions your app requests. More information about this below.
developer	Who developed this application
icons	The icons used by the app in many different sizes.

The most interesting part of this manifest is the permissions field where we ask for the *storage* permission that allows us to use IndexedDB without size restrictions[62] (thanks to that permission we can store as many notes as we want - though we should be mindful not to use too much of the user's disk space!).

Now that the manifest is ready let's move on to the HTML.

[62]To learn more about permissions read the page on MDN about app permissions.

Building the HTML

Before we start working on the HTML, let's take a brief detour to talk quickly about the Gaia Building Blocks[63], which are a collection of reusable CSS and JS with the *look and feel* of Firefox OS that we can use on our own apps.

Just like on the Web, you're not required to use the *look and feel* of Firefox OS in your own app. Using or not using the Gaia Building Blocks is a personal decision - and a good applications should have its own distinctive style and user experience. The important thing to understand is that your app will not suffer any type of prejudice or penalty on the Firefox Marketplace by not using the Gaia look and feel. I am using it here because I am not a good designer so ready made UI toolkits appeal to me (it's either that or hiring a designer).

The HTML structure that we use in this application was built following the patterns adopted by the Gaia Building Blocks where each screen is a `<section>` and the elements follow a predefined format. If you haven't already, download the source code from the memos repository[64] so that you have the files (including the Building Blocks) to use. For those not confident with git and GitHub, the files are also available as a .zip file[65].

⚠ Warning: The version of the Gaia Building Blocks I used for this app is not the most up-to-date available from Mozilla. Trying to update to the current version will, unfortunately, break the Memos app. In your own projects, however, always use the latest version of the Gaia Building Blocks.

Including the Building Blocks

Before doing anything else copy the **shared** and the **styles** folders that you obtained by downloading the Memos repository to the **memos** folder you created. This will allow use to use the Gaia Building Blocks in our app.

Let's begin our **index.html** files by including the needed bits.

[63]http://buildingfirefoxos.com/building-blocks
[64]https://github.com/soapdog/memos-for-firefoxos
[65]https://github.com/soapdog/memos-for-firefoxos/archive/master.zip

```
 1  <!DOCTYPE html>
 2  <html>
 3  <head>
 4      <meta charset="utf-8">
 5      <link rel="stylesheet" type="text/css" href="style/base.css" />
 6      <link rel="stylesheet" type="text/css" href="style/ui.css" />
 7      <link rel="stylesheet" type="text/css" href="style/building_blocks.css" />
 8      <link rel="stylesheet" type="text/css"
 9            href="shared/style/headers.css" />
10      <link rel="stylesheet" type="text/css"
11            href="shared/style_unstable/lists.css" />
12      <link rel="stylesheet" type="text/css"
13            href="shared/style_unstable/toolbars.css" />
14      <link rel="stylesheet" type="text/css"
15            href="shared/style/input_areas.css" />
16      <link rel="stylesheet" type="text/css"
17            href="shared/style/confirm.css" />
18      <title>Memos</title>
19  </head>
```

On *line 01* we declare the DOCTYPE as HTML5. From *line 05 up to 15* we include the CSS from the various components that we're going to use in our app such as headers, lists, text entry fields and more.

Building the main screen

Now we can start building the various screens. As mentioned earlier, each screen used by our app is a `<section>` inside the HTML `<body>`. The body tag must have an attribute *role* with its value equal to *application* because that is used by the CSS selectors to build the interface, so our body tag will be `<body role="application">`. Let's build the first screen and declare our body tag as well.

```
<body role="application">

<section role="region" id="memo-list">
    <header>
        <menu type="toolbar">
            <a id="new-memo" href="#"><span class="icon icon-add">add</span></a>
        </menu>
        <h1>Memos</h1>
    </header>
    <article id="memoList" data-type="list"></article>
</section>
```

Our screen has a `<header>` containing a button to add new notes and the application name. The screen also has an `<article>` which will be used to hold the list of stored notes. We're going to use the button and the article IDs to capture events when we reach the JavaScript implementation part.

Be aware that each screen is a fairly straight forward HTML chunk. Building these same screens in many languages usually requires a lot more work. All we're doing is declaring our containers and giving them IDs when we need to reference them later.

Now that the main screen is done, let's build the editing screen.

Building the editing screen

The editing screen is a bit more complex because it also holds the dialog box used when the user tries to delete a note.

```
<section role="region" id="memo-detail" class="skin-dark hidden">
    <header>
        <button id="back-to-list"><span class="icon icon-back">back</span>
        </button>
        <menu type="toolbar">
            <a id="share-memo" href="#"><span class="icon icon-share">share</spa\
n>
            </a>
        </menu>
        <form action="#">
            <input id="memo-title" placeholder="Memo Title" required="required"
                   type="text">
            <button type="reset">Remove text</button>
        </form>
    </header>
    <p id="memo-area">
        <textarea placeholder="Memo content" id="memo-content"></textarea>
    </p>
    <div role="toolbar">
        <ul>
            <li>
                <button id="delete-memo" class="icon-delete">Delete</button>
            </li>
        </ul>
    </div>
    <form id="delete-memo-dialog" role="dialog" data-type="confirm"
          class="hidden">
        <section>
```

```
                <h1>Confirmation</h1>
                <p>Are you sure you want to delete this memo?</p>
            </section>
            <menu>
                <button id="cancel-delete-action">Cancel</button>
                <button id="confirm-delete-action" class="danger">Delete</button>
            </menu>
        </form>
    </section>
```

At the top of the screen, represented by the `<header>` element, we have:

- a back button to return to the main screen,
- a text entry field that is used to hold the title of the given note,
- and a button that is used to share the note over email.

Below the top toolbar, we have a paragraph holding a `<textarea>` that holds the content of the note and then another toolbar with a trashcan button used to delete the current viewed note.

These three elements and their child nodes are the editing screen. After them we have a `<form>` that is used as a dialog box containing the confirmation screen that is presented to the user when he or she tries to delete a note. This dialog box is pretty simple, it only contains the text of the confirmation prompt and two buttons; one for deleting the note and another for canceling the action.

Now that we're closing this `<section>` we have all our screens implemented and the remaining HTML code is only there to include the JavaScript files and close the html file.

```
<script src="/js/model.js"></script>
<script src="/js/app.js"></script>
</body>
</html>
```

Crafting the JavaScript code

Now we're going to breathe life into our app by adding JavaScript. To better organize this code, I've divided the JavaScript code into two files:

- **model.js**: contains the routines to deal with storage and retrieval of notes but does not contain any app logic or anything related to the interface or data entry. In theory, we could reuse this same file in other apps that required text notes.
- **app.js**: attaches the HTML elements with their event handlers and contains the app logic.

Both files should be placed inside a **js** folder next to the **style** and **shared** folders.

model.js

We're going to use IndexedDB[66] to store our notes. Since we asked the *storage* permission on the app manifest we can store as many notes as we want - however, we should not abuse this! Firefox OS devices generally have very limited storage space, so you always need to be mindful of what data you store (users will delete and down-rate your app if it uses too much storage space!). And storing excessive amounts of data will have a performance penalty, which will make your app feel sluggish. Please also note that when you submit an application to the Firefox OS Marketplace, reviewers will ask you why you need unlimited storage space - if you can't justify why, your application will be rejected.

The part of the code from *model.js* that is shown below is responsible for opening the connection and creating the storage.

> Important: This code was written to be understood easily and does not represent the best practices for JS programming. Some global variables are used (I'm so going to hell for this) among other tidbits. The error handling code is basically non-existant. The main objective of this book is to teach the *workflow* of developing apps for Firefox OS and not teaching best JS patterns. That being said, depending on feedback, I will update the code in this book to better reflect best practices if enough people think it will not impact the beginners.

```
var dbName = "memos";
var dbVersion = 1;

var db;
var request = indexedDB.open(dbName, dbVersion);

request.onerror = function (event) {
    console.error("Can't open indexedDB!!!", event);
};
request.onsuccess = function (event) {
    console.log("Database opened ok");
    db = event.target.result;
};

request.onupgradeneeded = function (event) {

    console.log("Running onUpgradeNeeded");

```

[66]https://developer.mozilla.org/en-US/docs/IndexedDB/Using_IndexedDB

```
    db = event.target.result;

    if (!db.objectStoreNames.contains("memos")) {

        console.log("Creating objectStore for memos");

        var objectStore = db.createObjectStore("memos", {
            keyPath: "id",
            autoIncrement: true
        });
        objectStore.createIndex("title", "title", {
            unique: false
        });

        console.log("Adding sample memo");
        var sampleMemo1 = new Memo();
        sampleMemo1.title = "Welcome Memo";
        sampleMemo1.content = "This is a note taking app. Use the plus sign " +
                              "in the topleft corner of the main screen to " +
                              "add a new memo. Click a memo to edit it. All " +
                              "your changes are automatically saved.";

        objectStore.add(sampleMemo1);
    }
}
```

Important: Forgive me again for the globals, this is an educational resource only. Another detail is that I removed the comments from the source code to save space in the book. If you pick the source from GitHub you will get all the comments.

The code above creates a *db* object and a *request* object. The *db* object is used by other functions in the source to manipulate the notes storage.

On the implementation of the `request.onupgradeneeded` function we also create a welcome note. This function is executed when the application runs for the first time (or when the database version changes). This way once the application launches for the first time, the database is initialized with a single welcome note.

With our connection open and the storage initialized its time to implement the basic functions for note manipulation.

```
function Memo() {
    this.title = "Untitled Memo";
    this.content = "";
    this.created = Date.now();
    this.modified = Date.now();
}

function listAllMemoTitles(inCallback) {
    var objectStore = db.transaction("memos").objectStore("memos");
    console.log("Listing memos...");

    objectStore.openCursor().onsuccess = function (event) {
        var cursor = event.target.result;
        if (cursor) {
            console.log("Found memo #" + cursor.value.id +
                                    " - " + cursor.value.title);
            inCallback(null, cursor.value);
            cursor.continue();
        }
    };
}

function saveMemo(inMemo, inCallback) {
    var transaction = db.transaction(["memos"], "readwrite");
    console.log("Saving memo");

    transaction.oncomplete = function (event) {
        console.log("All done");
    };

    transaction.onerror = function (event) {
        console.error("Error saving memo:", event);
        inCallback({
            error: event
        }, null);

    };

    var objectStore = transaction.objectStore("memos");

    inMemo.modified = Date.now();

    var request = objectStore.put(inMemo);
    request.onsuccess = function (event) {
        console.log("Memo saved with id: " + request.result);
        inCallback(null, request.result);

    };
}

function deleteMemo(inId, inCallback) {
    console.log("Deleting memo...");
    var request = db.transaction(["memos"],
                  "readwrite").objectStore("memos").delete(inId);

    request.onsuccess = function (event) {
        console.log("Memo deleted!");
        inCallback();
    };
}
```

On the piece of code above we create a constructor function that creates new Memos with some fields already initialized. After that we implement functions for listing, saving and removing notes. Many of these functions receive a callback parameter called `inCallback` which is a function to be called after the function does its thing. This is needed due to the asynchronous nature of IndexedDB. All callbacks have the same signature which is `callback(error, value)` where one of the values is null depending on the outcome of the previous function.

Since this is a beginner book I've opted not to use *Promises*[a] since many beginners are not familiar with the concept. I recommend using such concepts to create easier to maintain code that is more pleasant to read.

[a]https://developer.mozilla.org/en-US/docs/Mozilla/JavaScript_code_modules/Promise.jsm/Promise

Now that our note storage and manipulation functions are ready, let's implement our app logic in a file called **app.js**.

app.js

This file will contain our app logic. Since the source code is too large for me to place it all at once on the book, I will break it in parts and explain each part piece by piece.

```
var listView, detailView, currentMemo, deleteMemoDialog;

function showMemoDetail(inMemo) {
    currentMemo = inMemo;
    displayMemo();
    listView.classList.add("hidden");
    detailView.classList.remove("hidden");
}


function displayMemo() {
    document.getElementById("memo-title").value = currentMemo.title;
    document.getElementById("memo-content").value = currentMemo.content;
}

function shareMemo() {
    var shareActivity = new MozActivity({
        name: "new",
        data: {
            type: "mail",
            body: currentMemo.content,
            url: "mailto:?body=" + encodeURIComponent(currentMemo.content) +
                        "&subject=" + encodeURIComponent(currentMemo.title)

        }
    });
    shareActivity.onerror = function (e) {
        console.log("can't share memo", e);
    };
}

function textChanged(e) {
    currentMemo.title = document.getElementById("memo-title").value;
    currentMemo.content = document.getElementById("memo-content").value;
    saveMemo(currentMemo, function (err, succ) {
        console.log("save memo callback ", err, succ);
        if (!err) {
            currentMemo.id = succ;
        }
    });
}

```

```
function newMemo() {
    var theMemo = new Memo();
    showMemoDetail(theMemo);
}
```

At the beginning we declare some global variables (yuck!!!) to hold references to some DOM elements that we want to use later inside some functions. The most interesting global is `currentMemo` which is an object that holds the current note that the user is reading.

The `showMemoDetail()` and `displayMemo()` functions work together. The first one loads the selected note into the `currentMemo` and manipulates the CSS of the elements so that the editing screen is shown. The second one picks the content from the `currentMemo` variable and places it on the screen. We could do both things on the same function but having them separate makes it easier to experiment with new implementations.

The `shareMemo()` function uses a WebActivity[67] to open the email application with a new message pre-filled with the selected notes content.

The `textChanged()` function picks the data from the entry fields and place them into the `currentMemo` object and then saves the note. This is done because the application is an `auto-save` app where your content is always saved. All alterations on the content or title of the note will trigger this function and the note will always be saved on the IndexedDB storage.

The `newMemo()` function creates a new note and opens the editing screen with it.

```
function requestDeleteConfirmation() {
    deleteMemoDialog.classList.remove("hidden");
}

function closeDeleteMemoDialog() {
    deleteMemoDialog.classList.add("hidden");
}

function deleteCurrentMemo() {
    closeDeleteMemoDialog();
    deleteMemo(currentMemo.id, function (err, succ) {
        console.log("callback from delete", err, succ);
        if (!err) {
            showMemoList();
        }
    });
}

```

[67] https://hacks.mozilla.org/2013/01/introducing-web-activities/

```
function showMemoList() {
    currentMemo = null;
    refreshMemoList();
    listView.classList.remove("hidden");
    detailView.classList.add("hidden");
}
```

The `requestDeleteConfirmation()` function is responsible for showing the note removal confirmation dialog.

The `closeDeleteMemoDialog()` and `deleteCurrentMemo()` are triggered by the buttons on the removal confirmation dialog.

The `showMemoList()` function does some clean up before showing the list of stored notes. For example, it cleans the content of `currentMemo` since we're not reading any memo yet.

```
function refreshMemoList() {
    if (!db) {
        // HACK:
        // this condition may happen upon first time use when the
        // indexDB storage is under creation and refreshMemoList()
        // is called. Simply waiting for a bit longer before trying again
        // will make it work.
        console.warn("Database is not ready yet");
        setTimeout(refreshMemoList, 1000);
        return;
    }
    console.log("Refreshing memo list");

    var memoListContainer = document.getElementById("memoList");


    while (memoListContainer.hasChildNodes()) {
        memoListContainer.removeChild(memoListContainer.lastChild);
    }

    var memoList = document.createElement("ul");
    memoListContainer.appendChild(memoList);

    listAllMemoTitles(function (err, value) {
        var memoItem = document.createElement("li");
        var memoP = document.createElement("p");
        var memoTitle = document.createTextNode(value.title);

        memoItem.addEventListener("click", function (e) {
            console.log("clicked memo #" + value.id);
            showMemoDetail(value);

        });

        memoP.appendChild(memoTitle);
        memoItem.appendChild(memoP);
        memoList.appendChild(memoItem);


    });
}
```

The `refreshMemoList()` function modifies the DOM by building element by element the list of notes that is displayed on the screen. It would be a lot easier to use some templating aid such as handlebars[68] or underscore[69] but since this app is built using nothing but *vanilla javascript* we're doing everything by hand. This function is called by `showMemoList()` that was shown above.

These are all the functions used by our app. The only part of the code that is missing is the initialization of the event handlers and the initial call of `refreshMemoList()`.

```
window.onload = function () {
    // elements that we're going to reuse in the code
    listView = document.getElementById("memo-list");
    detailView = document.getElementById("memo-detail");
    deleteMemoDialog = document.getElementById("delete-memo-dialog");

    // All the listeners for the interface buttons and for the input changes
    document.getElementById("back-to-list")
            .addEventListener("click", showMemoList);
    document.getElementById("new-memo")
            .addEventListener("click", newMemo);
    document.getElementById("share-memo")
            .addEventListener("click", shareMemo);
    document.getElementById("delete-memo")
            .addEventListener("click", requestDeleteConfirmation);
    document.getElementById("confirm-delete-action")
            .addEventListener("click", deleteCurrentMemo);
```

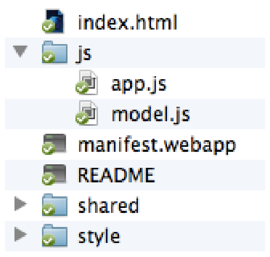
[68]http://handlebarsjs.com/
[69]http://underscorejs.org/

```
    document.getElementById("cancel-delete-action")
            .addEventListener("click", closeDeleteMemoDialog);
    document.getElementById("memo-content")
            .addEventListener("input", textChanged);
    document.getElementById("memo-title")
            .addEventListener("input", textChanged);

    // the entry point for the app is the following command
    refreshMemoList();

};
```

Now all files are ready and we can begin trying our application on the simulator.

Testing the app with the Web IDE

Before we try our application on the simulator we'd better check out if the files are in the correct place. Your memos folder should look like this one:

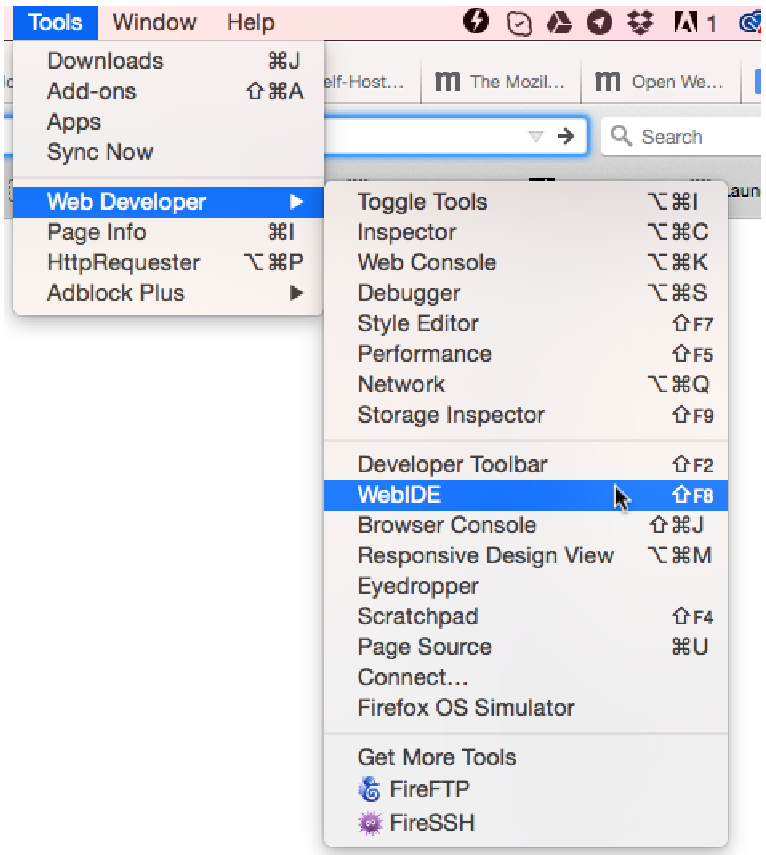


List of files used by Memos

If you have a hunch that you wrote something wrong, just compare your version with the one on the memos github repository[70] (There is also a copy of the source code in a folder called **code** on the book repository[71]).

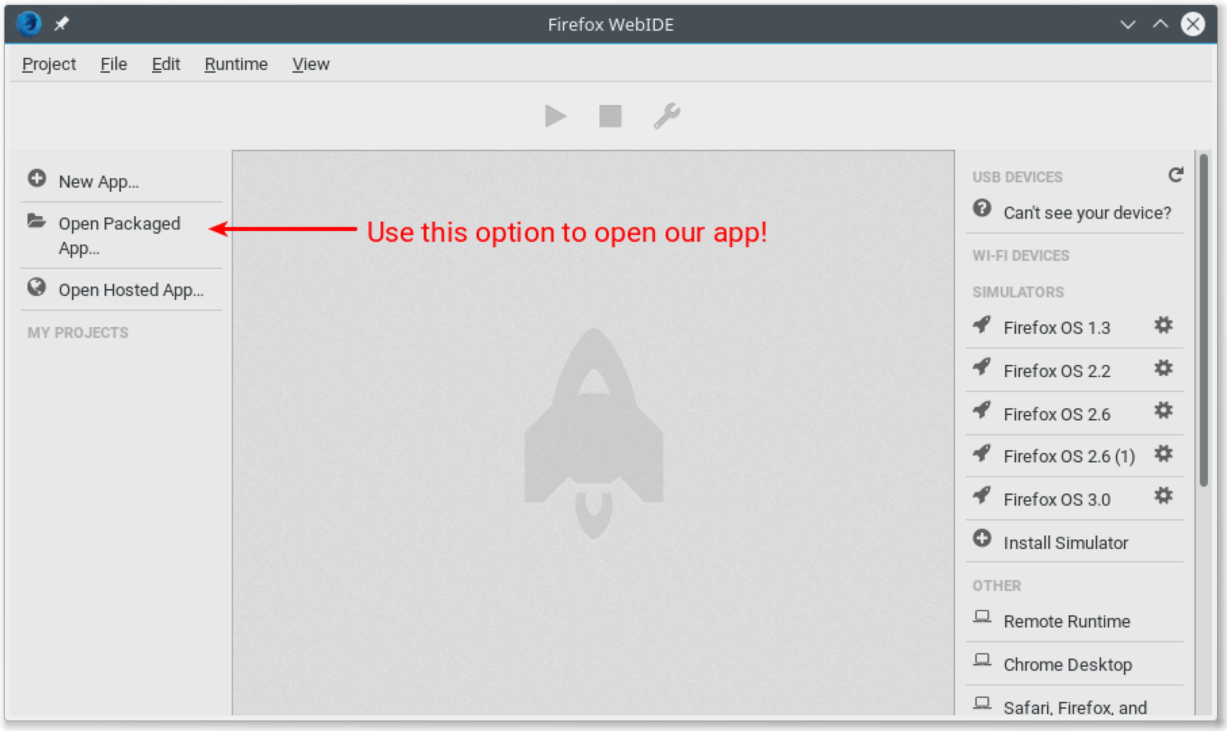
To open the *Web IDE* go to the menu for **Tools** -> **Web Developer** -> **Web IDE**.

[70]https://github.com/soapdog/memos-for-firefoxos
[71]https://github.com/soapdog/guia-rapido-firefox-os



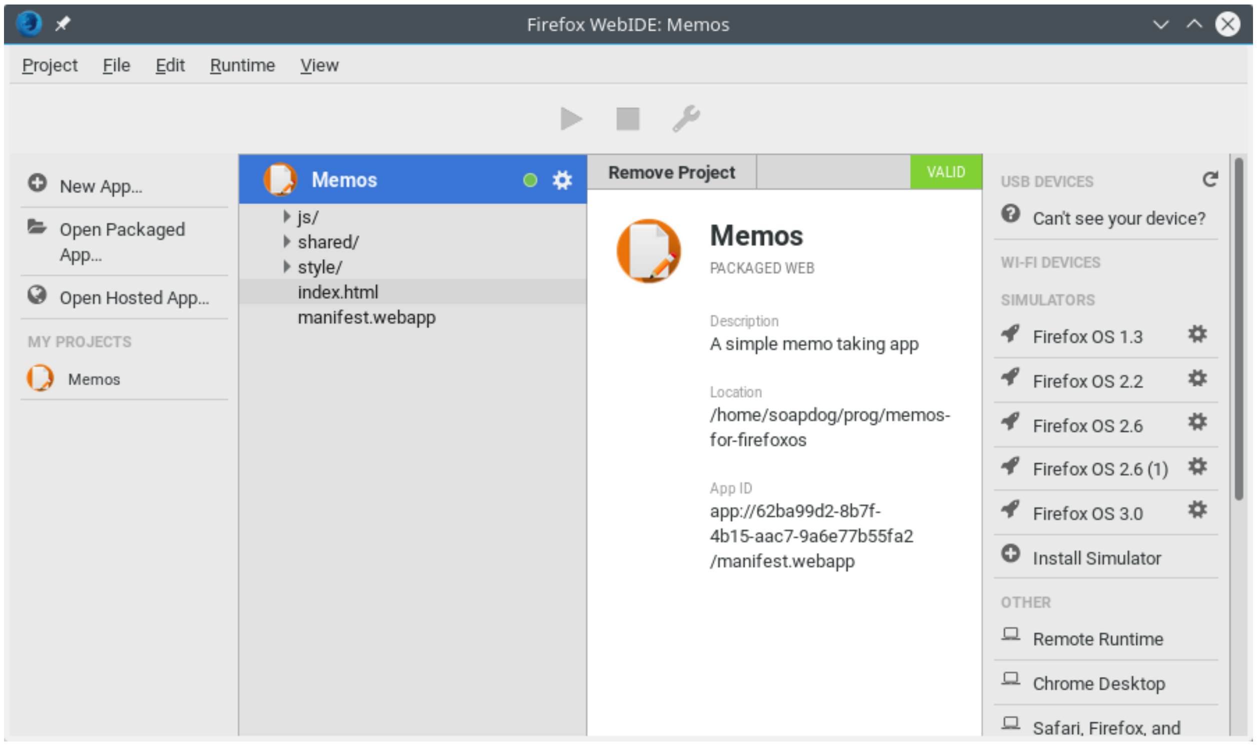
Where you can find the Web IDE

With the WebIDE open, click the **Add Packaged App** option on the **Apps tab** and browse to where you placed the memos files and select that folder.



Adding a new app

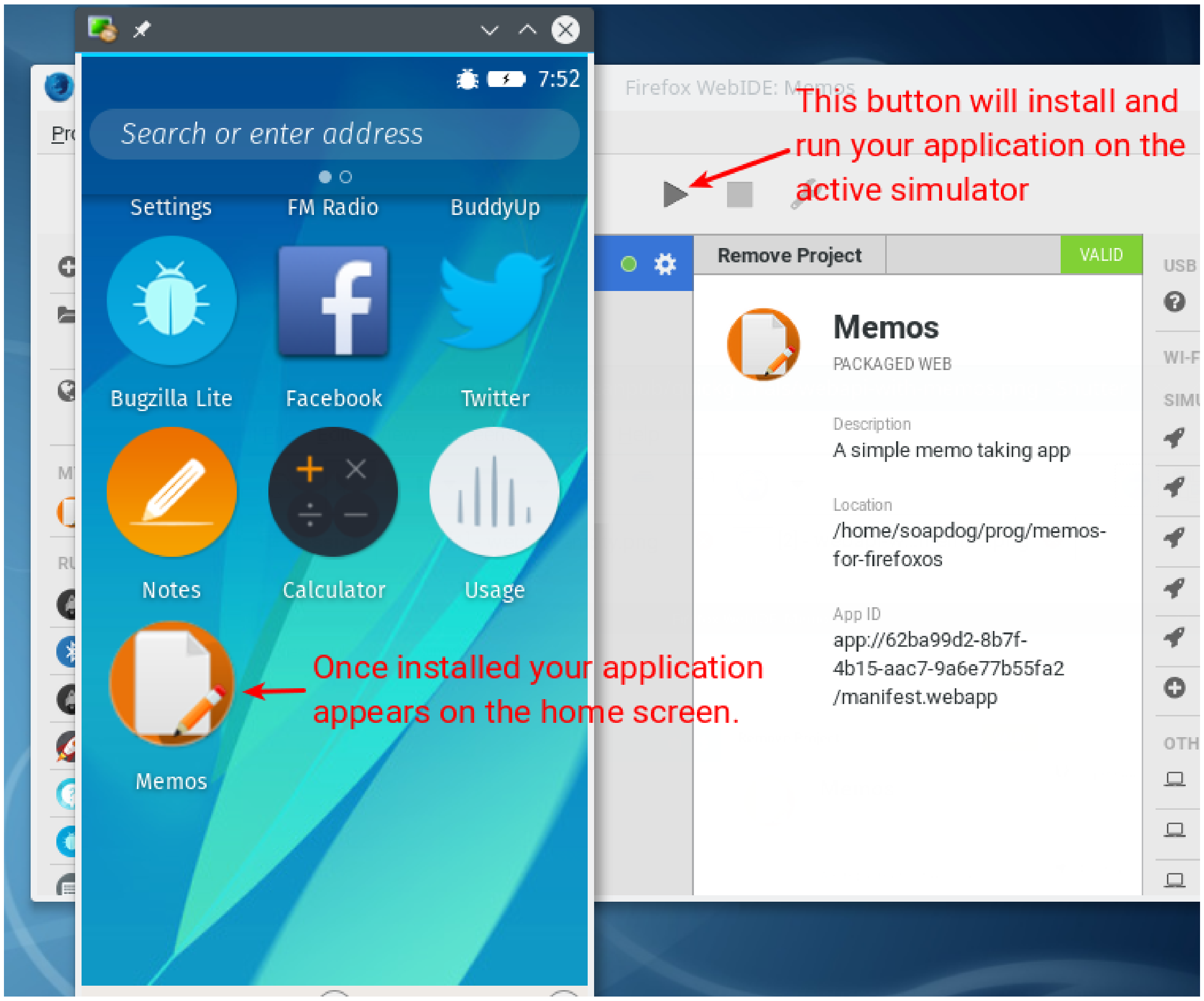
If everything works as expected you will see the Memos app on the list of apps.



Memos showing on the Web IDE

After adding your application, use the options on the right side of the WebIDE to run one of your installed simulators. If you haven't installed any simulator yet, I suggest you follow the instructions on screen and install them all.

With the Simulator running press the **Play** button on the memos listing on the **Web IDE** to install memos on the running Simulator. After the installation the memos icon will appear at the Simulator home screen. You can just click it to run your app.


Firefox WebIDE: Memos
This button will install and run your application on the active simulator
7:52
Search or enter address
Settings
FM Radio
BuddyUp
Bugzilla Lite
Facebook
Twitter
Notes
Calculator
Usage
Memos
Once installed your application appears on the home screen.
Remove Project
VALID
Memos
PACKAGED WEB
Description
A simple memo taking app
Location
/home/soapdog/prog/memos-for-firefoxos
App ID
app://62ba99d2-8b7f-4b15-aac7-9a6e77b55fa2/manifest.webapp
USB
WI-F
SIMU
OTH

Memos installed on the Simulator

Congratulations! You created and tested your first app. It's not a complex or revolutionary app - but I hope it helped you understand the development workflow of Firefox OS. As you can see, it's not very different from standard Web development.

Remember that whenever you alter some of the source files you need to press the **Reload** button to update the copy of the app that is stored on the running Simulator.

Summary

In this chapter we built our first application for Firefox OS and saw it running on the simulator. In the next chapter we're going to check out the developer tools that comes bundled with Firefox, they will make your life a lot easier when developing applications.

Developer Tools

Firefox has many tools to help web developers do their job. Many people are still using FireBug[72] and haven't realize that Firefox now has its own built-in tools. In this chapter we're going to review the tools that are most useful for developing apps for Firefox OS.

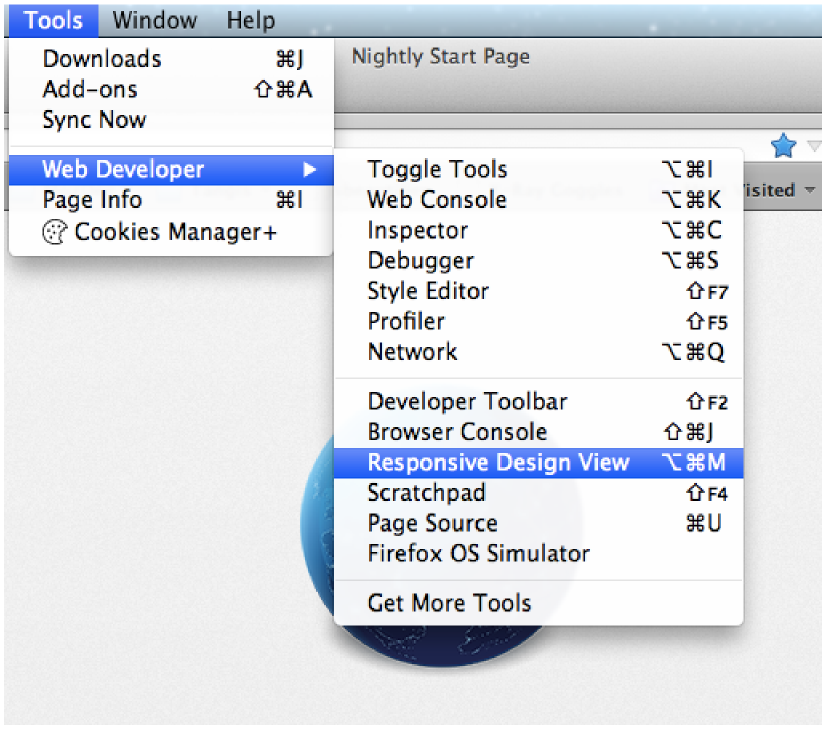
If you're interested in learning more about these tools, and what other dev tools goodness is about to land in Firefox, check out the developer tools[73] page on Mozilla's Developer Network (really, go check that link! I will wait).

Introducing the Responsive Design View

A common workflow in Web development is changing a HTML file and then reloading the page in the browser to see what's changed. Unless you are using something like Grunt or Volo, generally there won't be the need for a compilation step or similar. Even though the Firefox OS Simulator permits you to use that same workflow, the emulator is currently restricted to one resolution (480x320). This is less than ideal if you are also designing your application to work on tablets, phablets, giant TVs, or anything in between.

To check how your app will look on any screen resolution, you can use Firefox's **Responsive Design View** tool to change the screen (and viewport). It can be enabled by going to the **Tools menu** -> **Web Developer** -> **Responsive Design View** as shown in the image below. When you activate this tool, the window will change so that you can alter the viewport size using the drag corners or the selection box.

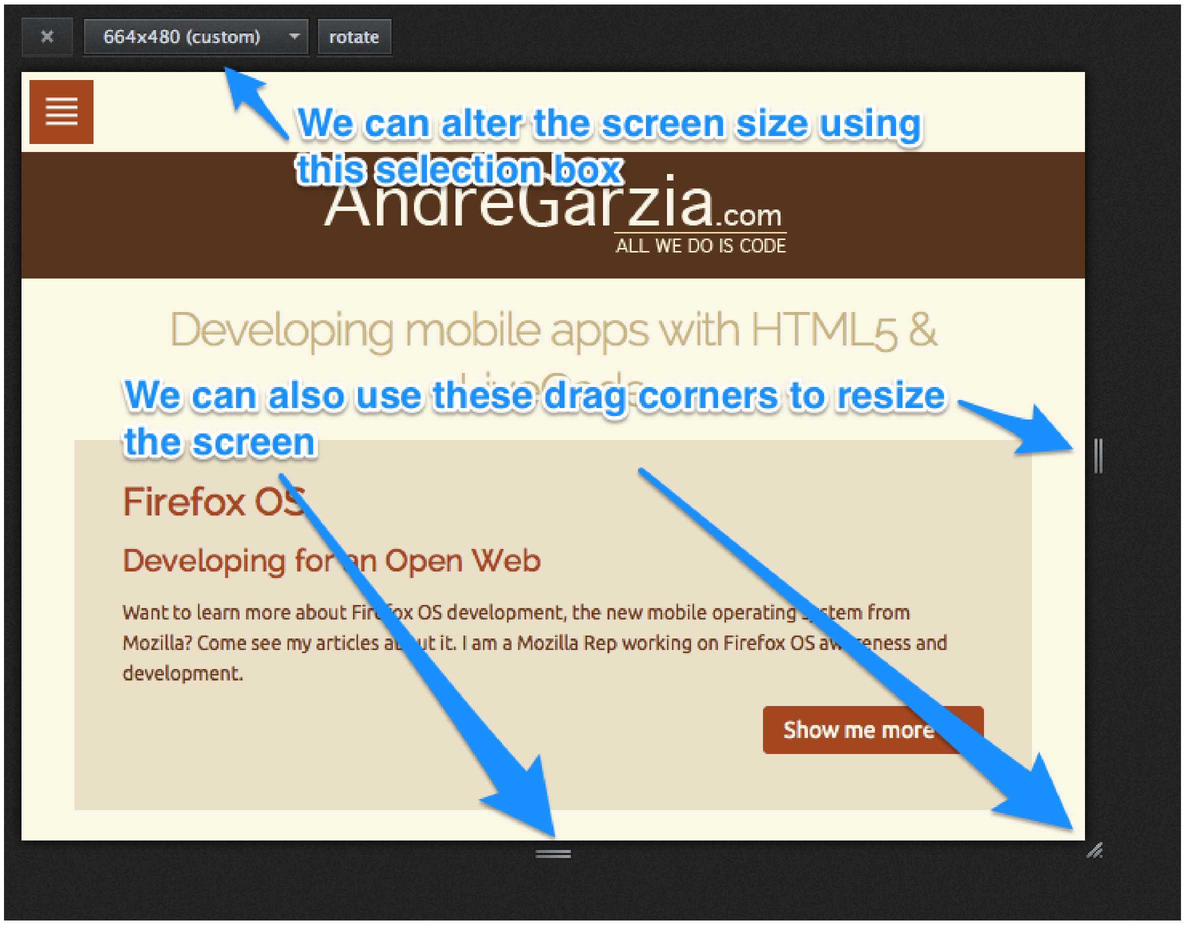
[72] https://addons.mozilla.org/pt-BR/firefox/addon/firebug/
[73] https://developer.mozilla.org/en-US/docs/Tools



Activating Responsive Design View

Using the responsive design view is specially useful for testing out **media queries**[74], as it allows you to resize the screen and see how your site responds to changes in layout in real time. Another great feature of the Responsive Design View is that you can save predefined sizes. If you know what viewport sizes you are targetting, then your can quickly check different viewport sizes without needing to resize the actual browser window.

[74]https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries


664x480 (custom)
rotate
We can alter the screen size using this selection box
AndreGarzia.com
ALL WE DO IS CODE
Developing mobile apps with HTML5 &
We can also use these drag corners to resize the screen
Firefox OS
Developing for an Open Web
Want to learn more about Firefox OS development, the new mobile operating system from Mozilla? Come see my articles about it. I am a Mozilla Rep working on Firefox OS awareness and development.
Show me more

Responsive Design View Sample

At the time of writing, most of the Firefox OS phones that are on the market are running on a 480x320 resolution - and at a pixel density of about 96 pixels-per-inch. However, you should expect this to change as new Firefox OS hardware becomes available as time goes on: screens will likely pack more pixels and have higher pixel densities (just like Apple's retina displays).

To future proof your app, don't hard-code your CSS to any resolution or pixel density. Instead, you should use media queries and the responsive design methodology to create apps that adapt to any screen size. To learn more about responsive design, I recommend the following books Responsive Web Design[75] and Mobile First[76].

In summary, the responsive design view allows us to test our web applications using many different screen sizes without the need to resize the Firefox browser window itself. In my humble opinion this is one of the most useful web developer tools available - but it has one big limitation: it does not currently allow you to test different pixel densities (i.e. to see what your site would look like on a "retina" display or better).

[75]http://www.abookapart.com/products/responsive-web-design
[76]http://www.abookapart.com/products/mobile-first

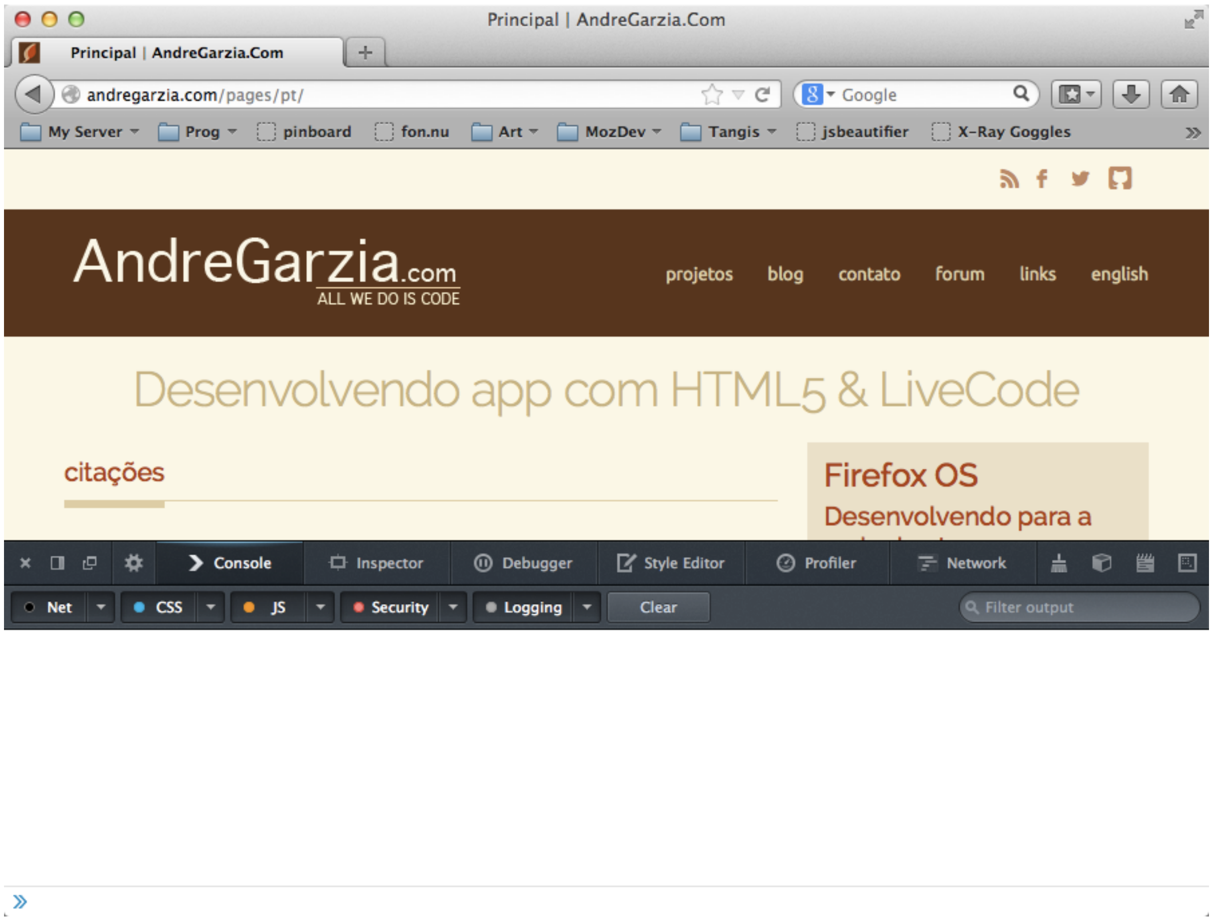
Developer tools

Firefox's developer tools are similar to FireBug and the tools available in other modern browsers. Using these tools you can execute and debug your JavaScript using the console[77], and manipulate both the DOM and the CSS on the current page.

To bring up the console, you can either:

```
* Go to "Tools menu > Web Developer > Web Console".
```

- right-click on the page you want to debug, and select "Inspect Element", then click on "Console" tab.



JavaScript Console

[77]https://developer.mozilla.org/en-US/docs/Web/API/console

Besides the *JavaScript Console* there are many other tools available such as *the style editor*[78], *the network monitor*[79], *the JavaScript profiler*[80], *the JavaScript debugger*[81], *the page inspector*[82] and many others.

In the application we've built in the previous chapter, we used the console to check the progress of our application. This is a pretty powerful way to debug our apps - but some developers are still using `alert()` all over their JavaScript code as their "debug tool".

Using `alert()` is really bad because if one forgets to remove any `alert()`s, it's the user's who will ultimately pay the price. Using the console avoids this problem as it harmlessly (and silently!) routes all messages to a place that user's don't normally access - so it doesn't disrupt the user experience. Using the console also means you don't need to remove your console messages from your code, unless you really want to. This can help with code maintenance and debugging if things do go wrong (as they generally do with any software!).

Learning how to properly use the developer tools bundled with Firefox (or whatever browser you're using) is an important step in becoming a better developer. That's why I advise everyone to check the links above and get more familiar with the various tools available in Firefox.

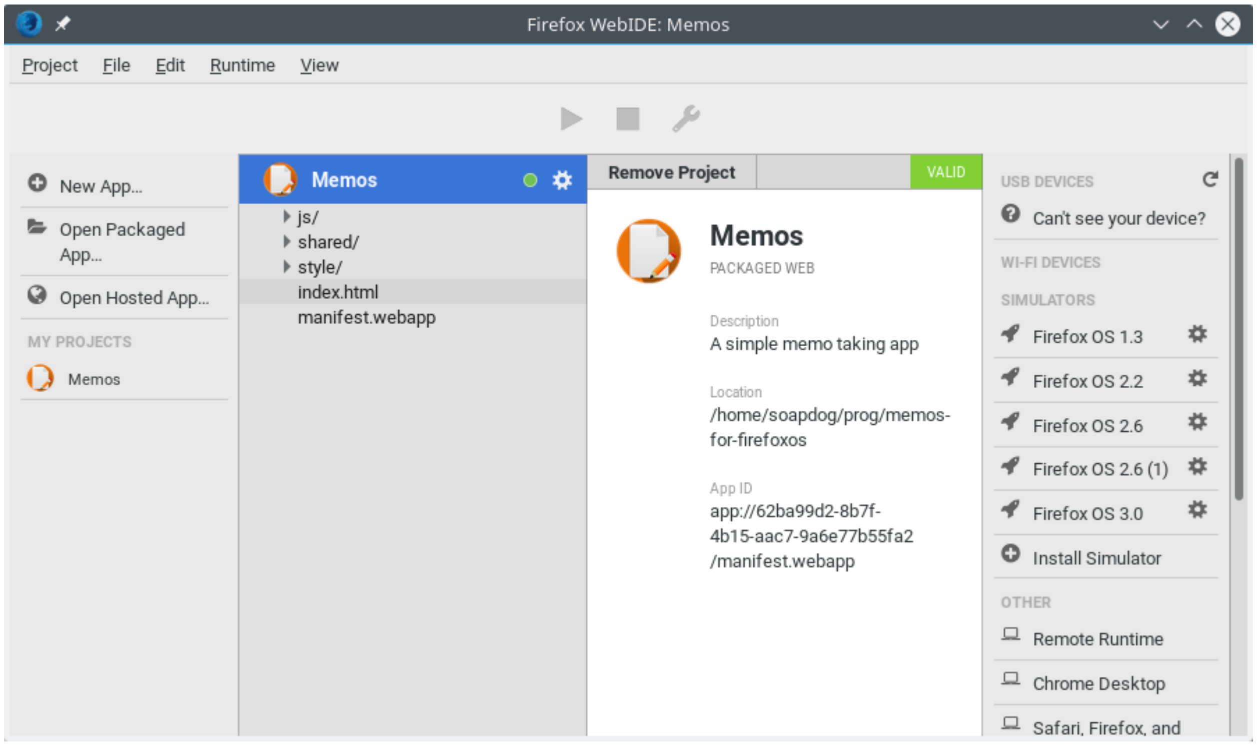
One special tool that was not mentioned above is the *remote debugger*[83]. That tool allows us to connect to a phone running Android or Firefox OS, and use the developer tools to debug the code that is running on the device in real time.

Summary

This chapter provided a brief tour of the developer tools that come bundled with Firefox. Using these tools will make your development process easier, specially when you use them together with the Firefox OS simulator. They are an indispensable combination for putting together an application. In the next chapter we're going to learn more about the simulator and how to make the best use of it.

[78]https://developer.mozilla.org/en-US/docs/Tools/Style_Editor
[79]https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor
[80]https://developer.mozilla.org/en-US/docs/Tools/Profiler
[81]https://developer.mozilla.org/en-US/docs/Tools/Debugger
[82]https://developer.mozilla.org/en-US/docs/Tools/Page_Inspector
[83]https://developer.mozilla.org/en-US/docs/Tools/Remote_Debugging

The WebIDE



Memos
PACKAGED WEB

Description
A simple memo taking app

Location
/home/soapdog/prog/memos-for-firefoxos

App ID
app://62ba99d2-8b7f-4b15-aac7-9a6e77b55fa2/manifest.webapp

The WebIDE

We've setup the WebIDE in the chapter about preparing the environment and we used it on the chapter about building our first app. Now we're going to take a deeper look into the WebIDE features and learn how to do the most common tasks.

To learn more about it, check out the WebIDE page[84] on MDN.

Remember: that if you are using a device running Firefox OS 1.1 or older then you need to use the Firefox OS 1.1 Simulator extension with it and not the WebIDE. This Simulator is explained in Appendix 2: The Firefox OS Simulator.

Adding Apps

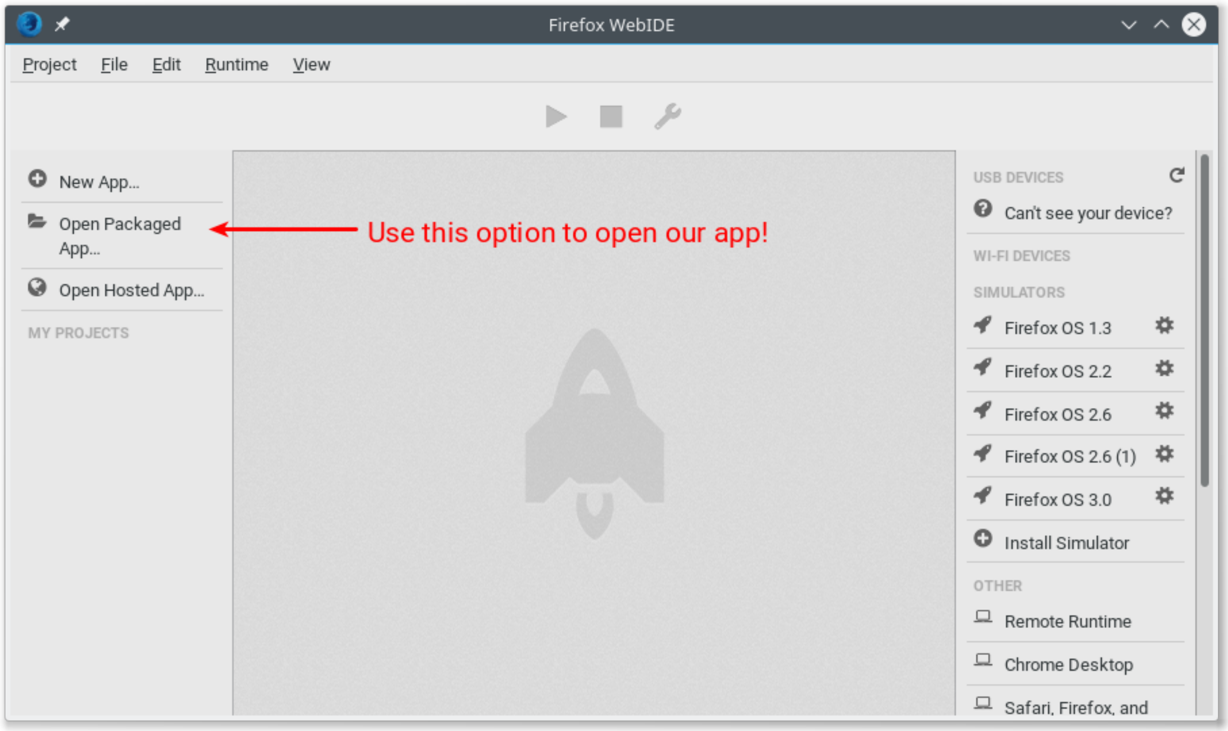
You can add both hosted and packaged apps to the WebIDE. Lets see how to add each type of app:

[84]https://developer.mozilla.org/en-US/docs/Tools/WebIDE

Adding packaged apps

You already saw how to add packaged apps to the WebIDE during our first app creation, but we're going to do a recap so I can show you what else is possible.

To add a new packaged application click the **Open Packaged App** button on the **WebIDE main screen** as shown in the screenshot below.



Showing the *Open Packaged App* option that adds a packaged app to the WebIDE

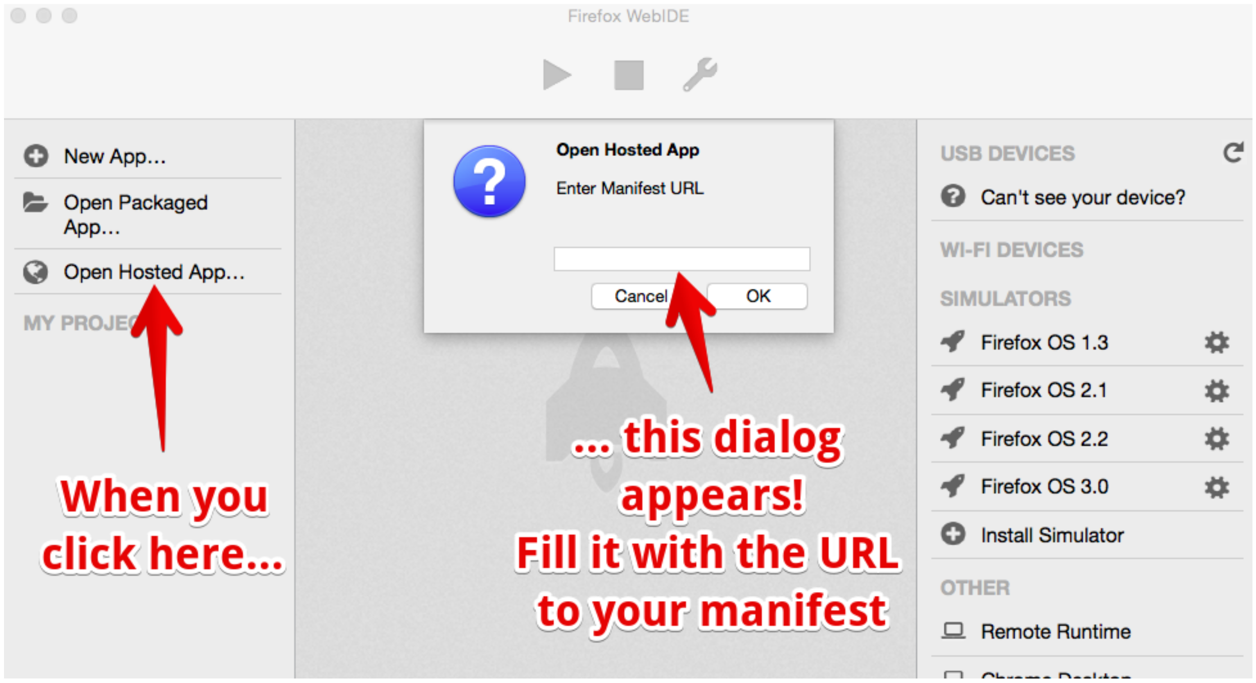
When you click on the button highlighted on the image, Firefox opens a file selection dialog. You should browse your hard drive and select the **folder that contains the manifest file** for the application that you want to add to the WebIDE. If there are no issues with your manifest then your app will be added to the list on screen.

Adding hosted apps

If you're building a hosted app then you should test it by using a web server. Do not try to use the method described above for hosted apps because you may miss some errors that will only happen on a hosted environment - such as serving the manifest with the wrong *MIME type*. Note that the simulator won't warn you about things like incorrect MIME types, but it's important to get such things right if you submit your app to the Mozilla Marketplace.

Most of the hosted apps are not applications built exclusively for Firefox OS but responsive design based websites that are able to adapt themselves to different devices and resolutions. These web apps usually have a complex backend that needs to be in-place for the application to work and that's why you need to test the app using a real web server running your backend stuff.

To run your app in the simulator, click the **Open Hosted App** button on the **WebIDE main screen** and then fill the URL of your application in the text entry box and click the **OK** button on that dialog box.



Adding a hosted app to the WebIDE

After clicking the button, the manifest is verified and if it is correct the application will be added and the WebIDE.

Running your app

Before you can run your application, you need to start your selected simulator. Choose one of your installed options listed on the right side of the **WebIDE main screen**

Once you have a Simulator running you can click the **Play** button located at the top of the WebIDE window and it will install and run the app on the active Simulator.

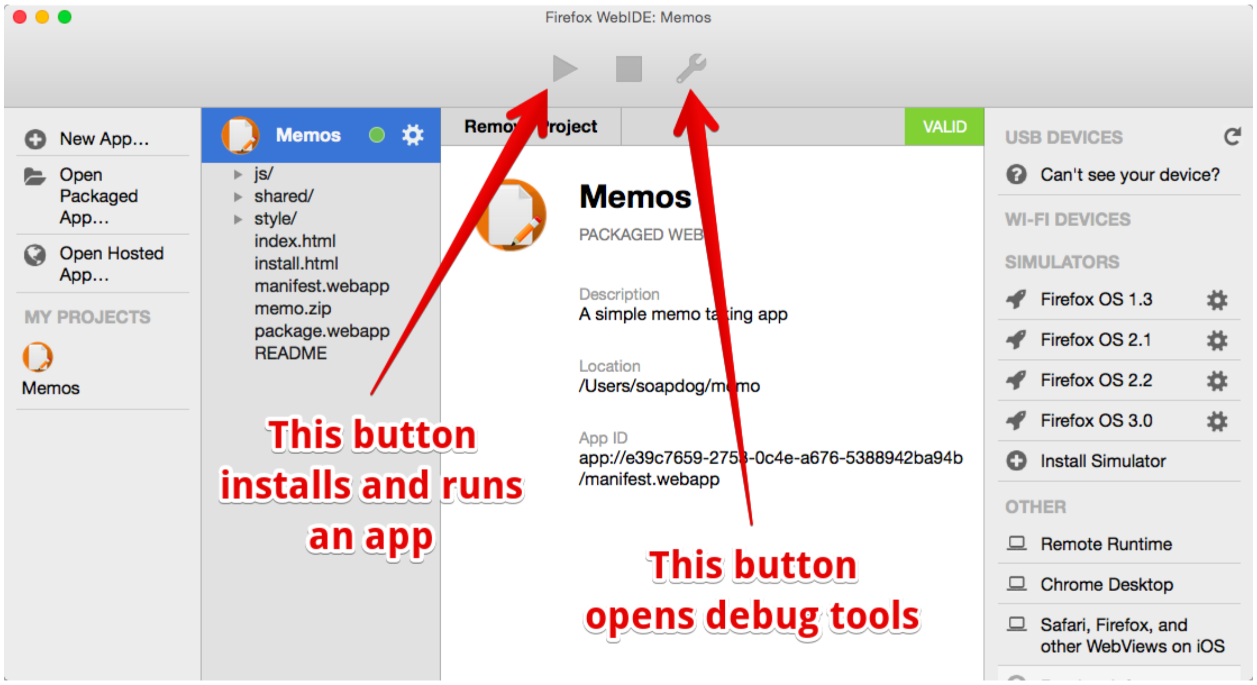
The application icon will appear at the home screen of the Simulator once the installation is complete. You can just click it to run.

Updating your app

Every time you change some of your files and want to test again on the Simulator you need to press the **Reload** button to update the installation of your app on the active Simulator.

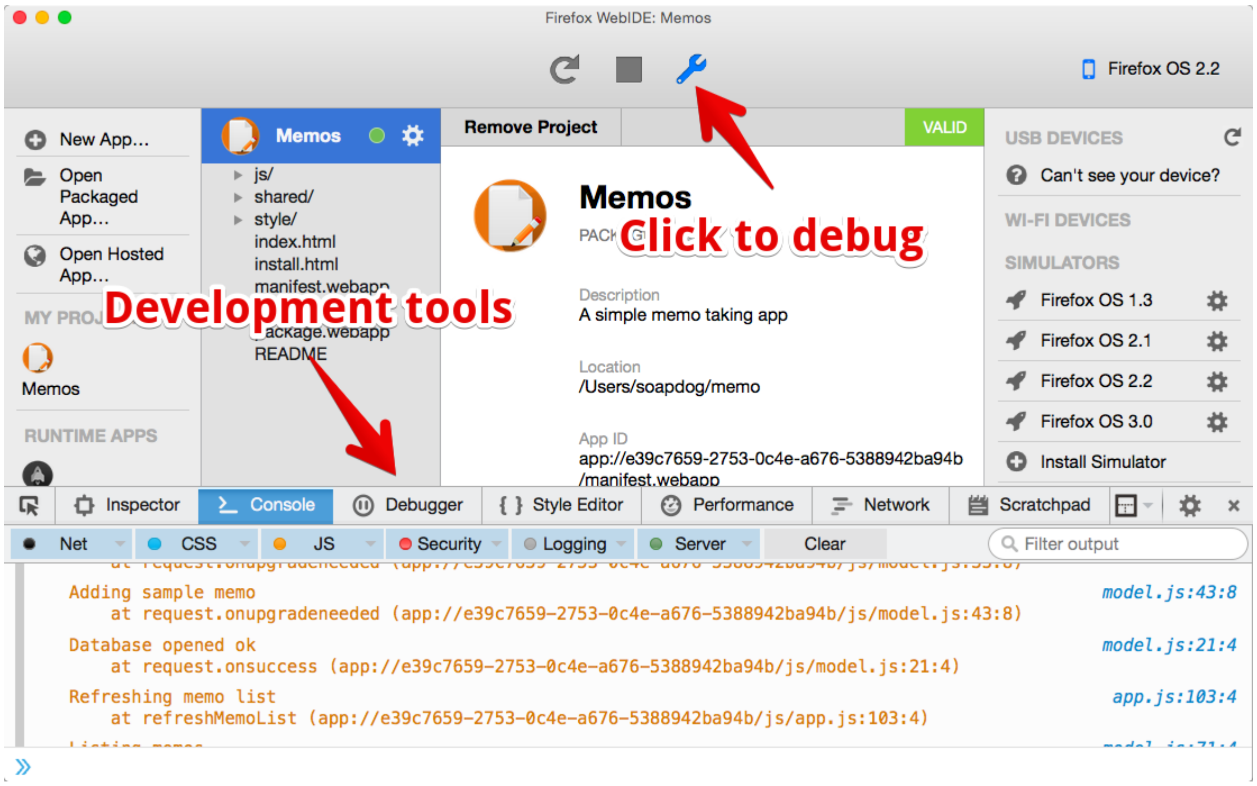
Debugging

After the application is added to the active Simulator we're able to debug it by clicking the **Wrench** button in the WebIDE window. This will launch the your app on the running Simulator and open a **JavaScript Console** connected to your app.



What button to press

After pressing this button you will see a screen like this:



Memos

Description
A simple memo taking app

Location
/Users/soapdog/memo

App ID
app://e39c7659-2753-0c4e-a676-5388942ba94b
/manifest.webapp

```
Adding sample memo                                                                          model.js:43:8
    at request.onupgradeneeded (app://e39c7659-2753-0c4e-a676-5388942ba94b/js/model.js:43:8)
Database opened ok                                                                          model.js:21:4
    at request.onsuccess (app://e39c7659-2753-0c4e-a676-5388942ba94b/js/model.js:21:4)
Refreshing memo list                                                                        app.js:103:4
    at refreshMemoList (app://e39c7659-2753-0c4e-a676-5388942ba94b/js/app.js:103:4)
```

Developer Tools connected to the app running on the simulator

With the tools connected to your app you can test your JavaScript, debug your DOM, edit styles, etc. Like those startup guys like to say: *pivot until your app is good.*

Once your app is running well on the simulator it's time to test on a real device.

Testing apps on a real device

Nothing replaces testing on a real device. On the simulator, you test things by using a mouse and clicking on a computer screen; while on a real device you use your fingers on a touchscreen and by using physical buttons. Its a very different user and development experience.

As an example why this type of testing matters, let me tell you a brief story: Many years ago, Raphael Eckhardt (the designer who created the cover of this book) and I were building a puzzle game not that disimilar to Bejeweled. Our game involved dragging and dropping some pieces on a board and was working pretty well on the simulator.

When we then tested the game on an actual phone we realized our game components were not touch friendly at all. When placing a hand over the screen the board would be hidden below the hand. Even worst, the pieces the users were supposed to drag were smaller than the user's finger tip, so the user couldn't see what they were doing! In summary, our UX sucked very badly. That happened because we kept trying things only on the simulator using a mouse that had a tiny cursor. When we decided to try with our fatter-than-a-cursor fingers we realized that we need to rework our UI.

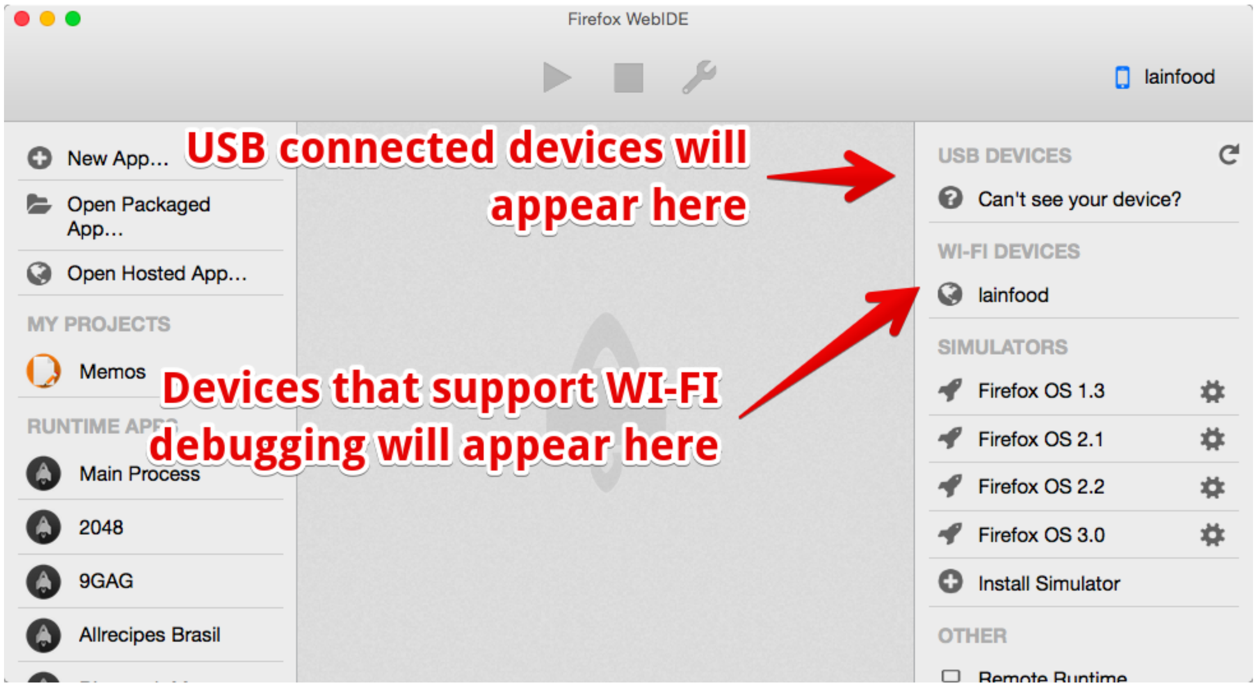
To avoid having a similarly frustrating experience, always test on a real device… or two, or more if you can get your hands on multiple devices. Test often with simple prototypes. Otherwise, you can waste valuable time and money having to recreate assets.

Tip: Many cities have community run device labs where you can go and test your app in multiple devices. You can check for *Open Device Labs* near you at https://opendevicelab.com/[a].

[a] https://opendevicelab.com/

Connecting with a Firefox OS device

If you have a Firefox OS device (and have any needed drivers installed) then you can push apps directly from the WebIDE to the device using a USB connection of a WI-FI connection depending on which version of Firefox OS you are running. In both cases you need to go to the **Settings app on the device** -> **Developer** and mark the option to debug using **ADB and DevTools**. If your device supports it then you can also mark the option to allow **WI-FI debugging**. Once your device is recognized, it will appear on the right side area of the **WebIDE main screen**



Device Connected!

Clicking on a device (such as *lainfood* on that screenshot) will connect the WebIDE to it. After that you can use the normal **Play**, **Stop and Debug** buttons to interact with your app and your device.

Summary

In summary, the WebIDE is fantastic. Its much better than the old Firefox OS 1.1 Simulator Extension since it has better developer tools and can run multiple Firefox OS versions. We can envision the WebIDE getting better and better with its built-in editors and more.

Besides feeling awesome and empowered, by this point in the book you hopefully have a good grasp of the workflow for building apps for Firefox OS.

In the next chapter we'll talk about distributing your application. It is time to spread your app around the world!

Distributing Your Apps

Now that our application is ready we need to figure out a way to get it to our users. In the introduction chapter I mentioned that, unlike Apple, Mozilla does not force you to use their distribution channels - we're free to spread our creations as we wish. In this chapter we're going to learn how to distribute our app **outside the Firefox Marketplace**[85].

In my humble opinion, distributing your application outside the Mozilla Marketplace makes sense in the following two situations.

1. You're developing an application for internal use within your company, or to a restricted/limited group of users. If you ship it to the marketplace then it will be available to anyone and if you want to restrict the usage of the app to a group of people then you will need some kind of authentication scheme with a server backend or something similar. For example, when the *Evernote* application is launched for the first time, it asks the user to log in their servers.
2. You already have a huge user-base that you can tap into for your app distribution. An example of this would be a news paper, like the *Financial Times*, which can simply distribute their app on their own website and reach most of their users. Remember that you can distribute your application outside the marketplace and in the marketplace at the same time, so if you already have your own marketing channel you can leverage that while still using the marketplace for reaching new users outside your own channel.

The distribution process for hosted and packaged apps is similar, but it uses different functions. Thats why I'm discussing them separately. Regardless if your app is hosted or packaged, the workflow is usually the same: you provide a button or link on your own home page that says something similar to **Click to Install Our App**, or you use a special URL that when launched causes the installation routine to run. In both cases, a dialog is presented to the user asking him or her to confirm that they want to install the given app.

Hosted Apps

[85]http://marketplace.firefox.com

Code for hosted app installation

```
var installapp = navigator.mozApps.install(manifestURL);
installapp.onsuccess = function(data) {
  // An App was installed.
};
installapp.onerror = function() {
 // An App was not installed, more information at
 // installapp.error.name
};
```

In the sample above `manifestURL` contains the address for the manifest file. When this code runs, the system asks the user to confirm his desire to install the given application and depending on the choice of the user it runs the success or the error callback.

To learn more about this API check the MDN page about application installation[86].

Packaged Apps

Packaged app installation is similar but instead of calling `mozApps.install()` we call `mozApps.installPackage()` as shown in the sample code below.

Code for packaged app installation

```
// Absolute url to package:
var packageURL = "http://myapp.example.com/myapp.zip";
var installapp = navigator.mozApps.installPackage(packageURL);
```

Warning: I have the impression that packaged app installation outside of the marketplace is not possible on Firefox OS version 1.0.1. Even though the API is documented, I have never tried it. Please if you try it, send me feedback so that I can update this book.

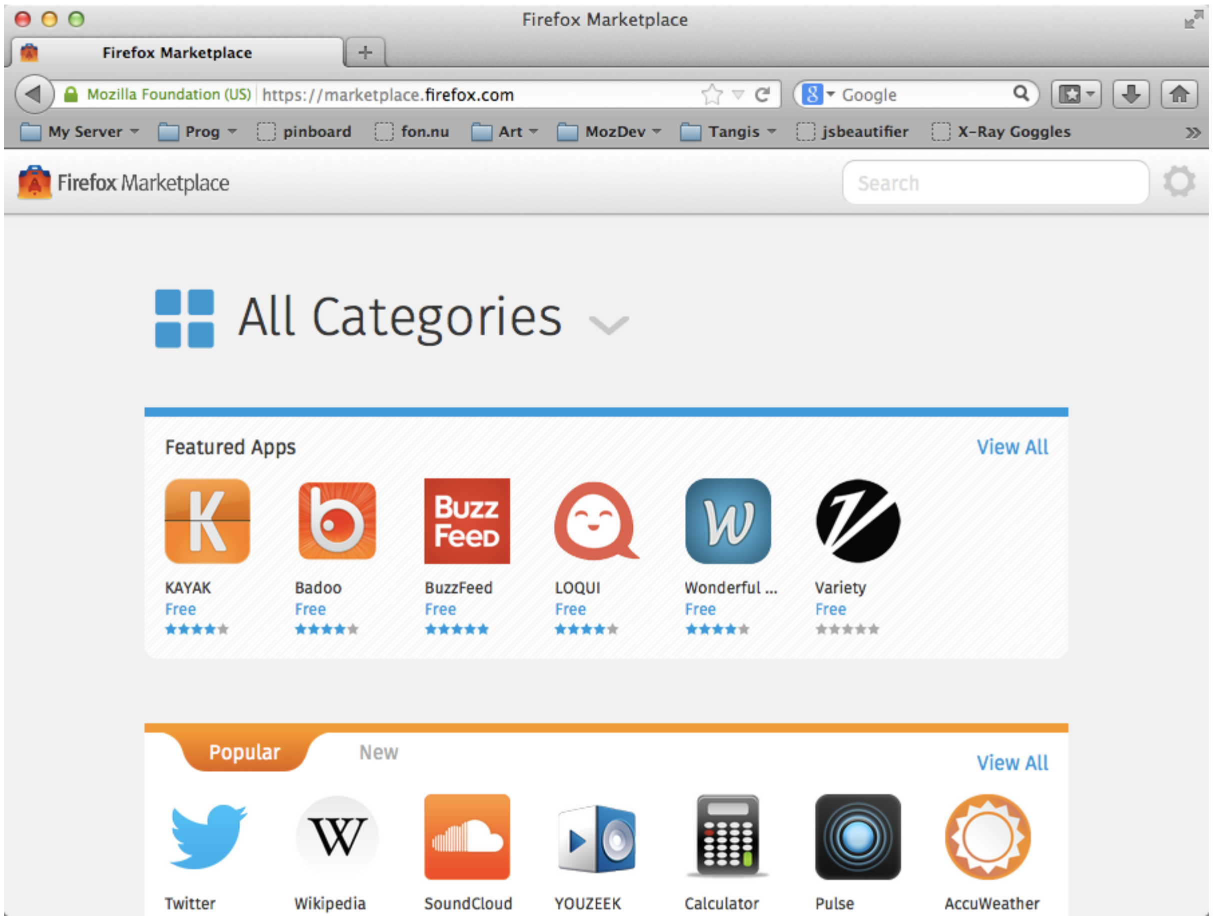
Summary

This chapter discussed options for distributing applications outside of the Firefox Marketplace by using the installation and management APIs for *Open Web Apps*. There are many other routines available to do things such as checking if your application is installed (so that you can hide that *Click Here To Install* button). To learn more about those APIs check out the MDN page about application installation[87] (yes, gave you this link before - this time, click it! There is important stuff there).

In the next chapter we're going to learn how to distribute our apps through the Firefox Marketplace.

[86]https://developer.mozilla.org/docs/Apps/JavaScript_API
[87]https://developer.mozilla.org/docs/Apps/JavaScript_API

The Firefox Marketplace



Firefox Marketplace

The Firefox Marketplace[88] is the online shop where you can buy or download applications for Firefox OS, Firefox, and Firefox For Android. This is the main channel for distributing Firefox OS applications, but you're not required to use it. If you want to distribute things outside the marketplace, read the previous chapter.

To place your apps on the marketplace you need to be identified via Mozilla Persona[89]. Just click **Sign Up** and follow the instructions. Once you're identified, you will be ready to submit apps to the Firefox Marketplace.

[88] http://marketplace.firefox.com
[89] https://login.persona.org/about

Checklist before even thinking about sending an app to the marketplace

All applications that are submitted to the marketplace go through an approval process (less scary than it sounds!). Hosted web applications go through a lighter process than privileged apps because they use less sensitive APIs. Before sending your application to the marketplace check out the marketplace review criteria[90]. The most important parts are (IMHO):

- Firefox OS devices do not have a **back button** like Android and your desktop browser. If the user navigates to a screen inside your app where there is no way for them to get back to the previous place (i.e. the user gets stuck), your app will be rejected.
- Your app should have a 60x60 icon and clear descriptions.
- Your app should do what the description says. Saying one thing and providing something else will get your app rejected.
- If your app asks for a given permission then you should use it somewhere in your code. Flagging your application as a privileged app and not using any privileged app API will cause your app to be rejected with a request that you submit again as a plain app.
- Your application needs to have a *privacy policy* in place.
- Manifest files should be served with the correct MIME type and come from the same domain as the app for hosted apps.

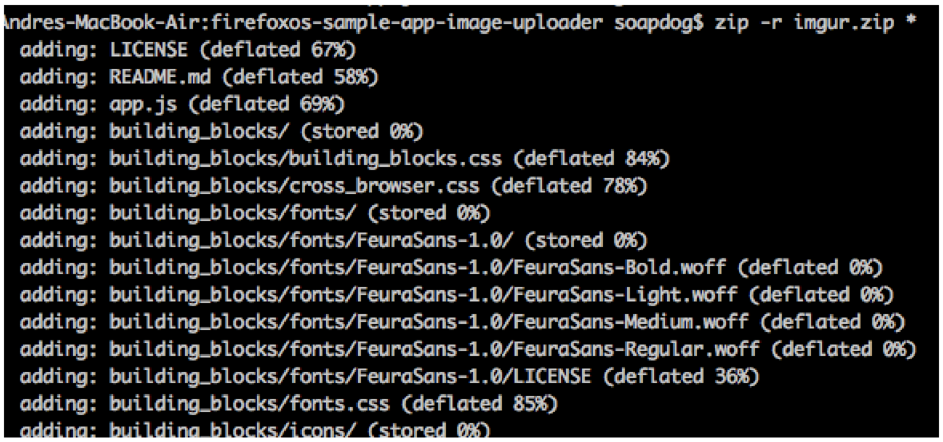
There are other criteria discussed in the link above - and the rules can change without notice. It will be worth your time to read that page. Getting your application rejected because of small stuff that is easy to fix is a huge waste of time. Better get things right the first time (reviewers love to approve good apps!).

Preparing your app for submission

The steps required to submit your application to the marketplace are different depending on whether it's a hosted or a packaged app. For a hosted app, it just needs to be accessible on the Internet with the correct MIME type and manifest in place. Packaged apps need to be compressed using *zip* and deserve some extra attention.

Many developers make the mistake of selecting the folder containing the application files and zipping it. This causes the zip file to contain a folder and this folder to contain the app. This is not the correct way to zip a Firefox OS application. The correct way is to zip the files and folders needed so that the manifest is on the *root level* of the zip file. On Mac OS X and Linux you can use the terminal to navigate to your application folder and use a command such as `zip -r myapp.zip *` to compress things correctly as shown on the screenshot below.

[90] https://developer.mozilla.org/en-US/docs/Web/Apps/Publishing/Marketplace_review_criteria



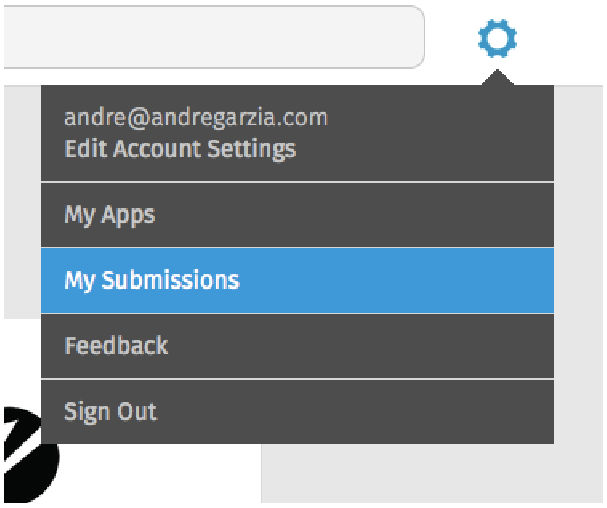
```
ndres-MacBook-Air:firefoxos-sample-app-image-uploader soapdog$ zip -r imgur.zip *
  adding: LICENSE (deflated 67%)
  adding: README.md (deflated 58%)
  adding: app.js (deflated 69%)
  adding: building_blocks/ (stored 0%)
  adding: building_blocks/building_blocks.css (deflated 84%)
  adding: building_blocks/cross_browser.css (deflated 78%)
  adding: building_blocks/fonts/ (stored 0%)
  adding: building_blocks/fonts/FeuraSans-1.0/ (stored 0%)
  adding: building_blocks/fonts/FeuraSans-1.0/FeuraSans-Bold.woff (deflated 0%)
  adding: building_blocks/fonts/FeuraSans-1.0/FeuraSans-Light.woff (deflated 0%)
  adding: building_blocks/fonts/FeuraSans-1.0/FeuraSans-Medium.woff (deflated 0%)
  adding: building_blocks/fonts/FeuraSans-1.0/FeuraSans-Regular.woff (deflated 0%)
  adding: building_blocks/fonts/FeuraSans-1.0/LICENSE (deflated 36%)
  adding: building_blocks/fonts.css (deflated 85%)
  adding: building_blocks/icons/ (stored 0%)
```

Correctly zipping the files

This zip file is what we send to the marketplace.

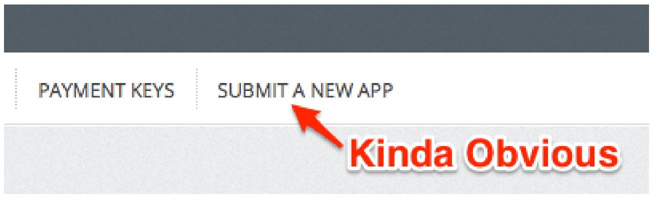
Submitting your app to the marketplace

Now with your application ready, and with the firm knowledge that it meets the review criteria, its time we send it to the marketplace. To do so browse to **My Submissions** using the gear button on the top of the marketplace page.



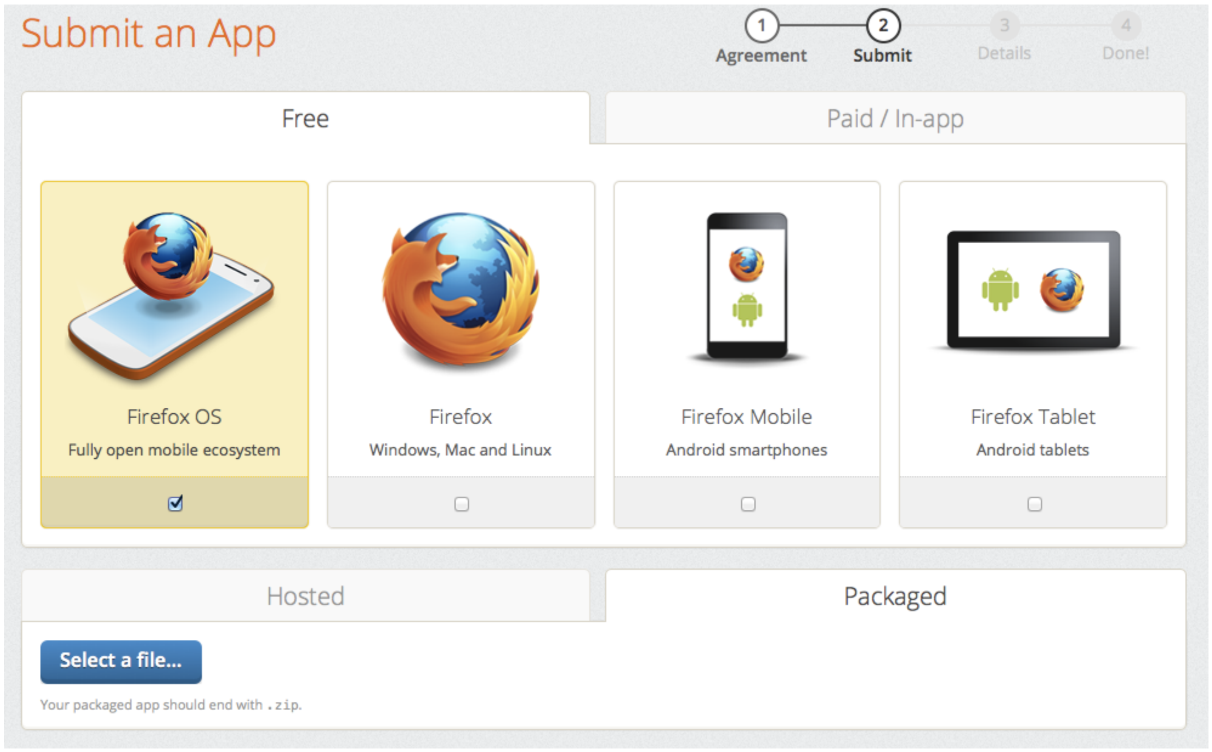
My Submissions

Inside the application management page, you need to click on **Submit An App** on the top menu.



Submit An App

This link will lead you to the form for submitting new apps, as seen in the screenshot below.



Submit New App

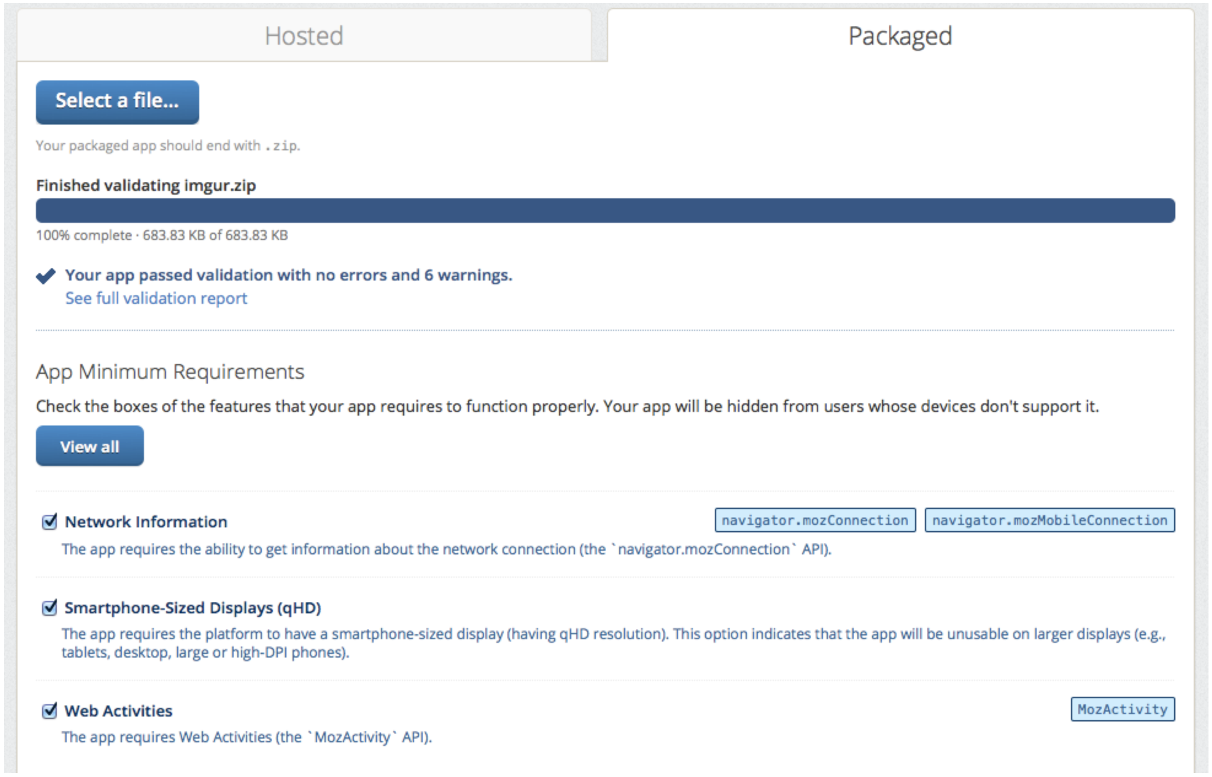
On this screen you will select the following options:

- If the application is hosted or packaged.
- If it is free or paid (or uses *in-app purchases*).
- What type of devices it is available for (Firefox OS, Firefox Desktop, Firefox for Mobile on phones, Firefox for Mobile on Tablets).

After making these choices you're driven to the second screen. On this book we're focusing on packaged apps but hosted apps are similar.

In the remaining text of this chapter we're assuming that we're shipping a free Firefox OS packaged app. In this case we need to upload the zip file we prepared on the previous section.

After uploading the file, it undergoes an automated process and a report is shown with many options.



Your packaged app should end with `.zip`.

Finished validating imgur.zip

100% complete · 683.83 KB of 683.83 KB

✓ **Your app passed validation with no errors and 6 warnings.**
See full validation report

App Minimum Requirements

Check the boxes of the features that your app requires to function properly. Your app will be hidden from users whose devices don't support it.

☑ **Network Information** `navigator.mozConnection` `navigator.mozMobileConnection`
The app requires the ability to get information about the network connection (the `navigator.mozConnection` API).

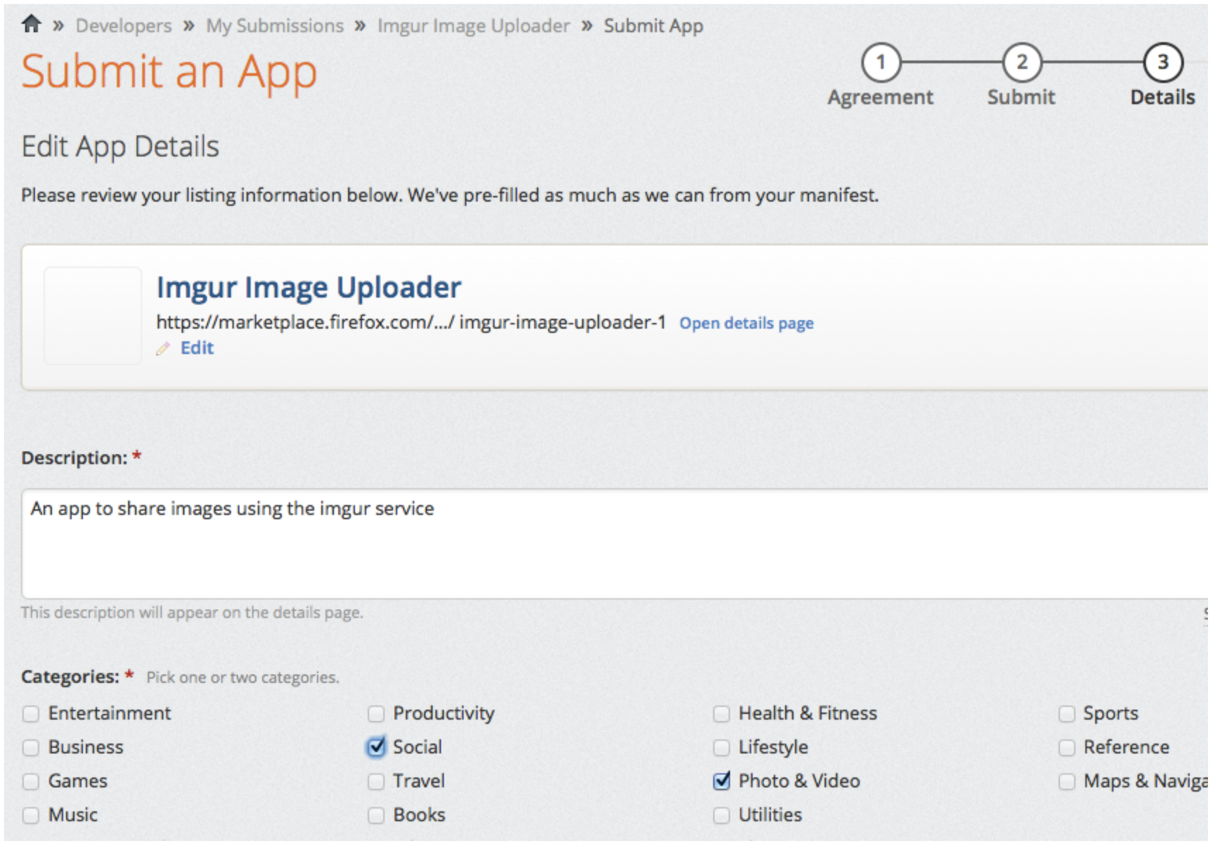
☑ **Smartphone-Sized Displays (qHD)**
The app requires the platform to have a smartphone-sized display (having qHD resolution). This option indicates that the app will be unusable on larger displays (e.g., tablets, desktop, large or high-DPI phones).

☑ **Web Activities** `MozActivity`
The app requires Web Activities (the `MozActivity` API).

After the zip upload

From the screen shot above we can see that the app I sent to the marketplace has no errors but contains six warnings. Ignoring the warnings for the sake of this text, lets check what the **minimum requirements** for this app are. In this case, the last option *Smartphone-Sized Displays (qHD)* should be unchecked because our application adapts to any screen size.

The next step is called **Step #3: Details** and it is where you fill the information about your application such as category, description, screen captures, etc.


» Developers » My Submissions » Imgur Image Uploader » Submit App
Submit an App
1 Agreement
2 Submit
3 Details

Edit App Details

Please review your listing information below. We've pre-filled as much as we can from your manifest.

Imgur Image Uploader
https://marketplace.firefox.com/.../ imgur-image-uploader-1 Open details page
Edit

Description: *

An app to share images using the imgur service

This description will appear on the details page.

Categories: * Pick one or two categories.

☐ Entertainment	☐ Productivity	☐ Health & Fitness	☐ Sports
☐ Business	✓ Social	☐ Lifestyle	☐ Reference
☐ Games	☐ Travel	✓ Photo & Video	☐ Maps & Naviga
☐ Music	☐ Books	☐ Utilities	

Filling details

After filling-in the details, the submission process is done. Now you just wait for the approval from the marketplace reviewers. Congratulations you shipped a Firefox OS application!!!

On the Application Management page[91] you can check the status of your submissions and alter details if needed.

To learn more about submitting applications to the Firefox Marketplace read this article on the Firefox OS developer hub[92].

Summary

Congratulations!!! You have a new application on the Firefox Marketplace, you're exploring a brand new market!

I hope you enjoyed this quick guide. I plan to update and expand this guide often - so keep your eyes open and register for the updates. If you downloaded this book from Leanpub then its all good because you will receive emails about any updates. If you downloaded it from somewhere else then

[91]https://marketplace.firefox.com/developers/submissions
[92]https://marketplace.firefox.com/developers/docs/submission

please consider fetching it from the official page at Leanpub[93] and registering your email. It's free and, no, you won't get any spam. Promise.

Please send me feedback. This book was written by pulling all-nights before a tech conference so you can infer how much I enjoy this project and want to see it succeed. I can be reached for feedback on my Twitter account at @soapdog[94] and over email at fxosquickguide@andregarzia.com[95]. My home page is at http://andregarzia.com[96].

Now that you're a part of the group of Firefox OS app creators, come be a part of the greater Mozilla community: Help keep the web free and open made by users for users. Join us at http://www.mozilla.org/contribute/[97] and help Firefox OS grow!

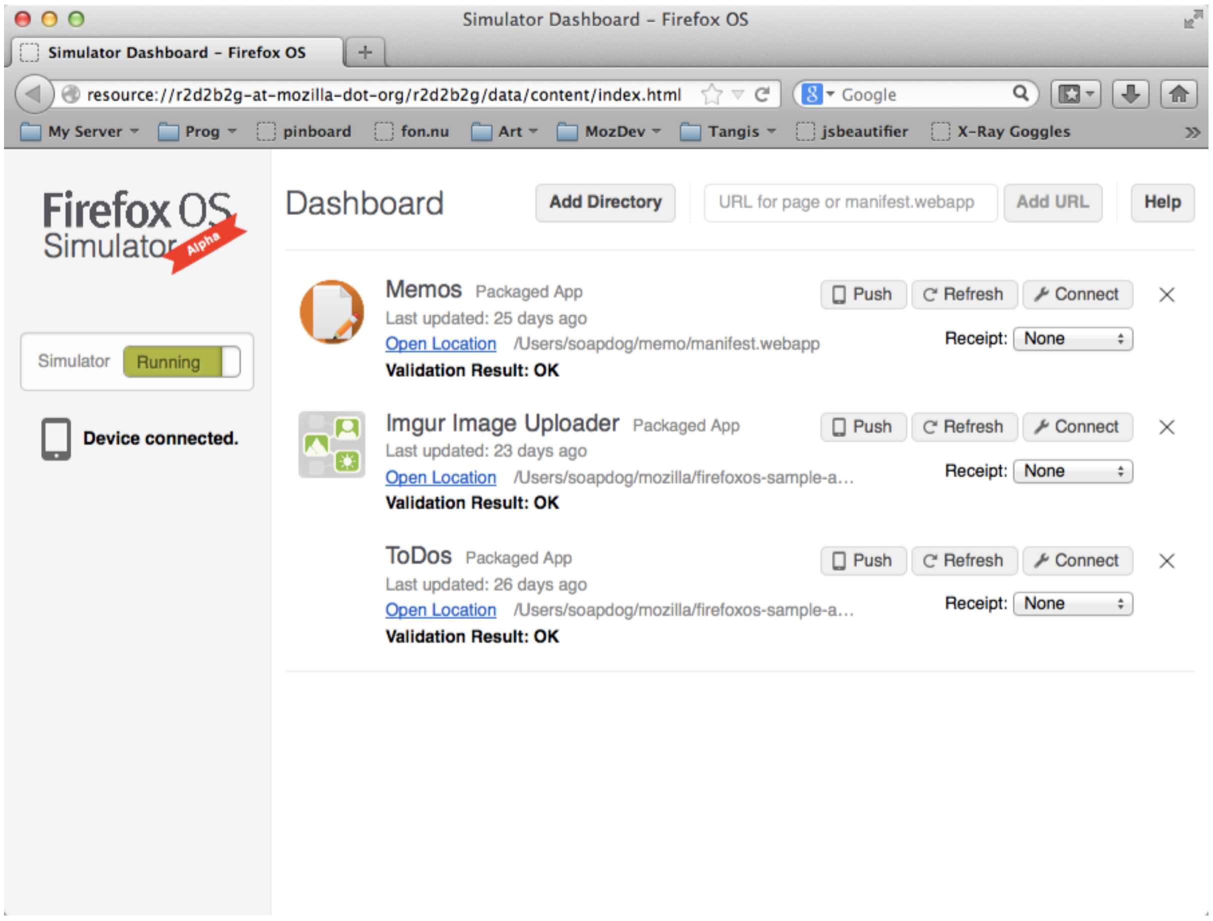
[93]http://leanpub.com/quickguidefirefoxosdevelopment
[94]http://twitter.com/soapdog
[95]mailto:fxosquickguide@andregarzia.com
[96]http://andregarzia.com
[97]http://www.mozilla.org/en-US/contribute/

Appendix 1: Useful links

- Mozilla[98]
- Mozilla Brasil Community Page[99]
- Firefox OS Developer Hub[100]
- Mozilla Developers Network[101]: Best documentation ever!
- Firefox OS[102]
- WebAPI on Mozilla Wiki[103]
- Books about Firefox OS[104]

[98] http://mozilla.org
[99] http://mozillabrasi.org.br
[100] http://marketplace.firefox.com/developers
[101] http://developer.mozilla.org/
[102] http://www.mozilla.org/pt-BR/firefox/os/
[103] http://wiki.mozilla.org/WebAPI
[104] http://firefoxosbooks.org

Appendix 2: The Firefox OS Simulator



Memos Packaged App
Last updated: 25 days ago
Open Location /Users/soapdog/memo/manifest.webapp
Validation Result: OK

Imgur Image Uploader Packaged App
Last updated: 23 days ago
Open Location /Users/soapdog/mozilla/firefoxos-sample-a…
Validation Result: OK

ToDos Packaged App
Last updated: 26 days ago
Open Location /Users/soapdog/mozilla/firefoxos-sample-a…
Validation Result: OK

Firefox OS Simulator Dashboard

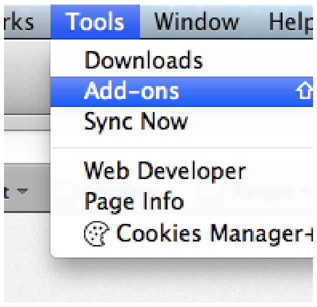
Attention: This chapter is here for compatibility with devices running Firefox OS 1.1. The current method for testing and debugging apps is the **WebIDE** which we talked about in the previous chapter. The content of this chapter is just for people testing stuff on Firefox OS version 1.1.

Attention: If you're running **Firefox 29 or newer** and you have a device running **Firefox OS 1.1 or earlier** then you will need another version of the **Firefox OS 1.1 Simulator** that is not currently listed on the add-ons marketplace. This version is **BETA** but its the best we've got right now. You can fetch it for Mac OS X[105], Linux[106] or Windows[107]. Just drop the xpi file on Firefox and follow the installation instructions. If you want to follow up on the quest of making the **Firefox OS 1.1 simulator** work on **Firefox 29** then check out bug request #1001590 it[108].

Setup

If you have a device running **Firefox OS 1.1** then you need to install the **Firefox OS 1.1 Simulator** because your device can't communicate with the new **Web IDE**.

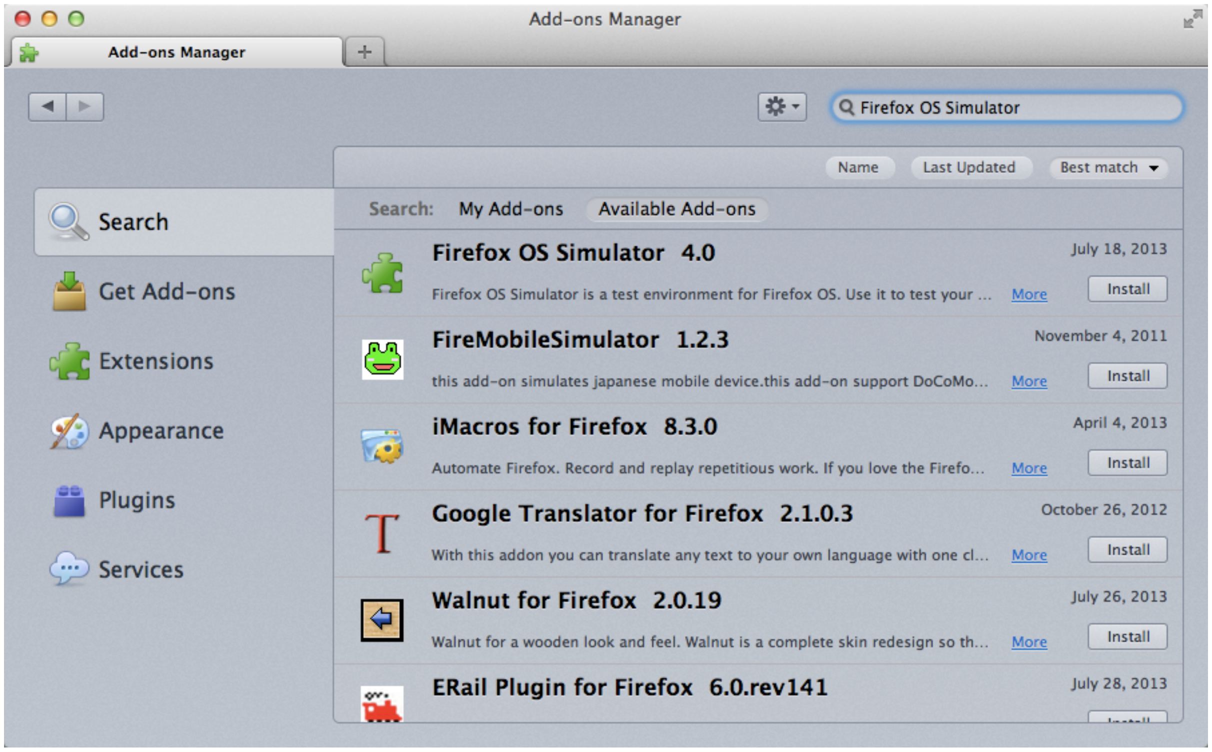
After installing Firefox, the next step is the installation of the Firefox OS Simulator that will be used to test our applications. With Firefox installed and running, go to the **Tools** menu and select **Add-ons**.



***Tools* menu with *Add-ons** menu selected**

Using the search box on the top right corner, search for **Firefox OS Simulator** and install the add-on by clicking the install button.

[105]http://ftp.mozilla.org/pub/mozilla.org/labs/r2d2b2g/r2d2b2g-5.0pre7-mac.xpi
[106]http://ftp.mozilla.org/pub/mozilla.org/labs/r2d2b2g/r2d2b2g-5.0pre7-linux.xpi
[107]http://ftp.mozilla.org/pub/mozilla.org/labs/r2d2b2g/r2d2b2g-5.0pre7-windows.xpi
[108]https://bugzilla.mozilla.org/show_bug.cgi?id=1001590



Firefox OS Simulator 4.0 July 18, 2013
Firefox OS Simulator is a test environment for Firefox OS. Use it to test your ... More Install

FireMobileSimulator 1.2.3 November 4, 2011
this add-on simulates japanese mobile device.this add-on support DoCoMo... More Install

iMacros for Firefox 8.3.0 April 4, 2013
Automate Firefox. Record and replay repetitious work. If you love the Firefo... More Install

Google Translator for Firefox 2.1.0.3 October 26, 2012
With this addon you can translate any text to your own language with one cl... More Install

Walnut for Firefox 2.0.19 July 26, 2013
Walnut for a wooden look and feel. Walnut is a complete skin redesign so th... More Install

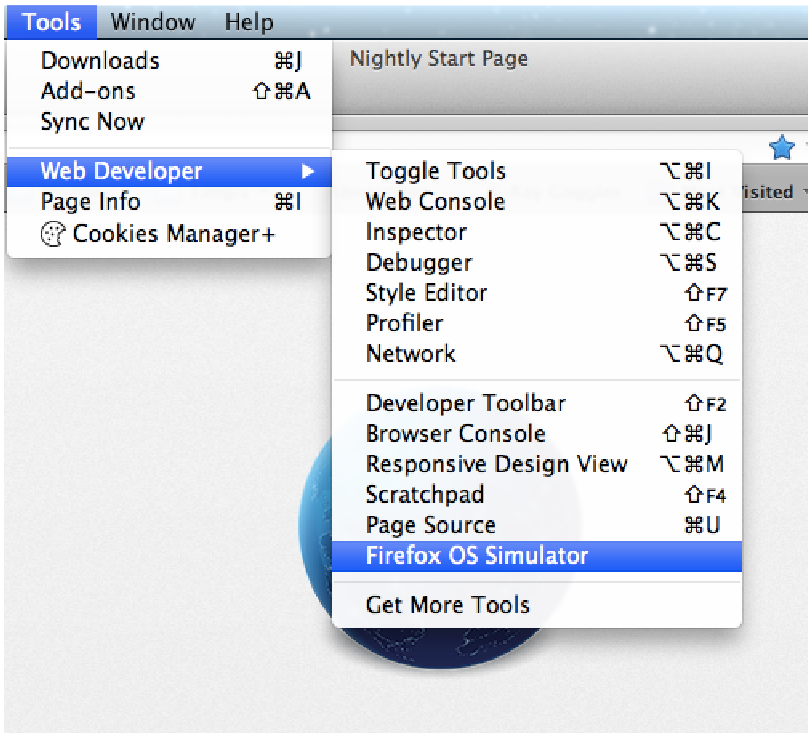
ERail Plugin for Firefox 6.0.rev141 July 28, 2013

Add-on manager showing the simulator add-on

Attention: If you're running **Firefox 29 or newer** and you have a device running **Firefox OS 1.1 or earlier** then you will need another version of the **Firefox OS 1.1 Simulator** that is not currently listed on the add-ons marketplace. This version is **BETA** but its the best we've got right now. You can fetch it for Mac OS X[109], Linux[110] or Windows[111]. Just drop the xpi file on Firefox and follow the installation instructions. If you want to follow up on the quest of making the **Firefox OS 1.1 simulator** work on **Firefox 29** then check out bug request #1001590 it[112].

After the installation of the add-on, you will be able to access the simulator by going to the menu **Tools -> Web Developer -> Firefox OS Simulator**.

[109]http://ftp.mozilla.org/pub/mozilla.org/labs/r2d2b2g/r2d2b2g-5.0pre7-mac.xpi
[110]http://ftp.mozilla.org/pub/mozilla.org/labs/r2d2b2g/r2d2b2g-5.0pre7-linux.xpi
[111]http://ftp.mozilla.org/pub/mozilla.org/labs/r2d2b2g/r2d2b2g-5.0pre7-windows.xpi
[112]https://bugzilla.mozilla.org/show_bug.cgi?id=1001590



Where you can find the simulator after is installed

Alternatively, you can navigate to the Firefox OS Simulator[113] addon page, and download the simulator from there.

We've installed the Firefox OS Simulator in the chapter about preparing the environment and we used it on the chapter about building our first app. Now we're going to take a deeper look into the simulator features and learn how to do the most common tasks.

To learn more about it, check out the Firefox OS Simulator documentation[114] on MDN.

Adding Apps

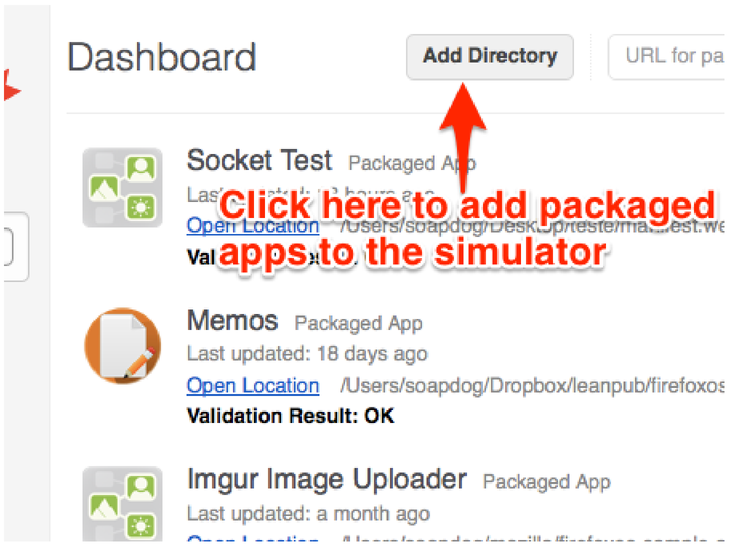
You can add both hosted and packaged apps to the simulator. Lets see how to add each type of app.

Adding packaged apps

You already saw how to add packaged apps to the simulator during our first app creation, but we're going to do a recap so I can show you what else is possible.

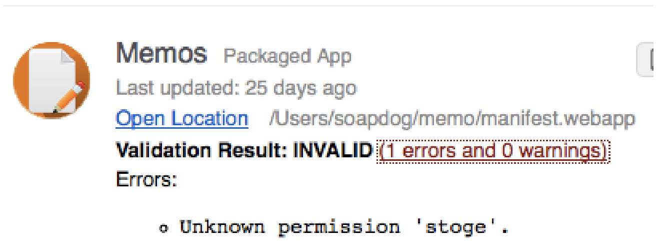
To add a new packaged application click the **Add Directory** button on the **Simulator Dashboard** as shown in the screenshot below.

[113]https://addons.mozilla.org/en-US/firefox/addon/firefox-os-simulator/
[114]https://developer.mozilla.org/en-US/docs/Tools/Firefox_OS_Simulator


Dashboard
Add Directory
URL for pa
Click here to add packaged apps to the simulator
Socket Test Packaged App
Open Location
Memos Packaged App
Last updated: 18 days ago
Open Location /Users/soapdog/Dropbox/leanpub/firefoxos
Validation Result: OK
Imgur Image Uploader Packaged App
Last updated: a month ago

Showing the *Add Directory* button that adds a packaged app to the simulator

When you click on the button highlighted on the image, Firefox opens a file selection dialog. You should browse your hard drive and select the **app manifest file** for the application that you want to add to the simulator. If there are no issues with your manifest and your start file is ok, the application will be added and the simulator will launch with your app running. If there is anything wrong with your manifest, or some other issue, then an error report will be shown on the dashboard.


Memos Packaged App
Last updated: 25 days ago
Open Location /Users/soapdog/memo/manifest.webapp
Validation Result: INVALID (1 errors and 0 warnings)
Errors:

```
◦ Unknown permission 'stoge'.
```

Example of an invalid manifest

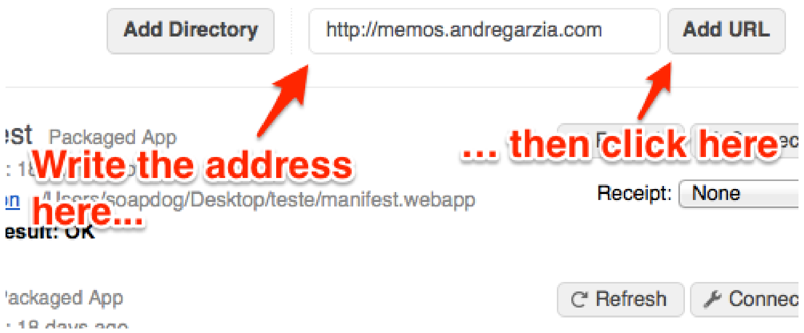
Whenever you update your application you should click **Refresh** to update the version of the app on the simulator (you can also press CMD/CTRL+R on the simulator window to refresh).

Adding hosted apps

If you're building a hosted app then you should test it by using a web server. Do not try to use the method described above for hosted apps because you may miss some errors that will only happen on a hosted environment - such as serving the manifest with the wrong *MIME type*. Note that the simulator won't warn you about things like incorrect MIME types, but it's important to get such things right if you submit your app to the Mozilla Marketplace.

Most of the hosted apps are not applications built exclusively for Firefox OS but responsive design based websites that are able to adapt themselves to different devices and resolutions. These web apps usually have a complex backend that needs to be in-place for the application to work and that's why you need to test the app using a real web server running your backend stuff.

To run your app in the simulator, fill the URL of your application in the text entry box on the top and click the **Add URL** button.


Add Directory
http://memos.andregarzia.com
Add URL
Write the address here...
... then click here
Packaged App
/Users/soapdog/Desktop/teste/manifest.webapp
Result: OK
Receipt: None
Refresh
Connect

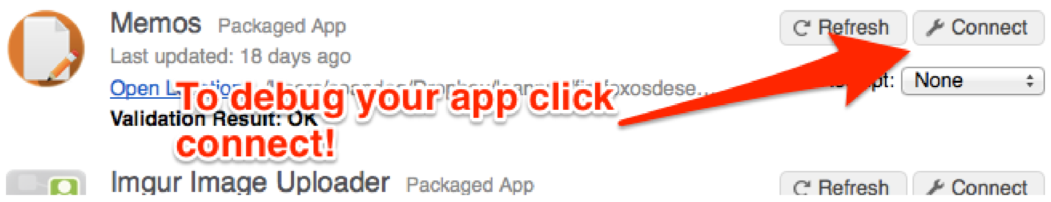
Adding a hosted app to the simulator

After clicking the button, the manifest is verified and if it is correct the application is added and the simulator is launched with your application running. Like when we're adding packaged apps, if something wrong happens with in the manifest you will see a report (e.g. "app submission to the marketplace needs at least an 128 icon").

As with packaged apps, whenever you update your application you should click **Refresh** to update the version of the app on the simulator (you can also press CMD/CTRL+R on the simulator window).

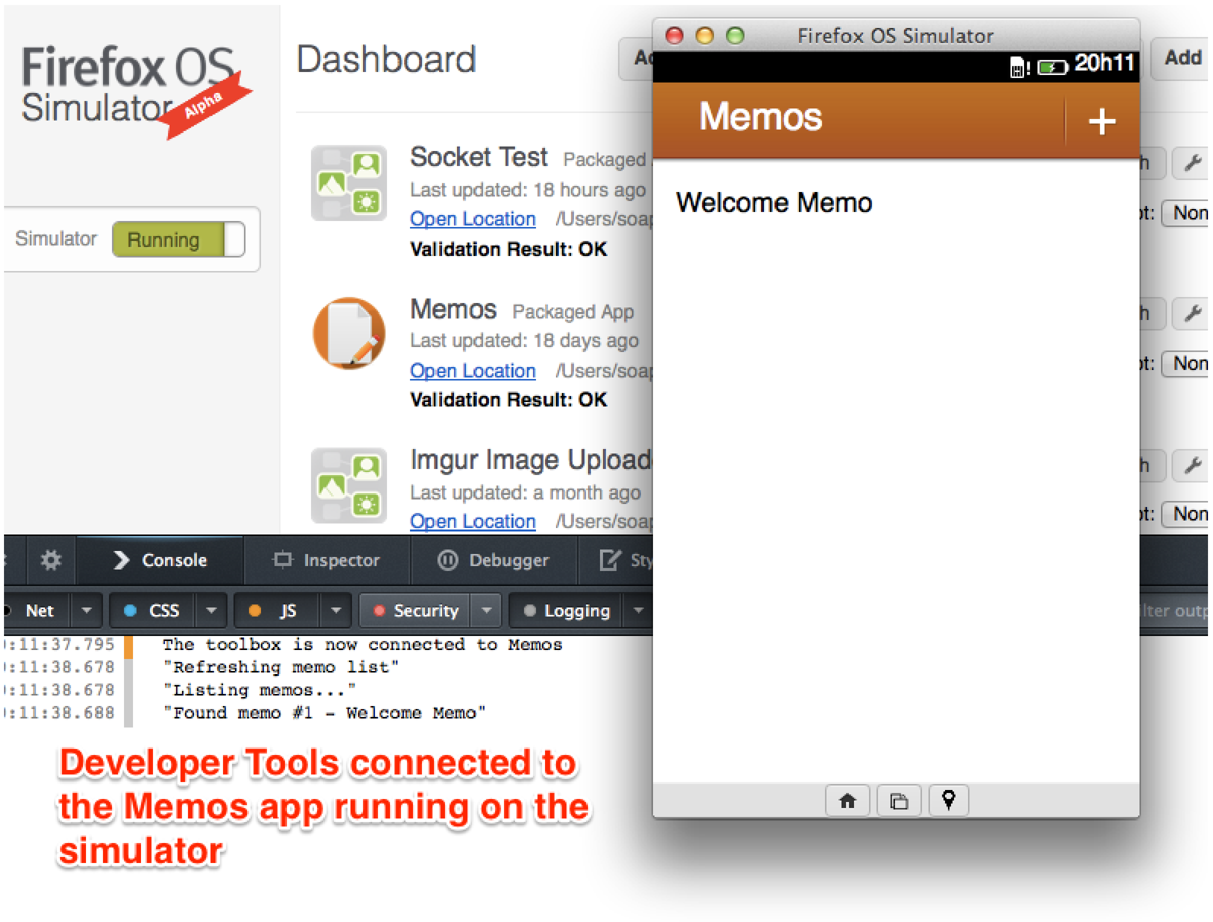
Debugging

After the application is added to the simulator we're able to debug it by clicking the **Connect** button next to the application listing on the dashboard. This will launch the simulator with your application running and the **JavaScript Console** open and connected to your app.



What button to press

After pressing this button you will see a screen like this:



```
:11:37.795  The toolbox is now connected to Memos
:11:38.678  "Refreshing memo list"
:11:38.678  "Listing memos..."
:11:38.688  "Found memo #1 - Welcome Memo"
```

Developer Tools connected to the app running on the simulator

With the tools connected to your app you can test your JavaScript, debug your DOM, edit styles, etc. Like those startup guys like to say: *pivot until your app is good.*

Once your app is running well on the simulator it's time to test on a real device.

Testing apps on a real device

Nothing replaces testing on a real device. On the simulator, you test things by using a mouse and clicking on a computer screen; while on a real device you use your fingers on a touchscreen and by using physical buttons. Its a very different user and development experience.

As an example why this type of testing matters, let me tell you a brief story: Some years ago, Raphael Eckhardt (the designer who created the cover of this book) and I were building a puzzle game not that disimilar to Bejeweled. Our game involved dragging and dropping some pieces on a board and was working pretty well on the simulator.

When we then tested the game on an actual phone we realized our game components were not touch friendly at all: when placing a hand over the screen the board would vanish behind the hand.

Even worst, the pieces the users were supposed to drag were smaller than the user's finger tip, so the user couldn't see what they were doing! In summary, our UX sucked very badly. That happened because we're kept trying things only on the simulator with a mouse that used a tiny cursor. When we decided to try with our fatter-than-a-cursor fingers we realized that we need to rework our UI.

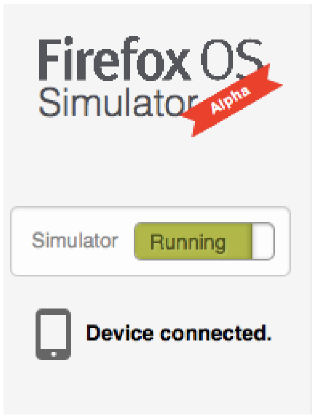
To avoid having a similarly depressing experience, always test on a real device… or two, or more if you can get your hands on some. And test often with simple prototypes: otherwise, you can waste valuable time and money having to recreate assets.

You can buy a developer preview phone running Firefox OS from the Geeksphone Shop[115]. I recommend using a Geeksphone Keon[116] because this device has similar specs to the devices that are being launched by Mozilla's partners.

You can also buy a device targeted at consumers if you happen to live in one of the countries where they are already available. A third way is that you can replace Android with Firefox OS on some devices (some specific devices only, chance of bricking, don't blame me!) - but I don't recommend this unless you're a power user and like to spend a lot of time hacking.

Connecting with a Firefox OS device

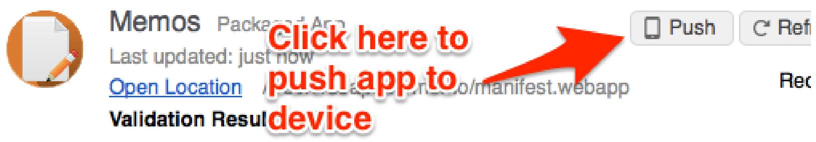
If you have a Firefox OS device (and have any needed drivers installed) then you can push apps directly from the simulator to the device if the device is connected to your computer. When the simulator detects that you plugged a Firefox OS phone, it will display a notice saying **Device Connected**.



Device Connected!

If your phone is connected (and detected) the simulator will add a new button next to **Refresh** and **Connect** called **Push**. When you press this button, a **permission request dialog** appears on the device screen asking for confirmation to install the pushed app.

[115]http://shop.geeksphone.com/en/
[116]http://www.geeksphone.com/



Which button to press to push apps to the connected device

And below we can see the permission request screen.

An incoming request to permit remote debugging connection was detected. Allow connection?

Not the best picture in the world but shows the permission screen (sorry for the face it was 4:25 AM)

With the application running on the device you can use *remote debugging* to connect a JavaScript console and debug the app.

Summary

In summary, the Firefox OS Simulator is awesome for building Firefox OS specific apps - but has some limitations if you are trying to build for a range of devices (e.g. currently, you can't emulate what Firefox OS would feel like on a tablet).

Besides feeling awesome and empowered, by this point in the book you hopefully have a good grasp of the workflow for building apps for Firefox OS.