WebFX Documentation

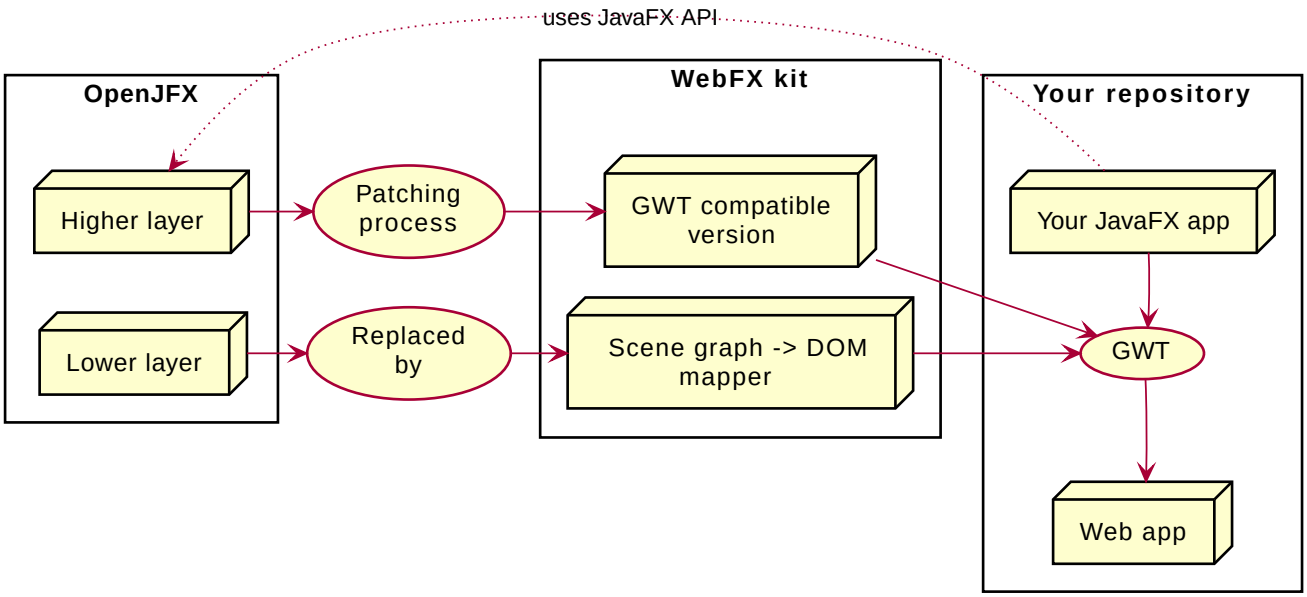
Table of Contents

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 may take some time to complete its coverage - thank you for your understanding. In the near future we hope to release a set of contributor guidelines and will endeavour to build an online community of developers around the project.

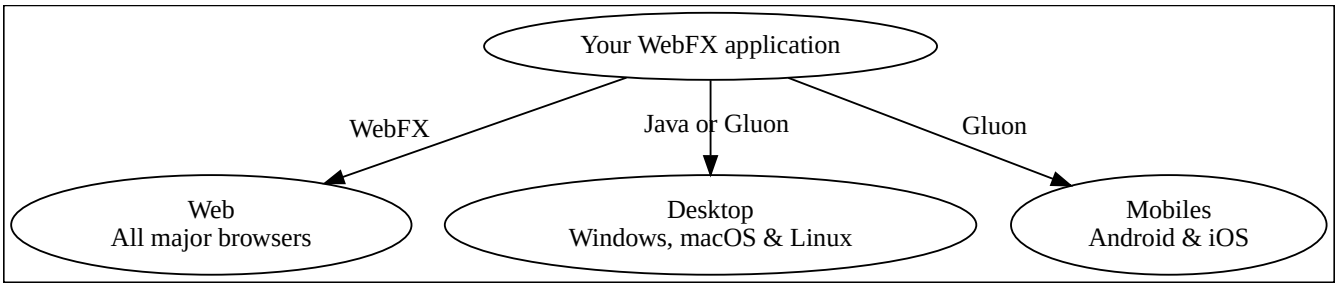
Benefits

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 the benefit of WebFX, and probably the main reason why you would prefer it over the other existing solutions.

Cross-platform



In addition to the web platform, a WebFX application can also run on desktops & mobiles. WebFX will invoke the standard Java toolchain to produce desktop executables (embedding an optimized JRE), and the Gluon toolchains to produce native executables for the desktop and mobiles.

Some alternative technologies or JVM languages allow you to do the same but only for the application logic, because they don't offer a cross-platform UI toolkit. With WebFX, you can do a full cross-platform development of your entire application from a single-source codebase.

Java full-stack

Writing your whole stack in Java is a big advantage, keeping your environment simple and homogenous from a single Java IDE. This prevents you from having to master other complex ecosystems such as JavaScript or TypeScript, and also allows you to share common code between your backend and frontend using the Java module system - a great advantage compared to heterogeneous systems.

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

WebFX application	JS size *
Website	218 kB

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

The secret? Three things:

- 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 (tree-shaking) process, which removes the JavaFX classes not used by the WebFX application.
- GWT produces amazingly compact and optimized JS code, leading to highly-performant web apps.

Gradual learning curve

WebFX is not an additional UI toolkit to learn - it's an emulation of the well-known and well-documented JavaFX API. All the powerful JavaFX features that you know and love are available for your web app. You will feel right 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 released under Apache 2.0 license.

Getting started

Prerequisites

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

Introducing the WebFX CLI

The WebFX CLI is an essential 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)
```

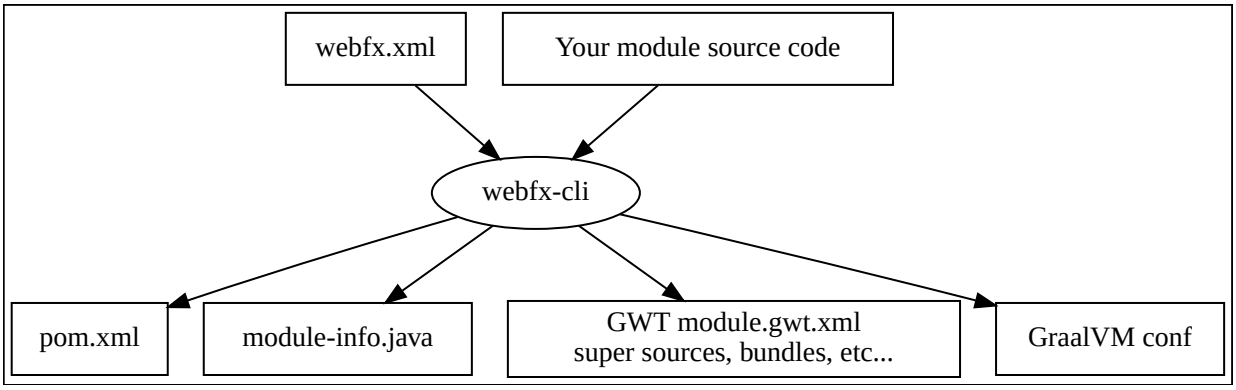
① This module contains the JavaFX code of your application. It is cross-platform (not yet bound to a specific platform) and therefore not executable.

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 the OS of your local machine).

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 Java toolchain to produce the desktop executables (Windows, macOS or Linux) with an embed JRE.

You can create several WebFX applications in the same repository. As 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:



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). It is at this point, for example, that the CLI will replace the OpenJFX modules with the WebFX kit modules in your GWT application module.

Installing the WebFX CLI

Since we haven't published an official WebFX release 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

SSH

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

HTTPS

```
git clone https://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
```

The `webfx.sh` and `webfx.bat` script files (located under the webfx-cli directory) are simply executing the same fat jar. We will use them in the next step.

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='sh \"$(cd "$(dirname "$1")" && pwd -P)/$(basename
"$1")/webfx.sh\"'" >> ~/.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='sh \"$(cd "$(dirname "$1")" && pwd -P)/$(basename
"$1")/webfx.sh\"'" >> ~/.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='sh \"$(cd "$(dirname "$1")" && pwd -P)/$(basename
"$1")/webfx.sh\"'" >> ~/.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)] `$params) { .
'$((Get-Item .).fullName)\webfx.bat' `$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

Windows (manual)

```
You can simply add the webfx-cli repository to your environment path, so webfx.bat
will be directly executed when typing the webfx command.
```

Now 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 so far the only command that uses `git` (a `git pull` of the webfx-cli repository). The CLI 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 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. Note that if this is the first time you have used the CLI, it will download some other files through Maven to retrieve essential information about the available WebFX modules.

Creating your application modules

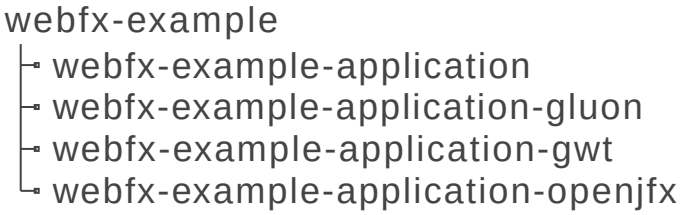
When we create an application, we pass the fully qualified name of the JavaFX class we want to

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

The following command will do a simple build of your application (without generating any final executable):

```
webfx build
```

To generate the executables for the different platforms, you need to pass some extra build options. You can have the list in the build help:

```
webfx build --help
```

Some builds require the installation of third-party software, as shown in the following table:

Build platform	Target platform	Install command	Build option	Executable file(s)
Linux, macOS or Windows	Web		--gwt	html
Linux, macOS or Windows	Any desktop with Java		--openjfx-fatjar	fat jar
Linux	Linux (embed JRE)	ubuntu-tools*	--openjfx-desktop	executable, .rpm, .deb
macOS	macOS (embed JRE)	xcode-tools	--openjfx-desktop	executable, .dmg, .pkg
Windows	Windows (embed JRE)	wix*, inno*	--openjfx-desktop	executable, .msi, .exe
Linux	Linux (native)	graalvm, ubuntu-tools	--gluon-desktop	executable
macOS	macOS (native)	graalvm, xcode-tools	--gluon-desktop	executable, .dmg, .pkg
Windows	Windows (native)	graalvm, vs-tools, wix*	--gluon-desktop	executable, .msi
Linux	Android (native)	graalvm, ubuntu-tools	--gluon-android (or --gluon-mobile)	.apk
macOS	iOS (native)	graalvm, xcode-tools	--gluon-ios (or --gluon-mobile)	.ipa

* these tools are optional, they are used to create installers (.rpm, .deb, .dmg, .pkg, .msi or .exe)

We will now cover each target platform in detail.

Web platform

You can use the following commands to target the Web platform:

Long syntax

```
webfx build --gwt ①
webfx build --gwt --locate ②
webfx build --gwt --reveal ③
webfx run --gwt ④
```

① Build the html executable file with the GWT Maven plugin

② Locate the generated executable file

③ Reveal the generated executable file in the file explorer

④ Execute the generated executable in the browser

Short syntax

```
webfx build -g ①
webfx build -gl ②
webfx build -gr ③
webfx run -g ④
```

① Build the html executable file with the GWT Maven plugin

② Locate the generated executable file

③ Reveal the generated executable file in the file explorer

④ Execute the generated executable in the browser

As opposed to `--reveal`, `--locate` works even before the build, as it prints the expected location, whether the executable file is present or not.

Desktop platform (JAR)

This executable requires Java installed on the target machine.

You can use the following commands to target the Desktop platform (JAR):

Long syntax

```
webfx build --openjfx-fatjar
webfx run --openjfx-fatjar
```

Short syntax

```
webfx build -f
webfx run -f
```

The `run` command has the same options as the `build` command. So the `--reveal` and `--locate` options work with both commands.

You can combine the build options. For example `webfx build --gwt --openjfx -fatjar` (short syntax: `webfx build -gf`) will build both the GWT html and OpenJFX fat jar executables.

Desktop platform (embed JRE)

As opposed to the JAR, the embed JRE will be specific to the target platform. Therefore, a Linux machine will build a Linux executable, a Mac a macOS executable, and a Windows machine a Windows executable.

Prerequisite: you can install the following software if you wish to also generate the installers:

Linux

```
webfx install ubuntu-tools ①
```

① Optional. These tools are used to generate the .deb and .rpm installers.

macOS

```
webfx install xcode-tools ①
```

① Required for code signing

Windows

```
webfx install wix ①
webfx install inno ②
```

① Optional. The WiX Toolset is used to create a simple .msi installer.

② Optional. Inno Setup is used to create a more elaborate .exe installer.

The CLI will help you to customize these installers in a next version. For now, we just use the default settings.

Then, you can use the following commands to target the Desktop platform (embed JRE):

Long syntax

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

Short syntax

```
webfx build -k
webfx run -k
```

Desktop platform (native)

Like for the embed JRE, 3 different machines are required to target the Linux, macOS, and Windows executables.

Prerequisites: you must install the following software for a successful build:

Linux

```
webfx install graalvm ①
webfx install ubuntu-tools ①
```

① Required for the Gluon toolchain

macOS

```
webfx install graalvm ①
webfx install xcode-tools ②
```

① Required for the Gluon toolchain

② Required for code signing

Windows

```
webfx install graalvm ①
webfx install vs-tools ①
webfx install wix ②
```

① Required for the Gluon toolchain

② Optional. WiX Toolset is used to create a simple .msi installer.

Then, you can use the below commands to target the Desktop platform (native):

Long syntax

```
webfx build --gluon-desktop
webfx run --gluon-desktop
```

Short syntax

```
webfx build -t
webfx run -t
```

There is an issue with the Gluon Maven plugin which may cause the build to fail on some versions of macOS. Details are provided in the GitHub issue here.

Android platform

A Linux machine is required to build the Android executable.

Prerequisites: you must install the following software for a successful build:

```
webfx install graalvm
webfx install ubuntu-tools
```

Then, you can use the following commands to target the Android platform (native):

Long syntax

```
webfx build --gluon-android
webfx run --gluon-android ①
```

① Will invoke the Gluon Maven plugin to install and run the executable on your Android device connected to your Linux machine via USB.

Short syntax

```
webfx build -a
webfx run -a ①
```

① Will invoke the Gluon Maven plugin to install and run the executable on your Android device connected to your Linux machine via USB.

Alternatively, you can use the `--gluon-mobile` (short: `-b`) cross-platform option. It will be interpreted as `--gluon-android` on Linux machines.

iOS platform

A Mac is required to build the iOS executable.

Prerequisite: you must install the following software for a successful build:

```
webfx install graalvm
```

Then, you can use the following commands to target the iOS platform (native):

Long syntax

```
webfx build --gluon-ios
webfx run --gluon-ios ①
```

① Will invoke the Gluon Maven plugin to install and run the executable on your iOS device connected to your Mac via USB.

Short syntax

```
webfx build -i
webfx run -i ①
```

① Will invoke the Gluon Maven plugin to install and run the executable on your iOS device connected to your Mac via USB.

Alternatively, you can use the `--gluon-mobile` (short: `-b`) cross-platform option. It will be interpreted as `--gluon-ios` on Macs.

GitHub workflows

In a future release of the CLI we plan to support automatic generation of GitHub workflows. Each push to the main branch of your GitHub repository will then trigger the generation of all your executables on the Linux, macOS and Windows GitHub servers.

For now, you can check out the demos to see sample GitHub workflow configuration. For example, the FX2048 GitHub workflow and the generated executables (expand the `Assets` link to view them).

Developing in your IDE

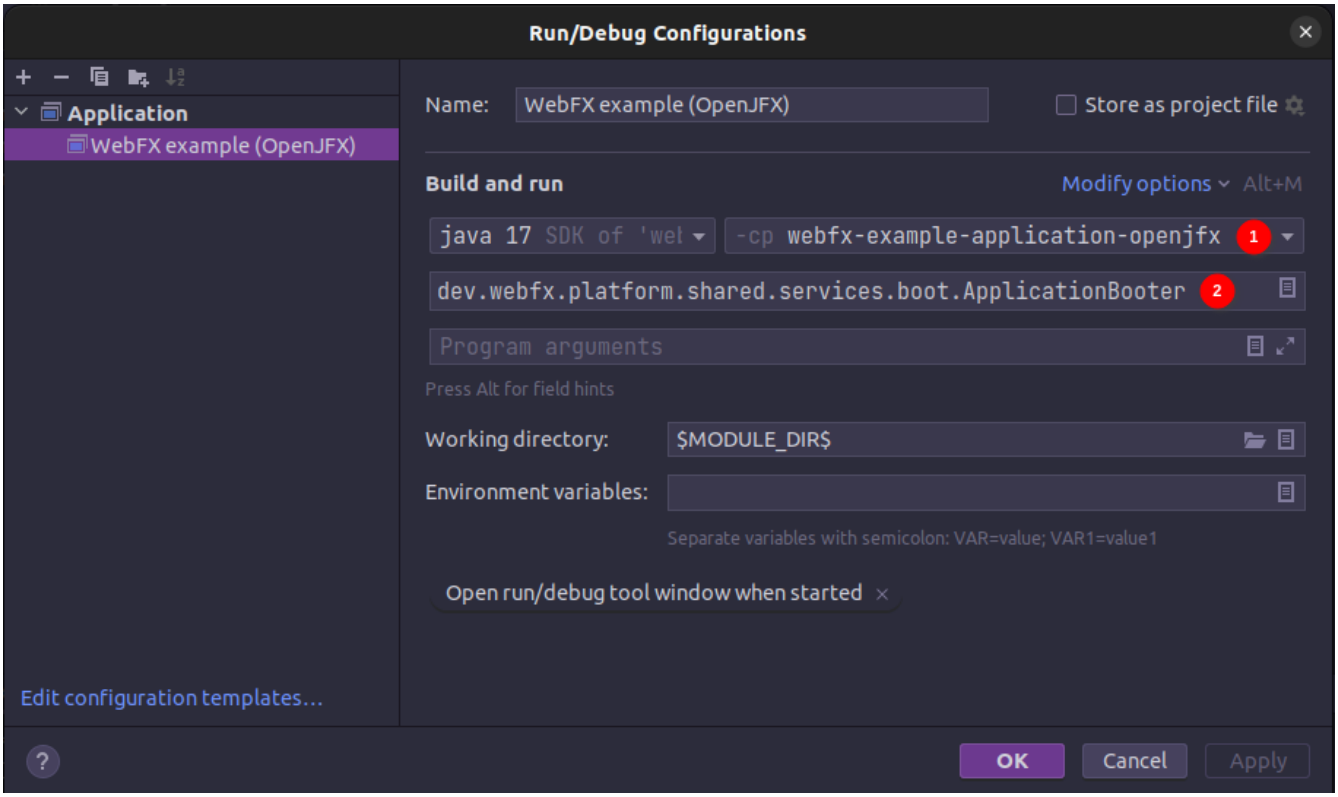
Instructions for IntelliJ IDEA are provided below, which can be quite easily transposed into other Java IDEs.

Opening the project

Open the webfx-example directory from your Java IDE. It should recognize the directory 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.

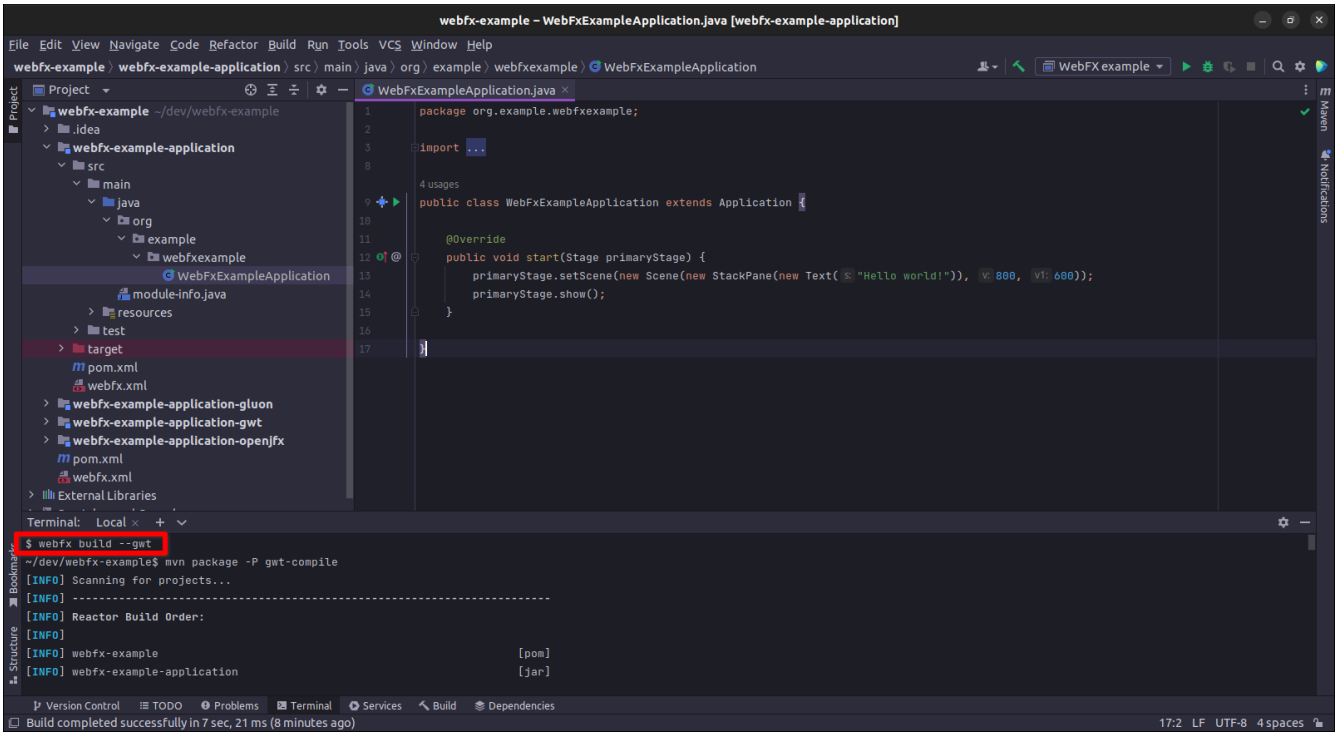
The way to boot GWT and OpenJFX applications is different, but WebFX offers a cross-platform way to do it. For this reason, the main class of a WebFX application is always dev.webfx.platform.shared.services.boot.ApplicationBooter. It will find your JavaFX application because it has been automatically 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 the JS library you want for your web app. We will document this later.

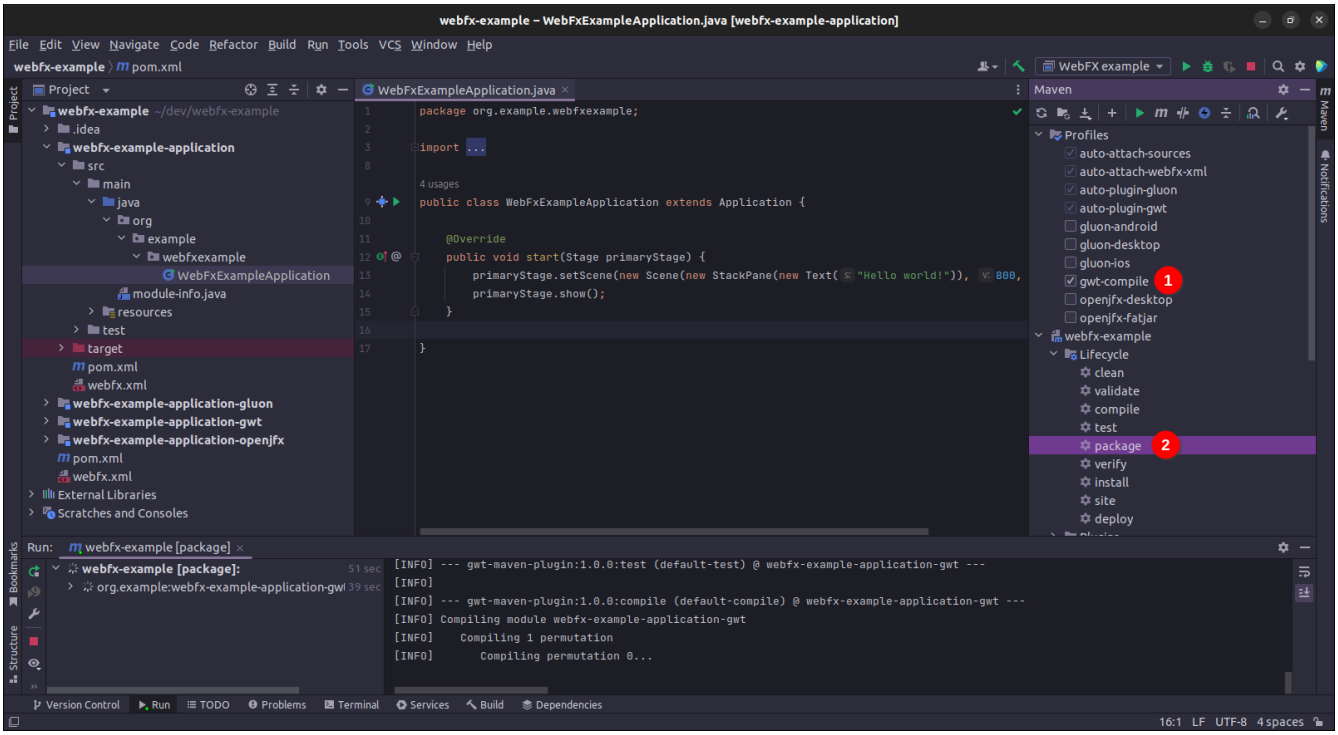
If you run this configuration, it will build and run your WebFX application in your IDE using the OpenJFX runtime. This is the configuration that you will use to develop and debug your application.

Building and running the GWT application

As opposed to the OpenJFX version where the IDE can use its own Java build system, the GWT version can only be built through Maven. One way to build it is to open a terminal window in your IDE, and to type `webfx build --gwt`.



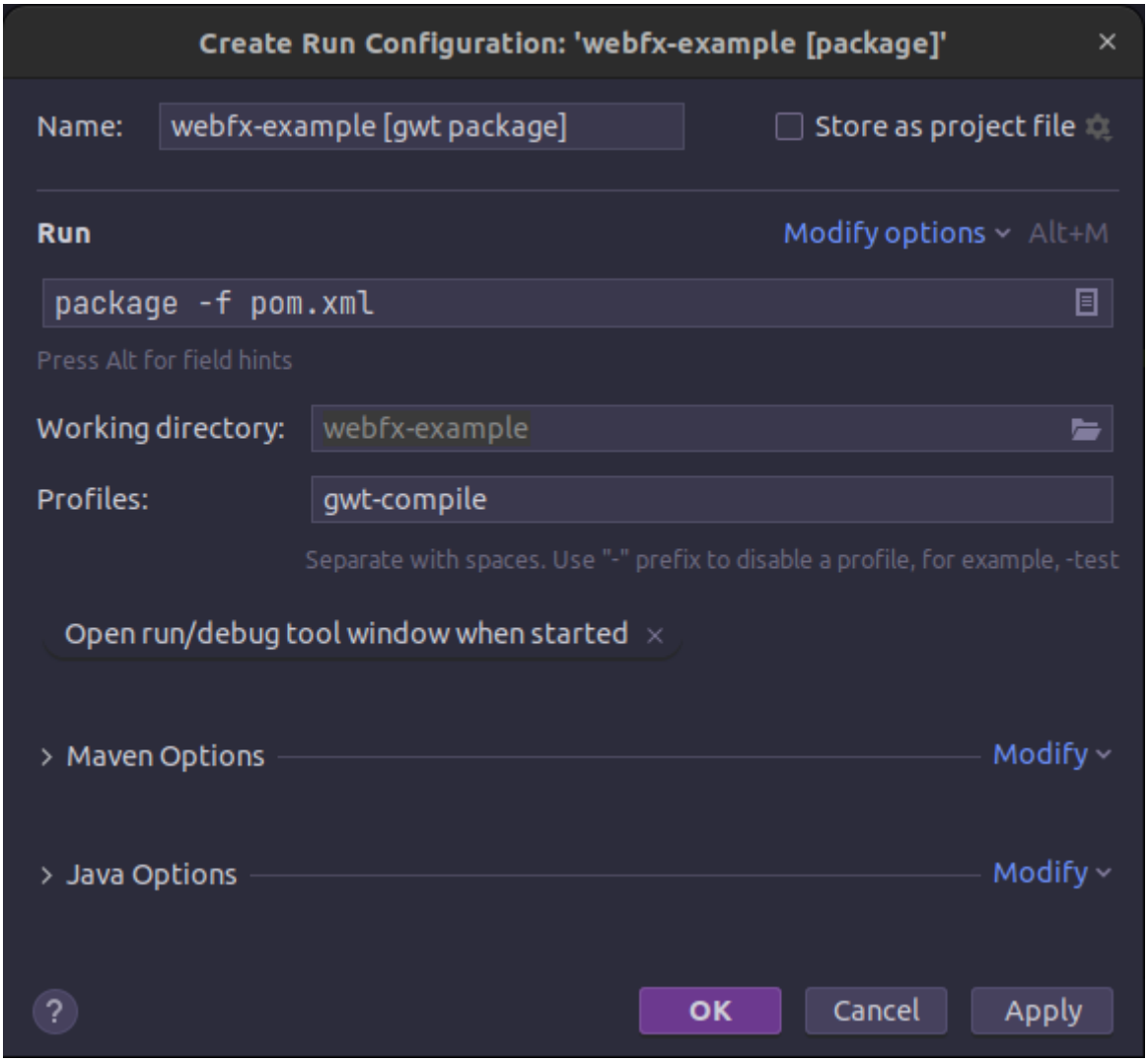
Another way is to trigger the build manually through the IDE Maven window.



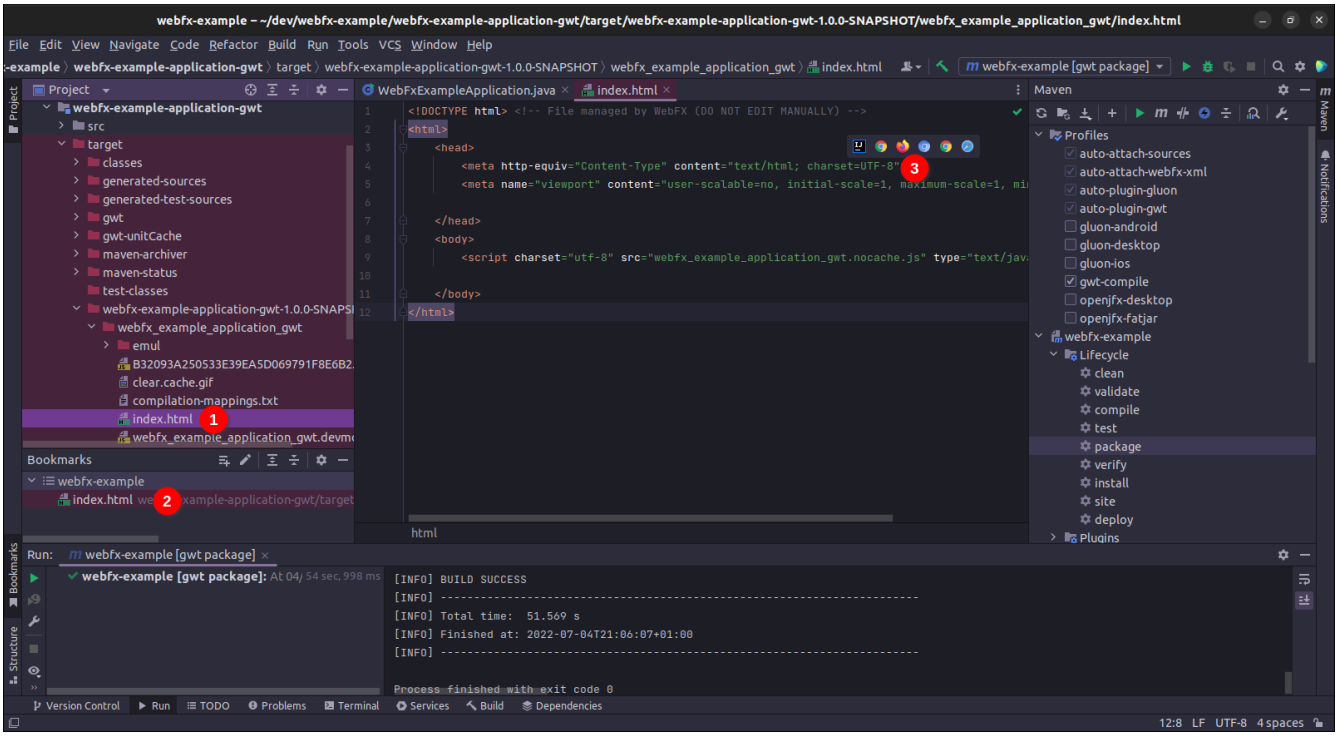
① Activate the `gwt-compile` Maven profile

② Run the `package` Maven goal

It can be a good idea to create a configuration by selecting `Modify Run Configuration` in the context menu of that `package` goal, and press OK in the window below. You can then easily run this configuration to trigger the GWT build.



Once built, you can ask your IDE to open the generated html file in a browser.



① Locate the generated html file

② You can drag & drop it in the Bookmarks window for later

③ Select the browser you want to use to open it

Making changes

Let's do a first little change by replacing the Text with a Button. At first sight, it looks an insignificant change, but actually it's not, because this involves a new dependency to the `javafx-controls` module (our application was only using the `javafx-graphics` module so far). To take advantage of the CLI, let's enter the fully qualified Button class name like this:

```
public class ExampleApplication extends Application {

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

}
```

The IDE doesn't recognize this class yet, but let's ask the CLI to update the build chain from the terminal window:

```
webfx update
```

The CLI should make all necessary changes in the `pom.xml`, `module-info.java`, and `module.gwt.xml` files. After a few seconds, your IDE should automatically detect and consider these changes. Then it should recognize the Button class, and allow you to import it.

Finally, let's add a simple code to react to the button:

```
public class ExampleApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button button = new Button("Hello world!");
        button.setOnAction(event -> button.setText("You clicked me!"));
        primaryStage.setScene(new Scene(new StackPane(button), 800, 600));
        primaryStage.show();
    }

}
```

You can first check the code is working as expected with the OpenJFX runtime by running the OpenJFX configuration. And then build and run the GWT version using one of the methods already

explained.

Our application looks a bit sad? Maybe you can try this little code that adds a bit of colors, animation and effect?

Some more creative ideas pop up in your mind? Now is the time for you to play with WebFX!