JavaFX Migration Guide

Using JavaFX with the latest JDK versions: A guide for developers familiar with JavaFX on JDK 8

This document serves as a guide for developers familiar with JavaFX from JDK 8 who wish to take advantage of the open source, stand alone, JavaFX bundle available for the latest JDK Feature releases.

Note : Please see the JDK migration guide for information on migrating your application from JDK 8u to newer JDK versions. This document covers the additional information specific to JavaFX.

Modifications required to JavaFX application

While JavaFX generally takes care of the backward compatibility, a migration from JavaFX-8u to the current version of JavaFX is a big leap. A lot has changed between JDK 8 and later releases. Almost all of those changes will be beneficial, but there are a few that will require some adjustments to application's code.

Why is this a significant leap?

- Modularization was introduced in JavaFX 9 (part of JDK 9). This version introduced encapsulation resulting in access removal of internal APIs which were public in JavaFX-8u.
- JavaFX 11 was un-bundled from JDK from version JDK 11. JavaFX was redesigned to be available as a stand-alone library rather than being included with the JDK.
- Accumulated changes from JavaFX 11 to the current version of JavaFX - some deprecation and new API additions.
- The JavaFX API has evolved iteratively over these intermediate versions and you might find that some of the APIs you are familiar with have been removed or deprecated.

A list of deprecated classes and APIs is available at -

https://download.java.net/java/GA/javafx24/docs/api/deprecated-list.html

An exhaustive list of new classes and APIs is available at -

https://download.java.net/java/GA/javafx24/docs/api/new-list.html

Here is a list of most important changes -

- Encapsulation was introduced in JDK 9. It did three things -
  - Revoked the access of non-public APIs (classes under com.sun.* packages)
  - Removed several deprecated and undocumented "impl_*" methods. (Refer - JDK-8144585)
  - Removed the internal Skin and CSS APIs from `com.sun.javafx.scene.control.skin` and `com.sun.javafx.css`.
- Important APIs were added to provide replacements for earlier internal APIs
  - A public Skin API was created in the `javafx.scene.control.skin` package, replacing the internal Skin classes formerly in `com.sun.javafx.scene.control.skin`. (See JDK-8077916)
  - A public CSS API was created in the `javafx.css` package, replacing the internal CSS classes formerly in `com.sun.javafx.css`. (See JDK-8077918)

- The Java Deployment stack was removed in JDK 11. There is no ability to run a JavaFX application as an Applet or a WebStart application.
- The JavaFX builder classes, which were previously deprecated in JDK 8 with the stated intention to remove them, have been removed from JDK 9. JavaFX applications that use the builder classes should instead construct the needed scene graph objects directly and set the desired properties with the equivalent method calls. (Refer - JDK-8092861)
- Support for VP6 video encoding format and FXM/FLV container has been removed in JavaFX Media. Users are encouraged to use H.264/AVC1 in the MP4 container or HTTP Live Streaming instead. (Refer - JDK-8187637)
- FX Media support for libavcodec 53 and 55 was removed. These libraries are not present on supported Linux platforms by default, and are no longer needed. (Refer JDK-8194062)
- JavaFX Requires GTK 3 on Linux (GTK 2 support was removed) (Refer - JDK-8299595)
- The `HostServices::getWebContext` method has been removed. There is no replacement for this functionality. This method was only used when running a JavaFX Applet, which is no longer available" (Refer - JDK-8187149)
- The security manager, which was used by some client applications, is no longer available. See https://openjdk.org/jeps/486 for details.

Important new APIs -

javafx.controls	javafx.scene.control.skin	New package containing several Skin classes
javafx.graphics	javafx.css	New package containing several CSS related classes

javafx.graphics	javafx.application	Platform.enterNestedEventLoop(Object) Platform.exitNestedEventLoop(Object, Object) Platform.isNestedLoopRunning() Platform.requestNextPulse() Platform.startup(Runnable)
javafx.graphics	javafx.stage	Window.getWindows()
javafx.fxml	javafx.fxml	FXMLLoader.getLoadListener() FXMLLoader.setLoadListener(LoadListener)
javafx.graphics	javafx.scene.text	Font.loadFonts(InputStream, double) Font.loadFonts(String, double) Text.caretBiasProperty() Text.caretPositionProperty() Text.caretShape(int, boolean) Text.caretShapeProperty() Text.hitTest(Point2D) Text.rangeShape(int, int) Text.selectionEndProperty() Text.selectionFillProperty() Text.selectionShapeProperty() Text.selectionStartProperty() Text.underlineShape(int, int) TextFlow.caretShape(int, boolean) TextFlow.hitTest(Point2D) TextFlow.rangeShape(int, int)

Building a JavaFX application with JavaFX 24

- Starting with JDK 11, JavaFX was open sourced and redesigned to be available as a stand-alone library rather than being included with the JDK.
- The JavaFX runtime is delivered as an SDK and as a set of jmods for each platform.
- There are two ways you can use the JavaFX runtime with your application. Both of these methods presume that you have already downloaded JDK 24, set JAVA_HOME to point to it, and put JDK 24 in your PATH.

1. Use the JavaFX SDK to compile and run your JavaFX application

Download & unzip the SDK for your platform from https://jdk.java.net/javafx24

Put the javafx modules on your module-path when you compile or run, and list the javafx modules you need using --add-modules

```
$ javac --module-path javafx-sdk-24/lib --add-modules javafx.controls MyFXApp.java
$ java --module-path javafx-sdk-24/lib --add-modules javafx.controls MyFXApp
```

Modular apps don't have to specify --add-modules, as the needed modules are in module-info.java of the application.

2. Create a JDK using the JavaFX JMODs

An even easier way to compile and run JavaFX applications is to create a custom JDK that includes the JavaFX modules. You can optionally add your modular application to this custom JDK.

Download & unzip jmods for your platform from https://jdk.java.net/javafx24

Run jlink to produce a JDK that includes the JavaFX modules:

```
$ jlink --output jdk-24+javafx-24 \
        --module-path javafx-jmods-24:$JAVA_HOME/jmods \
        --add-modules ALL-MODULE-PATH
```

Compile and run your application as follows:

```
jdk-24+javafx-24/bin/javac MyFXApp.java
jdk-24+javafx-24/bin/java MyFXApp
```