JavaFX and its Deployment

Jan Valenta

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

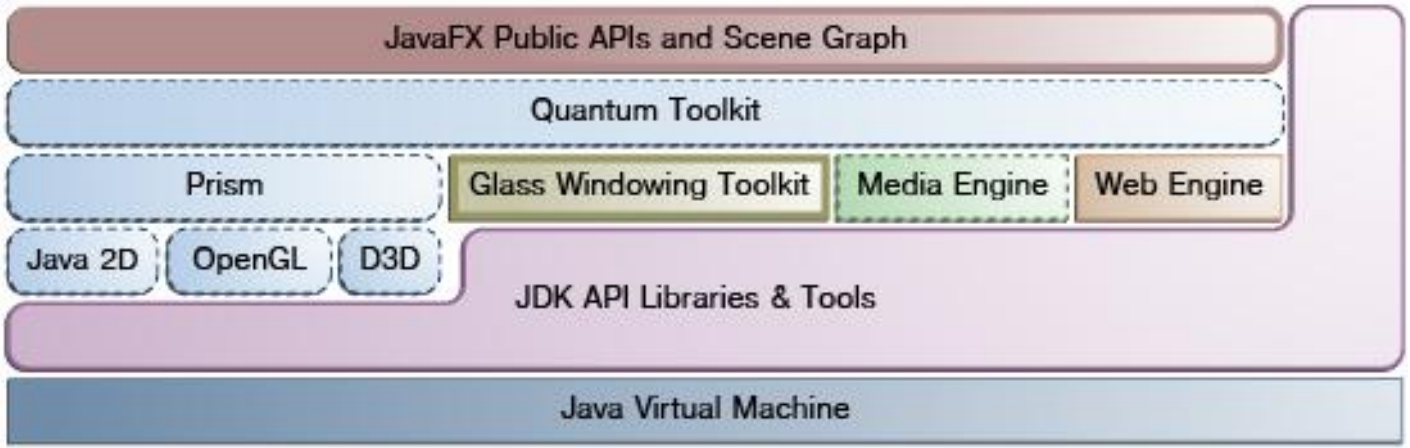
- JavaFX in general
- Deployment
- Execution Modes and Application Startup
- Packaging
- Browser Deployment
- NetBeans IDE
- Documentation

What You Will Learn

- JavaFX toolkit
- Deployment
  - Execution modes
  - Deployment tools
  - Effective documenting

JavaFX

- Graphics
- Media
- Rich Client/Internet Applications
- Architecture



JavaFX API features

- Full Java API access (generics, annotations, threads, 3rd party libraries)
- Binding support
  - Lazy binding, binding expressions
- Collections extension
  - Observable list and maps
- Animations
- Multitouch support
- UI controls
- Layouts, canvas support

Key ecosystem features

- FXML and SceneBuilder
  - XML-based declarative language to describe GUI
  - Can be generated by the SceneBuilder visual application
- Webview
  - based on Webkit, JavaScript call/be called from JavaFX
- Swing interoperability
- 2D & 3D, effects
- HW-accelerated graphics layer
- Media engine
- Possibility of self-contained applications that include the VM

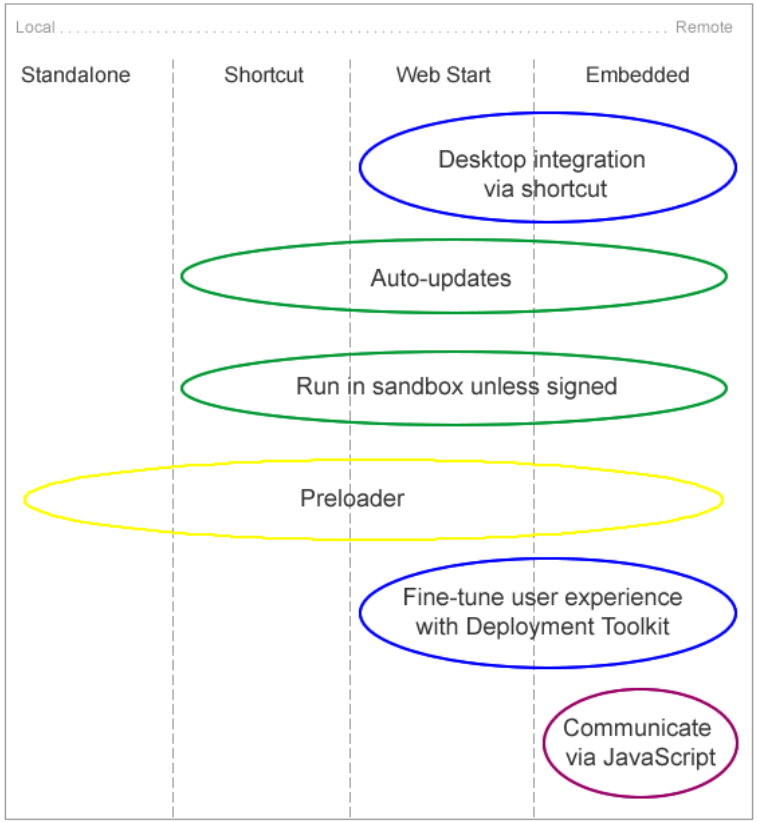
JavaFX Ensemble

JavaFX Deployment

Execution Modes

- Standalone program
  - Package available on a local drive including Java launcher (`java -jar MyApp.jar`, doubleclicking)
  - Native bundle
- Webstart
  - Link on a web page, package on remote server
- Applet
  - Embedded in a web page
- Desktop shortcut
  - For remote Webstart or embedded application

Execution Modes



- Web applications may ask the user to create a desktop shortcut (not applicable to a standalone app.)
- Auto-updates: every time the application starts
- Java Runtime protects the user (resources, connections, native libraries, ClassLoader creation, reading some System Properties)
- JavaScript: application can talk to the embedding web page

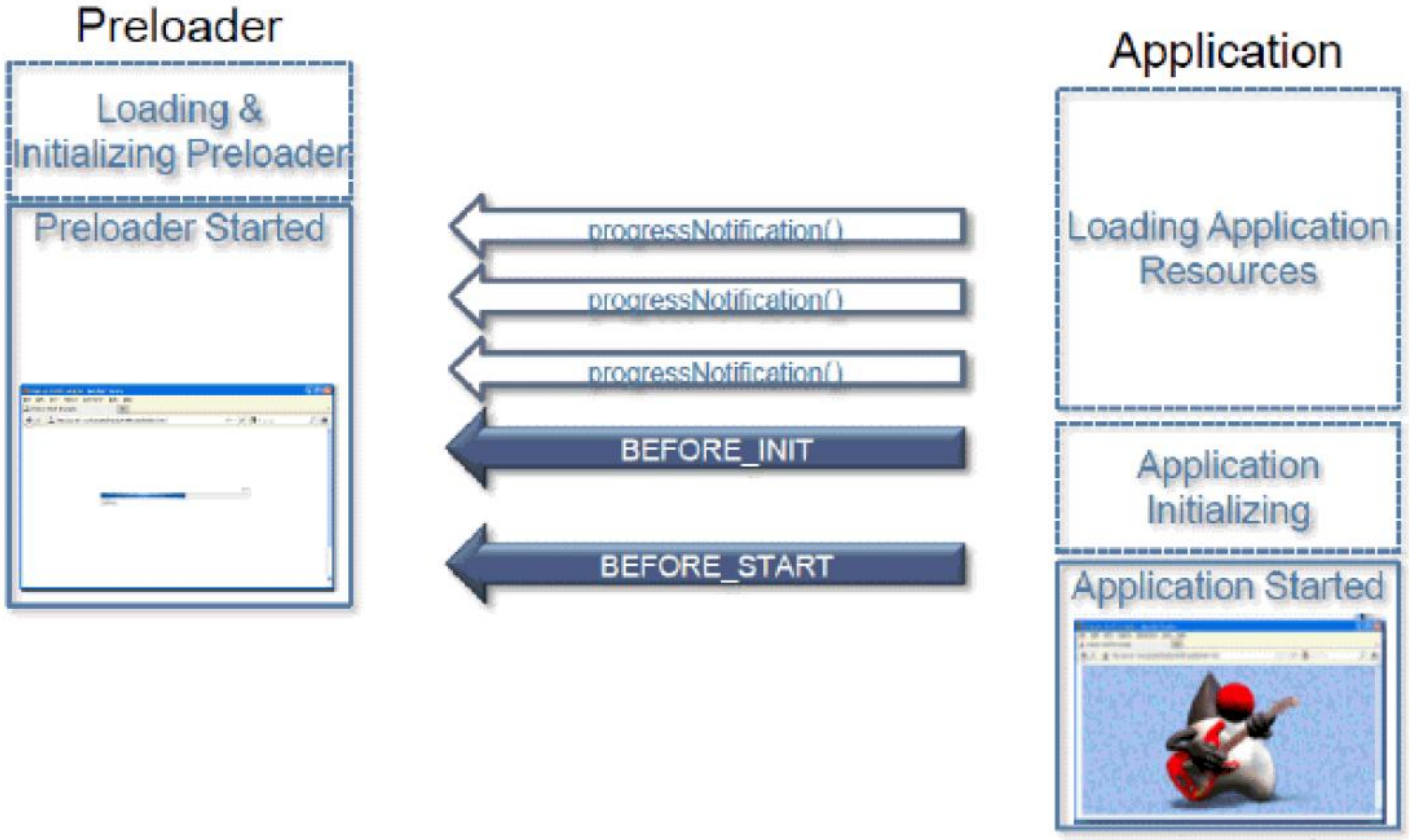
Application Startup

- User experience is very important for success of the applications
- Need to let the user know the status of loading and the progress toward running the application
- During initialization: JavaScript/HTML
- After initialization: customization of loading process possible (preloader)

Application Startup

- Startup sequence:
  - Splashscreen displayed
    - Check/Install/Update prerequisites
    - Init JVM and JavaFX
  - Preloader starts
    - Download Application JARs
    - Check permissions
  - Application Init
    - Application runs

Preloader



Initialization Customization

- Splashscreen: `onGetSplash` javascript callback
- Preloader: custom or default
  - Customizing default preloader via the CSS
    - `.default-preloader` class

```
.default-preloader {
  -fx-background-color: yellow;
  -fx-text-fill: red;
  -fx-preloader-graphic: url("http://host.domain/duke.gif");
  -fx-preloader-text: "Loading, loading, LOADING!";
 }
```

Initialization Customization

- Using Ant task

```
<fx:deploy ...>
  <fx:application ...>
  <fx:param name="javafx.default.preloader.stylesheet"
      value=".default-preloader { -fx-background-color:
 yellow; }"/>
  </fx:application>
</fx:deploy>
```

Preloader code

```
public class FirstPreloader extends Preloader {
  ProgressBar bar;
  Stage stage;
  private Scene createPreloaderScene() {
    bar = new ProgressBar();
    BorderPane p = new BorderPane();
    p.setCenter(bar);
    return new Scene(p, 300, 150);
  }
  public void start(Stage stage) throws Exception {
    this.stage = stage;
    stage.setScene(createPreloaderScene()); stage.show();
  }
  public void handleProgressNotification(ProgressNotification pn) {
    bar.setProgress(pn.getProgress());
  }
  public void handleStateChangeNotification(StateChangeNotification evt) {
    if (evt.getType() == StateChangeNotification.Type.BEFORE_START) {
      stage.hide();
    }
  }
}
```

Deployment tools
Packaging

Packaging

- Packaging ensures:
  - An easy-to-run instance of the program
    - Launching from the command line
    - Double-clickable jar
    - Desktop shortcut
    - Web page
  - All needed information/classes is available: required JRE and JavaFX Runtime, dependencies; autodownload/update
  - Visual feedback while loading

Packaging Tools

- Ant tasks
- NetBeans IDE
  - Most typical use cases, may be customized (`-post-jar` target)
- Command-line utility
  - Convenience tool

Packaging Steps

- CSS->BSS (optional)
  - Improves application runtime responsiveness
- Create jar file
  - Includes code and resources, including launcher for runtime detection, preloader, …
- Sign the jar file (optional)
  - Just when higher privileges needed
- Generate additional files for web deployment
  - Web page with javascript, JNLP file

Packaging Steps

	Ant task	CL tool	NetBeans
CSS->BSS	`<fx:csstobin>`	`javafxpackager -createbss`	"Binary Encode JavaFX CSS Files" checkbox
Create a jar	`<fx:jar>`	`javafxpackager -createjar`	Default build action, configurable
Sign a jar	`<fx:signjar>`	`javafxpackager -signjar`	"Request unrestricted access" checkbox, attach a certificate
Generate files for web deployment	`<fx:deploy>`	`javafxpackager -deploy`	Default build action, configurable

CSS Conversion

- Optional, improves performance, especially for large CSS files

- Loading a BSS file:

```
scene.getStylesheets().add(this.getClass()
                .getResource("mystyles.bss").toExternalForm());
```

CSS Conversion

- Ant task

```
<fx:csstobin outdir="build/classes">
     <fileset dir="build/classes" includes="**/*.css"/>
</fx:csstobin>
```

- Command line

```
javafxpackager -createbss -srcdir src\css -srcfiles stylesheet.css -
  outdir styles -outfile stylesheet
```

Create jar File

- The jar contains
  - Platform requirements (Java, JavaFX)
  - Java VM arguments
  - Application details (name of the main class, preloader class)
  - Application resources details
  - Class files and resources themselves
  - List of auxiliary jar files needed by the application

Create jar File

- Ant task

```
<fx:jar destfile="dist/application.jar">
  <fx:application name="Sample JavaFX application"
      mainClass="test.MyApplication"/> <!-- Details about application -->
  <fx:resources> <!-- Define what auxilary resources are needed -->
    <fx:fileset dir="dist" includes="lib/*.jar"/>
  </fx:resources>
  <fileset dir="build/classes"/>
  <manifest> <!-- Customize jar manifest (optional) -->
    <attribute name="Implementation-Vendor" value="Samples Team"/>
    <attribute name="Implementation-Version" value="1.0"/>
  </manifest>
</fx:jar>
```

Create jar File

- Command line

```
javafxpackager -createjar -appclass package.class -srcdir classes -outdir
  out -outfile outjar -v
```

Sign the jar File

- Optional, make sure it is really needed since it affects performance, pops out dialog boxes, etc.
- Similar to regular Java signing, with an option to sign the jar file as one object (saves about 10 percent of the jar size)
- See Java tutorial for signing details (keystores, signing keys)

Sign the jar File

- Ant task

```
<fx:signjar destdir="dist" keyStore="sampleKeystore.jks" storePass="****"
      alias="javafx" keyPass="****">
   <fileset dir='dist/*.jar'/>
</fx:signjar>
```

- Command line

```
javafxpackager -signjar --outdir dist -keyStore sampleKeystore.jks
      -storePass **** -alias javafx -keypass **** -srcdir dist
```

Generate Files for Web Deployment

- Deployment descriptor
- Basic or custom (to ease integration testing) html page
- Define details about the application, its parameters, preloader, stage size, platform requirements, needed permissions

Generate Files for Web Deployment

- Ant task

```
<fx:deploy width="600" height="400" outdir="web-dist" outfile="App">
   <fx:info title="Sample application"/>
   <fx:application name="SampleApp" mainClass="testapp.MainApp"
          preloaderClass="testpreloader.Preloader">
      <fx:param name="testVariable" value="10"/>
   </fx:application>
   <fx:resources>
      <fx:fileset requiredFor="preloader" dir="dist">
         <include name="preloader.jar"/>
      </fx:fileset>
      <fx:fileset dir="dist">
         <include name="helloworld.jar"/>
      </fx:fileset>
   </fx:resources>
</fx:deploy>
```

Generate Files for Web Deployment

- Command line

```
javafxpackager -deploy -outdir dist -outfile Sample -width 800 -
  height 600 -srcdir dist -srcfiles
  AnimatedCircles.jar;lib/somelibrary.jar;preloader.jar -
  appclass animatedcircles.AnimatedCircles -name "Animated
  Circles" -title "Animated Circles Demonstration" -vendor
  Oracle -embedjnlp -v
```

Web Page Templates

- Real web page testing

```
<html>
    <head>
        <title>Host page for JavaFX Application</title>
        #DT.SCRIPT.CODE#
        #DT.EMBED.CODE.ONLOAD#
    </head>
    <body>
        <h1>Embed JavaFX application into existing page</h1>
        <!-- application will be inserted here -->
        <div id="ZZZ"></div>
    </body>
</html>
```

Performance Tips

- Background (lazy) update check – default behavior
- Embedded JNLP in HTML (less connections)
- Embedded certificate into JNLP (no waiting for the security dialog)
- Sign jar as one binary object

JavaFX and JavaScript

- Calling JavaFX methods

```
var fxapp = document.getElementById("myApp")
var r = fxapp.doSomething()
```

- Calling JS functions, fields

```
public void resizeMyself(int w, int h) {
   JSObject jsWin = getHostServices().getWebContext();
   if (jsWin != null) {
      jsWin.eval("var m = document.getElementById('myApp');" +
      "m.width=" + w + "; m.height=" + h + ";");
   } else {
      // running as non embedded => use Stage's setWidth()/setHeight()
   }
}
```

Native Self-contained Bundles

- Includes all application resources
  - Including JRE runtime(!)
- JDK with bundled JavaFX SDK needed (i. e. > JDK7u6)
- Requested by parameters to `<fx:deploy>`, `javafxpackager` tool
- EXE, MSI, DMG, RPM, DEB
- Operating system requirements for system-dependent packages

Native Self-contained Bundles

Pros and Cons

- Pros
  - Resemble native applications
  - No JVM compatibility issues
  - No java runtime has to be installed beforehand
- Cons
  - Need to download, install and run instead of just run from webpage
  - Larger download size
  - Package per target platform
  - No autoupdate as known from JNLP mechanism

Native Self-contained Bundles

Examples

- Ant task

```
<fx:deploy width="${javafx.run.width}" height="${javafx.run.height}"
   nativeBundles="all" outdir="${basedir}/${dist.dir}"
  outfile="${application.title}">
```

  - Possible values are `all, deb, dmg, exe, image, msi, none, rpm`

- Command line tool

```
javafxpackager -deploy -native -outdir packages -outfile BrickBreaker
   -srcdir dist -srcfiles BrickBreaker.jar -appclass brickbreaker.Main
   -name "BrickBreaker" -title "BrickBreaker demo"
```

- Possible more precise specification after `-native`: image, installer, msi, exe, dmg, etc.

Bundle options

	Installation location	License support	Prerequisities
EXE	**Per user:** %LOCALAPPDATA% System: %ProgramFiles%	Yes (optional)	Windows Inno Setup 5 or later[1]
MSI	Per user: %LOCALAPPDATA% **System:** %ProgramFiles%	N/A	Windows WiX 3.0 or later[2]
DMG	Per user: home dir **System:** /Applications	Yes (optional)	Mac OS X
RPM	Per user: N/A System: /opt	N/A	Linux RPMBuild
DEB	Per user: N/A System: /opt	N/A	Linux Debian packaging tools

1 http://www.jrsoftware.org/isinfo.php
2 http://wixtoolset.org/

The Simple Way

- Packager's makeAll command
- Presumes sources in `src`, destination is `dist`

```
javafxpackager -makeall -appclass colorfulcircles.ColorfulCircles -name
  "Colorful Circles" -width 800 -height 600
```

- Compiles, packages and deploys
- Searches for Java and JavaFX
- NetBeans IDE

Documenting JavaFX

Documenting JavaFX Properties

- JavaFX SDK contains Javadoc doclet with additional JavaFX-related feature
- It copies documentation from property definition to its getters and setters, adds links.
- Doclet available in JavaFX SDK in `lib/javafx-doclet.jar`
- Will be available as standard JDK doclet in JDK8 – see the latest development builds
- To enable JavaFX features, pass `-javafx` command line argument (set `javafx.javadoc` environment variable for older releases)

JavaFX Properties

```
private DoubleProperty rate;
/**
  * Setter for the rate property.
  * Property description: Defines the direction/speed at which the {@code Timeline} is
  * expected to be played.
  * @defaultValue 1.0
  * @see #rateProperty
  */
 public final void setRate(double value) {}

/**
  * Almost(!) Copied documentation
  */
 public final double getRate() {}

/**
  * Almost(!) Copied documentation
  */
 public final DoubleProperty rateProperty() {}
```

JavaFX Properties

```
/**
 * Defines the direction/speed at which the {@code Timeline} is
 * expected to be played.
 * @defaultValue 1.0
 */
private DoubleProperty rate;

public final void setRate(double value) {}

public final double getRate() {}

public final DoubleProperty rateProperty() {}
```

Using the Doclet

- Command line

```
javadoc -docletpath <path-to-the sdk>/lib/javafx-doclet.jar -doclet
  com.javafx.lib.doclets.formats.html.HtmlDoclet -sourcepath src -d docs
  -J-Djavafx.javadoc=true packageName
```

- Ant task

```
<javadoc  destdir="docs" sourcepath="src" packagenames="packageName"
  access="private" additionalparam="-J-Djavafx.javadoc=true">
      <doclet name="com.javafx.lib.doclets.formats.html.HtmlDoclet"
  path="<path the the sdk>/lib/javafx-doclet.jar">
      </doclet>
</javadoc>
```

What You Have Learned

- JavaFX
- Execution modes
- Deployment tools
- Effective Documenting

References

- JavaFX
  - http://www.oracle.com/technetwork/java/javafx/overview/index.html
- Deploying JavaFX Applications
  - http://docs.oracle.com/javafx/2/deployment/jfxpub-deployment.htm
- Doclet guide
  - http://docs.oracle.com/javafx/2/doclet/jfxpub-doclet.htm

Q & A