DZone Refcardz

brought to you by...
AnswerHub
Connecting People With Knowledge

CONTENTS INCLUDE:

JavaFX 2.0

By Stephen Chin

WHAT IS JAVAFX?

JavaFX is an exciting new platform for building Rich Internet Applications with graphics, animation, and media. Starting with JavaFX 2.0, all the APIs are written in pure Java, which lets you write JavaFX applications directly in a language you are already familiar with. This Refcard will help you get started writing JavaFX applications and also serve as a convenient reference to some of the more advanced APIs.

To get started, download the latest JavaFX SDK from the JavaFX website at http://javafx.com/

If you are new to JavaFX, the easiest way to get started is to download the NetBeans bundle. This gives you full IDE support for writing JavaFX applications out of the box, including library setup, deployment options, and an integrated UI debugger. However, it is possible to do everything from your favorite IDE, such as Eclipse or IntelliJ, or the command line directly.

JFXPOETRY, A SIMPLE EXAMPLE

To illustrate how easy it is to build an application that melds graphics, text, animation, and media, we will start with a simple tutorial. The goal will be to write an application that:

- Loads and displays an image from the internet
- Displays and animates a verse of poetry
- Makes use of graphic effects
- Plays media asynchronously

For the JFXPoetry theme, we will use "Pippa's Song", a well known excerpt from Robert Browning's Pippa Passes.

Beginning with a Stage

Stage and Scene are the building blocks of almost every JavaFX program. A Stage can either be represented as a Frame for desktop applications or a rectangular region for applications embedded in a browser. The visual content of a Stage is called a Scene, which contains a parent node that is the root of a graph or tree of the rest of the elements called the scene graph. The following program creates a basic Stage and Scene with a StackPane layout as the root element:

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

    @Override
    public void start(Stage stage) {
        stage.setTitle("Pippa's Song by Robert Browning");
        stage.setResizable(false);

        StackPane root = new StackPane();
        stage.setScene(new Scene(root, 500, 375));
        // add additional code here
        stage.show();
    }
}
```

This code illustrates the basic template for creating any JavaFX application. The necessary elements are:

- A class that extends javafx.application.Application
- A Java main method that calls the launch helper method
- An implementation of the start method that takes the primary stage

The start method is guaranteed to be called on the JavaFX UI thread, so you can take care of scene graph manipulation and finally show the Stage when you are ready. Compiling and running this application will create a window with the title "Pippa's Song by Robert Browning" as shown in Figure 1.

Hot Tip The JavaFX scene graph is an extremely powerful concept, which can be used to apply transformations and effects that change the rendering of all the child nodes.

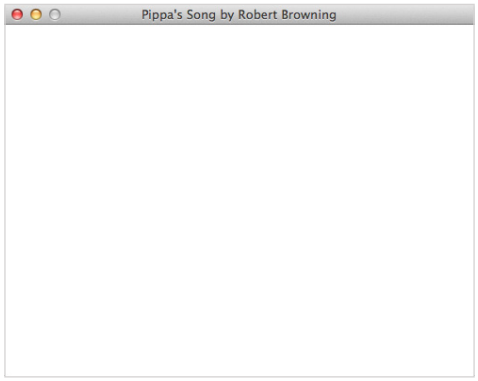


Figure 1. A simple Stage with a descriptive title

LOADING AN IMAGE

Loading and displaying images in JavaFX is extremely easy to do. The Image class allows you to load an image from a local file or a remote server simply by specifying the URL. This can then be set on an ImageView to be displayed inside of a Scene. The following code should be added before showing the Stage in replacement of the "add additional code here" comment:

```
Image image = new Image("http://farm1.static.flickr.com/39/121693644_75491b23b0.
jpg");
ImageView imageView = new ImageView(image);
root.getChildren().add(imageView);
```

Notice that to add the Image element to the root StackPane, you must first get the list of children and then add an element. This is a common pattern used throughout the JavaFX APIs that is possible because of Observable Lists where any change to the list contents is automatically updated in the UI.

For the background image, we have chosen a picture of a misty morning in Burns Lake, BC, Canada taken by Duane Conlon. The running example can be seen in Figure 2.

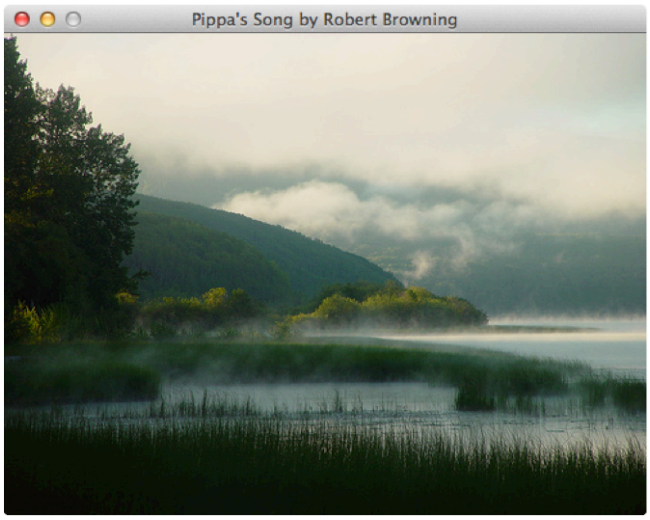


Figure 2. Image loaded from the network

DISPLAYING TEXT WITH EFFECTS

Displaying Text in JavaFX is as simple as constructing a Text Node and passing in a String to the constructor. There are many properties available on Text; however, for this example, we will set the font and fill color and add a Drop Shadow effect to make the text stand out on the background.

```
Text text
 = new Text(
      "The year's at the spring,\n" +
      "And day's at the morn;\n" +
      "Morning's at seven;\n" +
      "The hill-side's dew-pearled;\n" +
      "The lark's on the wing;\n" +
      "The snail's on the thorn;\n" +
      "God's in His heaven--\n" +
      "All's right with the world!");
text.setFont(Font.font("Serif", FontWeight.BOLD, 30));
text.setFill(Color.GOLDENROD);
text.setEffect(DropShadowBuilder.create().radius(3).spread(0.5).build());
text.setCache(true);
root.getChildren().add(text);
```

Hot Tip By using the cache property on Node, you can improve performance and avoid rendering artifacts during animation.

To create the DropShadow in the above code, we chose to use the builder syntax, which is a fluent API for creating JavaFX UI elements. Every JavaFX class has a builder peer that can be used to set properties by chaining method calls and will return an instance of the class when the final build method is invoked. This allows us to change the radius and spread properties of the DropShadow in order to accentuate the letters. Figure 3 shows the updated example with Text overlaid on the Image.

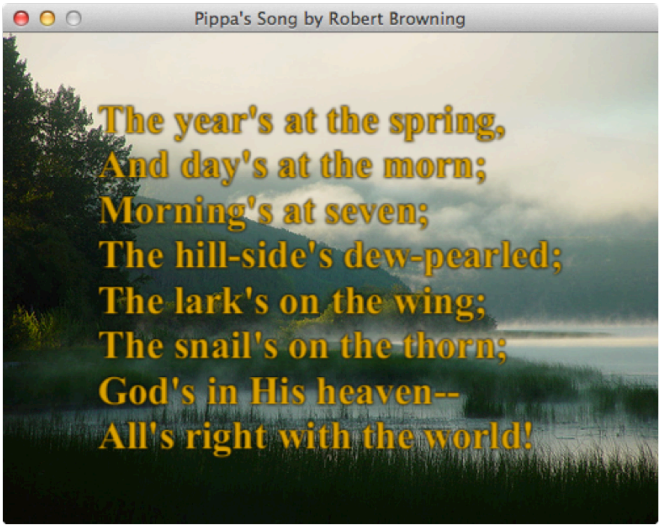


Figure 3. Updated example with a Text overlay

JavaFX offers a large set of graphics effects that you can easily apply to nodes to create rich visual effects. Table 1 lists all the available effects you can choose from.

Table 1. Graphics effects available in JavaFX

Transition	Description
Blend	Blends two inputs together using a pre-defined BlendMode
Bloom	Makes brighter portions of the node appear to glow
BoxBlur	Fast blur with a configurable quality threshold
ColorAdjust	Per-pixel adjustments of hue, saturation, brightness, and contrast
ColorInput	Fills a rectangular region with the given Color
DisplacementMap	Shifts each pixel by the amount specified in a displacement map

DropShadow	Displays an offset shadow underneath the node
GaussianBlur	Blurs the node with a configurable radius
Glow	Makes the node appear to glow with a given intensity level
ImageInput	Passes an image through to a chained effect
InnerShadow	Draws a shadow on the inner edges of the node
Lighting	Simulates a light source to give nodes a 3D effect
MotionBlur	Blurs the image at a given angle to create a motion effect
PerspectiveTransform	Maps a node to an arbitrary quadrilateral for a perspective effect
Reflection	Displays an inverted view of the node to create a reflected effect
SepiaTone	Creates a sepia tone effect to mimic aged photographs
Shadow	Similar to a DropShadow but without the overlaid image

ANIMATED TRANSITIONS

Animations in JavaFX can be accomplished either by setting up a Timeline from scratch or by using one of the prefabricated Transitions. To animate the Text rising onto the screen, we will use a TranslateTransition, which adjusts the position of a node in a straight line for the specified duration:

```
final TranslateTransition translate = TranslateTransitionBuilder.create()
    .duration(Duration.seconds(24))
    .node(text)
    .fromY(image.getHeight())
    .toY(0)
    .interpolator(Interpolator.EASE_OUT)
    .build();
translate.play();
```

Hot Tip Every JavaFX class has a matching builder, which you can use to make your code more readable when setting multiple properties.

By setting an interpolator of EASE_OUT, the text will start at full speed and gradually deaccelerate as it approaches its destination. To run the transition, all you need to do is call the play() function, which will animate the text as shown in Figure 4.

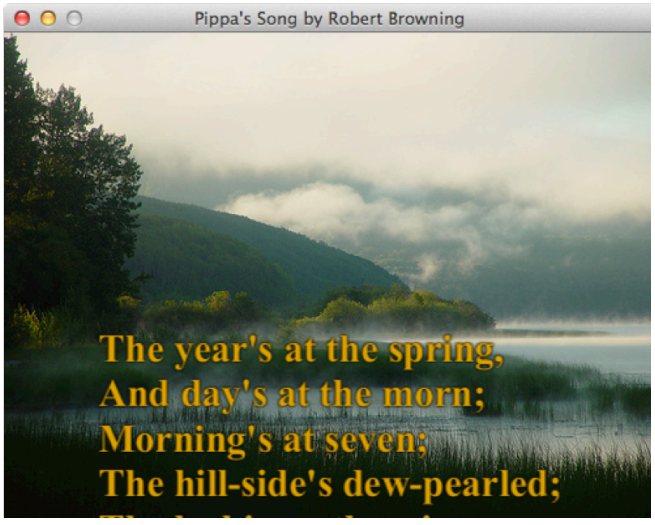


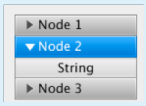
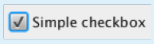
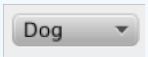
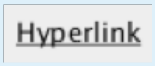
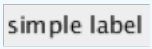
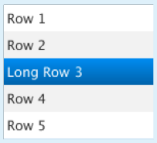
Table 2 lists all of the available transitions that are part of the JavaFX API. To get a feel for how the different transitions work, try adding a FadeTransition that will gradually fade the background in over a five-second duration.

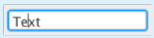
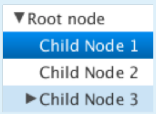
Table 2. Transitions Supported by JavaFX

Transition	Description
FadeTransition	Changes the opacity of a node over time
FillTransition	Animates the fill color of a shape
ParallelTransition	Plays a sequence of transitions in parallel
PathTransition	Animates nodes along a Shape or Path
PauseTransition	Executes an action after the specified delay
RotateTransition	Changes the rotation of a node over time
ScaleTransition	Changes the size of a node over time
SequentialTransition	Plays a sequence of transitions in series
StrokeTransition	Animates the stroke width of a shape
TranslateTransition	Changes the position of a node over time

INTERACTING WITH CONTROLS

The JavaFX 2 release features a complete library of skinnable controls that give you everything you will need for most web and business applications. Table 3 lists some of the controls and what they can be used for.

	Control	Description
	Accordion	The Accordion control lets you display a list of nodes that can be expanded by clicking the title. Since the title and content areas are both JavaFX nodes, you can embed images, controls, and even media in the accordion.
	Button	JavaFX buttons are extremely versatile and have complete control over all aspects of style including color, text, shape, and even images.
	CheckBox	The CheckBox control is a specialized type of button that includes a label text and a check selection area that can be checked, unchecked, or indeterminate, allowing for tri-state behavior.
	ChoiceBox	A ChoiceBox allows selection from a list of pre-defined items using a drop-down menu. This control supports a null (or undefined) state and sets the selected item to something that is not in the list.
	Hyperlink	Hyperlinks are a special type of button that mimic the behavior of a hyperlink in a web browser. This includes rollover animation and a special style for visited links.
	Label	A label is a basic control for displaying read-only text. It supports ellipses for content that is too long to be displayed and keyboard mnemonics for creating accessible UIs.
	ListView	The JavaFX ListView lets you display a vertical or horizontal list of items and is backed by an Observable List that will update automatically update the UI when the contents change. You can display Strings or any other type of object in a List by providing your own cell factory that converts objects into JavaFX nodes for display. Also, the list control supports single selection or multiple selection and lists where you can edit the contents inline.
	ProgressBar	The ProgressBar control gives feedback during long-running operations using a horizontal progress bar. This can either be as a percentage of complete when the duration is known or an indeterminate animation.

	ProgressIndicator	The ProgressIndicator control has the same functionality as the ProgressBar, except it displays the progress either as a small pie graph or an indeterminate spinning animation.
	RadioButton	The JavaFX RadioButton behaves like a two-state checkbox when used alone, but it is more commonly added to a ToggleGroup where only one of a set of radio buttons can be active at a time.
	TableView	The TableView is a very powerful table component for JavaFX that supports a two-dimensional grid of cells with column headers, column resizing, column reordering, multi-column sorting, nested columns, cell spans, editable cells, and many more features.
	TabPane	A TabPane displays a list of Tabs, only one of which can be active and displayed at a given time. The list can be positioned along the top, left, right, or bottom of the content area. If there are more tabs than fit, they will be displayed in a drop-down menu.
	TextField/ TextArea/ PasswordField	JavaFX provides a complete set of editable text area controls, including a basic one-line TextField, a multi-line TextArea, and a specialized PasswordField that masks characters as they are entered.
	ToolBar	The ToolBar lets you create a horizontal or vertical list of nodes styled like a typical application toolbar. You can add any type of node to the toolbar, but it is common to use different types of buttons spaced out with separators.
	TreeView	The TreeView shows a hierarchical view of TreeItems with controls to allow each level to be expanded in place. To create a TreeView with multiple top-level nodes, you can set showRoot to false, which will hide the top-most element of the model. Like the ListView control, it supports custom cell factories to display different content types inline.

In addition to these controls, JavaFX also supports a ScrollPane, Separator, Slider, SplitPane, Tooltip, Menu, HTMLEditor, and WebView. All of the JavaFX controls are fully skinnable, allowing you to customize the look and feel to match the design of your application using CSS or a custom Skin class.

The simplest control to use is a Button, which can easily be scripted to play the animation sequence again from the beginning. Adding a button to the scene graph is as simple as calling the constructor with the button text and adding it to the list of children.

```
Button play = new Button("Play Again");
root.getChildren().add(play);
play.visibleProperty().bind(translate.statusProperty()
  .isEqualTo(Animation.Status.STOPPED));
play.addEventHandler(ActionEvent.ACTION,
                new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        translate.playFromStart();
        // Uncomment once you finish the next section on Media:
        //mediaPlayer.stop();
        //mediaPlayer.play();
    }
});
```

To automatically hide the button while animation is in progress and show it after the animation completes, we make use of binding, a very powerful JavaFX feature that is exposed via a new set of APIs in JavaFX 2. By binding the visible property of the button to the animation status of the translation transition, the button will be shown as soon as the animation finishes.

Hot Tip Binding is a great alternative for event listeners and callbacks, because it allows you to create dynamic content with very little code.

Following this, we make use of an ActionEvent handler to play the animation and the media from the start, which is triggered once the button shown in Figure 5 is clicked.

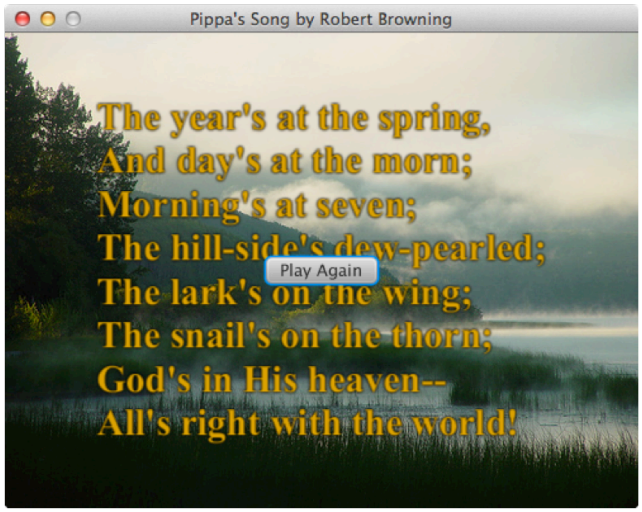


Figure 5: Button Control to Play the Animation Again

FINISHING WITH MEDIA

The finishing touch is to add some audio to the application. JavaFX has built-in media classes that make it very simple to play audio or video either from the local files or streaming off the network. To complete the example, we will add in a public domain clip of Indigo Bunting birds chirping in the background. Adding in the audio is as simple as creating a Media object with the source set to the URL and wrapping that in a MediaPlayer with autoPlay set to true.

```
Media media = new Media("http://video.fws.gov/sounds/35indigobunting.mp3");
final MediaPlayer mediaPlayer = new MediaPlayer(media);
mediaPlayer.play();
```

Hot Tip Besides loading media off a network, you can also access local files or resources in the application jar.

In this example, we are using an mp3 file, which is supported across platforms by JavaFX. Table 3 lists some of the common media formats supported by JavaFX, including all the cross-platform formats.

Table 3. Common Media Formats Supported by JavaFX

Type	Format	File Extension
Audio	MPEG-1, 2, 2.5 Audio Layer 3	mp3
Audio	Waveform Audio Format	wav
Audio	Audio Interchange File Format	aif, aiff
Video	Flash Video: VP6 video with MP3 audio	flv, f4v
Video	FX Media: VP6 video with MP3 audio	fxm

Note: This Refcard covers the latest changes through JavaFX 2.0.3. In addition to what is mentioned here, Oracle has announced support for AAC audio and H.264 video coming in JavaFX 2.1.

To try the completed example, complete with animation and audio, click the following url: http://steveonjava.com/refcard/JFXPoetry.jnlp

FXML AND CSS

In addition to writing applications in pure Java, JavaFX also supports a declarative markup format called FXML. Additionally, JavaFX lets you use CSS (cascading style sheets) to define control and component styles. The combination of FXML and CSS allows you to specify your user interface in a declarative and designer-friendly format while keeping your application logic in the Java language.

The following code snippet shows the JFXPoetry application converted to FXML

```
<StackPane prefHeight="375" prefWidth="500"
 xmlns:fx="http://javafx.com/fxml"
 fx:controller="steveonjava.Controller">
  <children>
    <ImageView fx:id="imageView">
      <image>
        <Image fx:id="image" url="http://farm1.static.flickr.com/…"/>
      </image>
    </ImageView>
    <Text fx:id="text" cache="true" text="
The year's at the spring,&#10;…"/>
    <Button fx:id="button" text="Play Again" onAction="#replay"/>
  </children>
</StackPane>
```

Notice that the element names match the JavaFX classes and that the attributes align with the properties of those classes. For complex types, you can instead specify them as nested elements, which allows you to create the scene graph hierarchy declaratively and specify complex types such as Images.

The fx namespace is defined in the root element and used to hook up a controller class that has the Java application logic to play the animations and media. The controller code is the same as shown earlier, so we have reduced the code to just the stub methods in the following listing:

```
public class Controller implements Initializable {
    private TranslateTransition translate;
    private FadeTransition fade;
    private MediaPlayer mediaPlayer;
    @FXML private Text text;
    @FXML private Image image;
    @FXML private ImageView imageView;
    @FXML private Button button;
    @FXML private void replay(ActionEvent event) {…}
    @Override public void initialize(URL url, ResourceBundle rb) {…}
}
```

The variables annotated with @FXML will get their values injected by the elements in the FXML file with the matching fx:id tags. Any method in the controller can be hooked up as an event handler, which is demonstrated by the replay method that is wired up to the onAction event of the button element in the FXML. Finally, you can override the initialize method to do application setup after all the variables have been initialized, such as setting up the transitions, binding, and media for this poetry example.

The FXML file is complemented by a CSS file that defines the styles for the application elements, as shown in the following listing:

```
#text {
    -fx-font-family: serif;
    -fx-font-weight: bold;
    -fx-font-size: 30pt;
    -fx-fill: goldenrod;
    -fx-effect: dropshadow(three-pass-box, black, 3, .5, 0, 0);
}

#button {
    -fx-background-color: linear-gradient(darkorange, derive(darkorange, -80%));
    -fx-background-radius: 24;
    -fx-padding: 12;
    -fx-font-size: 16pt;
    -fx-font-weight: bold;
    -fx-text-fill: white;
}
```

Note: Notice that the text fill is specified differently for a Text element (which inherits the Shape –fx-fill attribute) vs. a Button (which inherits the Control –fx-text-fill attribute)

JavaFX stylesheets are based on the W3C CSS2.1 specification with some additions inspired from CSS3. In addition, many JavaFX-specific properties are exposed by prefixing the property name with "-fx-", as shown in the above example to set the font size, fill, and drop shadow effect. In addition, the above CSS includes a new styling for the "Play Again" button so it matches the overall theme of the application better.

The final step is to load the FXML and CSS files in the start method of your application, as shown in the following code:

```
Parent root = FXMLLoader.load( ().getResource("JFXPoetry.fxml"));
Scene scene = new Scene(root);
scene.getStylesheets().add( ().getResource("JFXPoetry.css").toExternalForm());
```

Notice that the FXML loader expects a URL, while the stylesheet list on Scene is of type String. To convert a URL to a String for the stylesheet, you can call toExternalForm to get the correct URI syntax for the resource you are loading. Upon running you can see the new CSS button design as shown in Figure 6.

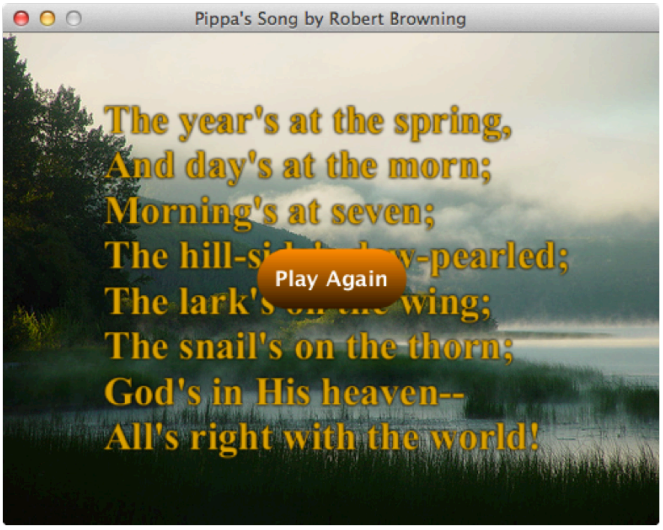


Figure 6: FXML version of JFXPoetry with a CSS-styled button

To try the completed FXML example, complete with animation and audio, click the following url: http://steveonjava.com/refcard/JFXPoetry-FXML.jnlp

For both the Java and FXML versions of this application, you can find the source code on GitHub: https://github.com/steveonjava/JFXPoetry

Hot Tip If you are a JVM language fan, you can also code your application in Scala, Groovy, Visage, or other languages that compile to JVM

API REFERENCE

An easy way to view and navigate the full JavaFX API is to use the Ensemble sample application. This comes with the JavaFX SDK and can be launched by double clicking the Ensemble.jar in the samples folder. Figure 7 shows what the Ensemble application looks like when you first open it.

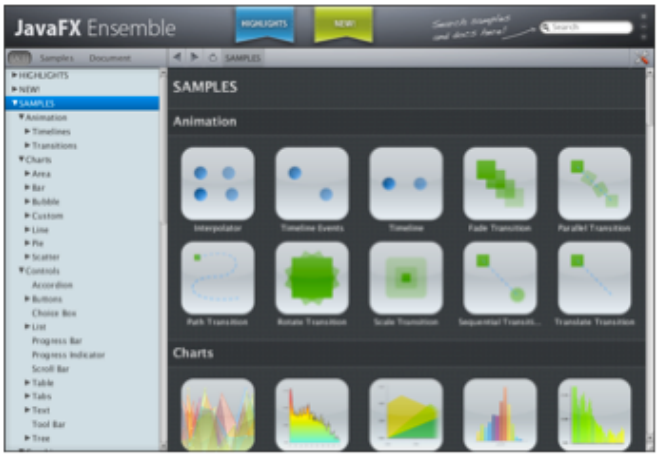


Figure 7: JavaFX Ensemble application

Ensemble comes with live code samples of all the major features of JavaFX as well as an inline API documentation viewer, which makes use of the new WebView component that provides a full-featured, embeddable browser that you can use in your own applications.

ADDITIONAL RESOURCES

- JavaFX API documentation: http://docs.oracle.com/javafx/2.0/api/index.html
- JavaFX FXML introduction: http://docs.oracle.com/javafx/2.0/api/javafx/fxml/doc-files/introduction_to_fxml.html
- JavaFX CSS reference guide: http://docs.oracle.com/javafx/2.0/api/javafx/scene/doc-files/cssref.html
- JFXtras, utilities and add-ons for JavaFX: http://jfxtras.org/
- Pro JavaFX Platform 2 book with free samples for download: http://projavafx.com/
- Examples and news direct from the JavaFX team: http://fxexperience.com/
- My blog covering all things JavaFX: http://steveonjava.com/

Compatible JVM language libraries:

- Scala language bindings: http://scalafx.org/
- Groovy language bindings: http://groovyfx.org/

ABOUT THE AUTHOR

Stephen Chin is a Java Technology Evangelist at Oracle specializing in UI and co-author of the Pro JavaFX Platform 2 title, which is the leading technical reference for JavaFX. He has been featured at Java conferences around the world including Devoxx, Codemash, OSCON, J-Fall, GeeCON, Jazoon, and JavaOne, where he twice received a Rock Star Award. In his evenings and weekends, Stephen is an open-source hacker, working on projects including ScalaFX, a DSL for JavaFX in the Scala language, Visage, a UI oriented JVM language, JFXtras, a JavaFXcomponent and extension library, and Apropos, an Agile Project Portfolio scheduling tool written in JavaFX. Stephen can be followed on twitter@steveonjava and reached via his blog: http://steveonjava.com/

RECOMMENDED BOOK

Pro JavaFX 2:

In Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology, Jim Weaver, Weiqi Gao, Stephen Chin, Dean Iverson, and Johan Vos show you how you can use the JavaFX platform to create rich-client Java applications. You'll see how JavaFX provides a powerful Java-based UI platform capable of handling large-scale data-driven business applications. http://www.apress.com/

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, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
150 Preston Executive Dr.
Suite 200
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-50-9
ISBN-10: 1-936502-50-X
50795
9 781936 502509
$7.95

Version 1.0