CONTENTS

»"Hello World" JavaFX Application
»JavaFX Properties
»Binding API
»UI Components
»Shapes
»Controls, and more...

JavaFX 8

By Hendrik Ebbers & Michael Heinrichs

GENERAL

JavaFX is the new UI toolkit for Java-based client applications running on desktop, embedded, and mobile devices. It is part of the JDK 8 and is provided as a pure Java API. Among others, the following features are supported:

- Accelerated 2D and 3D graphics
- UI controls, layouts, and charts
- Audio and video support
- Effects and animations
- HTML5 support
- Bindings, CSS, FXML, and more...

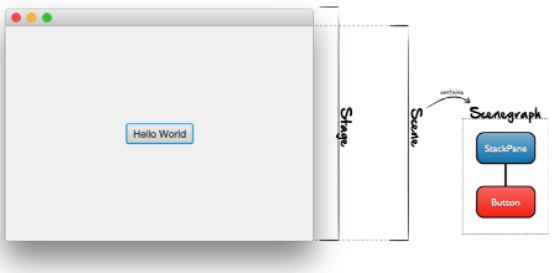
To provide maximum performance, JavaFX uses different native rendering engines depending on the platform it is running on. On Windows for example, Direct3D is used, while on most other systems it uses OpenGL.

"HELLO WORLD" JAVAFX APPLICATION

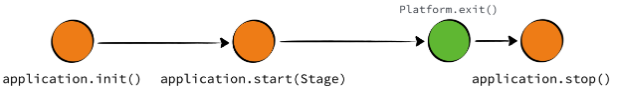
This "Hello World" example will show a window with a button. Clicking the button will print "Hello World" to the console.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button button = new Button("Hello World");
        button.setOnAction (e -> System.out.
println("Hello World"));
        StackPane myPane = new StackPane();
        myPane.getChildren().add(button);

        Scene myScene = new Scene(myPane);

        primaryStage.setScene(myScene);
        primaryStage.setWidth(400);
        primaryStage.setHeight(300);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The following diagram shows the structure of the program. The main window corresponds to a Stage, which contains a Scene, which contains a scene graph. The scene graph is a tree of UI nodes.



A JavaFX application runs in the default JavaFX lifecycle that is defined by the Application class. The `init()` method is called before the JavaFX application thread is created, therefore no UI-specific operations are allowed.



Each JavaFX application needs at least one stage with an internal scene graph. At runtime, one can add new windows by creating new stage instances.

JAVAFX PROPERTIES

JavaFX properties are based on regular JavaBeans properties. The JavaFX runtime provides default implementations for all property types, which can be used in one's own class. The following code example shows the definition of the "size" property.

```
private final IntegerProperty size =
            new SimpleIntegerProperty(this, "size", 42);

public int getSize() {
    return size.get();
} // continued ->
```

```
public void setSize(int newValue) {
    size.set(newValue);
}
public IntegerProperty sizeProperty() {
    return size;
}
```

BINDING API

JavaFX offers the possibility to create bindings between properties. Bindings synchronize the values of two properties automatically, avoiding the error-prone boilerplate code that is otherwise needed.

Properties can be bound unidirectionally or bidirectionally, implemented by the methods `bind()` and `bindBidirectional()` respectively. If a property A is unidirectionally bound to a property B, property A will always have the same value as B. Property A becomes read-only in this case. If the properties A and B are bound bidirectionally, changes are propagated in both directions.

In additon to binding two properties directly, it is also possible to bind a property against an expression. If one of the operands of the expression changes, the expression is re-evaluated automatically and the result is assigned to the property.

HIGH-LEVEL BINDING API

There are two approaches to define complex bindings. The high-level API allows bindings to be defined for many typical use cases in an easy way.

OPERATIONS	EXAMPLES	TYPE

Arithmetic Operations	`num1.add(num2)`
	`Bindings.divide(num1, num2)`
	`num1.negate()`
		Number

Boolean Operations	`Bindings.or(bool1, bool2)` `bool1.not()`	Boolean
Comparisons	`obj1.isEqualTo(obj2)` `Bindings.notEqual(obj1, obj2)`	All
	`num1.greaterThan(num2)` `num1.lessThanOrEqualTo(num2)`	Number, String
	`Bindings.equalIgnoreCase(s1, s2)` `s1.isNotEqualToIgnoreCase(s2)`	String
Conversions	`obj.asString()`	All
	`num.asString(format)` `Bindings.format(format, val...)`	Number, Object
Null Check	`obj.isNull()` `Bindings.isNotNull(obj)`	Object, String
String Operations	`Bindings.concat(s1, s2)`	String
Min / Max	`Bindings.min(num1, num2)` `Bindings.max(num1, num2)`	Number
Collections	`Bindings.valueAt(list, index)` `Bindings.size(collection)`	Collection
Select Binding	`Bindings.select(root, properties...)`	
Ternary Expression	`Bindings.when(cond).then(val1).otherwise(val2)`	

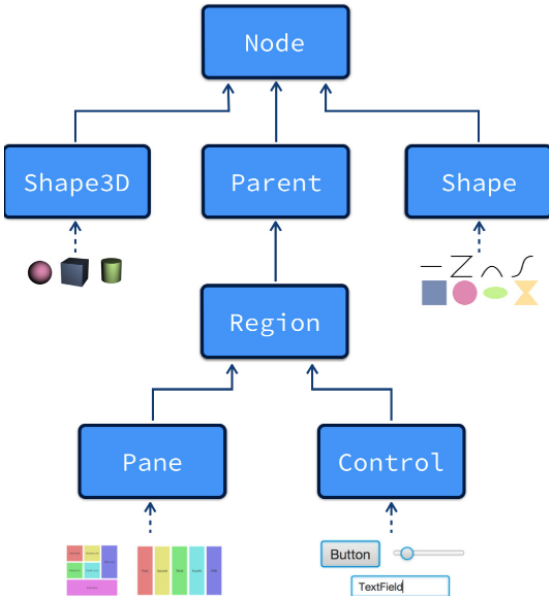
LOW-LEVEL BINDING API

With the low-level API, one can define bindings for arbitrary expressions. The following code sample shows how a binding can be defined that calculates the length of a vector (x, y) where x and y are two DoubleProperties.

```
length = Bindings.createBinding(
    () -> Math.sqrt(x.get() * x.get() + y.get() * y.get()),
    x, y
);
```

UI COMPONENTS

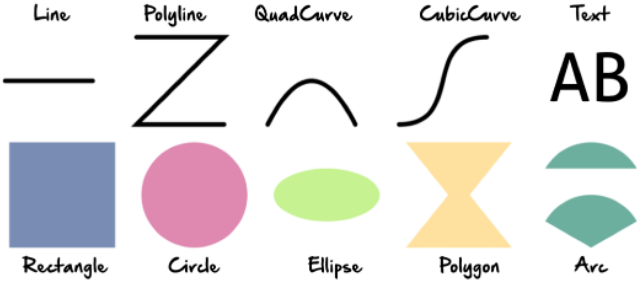
All UI components in JavaFX extend the base class `Node`. There are different types of nodes, as can be seen in the following class hierarchy:



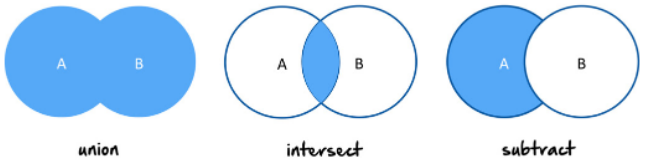
All geometric shapes, and text (which is just a very complex shape), extend either the class `Shape` or `Shape3D`. These are leaf nodes in the scene graph. The possibility to contain other nodes as children is defined in the abstract class `Parent`. UI controls and layout panes extend this class. Besides the properties that define the specific behavior of a node, all of them provide support for event handling and CSS styling.

SHAPES

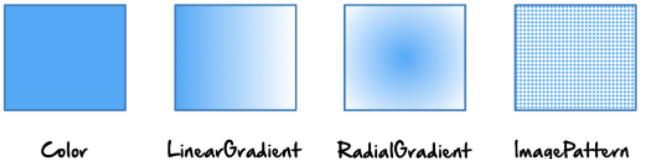
Shapes define the most basic nodes that can be shown in a JavaFX scene graph. The `Shape` class is the superclass of all geometric primitives and defines these basic features:



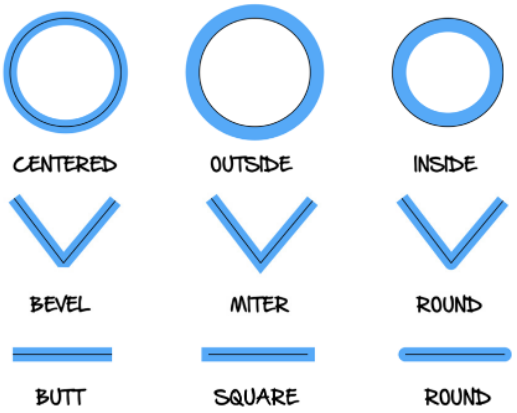
In addition, this class provides the boolean operations `union`, `intersect`, and `subtract` to create new shapes.



Beyond just setting the stroke and the fill of a shape to a color, you can use the `Paint` class's four implementations:

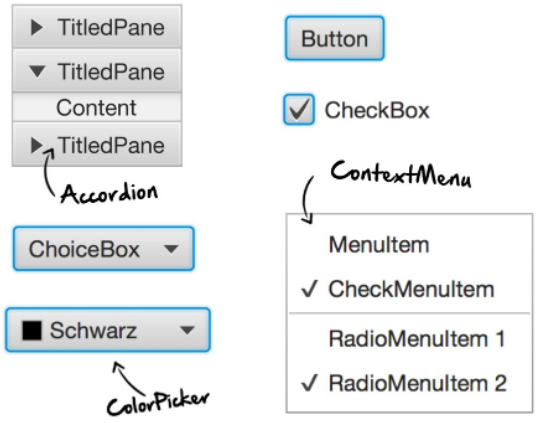


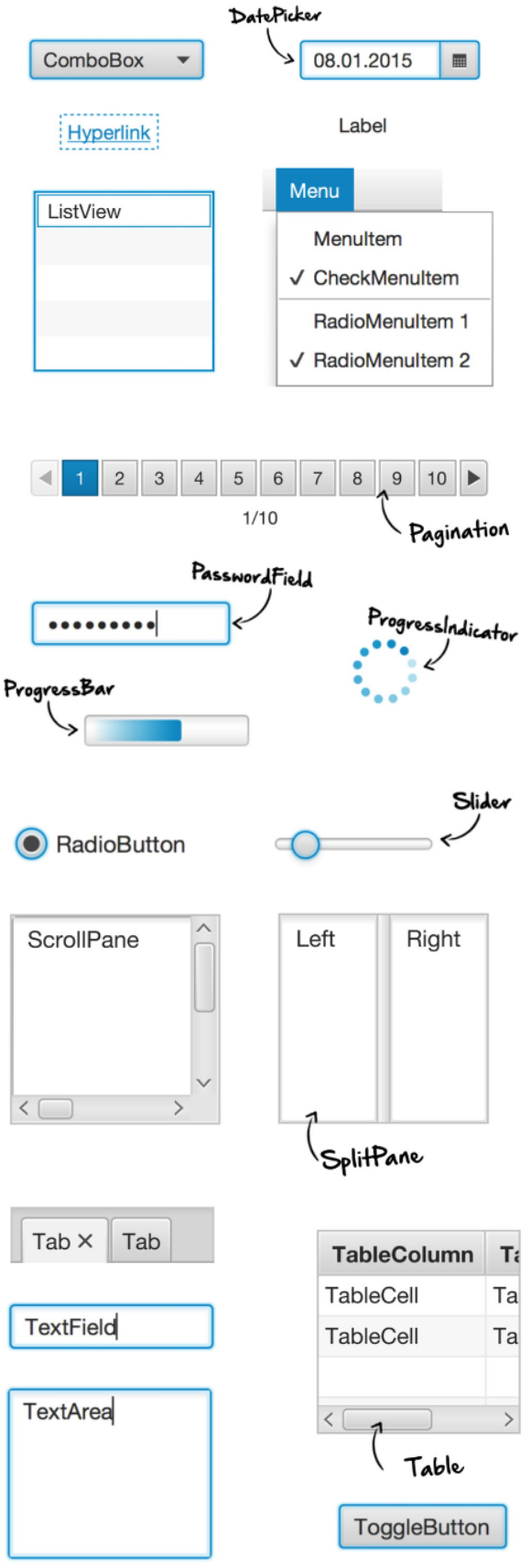
The stroke can further be configured by modifying the properties `strokeType`, `strokeLineJoin`, and `strokeLineCap`. It is also possbile to define a dashing pattern.

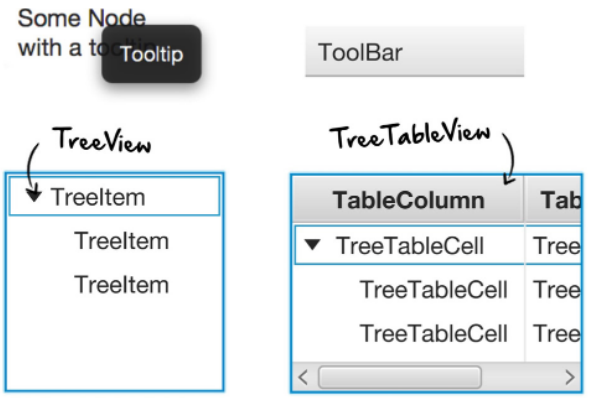


CONTROLS

Controls are the JavaFX UI nodes that a developer will use most of the time. All controls — like buttons, textfields, or tables — extend the `Control` class. By default, JavaFX controls will be rendered in a consistent and system-independent theme.







The speciLc style of a control is deLned with CSS. The default CSS theme in JavaFX, which is used to style all controls, is called "Modena." Internally, all controls are made up of primitive UI nodes like shapes and panes. In other words, each JavaFX control is vector-based and can be scaled without losing any sharpness or looking pixelated.

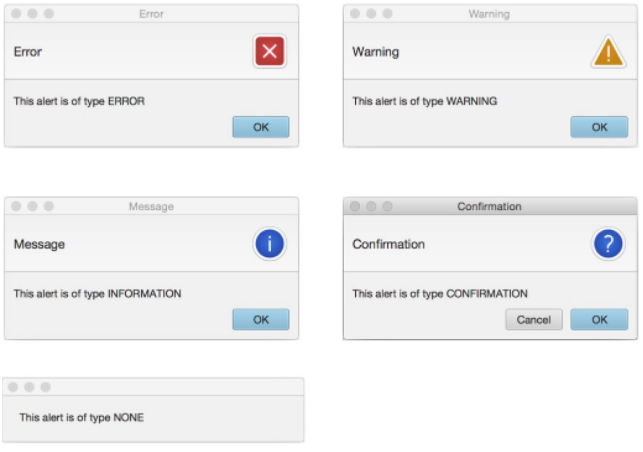
A tooltip can be speciLed for each control. A tooltip can contain any UI node, so you can place a pane with any content in a tooltip and create an awesome visual help dialog. For example, it is possible to add a `MediaView` to a tooltip and play a video instead of only showing text.

SIZE OF A CONTROL

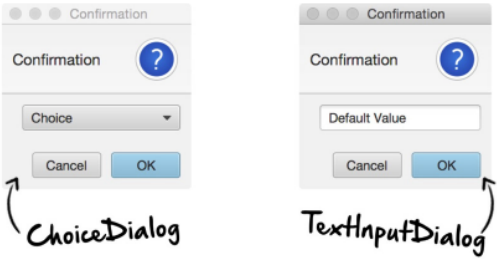
The position and size of a control is deLned by its parent node, which is typically a layout pane. A developer can inLuence size by setting the minimum, maximum, and preferred size of a control. By default, these values are calculated by the control. For example, the preferred width of a button depends on the text the button contains. JavaFX calculates a width and height that ensures the complete text Lts into the button.

DIALOGS AND ALERTS

JavaFX provides several alert types.

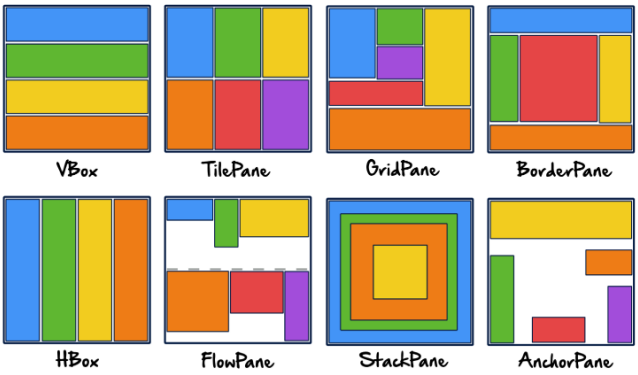


In addition to standard notiLcation alerts, JavaFX also contains special dialogs that allow you to request a value.



LAYOUT PANES

JavaFX provides several layouts out of the box, which can be seen in the following diagram:



A layout is a parent node in the scene graph that modiLes the position of the child and — if the child is resizable — also the size. For each layout — and for resizable components in general — three sizes can be speciLed: the preferred, the minimum, and the maximum size. Each layout algorithm tries to optimize the size of the child nodes toward the preferred size and will shrink or expand the child nodes according to the available space (adhering to the minimum and maximum bounds). Each pane stores its children in an `ObservableList`. In several pane types (e.g. `HBox`, `FlowPane`, and `StackPane`) the index in this list deLnes the position of the child in the pane. The basic `Node` class provides the methods `toFront()` and `toBack()` to put a child to the Lrst or last position.

ADDITIONAL COMPONENTS

In addition to shapes, layout panes, and controls, JavaFX provides several UI components for speciLc needs.

IMAGEVIEW

Two classes are required to show an image. The `Image` class encapsulates the raw data of an image and its properties. The `ImageView` class is a scene graph node, and it is responsible for showing the image on screen. This split is needed because it allows JavaFX to show an image several times on screen, while the data is kept in memory only once.

```
Image image = new Image("path/to/image");
ImageView imageView = new ImageView(image);
myPane.getChildren().add(imageView);
```

MEDIAVIEW

The `MediaView` class can be used to show a video on screen or play an audio Lle. JavaFX supports MP3, AIFF, WAV, and MPEG-4 as

audio codecs and FLV (Flash Video) or MPEG-4 (H.264/AVC) as video codecs. Two classes are required to play an audio stream: similar to the `Image` class, the `Media` class encapsulates the raw data, while `MediaPlayer` provides functionality to control the playback and contains other useful media information. To play a video, a third class is needed: `MediaView`. It is a regular scene graph node, which means one can even apply effects and animate videos.

```
Media media = new Media("path/to/media");
MediaPlayer player = new MediaPlayer(media);
MediaView mediaView = new MediaView(player);

player.setVolume(0.5);
player.play();
```

CHARTS

JavaFX contains a full-fledged Charts API that allows it to define and visualize charts. The following figure shows the type of charts that are available in the standard JDK:

CANVAS

In some situations, it is more efficient to have full control over rendering and draw to the screen directly. JavaFX provides the `Canvas` class for these scenarios. It provides a `GraphicsContext` with several methods to draw geometric figures or images directly. The `Canvas` component is comparable to HTML Canvas or the Java2D `Graphics2D` functionality.

WEBVIEW

The JavaFX `WebView` class can be used to embed any web content in your application. `WebView` uses WebKit internally to render web content and provide interaction with that content. Even rich HTML5 applications can be wrapped in a JavaFX window or pane. `WebView` provides a `WebEngine` object that allows the developer to directly interact with the HTML content and, for example, inject JavaScript or manipulate the DOM.

```
WebView webView = new WebView();
WebEngine engine  = webView.getEngine();

//Load a web page
engine.load("http://www.guigarage.org");

//Add the web view to the JavaFX view
myPane.getChildren().add(webView);

// Inject JavaScript
engine.executeScript("history.back()");
```

CSS

JavaFX provides styling by CSS. CSS support is based on the W3C CSS version 2.1, but there are some minor differences that can be found in the JavaFX CSS documentation (http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html).

In JavaFX, a stylesheet can be applied to the scene graph or to a specific node. A stylesheet that is applied to the scene graph will affect all nodes in the scene graph. If the stylesheet is applied to a node, it will affect this node and all its children (recursively). Several stylesheets can be applied to the scene graph or a node. The following code snippet shows how you can set a stylesheet for a scene graph or a node:

```
// load the stylesheet
String style = getClass().getResource("style.css").
toExternalForm();

// apply stylesheet to the scene graph
myScene.getStylesheets().addAll(style);

// apply stylesheet to a node
parentPanel.getStylesheets().addAll(style);
```

JavaFX also supports inline stylesheets for nodes. Here you can define the CSS rules directly in your Java code as a string, which can be helpful for debuging and testing. The `Node` class provides a method that can be used to set an inline style for a `Node` instance:

```
button.setStyle("-fx-background-color: green;");
```

A CSS rule is applied to a node if its selector matches. In the selector, one can use a combination of ID, element classes, style classes, and pseudo classes. A JavaFX node can have exactly one ID:

```
mySaveButton.setId("my-save-button");
```

Here is an example of a CSS rule that styles exactly this button:

```
/* The # sign in the selector defines an id */
#my-save-button {
    -fx-background-color: blue;
}
```

In addition to the ID, a node can have several style classes:

```
button.getStyleClass().add("toolbar-button");
```

All nodes that have the "toolbar-button" style class can be styled with a single rule in CSS:

```
/* The . sign in the selector defines a style class */
.toolbar-button {
    -fx-background-color: blue;
}
```

Pseudo classes can be defined for any node, too. Pseudo classes are activated and deactivated by using the Java API:

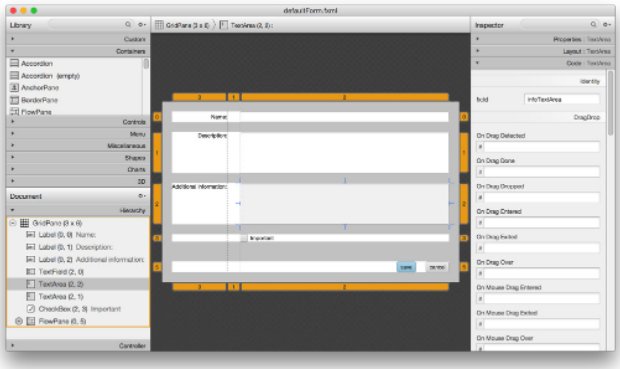
```
// Define the pseudo class
PseudoClass myPseudoClass = PseudoClass.
getPseudoClass("active");

//activate the pseudo class
myNode.pseudoClassStateChanged(myPseudoClass, true);

//deactivate the pseudo class
myNode.pseudoClassStateChanged(myPseudoClass, false);
```

In CSS you can use the pseudo class in a selector to deLne a speciLc style:

```
/* A colon signals a pseudo class */
.control:active {
    -fx-background-color: blue;
}
```

FXML

FXML is an XML-based language that deLnes the structure and layout of JavaFX UIs. It is tool-agnostic and can be edited with any editor, but it's especially helpful to use SceneBuilder, a WYSIWYG editor for FXML.



FXML allows you to create a clear separation between the view of an app and the logic. JavaFX provides an API to deLne MVC packages that are based on FXML and a Java controller. Here is a small FXML deLnition for a view that contains only one button in a StackPane:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>

<StackPane maxHeight="-Infinity" maxWidth="-
Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.
com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Button mnemonicParsing="false" text="ButtonTitle"
/>
   </children>
</StackPane>
```

Once the FXML view is created, it can be loaded with the `FXMLLoader` class:

```
FXMLLoader loader = new FXMLLoader(getClass().
getResource("demo.fxml"));
StackPane view = loader.load();
```

A Java-based controller can be bound to the FXML LLe. The `@FXML` annotation can be used to inject view nodes directly into the controller class. An injectable node must be marked with a unique fx:id in FXML:

```
<Button fx:id="myButton" mnemonicParsing="false"
text="ButtonTitle"/>
```

Once this is done, the `Button` instance can be injected in the controller by using the fx:id as the LLeld name:

```
public class ViewController {

    @FXML
    private Button myButton;
}
```

When loading the FXML stream, the controller can be passed to the loader. In this case, the `FXMLLoader` will automatically inject all LLelds that are annotated with `@FXML`:

```
ViewController controller = new ViewController();
FXMLLoader loader = new FXMLLoader(getClass().
getResource("demo.fxml"));
loader.setController(controller);
StackPane view = loader.load();
```

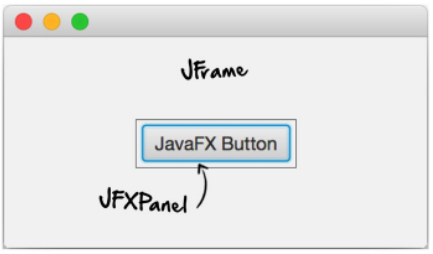
In addition, the FXML speciLcation supports several other features, like resource bundles, event handler linking, or nesting FXML LLes.

JAVAFX AND SWING

It is possible to use JavaFX components within a Swing application — and to use Swing components in a JavaFX application.

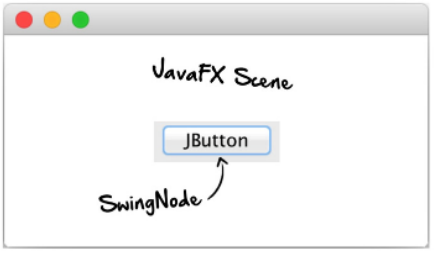
JAVAFX IN A SWING APPLICATION

`JFXPanel` extends Swing's `JComponent` and can therefore be integrated into any Swing application. The scene property of `JFXPanel` allows you to set a JavaFX Scene, which will be visualized within the JFXPanel.



SWING IN A JAVAFX APPLICATION

A SwingNode can be used to add Swing components to a JavaFX application. `SwingNode` is a regular node and can be added anywhere in the scene graph. It has a content property that can take any JComponent.



THREADING

Unfortunately, both Swing and JavaFX require their own main application thread, and changes have to be made on the right thread. This can be achieved with the helper methods `Platform.runLater()` and `SwingUtilities.invokeLater()`, respectively.

PACKAGING AND DEPLOYMENT

Java 8 offers support to deploy your JavaFX application as a native app. In this case, the JRE will be bundled with your application. A native executable will be created (e.g., an EXE on Windows or a DMG on Mac), and a user can execute it without the need for Java on the client system. Additional information, like metadata or application icons, can be defined for the native app, too. The bin folder of the JDK contains the `javafxpackager` executable that must be used to create such a bundled application. There are plugins for Ant, Maven, and Gradle available to support this feature automatically in your build.

TOOLS

Since JavaFX is completely Java based, you can use any Java IDE—or just a text editor and the JDK—to create JavaFX applications. Beyond that, there are also some nice tools, which are part of the JavaFX ecosystem, that can help you create JavaFX applications. The following table gives a short overview of some of these tools:

TOOL	DESCRIPTION
Scene Builder	Scene Builder is an open-source WYSIWYG editor for JavaFX that creates FXML-based views. It also has very good CSS support and contains a CSS analyzer.
Scenic View	Scenic View is a tool that helps you debug your application. The tool finds running JavaFX applications and allows you to navigate through scene graphs and inspect the properties of specific nodes.
JavaFX Ensemble	This application provides demos and examples for most JavaFX features, including their sources. Most of the demos are interactive and let you directly test how a feature behaves at runtime.
e(fx)clipse	A plugin for Eclipse that adds a lot of useful JavaFX support to the IDE.

ABOUT THE AUTHORS

Hendrik Ebbers is Software Engineer at Canoo Engineering AG and lives in Dortmund, Germany. His main focus besides research and development is primarily in the areas of UI technologies, Middleware and DevOps. Additionally, Hendrik Ebbers is founder and leader of the Java User Group Dortmund and gives talks and presentations in User Groups and Conferences. He's blogging about UI related topics at www.guigarage.com (or on Twitter @hendrikEbbers) and contributes to some open-source Projects: DataFX, AquaFX and Dolphin Platform. Hendrik's JavaFX book "Mastering JavaFX 8 Controls" was released 2014 by Oracle press. Hendrik is a JavaOne Rockstar and JSR expert group member.

Michael Heinrichs is a user interface creator by passion. He is convinced: no matter which technology and which device, if it has a screen, one can build a truly amazing experience. And pure magic.

Michael works at the Canoo Engineering AG as a software engineer on-next generation user interfaces. Before that, he was responsible for performance optimizations in JavaFX Mobile at Sun Microsystems and later became the technical lead of the JavaFX core components at Oracle.

Michael loves to spend time with his family and cooking. You can find him on Twitter @net0pyr and occasionally he blogs at http://blog.netopyr.com.

CREDITS:
Editor: G. Ryan Spain | Designer: Yassee Mohebbi | Production: Chris Smith | Sponsor Relations: Chris Brumfield | Marketing: Chelsea Bosworth

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

VERSION 1.0 $7.95