

JavaFX

Using Text and Text Effects in JavaFX

Release 2.1

E22627-06

June 2013

JavaFX/Using Text and Text Effects in JavaFX, Release 2.1

E22627-06

Copyright © 2011, 2013, Oracle and/or its affiliates. All rights reserved.

Primary Author: Irina Fedortsova

Contributor: Phil Race

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

1 Using Text and Text Effects in JavaFX

Introduction	1-1
Adding Text	1-1
Setting Text Font and Color	1-2
Making Text Bold or Italic	1-3
Using Custom Fonts	1-3
Setting LCD Text Support	1-3
Applying Effects	1-4
Perspective Effect	1-4
Blur Effect	1-5
Drop Shadow Effect	1-5
Inner Shadow Effect	1-6
Reflection	1-6
Combining Several Effects	1-7
View and Download Application Files	1-9

Using Text and Text Effects in JavaFX

This article explains how to add text and text effects to your JavaFX 2 applications. It also includes code samples to illustrate the APIs being used.

Introduction

The graphical content of JavaFX 2 applications consists of objects organized in a tree-like structure called a **scene graph**. A single element in the scene graph is called a **node**. Nodes can handle different types of content, including text. Nodes can be transformed and animated. You can also apply various effects to nodes. Using features common to all node types enables you to provide sophisticated text content that meets the demands of modern rich Internet applications (RIAs).

The JavaFX 2 release provides the `javafx.scene.text.Text` class that is used to display text. The `Text` class inherits from the `Node` class. For this reason, you can apply effects, animation, and transformations to text nodes in the same way as to any other nodes. Because the `Node` class inherits from the `Shape` class, you can set a stroke or apply a fill setting to text nodes in the same way as to any shape.

Adding Text

To add a text object to your application, use any of the constructors shown in [Example 1-1](#) through [Example 1-3](#).

Example 1-1

```
Text t = new Text();
t.setText("This is a text sample");
```

Example 1-2

```
Text t = new Text("This is a text sample");
```

Example 1-3

```
Text t = new Text (10, 20, "This is a text sample");
```

You can also create text objects by using the `javafx.scene.text.TextBuilder` class as shown in [Example 1-4](#).

Example 1-4

```
Text t = TextBuilder.create().text("This is a text sample").build();
```

Setting Text Font and Color

When adding text, you can also set some of its properties. To set the font, you can use an instance of the `javafx.scene.text.Font` class. The `Font.font()` method enables you to specify the font family name and size. You can also set the text color as shown in [Example 1-5](#).

Example 1-5

```
t.setText("This is a text sample");  
t.setFont(Font.font("Verdana", 20));  
t.setFill(Color.RED);
```

Alternatively, you may want to use a system font, which varies depending on the platform. For this purpose, call the `Font.getDefault()` method.

In the production code, Oracle recommends that you set the styles using cascading style sheets (CSS). For example, to be able to apply a linear gradient fill to your text objects, add the style with the required rules to your CSS as shown in [Example 1-6](#).

Example 1-6

```
#fancytext {  
    -fx-font: 100px Tahoma;  
    -fx-fill: linear-gradient(from 0% 0% to 100% 200%, repeat, aqua 0%, red 50%);  
    -fx-stroke: black;  
    -fx-stroke-width: 1;  
}
```

In your code, create a text object and apply the style from CSS as shown in [Example 1-7](#).

Example 1-7

```
Text t = new Text("Stroke and Fill");  
t.setId("fancytext");
```

This code creates the text shown in [Figure 1-1](#).

Figure 1-1 Text with a Linear Gradient Fill



For more details about using CSS in JavaFX applications, see *Skinning JavaFX Applications with CSS*.

Making Text Bold or Italic

To make the text look bold, use the `FontWeight` constant of the `font` method as shown in [Example 1-8](#).

Example 1-8

```
t.setFont(Font.font("Verdana", FontWeight.BOLD, 70));
```

To display text in italic, use the `FontPosture` constant as shown in [Example 1-9](#).

Example 1-9

```
t.setFont(Font.font("Verdana", FontPosture.ITALIC, 20));
```

Using Custom Fonts

If you need to use a unique font that might not be installed on another computer, you can include a TrueType font (.ttf) or an OpenType (.otf) in your JavaFX 2 application.

To include a TrueType or OpenType font as a custom font, use the following procedure:

1. Create a `resources/fonts` folder in your project folder.
2. Copy your font files to the `fonts` subfolder in your project.
3. In your source code, load the custom font as shown in [Example 1-10](#).

Example 1-10

```
text.setFont(Font.loadFont("file:resources/fonts/isadoracyr.ttf", 120));
```

This code provides the font for the text shown in [Figure 1-2](#).

Figure 1-2 Custom Font



Setting LCD Text Support

LCD (liquid crystal display) text is an anti-aliased text that takes advantage of the properties of LCD panels to render smoother text. You can take advantage of the LCD text on the text nodes by using the API shown in [Example 1-11](#).

Example 1-11

```
text.setFontSmoothingType(FontSmoothingType.LCD);
```

Alternatively, you can provide this setting in a .css file by using the syntax shown in [Example 1-12](#).

Example 1–12

```
.text { -fx-font-smoothing-type: lcd; }
```

Applying Effects

The JavaFX 2 release provides a wide set of effects that reside in the `javafx.scene.effect` package. As already mentioned, you can apply effects to your text nodes. For a complete set of available effects, see the API documentation. You can see some of the effects in action in the `TextEffects` demo application. This application displays text nodes with various effects. Download the `texteffects.zip` file using the link in the sidebar, extract the files, save them on your computer, and open a NetBeans project in the NetBeans IDE.

Perspective Effect

The `PerspectiveTransform` class enables you to imitate a three-dimensional effect for your two-dimensional content. The perspective transformation can map an arbitrary quadrilateral into another quadrilateral. An input for this transformation is your node. To define the perspective transformation, you specify the x and y coordinates of output locations of all four corners. In the `TextEffects` application, the `PerspectiveTransform` effect is set for a group consisting of a rectangle and text as shown in [Example 1–13](#).

Example 1–13

```
PerspectiveTransform pt = new PerspectiveTransform();
pt.setUlx(10.0f);
pt.setUly(10.0f);
pt.setUrx(310.0f);
pt.setUry(40.0f);
pt.setLrx(310.0f);
pt.setLry(60.0f);
pt.setLlx(10.0f);
pt.setLly(90.0f);

g.setEffect(pt);
g.setCache(true);

Rectangle r = new Rectangle();
r.setX(10.0f);
r.setY(10.0f);
r.setWidth(280.0f);
r.setHeight(80.0f);
r.setFill(Color.BLUE);

Text t = new Text();
t.setX(20.0f);
t.setY(65.0f);
t.setText("Perspective");
t.setFill(Color.YELLOW);
t.setFont(Font.font(null, FontWeight.BOLD, 36));

g.getChildren().add(r);
g.getChildren().add(t);
return g;
```

You can see the result of the perspective transformation in [Figure 1–3](#).

Figure 1–3 Text with a Perspective Effect

Blur Effect

The `GaussianBlur` class provides a blur effect based on a Gaussian convolution kernel.

[Example 1–14](#) shows a text node with an applied Gaussian blur effect implemented in the `TextEffects` application.

Example 1–14

```
Text t2 = new Text();
t2.setX(10.0f);
t2.setY(140.0f);
t2.setCache(true);
t2.setText("Blurry Text");
t2.setFill(Color.RED);
t2.setFont(Font.font(null, FontWeight.BOLD, 36));
t2.setEffect(new GaussianBlur());
return t2;
```

You can see the result of the Gaussian blur effect in [Figure 1–4](#).

Figure 1–4 Text with a Blur Effect

Drop Shadow Effect

To implement a drop shadow effect, use the `DropShadow` class. You can specify a color and an offset for the shadow of your text. In the `TextEffects` application, the drop shadow effect is applied to the red text and provides a three-pixel gray shadow. You can see the code in [Example 1–15](#).

Example 1–15

```
DropShadow ds = new DropShadow();
ds.setOffsetY(3.0f);
ds.setColor(Color.color(0.4f, 0.4f, 0.4f));

Text t = new Text();
t.setEffect(ds);
t.setCache(true);
t.setX(10.0f);
t.setY(270.0f);
```

```
t.setFill(Color.RED);
t.setText("JavaFX drop shadow...");
t.setFont(Font.font(null, FontWeight.BOLD, 32));
```

You can see the result of the applied effect in [Figure 1-5](#).

Figure 1-5 Text with a Drop Shadow Effect

The text "JavaFX drop shadow..." is displayed in a large, bold, red font. The text has a soft, grey drop shadow effect applied to it, making it stand out against the white background.

Inner Shadow Effect

An inner shadow effect renders a shadow inside the edges of your content. For text content, you can specify a color and an offset. See how the inner shadow effect with four-pixel offsets in the x and y directions is applied to a text node in [Example 1-16](#).

Example 1-16

```
InnerShadow is = new InnerShadow();
is.setOffsetX(4.0f);
is.setOffsetY(4.0f);

Text t = new Text();
t.setEffect(is);
t.setX(20);
t.setY(100);
t.setText("InnerShadow");
t.setFill(Color.YELLOW);
t.setFont(Font.font(null, FontWeight.BOLD, 80));

t.setTranslateX(300);
t.setTranslateY(300);

return t;
```

The result of the applied effect is shown in [Figure 1-6](#).

Figure 1-6 Text with an Inner Shadow Effect

The text "InnerShadow" is displayed in a large, bold, yellow font. The text has a soft, grey inner shadow effect applied to it, making it stand out against the white background.

Reflection

The `Reflection` class enables you to display the reflected version of your text below the original text. You can adjust the view of your reflected text by providing additional parameters such as a bottom opacity value, a fraction of the input to be visible in the

reflection, an offset between the input text and its reflection, and a top opacity value. For details, see the API documentation.

The reflection effect is implemented in the `TextEffects` application as shown in [Example 1–17](#).

Example 1–17

```
Text t = new Text();
t.setX(10.0f);
t.setY(50.0f);
t.setCache(true);
t.setText("Reflections on JavaFX...");
t.setFill(Color.RED);
t.setFont(Font.font(null, FontWeight.BOLD, 30));

Reflection r = new Reflection();
r.setFraction(0.7f);

t.setEffect(r);

t.setTranslateY(400);
return t;
```

You can see the text node with the reflection effect in [Figure 1–7](#).

Figure 1–7 Text with a Reflection Effect



Combining Several Effects

In the previous section you learned how to apply a single effect to a text node. To further enrich your text content, you can compose several effects and apply a chain of effects to achieve a specific visual result. Consider the `NeonSign` application shown in [Figure 1–8](#).

Figure 1–8 *The Neon Sign Application Window*

The graphical scene of the `NeonSign` application consists of the following elements:

- An image of a brick wall used as a background
- A rectangle that provides a radial gradient fill
- A text node with a chain of effects
- A text field used for entering text data

The background is set in an external `.css` file.

This application uses a binding mechanism to set the text content on the text node. The text property of the text node is bound to the text property of the text field as shown in [Example 1–18](#).

Example 1–18

```
Text text = new Text();
TextField textField = new TextField();
textField.setText("Neon Sign");
text.textProperty().bind(textField.textProperty());
```

You can type in the text field and view the changed contents of the text node.

A chain of effects is applied to the text node. The primary effect is a blend effect that uses the `MULTIPLY` mode to mix two inputs together: a drop shadow effect and another blend effect, `blend1`. Similarly, the `blend1` effect combines a drop shadow effect (`ds1`) and a blend effect (`blend2`). The `blend2` effect combines two inner shadow effects. Using this chain of effects and varying color parameters enables you to apply subtle and sophisticated color patterns to text objects. See the code for the chain of effects in [Example 1–19](#).

Example 1–19

```
Blend blend = new Blend();
```

```
blend.setMode(BlendMode.MULTIPLY);

DropShadow ds = new DropShadow();
ds.setColor(Color.rgb(254, 235, 66, 0.3));
ds.setOffsetX(5);
ds.setOffsetY(5);
ds.setRadius(5);
ds.setSpread(0.2);

blend.setBottomInput(ds);

DropShadow ds1 = new DropShadow();
ds1.setColor(Color.web("#f13a00"));
ds1.setRadius(20);
ds1.setSpread(0.2);

Blend blend2 = new Blend();
blend2.setMode(BlendMode.MULTIPLY);

InnerShadow is = new InnerShadow();
is.setColor(Color.web("#feeb42"));
is.setRadius(9);
is.setChoke(0.8);
blend2.setBottomInput(is);

InnerShadow is1 = new InnerShadow();
is1.setColor(Color.web("#f13a00"));
is1.setRadius(5);
is1.setChoke(0.4);
blend2.setTopInput(is1);

Blend blend1 = new Blend();
blend1.setMode(BlendMode.MULTIPLY);
blend1.setBottomInput(ds1);
blend1.setTopInput(blend2);

blend.setTopInput(blend1);

text.setEffect(blend);
```

In this article, you learned how to add text and apply various effects to text content. For a complete set of available effects, see the API documentation.

If you need to implement a text editing area in your JavaFX application, use the `HTMLTextArea` component. For more information about the `HTMLTextArea` control, see [Using JavaFX UI Controls](#).

View and Download Application Files

View Source Code

<http://docs.oracle.com/javafx/2/text/TextEffects.java.html>

<http://docs.oracle.com/javafx/2/text/NeonSign.java.html>

Download Source Code

<http://docs.oracle.com/javafx/2/text/texteffects.zip>

<http://docs.oracle.com/javafx/2/text/neonsign.zip>

