JavaFX

User's Guide

Release 26

G50610-01

JavaFX User's Guide, Release 26

G50610-01

Copyright © Oracle and/or its affiliates.

Contents

1 What is JavaFX?

2 Supported JDK Versions

3 Downloading JavaFX

4 Compiling and Running a JavaFX Application

Creating a Custom JRE for a JavaFX Application 2

5 JavaFX Sample Applications

6 JavaFX API Documentation

1 What is JavaFX?

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.

You can customize the look and feel of JavaFX applications. Cascading Style Sheets (CSS) separate appearance and style from implementation so that developers can concentrate on coding. Graphic designers can easily customize the appearance and style of the application through the CSS. See JavaFX CSS Reference Guide.

If you have a web design background, or if you would like to separate the user interface (UI) and the back-end logic, then you can develop the presentation aspects of the UI in the FXML scripting language and use Java code for the application logic. See Introduction to FXML

Key Features

- **WebView.** This is the JavaFX embedded browser, which is a user interface component that provides a web viewer and full browsing functionality through its API. It uses WebKitHTML technology to make it possible to embed web pages within a JavaFX application. JavaScript running in WebView can call Java APIs, and Java APIs can call JavaScript running in WebView. It supports HTML5 features, including Web Sockets, Web Workers, Web Fonts, and printing capabilities.
- **Swing interoperability**: Existing Swing applications can be updated with JavaFX features, such as rich graphics media playback and embedded Web content. Use the `SwingNode` class to embed Swing content into JavaFX applications
- **Built-in UI controls and CSS**: JavaFX provides all the major UI controls that are required to develop a full-featured application. Components can be skinned with standard Web technologies such as CSS.
- **3D Graphics Features**: The JavaFX 3D graphics APIs provide a general purpose three-dimensional graphics library for the JavaFX platform. You can use 3D geometry, cameras, and lights to create, display, and manipulate objects in 3D space.
- **Canvas API**: The JavaFX Canvas API provides a custom texture that you can write to.
- **Printing API**: The `javafx.print` package enables JavaFX applications to show standard print dialogs, discover printers and paper and media capabilities, and render JavaFX `Node` instances to a printer through a `PrinterJob`.
- **Rich Text Support**: JavaFX's enhanced text support includes bidirectional text and complex text scripts, such as Thai and Hindu in controls and multiline and multistyle text in text nodes.
- **Multitouch Support**: JavaFX provides support for multitouch operations, based on the capabilities of the underlying platform.
- **Hi-DPI support**: JavaFX supports Hi-DPI displays.
- **High-performance media engine**: The media pipeline supports the playback of web multimedia content. It provides a stable, low-latency media framework that is based on the GStreamer multimedia framework.

- **Self-contained application deployment model**: See Creating a Custom JRE for a JavaFX Application, which enables you to distribute your JavaFX application as a self-contained application.

2
Supported JDK Versions

The release number of JavaFX corresponds to the JDK release that it supports. For example, JavaFX 26 supports JDK 26.

3 Downloading JavaFX

Download the JavaFX SDK from JavaFX Downloads. You can also download JavaFX JMOD files as a compressed archive file, which you can use to create a custom runtime image for your JavaFX application. See Creating a Custom JRE for a JavaFX Application.

After downloading the JavaFX SDK or the JavaFX JMOD archive file, unzip it or extract its files into a directory on your computer.

4 Compiling and Running a JavaFX Application

The following steps show you how to compile and run a simple JavaFX application.

1. Create the following directory tree:
   - `myapplication`
     - `src`
       - `helloworld`
     - `bin`

   Run all the commands in these steps in the directory `myapplication`.
2. Save the following code in a file named `src/helloworld/HelloWorldFX.java`:

```
package helloworld;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorldFX extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

3. Add an environment variable pointing to the `lib` directory of the JavaFX SDK. For example:
   - **Linux and macOS**:

     ```
     export JAVAFX_SDK=/path/to/javafx_sdk/lib
     ```

   - **Windows**:

     ```
     set JAVAFX_SDK="C:\path\to\javafx_sdk\lib"
     ```

4. Compile `HelloWorld.java`. In the `--module-path` command-line option, specify the `lib` directory of the JavaFX SDK. In the `--add-modules` command-line option, specify the JavaFX modules that your application uses. For example:
   - **Linux and macOS**:

     ```
     javac --module-path $JAVAFX_SDK --add-modules javafx.controls \
           -d bin HelloWorld.java
     ```

   - **Windows**:

     ```
     javac --module-path %JAVAFX_SDK% --add-modules javafx.controls ^
           -d bin src/helloworld/HelloWorldFX.java
     ```

5. Run the application.
   - **Linux and macOS**:

     ```
     java --module-path $JAVAFX_SDK --add-modules javafx.controls \
          --enable-native-access=javafx.graphics -cp bin
     helloworld.HelloWorldFX
     ```

   - **Windows**:

     ```
     java --module-path %JAVAFX_SDK% --add-modules javafx.controls ^
          --enable-native-access=javafx.graphics -cp bin
     helloworld.HelloWorldFX
     ```

> **Note**
>
> Some JavaFX modules, such as `javafx.graphics` call restricted methods, which causes the Java runtime to print a warning message. This example uses the command-line option `--enable-native-access` to suppress this warning.
> See Restricted Methods in *Java Platform, Standard Edition Core Libraries* for more information about restricted methods and the `--enable-native-access` option.

Creating a Custom JRE for a JavaFX Application

You can also download JavaFX JMOD files, which you can use to create a custom runtime image for your JavaFX application.

These steps show you how to create a custom JRE for the `HelloWorldFX` sample application described in Compiling and Running a JavaFX Application. Run all the commands from the `myapplication` directory.

1. In the directory `myapplication`, create a subdirectory named `mods`.
2. Add an environment variable pointing to the directory that contains the JavaFX JMOD files. For example:
   - **Linux and macOS**:
     ```
     export JAVAFX_JMODS=/path/to/javafx_jmods
     ```
   - **Windows**:
     ```
     set JAVAFX_JMODS="C:\path\to\javafx_jmods"
     ```
3. Save the following code in a file named `src/module-info.java`
   ```
   module helloworld {
       requires javafx.controls;
       exports helloworld;
   }
   ```
4. Compile the `HelloWorldFX` application as a module:
   - **Linux and macOS**
     ```
     javac --module-path $JAVAFX_MODS -d mods/helloworld \
           src/module-info.java src/helloworld/HelloWorldFX.java
     ```
   - **Windows**
     ```
     javac --module-path %JAVAFX_MODS% -d mods\helloworld ^
           src\module-info.java src\helloworld/HelloWorldFX.java
     ```

   Note

   To compile all Java source files in a directory, use the following command:
   - **Linux and macOS**
     ```
     javac --module-path $JAVAFX_MODS -d mods/helloworld $(find src/ -
     name "*.java")
     ```
   - **Windows**
     ```
     dir /s /b src\*.java > source-files.txt & javac --module-path
     %JAVAFX_MODS% ^
         -d mods/helloworld @source-files.txt & del source-files.txt
     ```
5. Test the `HelloWorldFX` modular application by running it:

- **Linux and macOS**

  ```
  java --module-path $JAVAFX_SDK:mods --enable-native-
  access=javafx.graphics \
       -m helloworld/helloworld.HelloWorldFX
  ```

- **Windows**

  ```
  java --module-path "%JAVAFX_SDK%;mods" --enable-native-
  access=javafx.graphics ^
       -m helloworld/helloworld.HelloWorldFX
  ```

6. Create a custom JRE for the `HelloWorldFX` modular application with `jlink`:

   - **Linux and macOS**

     ```
     jlink --module-path %JAVAFX_MODS:mods --add-modules helloworld --output
     helloworld
     ```

   - **Windows**

     ```
     jlink --module-path "%JAVAFX_MODS%;mods" --add-modules helloworld --
     output helloworld
     ```

7. Run `HelloWorldFX` with the custom JRE you just created:

   - **Linux and macOS**

     ```
     helloworld/bin/java --enable-native-access=javafx.graphics -m
     helloworld/helloworld.HelloWorldFX
     ```

   - **Windows**

     ```
     helloworld\bin\java --enable-native-access=javafx.graphics -m
     helloworld/helloworld.HelloWorldFX
     ```

5

JavaFX Sample Applications

- The `jfx/apps/samples` directory in the OpenJDK GitHub repository
- Getting Started with JavaFX Sample Applications in the Java SE 8 documentation

6 JavaFX API Documentation

- JavaFX API Documentation
- JavaFX CSS Reference Guide
- Introduction to FXML