WebFX Documentation

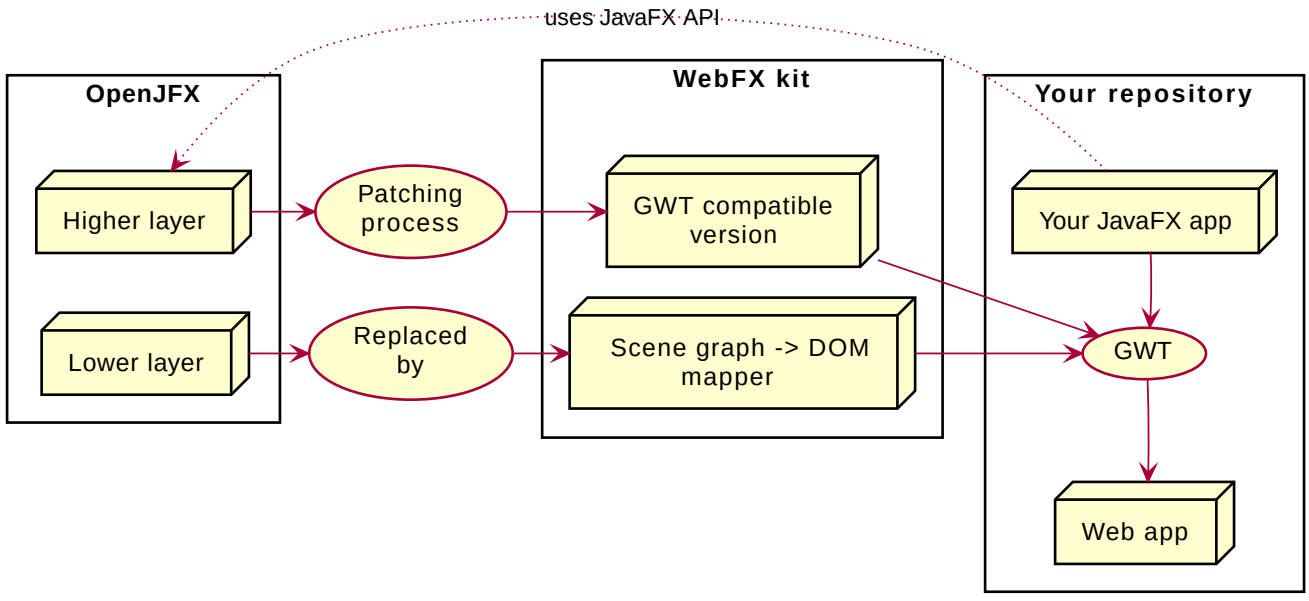
Table of Contents

Introduction

What is WebFX?

WebFX is a JavaFX application transpiler powered by GWT. It can transpile a JavaFX application into a traditional self-contained pure JavaScript web app (with no plugin or server required for its execution in the browser).

How it works



The webfx-kit module is the heart of WebFX. It's a modified version of OpenJFX that can be

transpiled. This is achieved by patching the higher layer of OpenJFX (which contains the main JavaFX features and API) to make it GWT compatible, and by replacing the lower layer (the graphic rendering pipeline) by a scene graph → DOM mapper (the DOM being finally rendered by the browser).

Limitations

The WebFX kit coverage is for now limited to the essential features of JavaFX. So to successfully compile to the web, your JavaFX app needs to meet these 2 requirements:

- use only the features covered by the WebFX kit (you can check out the JavaDoc to get an idea of this coverage)
- be compatible with GWT (no reflection, no multi-threading, no blocking code, etc...)

A JavaFX application that meets these 2 requirements is called a *WebFX application*, and it can be transpiled to the web simply by running a GWT compilation of it together with the WebFX kit.

Note for the impatient: OpenJFX is a very rich and big library (about 10MB) compared to standard JS frameworks (typically 100KB). It will take time to complete its coverage (some parts may not be possible). Thanks for your understanding. But compared to some frameworks, you can already do a lot with the current coverage.

Benefits

Performance

Despite the big size of OpenJFX, WebFX can produce lightweight web apps, as demonstrated by the demos and the website:

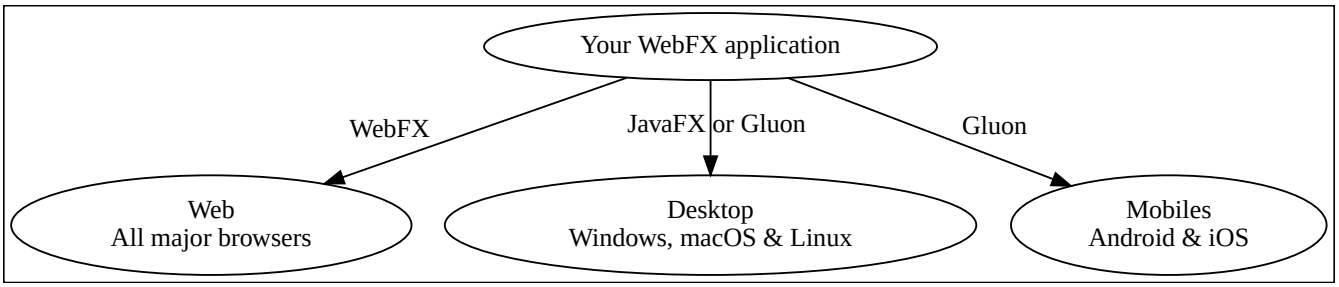
WebFX application	JS size *
Colorful circles demo	90.6 kB
Particles demo	90.3 kB
Tally counter demo	101 kB
Modern gauge demo	139 kB
Enzo clocks demo	253 kB
FX2048 demo	178 kB
SpaceFX demo	139 kB
Ray tracer demo	135 kB
Mandelbrot demo	142 kB
Website	218 kB

* compressed JS size transiting over the network, without eventual images or other resources

Here are the 3 secrets of this magic:

- The scene graph → DOM mapper is much thinner than the original OpenJFX lower layer
- GWT runs a dead code elimination, which removes the JavaFX classes you don't use
- GWT produces an amazingly compaq and optimized JS code

Cross-platform



In addition to the web platform, WebFX will help you to compile your application for the desktop & mobiles thanks to the JavaFX & Gluon toolchains. You can do a full cross-platform development from a single source code base.

Java full-stack

WebFX should please Java developers who are looking for a Java full-stack solution. Writing your whole stack in Java is a big advantage, keeping your development homogenous and simple, with all your code in your preferred Java IDE. Not only you don't need to master other complex ecosystems such as JavaScript or TypeScript, but you can also share the common code between your backend and frontends with the Java module system, a great advantage compared to heterogeneous systems.

Low learning curve

WebFX is not yet another UI toolkit to learn, but nothing else than an already well known and well documented UI API: JavaFX. All the features you love like JavaFX bindings available for your web app. You will just feel at home with WebFX!

No server

There are already great solutions to run Swing or JavaFX applications in the browser without plugins by actually running them on a server. And these solutions don't have the limitations WebFX has on the API. However, a traditional self-contained web app packaging is a much more simple, scalable and reliable execution model. This is where WebFX has a role to play.

Fast development cycles

You don't need to run regular GWT compilations like you would do with a traditional GWT development, because you can run and debug your WebFX application directly in your Java IDE with the OpenJFX runtime (like a standard JavaFX development). So you first develop your features in a traditional way, and only at the end of a development cycle, transpile your app to test the web version.

Free and open source

WebFX is an open source initiative under Apache 2.0 license.

Getting started

Prerequisite

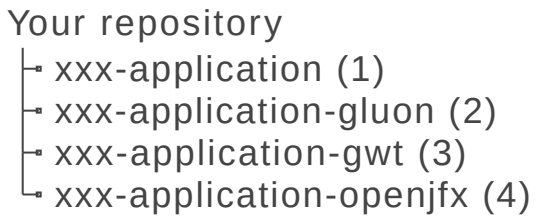
To develop WebFX applications, you will need the following software already installed on your development machine:

- JDK 13 or above
- Maven
- Git
- Your preferred Java IDE

Be sure that `java`, `mvn` and `git` are in the path of your terminal. The WebFX CLI will invoke these commands without specifying their full path.

Introducing the WebFX CLI

The WebFX CLI is a Command Line Interface tool that will assist you developing WebFX applications. It will create your application modules as follows:



① This module contains the JavaFX code of your application. It is cross-platform (not yet bound to a specific platform) and therefore not executable.

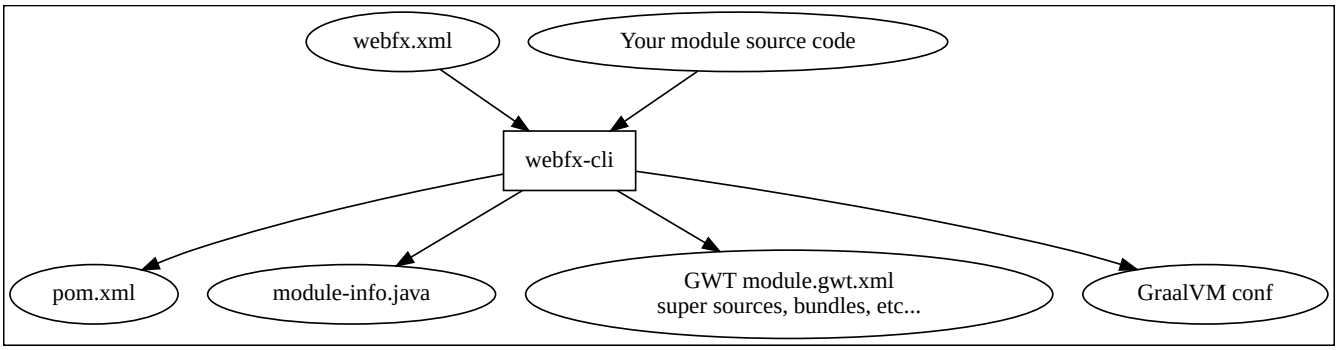
② This module targets the native desktop & mobile platforms. It binds your application with the OpenJFX runtime, and can call the Gluon toolchain to produce the Windows, macOS, Linux, Android & iOS native executables (depending on your OS).

③ This module targets the web platform. It binds your application with the WebFX kit, and can call GWT to produce the web app.

④ This module targets the standard desktop platform. It binds your application with the OpenJFX runtime. It is directly executable in your IDE. It can also call the standard JavaFX toolchain to produce the desktop executables (Windows, macOS or Linux) with an embed JVM.

You can create several WebFX applications in the same repository. If your application code grows,

you can split your code into more modules. The WebFX CLI will help you to create and maintain all your modules. For each module, it will create and maintain your build files as follows (when applicable to the module):



Your inputs will be centralized in the WebFX module files named webfx.xml (same location as pom.xml), and the WebFX CLI will generate the rest of the build chain from these webfx.xml files. For example, a typical directive in webfx.xml will be:

```
<dependencies>
    <used-by-source-modules/>
</dependencies>
```

This directive is asking the WebFX CLI to generate the list of your dependencies from an analysis of your source code, and automatically populate the dependencies in pom.xml.

In that process, the WebFX CLI takes care of the cross-platform aspects: when a feature is platform-dependent (a different implementation exists for different platforms), the tool will pick up the right modules (those whose implementation matches the target platform). In particular, it will replace the OpenJFX modules with the WebFX kit ones when targeting the web.

Installing the WebFX CLI

Since we haven't published any release at this stage yet, the way to install the WebFX CLI for now is to clone the webfx-cli repository, and build it with Maven.

We will distribute the WebFX CLI in a better way with the first official release.

Cloning the webfx-cli repository

HTTPS

```
git clone https://github.com/webfx-project/webfx-cli.git
```

SSH

```
git clone git@github.com:webfx-project/webfx-cli.git
```

Building webfx-cli with Maven

This is achieved by running the Maven *package* goal under the webfx-cli directory:

```
cd webfx-cli
mvn package
```

As previously mentioned, WebFX CLI requires JDK 13 or above to successfully compile.

This generates an executable fat jar in the target folder that we can execute with java:

```
java -jar target/webfx-cli-0.1.0-SNAPSHOT-fat.jar --help
```

Creating a permanent *webfx* alias

To easily invoke the WebFX CLI from a terminal, we need to create a permanent *webfx* alias. This is done with the following command (to run under the webfx-cli directory):

Linux

```
echo "alias webfx='java -jar $(cd "$(dirname "$1")" && pwd -P)/$(basename
"$1")/target/webfx-cli-0.1.0-SNAPSHOT-fat.jar'" >> ~/.bashrc ①

source ~/.bashrc ②
```

① Adding the alias to the shell profile

② Applying it to the current session

macOS >= Catalina

```
echo "alias webfx='java -jar $(cd "$(dirname "$1")" && pwd -P)/$(basename
"$1")/target/webfx-cli-0.1.0-SNAPSHOT-fat.jar'" >> ~/.zshrc ①

source ~/.zshrc ②
```

① Adding the alias to the shell profile

② Applying it to the current session

macOS < Catalina

```
echo "alias webfx='java -jar $(cd "$(dirname "$1")" && pwd -P)/$(basename
"$1")/target/webfx-cli-0.1.0-SNAPSHOT-fat.jar'" >> ~/.bash_profile ①

source ~/.bash_profile ②
```

① Adding the alias to the shell profile

② Applying it to the current session

Windows PowerShell

```
If (!(Test-Path $profile)) { New-Item -Path $profile -Force } ①

"`r`nfunction webfx([String[]] [Parameter(ValueFromRemainingArguments)] `$params) {
java -jar $((Get-Item .).fullName)\target\webfx-cli-0.1.0-SNAPSHOT-fat.jar `$params
}`r`n" >> $profile ②

If ($(Get-ExecutionPolicy) -eq "Restricted") { Start-Process powershell -Verb runAs
"Set-ExecutionPolicy -ExecutionPolicy RemoteSigned" -Wait } ③

. $profile ④
```

① Creating a PowerShell profile if it doesn't exist

② Adding the alias (implemented as a function) to it

③ Lowering the execution policy if necessary to execute the profile

④ Applying it to the current session

Then you should be able to invoke the CLI tool from the terminal:

```
webfx --help
```

Updating the WebFX CLI to the latest version

You can check for update at anytime by running:

```
webfx bump cli
```

If a new version is available, it will download it and build it.

This is the only command that uses `git`, and it's just a `git pull` of the webfx-cli repository. The tool will not call `git` on your own repositories.

Creating your first WebFX app

Creating and initializing your repository

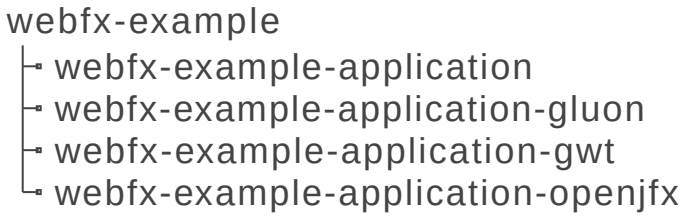
Let's create our first WebFX application. We need to create the repository folder and ask the WebFX CLI to initialize it, passing it the groupId, artifactId and version of our application.

```
mkdir webfx-example
cd webfx-example
webfx init org.example:webfx-example:1.0.0-SNAPSHOT
```

`webfx init org.example:1.0.0-SNAPSHOT` will also work as the tool takes the repository directory name as the artifactId if omitted in the command.

Creating your application modules

```
webfx create application --class org.example.webfxexample.WebFxExampleApplication
--helloWorld
```



Building your application

```
webfx build
```

Running your application

You can run the OpenJFX version of your application with the following command:

```
webfx -m webfx-example-application-openjfx run
```

You can run the GWT version of your application with the following command:

```
webfx -m webfx-example-application-gwt run
```

Developing in your IDE

We will give the instructions for IntelliJ IDEA, but you should be able to easily transpose them to other Java IDEs.

Opening the project

Configuring the OpenJFX application

A few explanation before going further: GWT has a different entry point than standard Java applications, but WebFX hides the hassle with a single cross-platform entry point that will correctly bootstrap your application. So the main class of a WebFX application will always be `dev.webfx.platform.shared.services.boot.ApplicationBooter` whatever the platform. It will find your JavaFX application because it has been automatically declared as a Java service by the WebFX CLI.

GWT normally doesn't support the Java service API, but WebFX does, because the CLI emulates it by generating a GWT super source. You can rely on this feature to declare and implement your own services. Your services can even be platform-dependent, and you can write a specific implementation for each platform. This can be a UI API, with an OpenJFX implementation, and then a GWT implementation using an alternative JS library for your web app.

Building and running the GWT application

Making changes

```
webfx update
```