JavaFX.Next

Live for the Code

Kevin Rushforth – Oracle
Johan Vos – Gluon
October 2018

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Agenda

- JavaFX: A Brief History
- JavaFX Unplugged
- JavaFX 11 releases
- Community Involvement
- JavaFX 12 and beyond
- Q & A

JavaFX: A Brief History

- JavaFX 1 scripting language (2008)
  - Applications written in new JavaFX script language
  - Released separately from JDK
  - Included support for applets, Web Start
- JavaFX 2 recast as Java API (2011)
  - Released as separate SDK
    - later “co-shipped” with JDK7, but not on default classpath
  - OpenJFX Project was born!
    - Project is under the OpenJDK umbrella
    - Parts of JavaFX were open-sourced in FX 2 (scene graph and UI controls)
      - This enabled non-Oracle contributions

JavaFX: A Brief History

- JavaFX 8 delivered as part of JDK 8, on default classpath (2014)
  - Entire platform (*) was open-sourced
    - (*) minus deploy/plugin code and a few optional bits that we couldn’t
  - Build is done using OpenJFX sources
- JavaFX 9 javafx modules linked into JDK 9 (2017)
  - This is for Oracle JDK only – the OpenJDK binaries don't include JavaFX
- JavaFX 10 was similarly included in Oracle JDK 10 (Mar 2018)
  - OpenJDK 10 binaries were released for all platforms – without JavaFX

JavaFX Unplugged

- Problem: JavaFX runs on Oracle JDK but not OpenJDK
  - Only way to include JavaFX in JDK is to build both
  - Even for Oracle JDK you often need to build JDK in order to edit / build / debug FX
- Solution: Split out the JavaFX modules from the JDK in JDK 11
  - IMPORTANT: This doesn't mean JavaFX is dead!
  - In a modular world, it doesn’t make sense to add all modules into a monolithic JDK
- Removed dependencies on internal interfaces from java.base, java.desktop
  - New public API in jdk.unsupported.desktop for FX / Swing interop
  - New public API in java.desktop for print dialog support
  - Deployment / plugin removed prior to FX (eliminated additional dependencies)

ORACLE

JavaFX Unplugged

- Oracle JDK 9 and Oracle JDK 10 shipped with a javapackager tool
  - Was built / delivered along with FX, and as such, was never in OpenJDK
- Removing FX from JDK build meant that javapackager tool was also gone
- The javapackager tool didn't fit as part of the standalone FX
  - It is a tool like jlink (with a dependency on a shared private interface)
  - Unlike JavaFX itself, the packager is a natural fit for the JDK
- Solutions for the packager tool:
  - Oracle is working a JEP for a new jpackager tool that we hope to deliver in JDK 12
    - The code review is in progress on core-libs-dev
  - Gluon is hosting a standalone javapackager, based on sources in 12, to fill the gap

JavaFX Unplugged

- JavaFX modules are now built separately from JDK with no odd interaction
  - (e.g., no more adding qualified exports to build.gradle when changing module-info.java; no need for JavaFX developers to build the JDK)
- JavaFX 11 was shipped just before JDK 11
- We plan to ship JavaFX 12 just before JDK 12
- Current plan is to release on a 6 month cadence matching JDK schedule
  - We could change this in future if there is a need (we aren't tied to it)
  - As long as we stay on same release schedule, we plan to use same numbering
    - 11, 12, 13, …

ORACLE

JavaFX 11

- JavaFX 11 delivered as a standalone release at https://openjfx.io/
  - javafx.* modules no longer included in the JDK
- Runs on both OpenJDK 11 and Oracle JDK 11
  - also runs on OpenJDK 10
- In addition to the separation from JDK, JavaFX 11 includes:
  - 90 bug fixes
  - 9 enhancements
  - 18 code contributors to GitHub
  - Win/Mac/Linux officially released
  - Embedded EA, mobile coming

JavaFX 11

- Three ways to run JavaFX apps now:
  - 1. Download the SDK, put it on module path when compiling / running your app
  - 2. Download the JMODs and create a custom Java runtime that:
    - Includes the javafx modules
    - Optionally includes your app, if modular
  - 3. Use maven or gradle to dynamically download the modules from Maven Central

JavaFX 11 – Not in JDK 11

```
package pkg;
import javafx.application.Application;
...
public class HelloFX extends Application {
    public void start(Stage stage) {
        StackPane root = new StackPane(new Button("click me"));
        stage.setScene(new Scene(root, 300, 200));
        stage.show();
    }
    ...
}

$ javac pkg/HelloFX.java
pkg/HelloFX.java:2: error: package javafx.application does not exist
```

- Now what?

JavaFX 11 – Running With SDK

- Download SDK for your platform from https://openjfx.io/
  - Unzip it to /SOMEDIR/javafx-sdk-11/
- Put the javafx modules on your module-path when you compile or run
  - Note that you also need to list the javafx modules you use: --add-modules

```
$ javac --module-path /SOMEWHERE/javafx-sdk-11/lib --add-modules javafx.controls pkg/HelloFX.java
$ java --module-path /SOMEWHERE/javafx-sdk-11/lib --add-modules javafx.controls pkg.HelloFX
```

  - Modular apps don't have to specify --add-module, they are in module-info.java

```
$ java --module-path /SOMEWHERE/javafx-sdk-11/lib:/MY/APP/dist –m myapp/pkg.HelloFX
```

JavaFX 11 – Running With JMODs

- Download jmods for your platform from https://openjfx.io/
  - Unzip it to /SOMEDIR/javafx-jmods-11/
- Use jlink to create custom JDK image with the modules you need
  - Optionally trim what you don't use
  - Add unbundled modules into the images (e.g., javafx.* modules)
  - You can even add your application into the image – if it is modular

JavaFX 11 – Running With JMODs

- Run jlink to produce JDK with modules you need:

```
$ jlink --output myjdk --bind-services \
    --module-path /SOMEWHERE/javafx-jmods-11 \
    --add-modules javafx.media,javafx.fxml,javafx.controls,java.se
```

- Since you added the javafx modules into your custom JDK, no need to specify them again when you compile or run your app:

```
$ myjdk/bin/javac pkg/HelloFX.java
$ myjdk/bin/java pkg.HelloFX
```

JavaFX 11 – Running With JMODs

- Example building modular app into custom runtime (has only modules you need)

```
module myapp {
    requires java.logging;
    requires javafx.fxml;         // if your app uses FXML
    requires javafx.controls;     // controls re-exports graphics and base
    exports pkg to javafx.graphics;
}

$ jlink --output myjdk --add-modules myapp \
    --module-path /SOMEWHERE/javafx-jmods-11:/MY/APP/dist
```

- The custom JDK includes your app, plus all dependent modules (JDK and JavaFX)
  - It doesn't include what you don't need

```
$ myjdk/bin/java –m myapp/pkg.HelloFX
```

JavaFX 11 – Running With Maven

- No need to download the JavaFX SDK
- List FX modules and versions you want in pom.xml as follows:

```
<url>http://maven.apache.org</url>
<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>11</version>
  </dependency>
</dependencies>
```

- You can download an example pom.xml file from https://openjfx.io/
- Run the application; build system downloads javafx modules, platform natives:

```
$ mvn compile exec:java
```

JavaFX 11 – Running With Gradle

- List the modules and versions you depend on in build.gradle as follows:

```
repositories {
  mavenCentral()
}
dependencies {
  compile "org.openjfx:javafx-base:11:$platform"
  compile "org.openjfx:javafx-graphics:11:$platform"
  compile "org.openjfx:javafx-controls:11:$platform"
}
```

- You can download an example build.gradle file from https://openjfx.io/
  - Has script to auto-detect platform and shows how to add modules to your module-path
- Run the application; build system downloads javafx modules, platform natives:

```
$ gradle run
```

JavaFX 11.x Update Releases

- Mainline jfx-dev repo is now for openjfx12
  - Most developers should move to the latest feature release
- Gluon will be releasing periodic updates of 11.x
  - Schedule is still being worked out
- A few critical fixes can be backported to 11-dev repo for 11.x
  - Security fixes
  - Other issues as deemed important
  - Project Lead approval required to backport

Community Involvement

- All code is fully open-sourced in OpenJFX (this was a prerequisite)
  - We removed the last closed code in 10 and the last closed build dependency in 11
- We have had a small number of larger contributions from the community:
  - Marlin (first delivered in FX 9, updated in 10 and again in 11)
  - Maven modules in 11
  - Public Robot API
- And several more smaller contributions:
  - Bug fixes, documentation fixes, missing functionality (e.g., .MathML support)
- We have been taking steps to encourage even more participation
  - Growing the community benefits all of us

Community Involvement

- GitHub mirror to encourage more contributions
  - https://github.com/javafxports/openjdk-jfx (mirrors HG openjfx/jfx-dev/rt repo)
  - Users have familiar tools for collaboration
  - When a user wants to contribute a fix or test an idea, they issue a pull request
  - Integrated CI build system runs a (partial) build / test on all three platforms
    - Makes it easier for contributors to submit fixes if they can't test on, e.g., win or mac
  - In order to track issues and maintain quality, we still require a code review
  - Contributor agreement required before PR is formally reviewed or accepted
  - Will align nicely with Project “Skara”
- Streamlined code review process for simple fixes
  - We want to combine high quality fixes with low administration overhead

Community Involvement

- New features need effort and commitment on the part of the contributor
  - Not simply proposing a change that your particular app would benefit from
  - We don't want "drive-by" contributions
  - Think in terms of API "stewardship"
- Inclusion of new features guided by OpenJFX project leads:
  - Kevin Rushforth (Oracle) and Johan Vos (Gluon)
- Overarching goals for new features:
  - Consistency in the core APIs
  - Maintainability
  - Ability to implement on a wide range of platforms (including mobile devices)

Community Involvement

- Community web site: https://openjfx.io/
  - Download releases
  - Documentation: javadocs + “getting started” guide
  - Example projects using JavaFX
- The content is in a project on GitHub:
  - https://github.com/openjfx/
  - Contributions to the site content are welcome

JavaFX 12

- Proposed schedule:
  - RDP1: Jan 7, 2019 (aka “feature freeze”)
  - RDP2: Feb 4, 2019
  - Freeze: Feb 25, 2019
  - GA: March 12, 2019
- Early access builds should be available soon (shortly after Code One)
- JavaFX 12 will run on JDK 11 and later

JavaFX 12 and Beyond

- Align with JDK releases
  - In general you can count on JavaFX *N* working with JDK *N*-1 and later
  - But we won't break older releases just to break them
- Focus on the core parts of JavaFX
  - Key point: Add functionality when it really makes sense
- Features that can be reasonably done by a library on top of FX should be
  - For example, a rich text editor can be implemented separately
  - If there is fundamental support missing in core, we can add that support

JavaFX 12 and Beyond

- Fix important bugs
  - Including bug fixes from developer community
- Platform support
  - Updating compilers
  - Fixing critical bugs when new macOS versions are released
  - Wayland on Ubuntu
  - Replacement for deprecated platform APIs (Metal replacing OpenGL)
- Updates to WebKit (and other third-party libraries)
- Modern rendering pipelines (e.g., Metal on Mac)

JavaFX 12 and Beyond

- Make sure third party libraries can work and can enhance OpenJFX core
  - Example: adding more public/protected API to VirtualFlow to enable subclassing
  - We are receptive to additions in the core API, if they are well grounded
- Improve documentation:
  - Getting started, IDE integrations, Build instructions for various platforms, etc.
  - openjfx.io site
  - We want others in the community to participate in this

JavaFX 12 and Beyond

Possible features for future releases

- Support for more platforms (mobile/embedded)
- Support for native integration (shared buffers)
  - NIO-backed writable image
- Support for scientific applications (e.g. AI)
- What else would you like to see?
  - Are you willing to help make it happen?

Live for the Code

Q & A

ORACLE

Links

- https://openjfx.io/ – community site for downloads, docs, etc.
- https://hg.openjdk.java.net/openjfx/jfx-dev/rt – official HG repo
- https://github.com/javafxports/openjdk-jfx -- GitHub mirror
- https://github.com/openjfx/ – GitHub project for openjfx.io site
- https://wiki.openjdk.java.net/display/OpenJFX -- OpenJFX Wiki on OpenJDK

- openjfx-dev@openjdk.java.net – developer mailing list
- openjfx-discuss@openjdk.java.net – discussion list for roadmaps, etc.

Live for the Code

ORACLE