WebFX Documentation

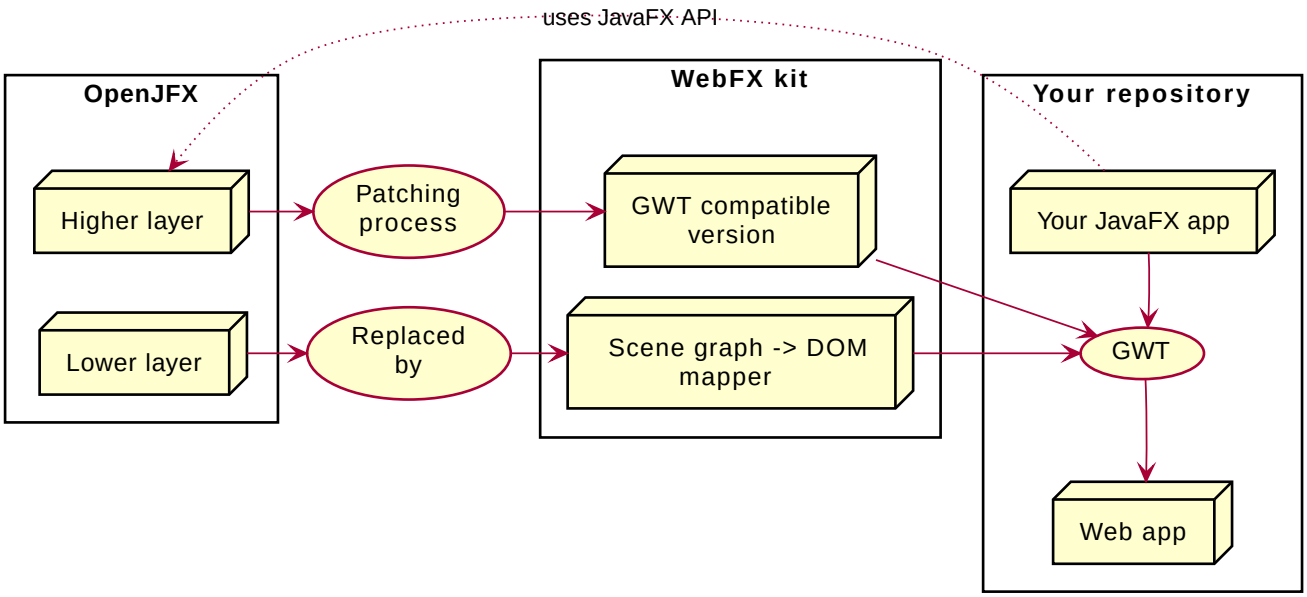
Table of Contents

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 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 code 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...)

When a JavaFX application meets these 2 requirements, we will call it 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 huge 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

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 currently has. However, a standard self-contained JS packaging is a much more simple, scalable and reliable execution model. This is precisely that benefit that WebFX is offering, and probably the

main reason why you would prefer it over the other existing solutions.

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
Medusa clock demo	180 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

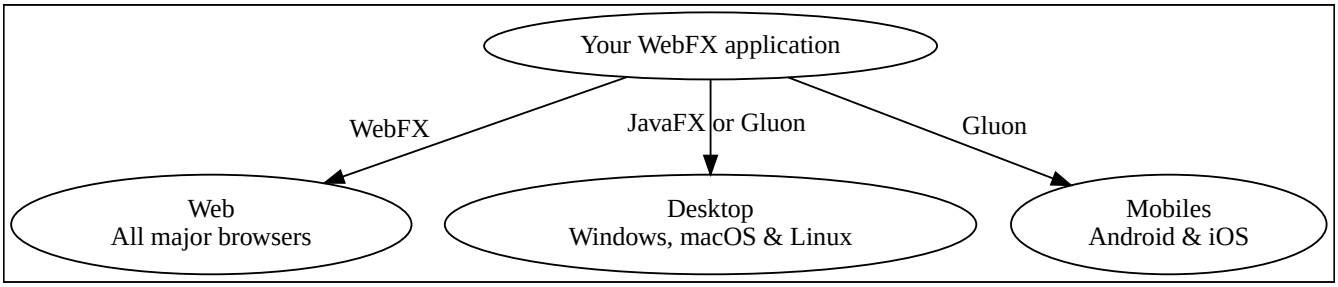
* compressed JS size transiting over the network, without eventual images or other resources

The secret? Here are the 3 main ingredients for this magic:

- The scene graph → DOM mapper is much thinner than the original OpenJFX lower layer which has to reimplement many features a browser already has.
- GWT runs a dead code elimination, which removes the JavaFX classes not used by the WebFX application.
- GWT produces an amazingly compaq and optimized JS code.

The later point also confers an excellent execution speed to your web app.

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

Writing your whole stack in Java is a big advantage, keeping your environment simple and homogenous from a single 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 the already well known and documented JavaFX API. All the powerful features you love like JavaFX bindings available for your web app. You will just feel at home with WebFX!

Fast development cycles

You don't need to run regular GWT compilations like you would do with a traditional GWT development, because you can already run and debug your WebFX application directly in your Java IDE with the OpenJFX runtime. You typically transpile your app only at the end of a development cycle to check the web version, after you have finished developing a feature using the standard JavaFX development model.

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:

```
Your repository
├─ xxx-application (1)
├─ xxx-application-gluon (2)
├─ xxx-application-gwt (3)
└─ xxx-application-openjfx (4)
```

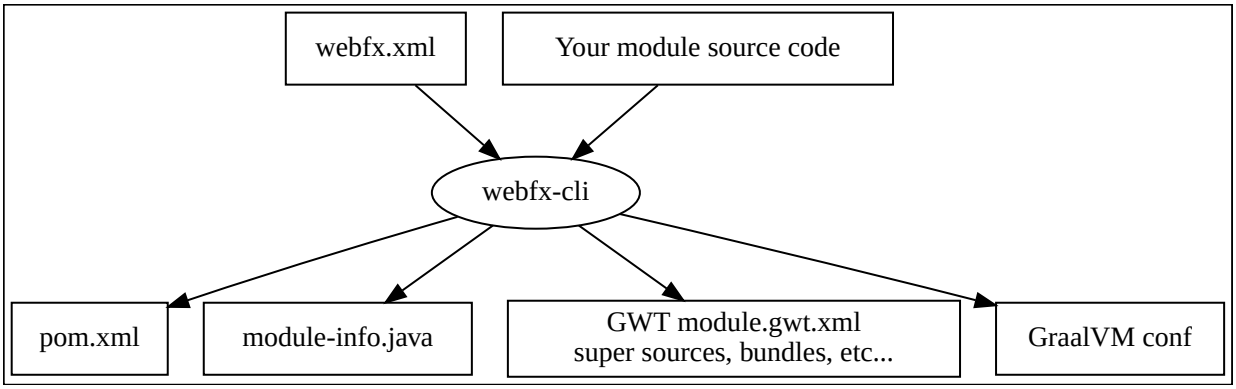
① This module contains the JavaFX code of your application. It is cross-platform (not yet bound to a specific platform) and therefore not executable.

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, and 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 CLI will help you to create and maintain all your modules. For each module, it will create and maintain your build chain 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 CLI will generate the rest of the build chain from them. For example, a typical directive in webfx.xml will be:

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

This directive is asking the CLI to identify the list of your dependencies from an analysis of your source code, and automatically populate the dependencies in pom.xml, module-info.java, module.gwt.xml, etc...

During that process, the CLI takes care of the cross-platform aspects: when a feature is platform-

dependent (a different implementation exists for different platforms), it will pick up the right modules (those whose implementation matches the target platform). This is at this point for example that it will replace the OpenJFX modules with the WebFX kit ones in your GWT application module.

Installing the WebFX CLI

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

We will distribute the CLI in a better way with the first WebFX 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 CLI from a terminal, we need to create a permanent *webfx* alias. This is done with the following commands (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 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 for you.

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

The CLI can display help at different levels. For example, `webfx bump --help` will display all available bump sub-commands, and `webfx bump cli --help` the usage of that specific command.

Creating your first WebFX app

Creating and initializing your repository

Let's create our first WebFX application. We need to create the repository directory and ask the 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 CLI takes the repository directory name as the artifactId when omitted in the command.

The init command creates only 2 files: webfx.xml and pom.xml. If this is the first time you use the CLI, it will download some other files through Maven to get the essential information about the WebFX modules, before completing this job.

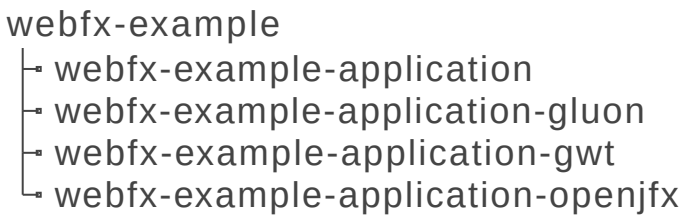
Creating your application modules

When we create an application, we pass the fully qualified name of the JavaFX class we want to create, and the prefix to use for the application modules:

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

we could omit the prefix here, because the CLI takes the parent module name in that case.

This command created the following modules:



The JavaFX class is located in the first module. Normally its `start()` method is empty at this stage, but because we specified the `--helloWorld` option, it has been populated with this simple template:

```
public class WebFxExampleApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setScene(new Scene(new StackPane(new Text("Hello world!")), 800,
600));
        primaryStage.show();
    }

}
```

Building your application

You can build your application using the `webfx build` command, passing it the following build flag(s):

Build platform	Build flag	Target platform	Bump command	Generated file(s)
Linux, macOS or Windows	--gwt	Web		html/JS + war
Linux, macOS or Windows	--openjfx-fatjar	Any desktop with Java		fat jar
Linux	--openjfx-desktop	Linux (embed JVM)		executable, .rpm, .deb
macOS	--openjfx-desktop	macOS (embed JVM)		executable, .dmg, .pkg

Build platform	Build flag	Target platform	Bump command	Generated file(s)
Windows	--openjfx-desktop	Windows (embed JVM)	wix*, inno*	executable, .msi, .exe
Linux	--gluon-desktop	Linux (native)	graalvm, ubuntu-devtools	executable
macOS	--gluon-desktop	macOS (native)	graalvm	executable, .dmg, .pkg
Windows	--gluon-desktop	Windows (native)	graalvm, wix*	executable, .msi
Linux	--gluon-mobile	Android (native)	graalvm, ubuntu-devtools	.apk, .aab
macOS	--gluon-mobile	iOS (native)	graalvm	.app, .ipa

* these installer tools are optional

Some builds require additional pieces of software to work. You can use `webfx bump graalvm`, `webfx bump wix`, etc… to install or upgrade them on your build machine. Also on macOS, for builds other than -gwt and -openjfx-fatjar, Xcode must be installed and configured for a proper code signing (otherwise the executables will refuse to run).

For now, let's just build the Web and OpenJFX desktop apps:

```
webfx build --gwt --openjfx-desktop
```

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

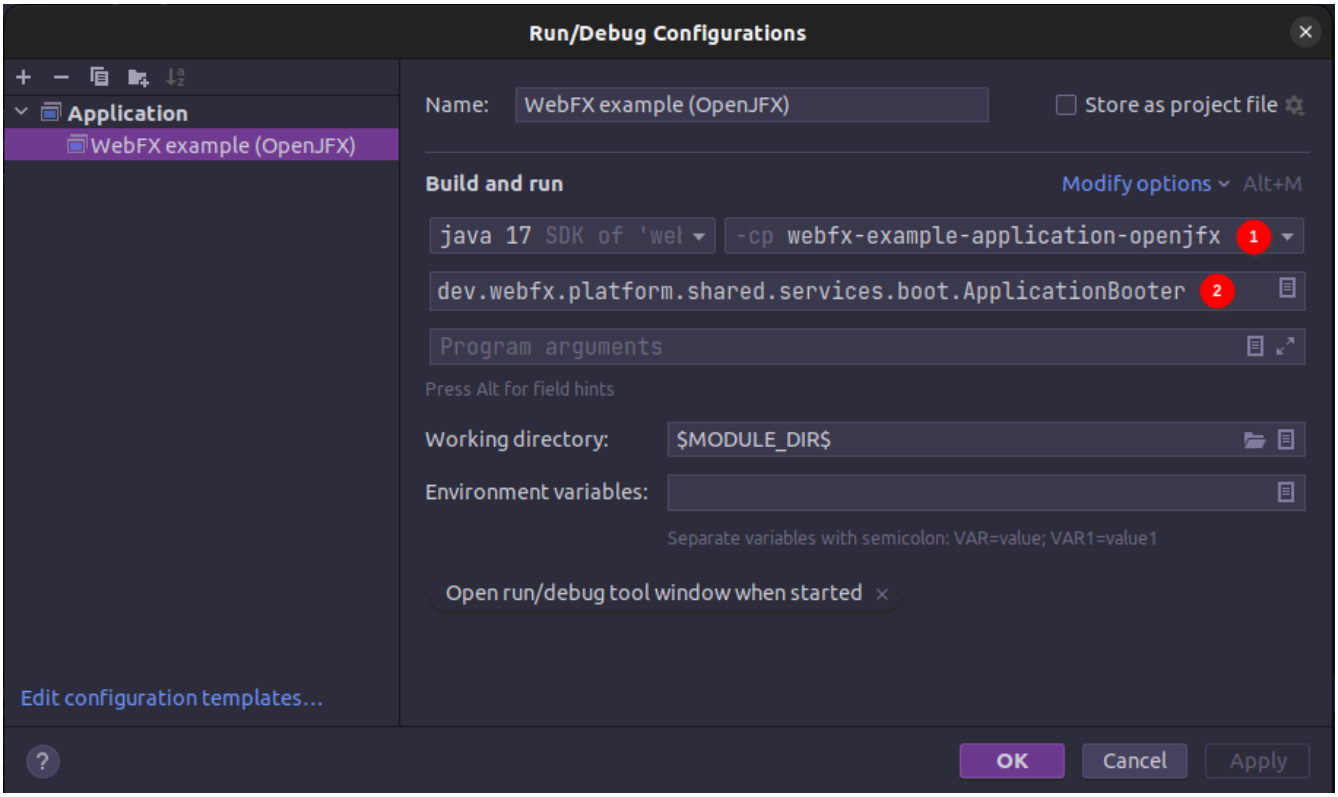
The screenshots will be taken from IntelliJ IDEA, but you can easily transpose them to other Java IDEs.

Opening the project

Open the webfx-example directory from your Java IDE. It will recognize it as a Maven project, and import it.

Building and running the OpenJFX application

Create an application configuration as follows:



① select the OpenJFX application module

② enter `dev.webfx.platform.shared.services.boot.ApplicationBooter` for the main class

You can just type `AB` for the main class, and your IDE should quickly find and suggest the WebFX ApplicationBooter class.

The way to boot GWT and OpenJFX applications are different, but WebFX offers a cross-platform way to do it. For this reason, the main class of a WebFX application is always dev.webfx.platform.shared.services.boot.ApplicationBooter. It will find your JavaFX application because it has been automatically declared as a Java service by the 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 have platform-dependent implementations. A service can be a cross-platform UI API for example, with an OpenJFX implementation, and a different GWT implementation using a JS library you want for your web app.

If you run this configuration, it will build and run your WebFX application in your IDE with the OpenJFX runtime. This is this configuration that you will use to develop and debug your application.

Building and running the GWT application

Making changes

```
webfx update
```

Building the standard executables (desktop)

```
webfx build --openjfx-desktop
```

```
macos: sudo spctl --master-disable
```

Building the native executables (desktop & mobiles)

Installing GraalVM

```
webfx bump graalvm
```

Building the native desktop app

```
webfx -m webfx-example-application-gluon build --gluon-desktop
```

Building the native mobile app

```
webfx -m webfx-example-application-gluon build --gluon-mobile
```