

JavaFX

Embedding Swing Content in JavaFX Applications

Release 8.0 Developer Preview

E41513-01

September 2013

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

Primary Author: Irina Fedortsova

Contributing Author:

Contributor: Anton Tarasov

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 END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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 failsafe, 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 of The Open Group.

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 Embedding Swing Content in JavaFX Applications

SwingNode Class	1-1
Embedding Swing Content and Handling Events	1-2
Adding Interoperability Between Swing and JavaFX Components	1-5
Conclusion	1-10

Embedding Swing Content in JavaFX Applications

This article describes how to embed Swing components in JavaFX applications. It discusses the threading restrictions and provides working applications that illustrate Swing buttons with HTML content embedded in a JavaFX application and interoperability between Swing and JavaFX buttons.

The ability to embed JavaFX content in Swing applications has existed since the JavaFX 2.0 release. To enhance the interoperability of JavaFX and Swing, JavaFX 8 introduces a new class that provides reverse integration and enables developers to embed Swing components in JavaFX applications.

Before you run any code from this article, install JDK 8 on your computer.

SwingNode Class

JavaFX 8 introduces the `SwingNode` class, which is located in the `javafx.embed.swing` package. This class enables you to embed Swing content in a JavaFX application. To specify the content of the `SwingNode` object, call the `setContent` method, which accepts an instance of the `javax.swing.JComponent` class. You can call the `setContent` method on either the JavaFX application thread or event dispatch thread (EDT). However, to access the Swing content, ensure that your code runs on EDT, because the standard Swing threading restrictions apply.

The code shown in [Example 1–1](#) illustrates the general pattern of using the `SwingNode` class.

Example 1–1

```
import javafx.application.Application;
import javafx.embed.swing.SwingNode;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
import javax.swing.JButton;
import javax.swing.SwingUtilities;

public class SwingFx extends Application {

    @Override
    public void start (Stage stage) {
        final SwingNode swingNode = new SwingNode();

        createSwingContent (swingNode);

        stage.setScene (new Scene (swingNode));
        stage.show ();
    }

    private void createSwingContent (SwingNode swingNode) {
        JButton button = new JButton ("Hello JavaFX!");
        swingNode.setContent (button);
    }
}
```

```
StackPane pane = new StackPane();
pane.getChildren().add(swingNode);

stage.setTitle("Swing in JavaFX");
stage.setScene(new Scene(pane, 250, 150));
stage.show();
}

private void createSwingContent(final SwingNode swingNode) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            swingNode.setContent(new JButton("Click me!"));
        }
    });
}
```

When run, this code produces the output shown in [Figure 1–1](#).

Figure 1–1 *Swing JButton Embedded in a JavaFX Application*



Embedding Swing Content and Handling Events

The ButtonHtmlDemo in the Swing tutorial adds font, color, and other formatting to three buttons shown in [Example 1–2](#) and [Example 1–3](#). The buttons respond to mouse and keyboard events as shown in [Example 1–5](#) and [Example 1–6](#). [Figure 1–2](#) shows the three buttons created using Swing in the ButtonHtmlDemo now embedded in a JavaFX Application (SwingNodeSample). You will create the SwingNodeSample application and ensure that all events are delivered to an appropriate Swing button and get processed.

Figure 1–2 *ButtonHtmlDemo Embedded in a JavaFX Application*



The left and right buttons have multiple lines of text implemented with the HTML formatting as shown in [Example 1–2](#).

Example 1–2

```
b1 = new JButton("<html><center><b><u>D</u>isable</b><br>"  
+ "<font color=#ffffdd>middle button</font>",  
leftButtonIcon);  
  
b3 = new JButton("<html><center><b><u>E</u>nable</b><br>"  
+ "<font color=#ffffdd>middle button</font>",  
rightButtonIcon);
```

The simple format of middle button does not require HTML, so it is initialized with a string label and an image as shown in [Example 1–3](#).

Example 1–3

```
b2 = new JButton("middle button", middleButtonIcon);
```

All three buttons have the tooltips and mnemonic characters as shown in [Example 1–4](#).

Example 1–4

```
b1.setToolTipText("Click this button to disable the middle button.");  
b2.setToolTipText("This middle button does nothing when you click it.");  
b3.setToolTipText("Click this button to enable the middle button.");  
  
b1.setMnemonic(KeyEvent.VK_D);  
b2.setMnemonic(KeyEvent.VK_M);  
b3.setMnemonic(KeyEvent.VK_E);
```

The left and right buttons are used to disable and enable the middle button respectively. To enable the application to detect and respond to user action on these buttons, attach action listeners and set action commands as shown in [Example 1–5](#).

Example 1–5

```
b1.addActionListener(this);  
b3.addActionListener(this);  
  
b1.setActionCommand("disable");  
b3.setActionCommand("enable");
```

Implement the actionPerformed method shown in [Example 1–6](#). This method is called when the user clicks the left or right button.

Example 1–6

```
public void actionPerformed(ActionEvent e) {  
    if ("disable".equals(e.getActionCommand())) {  
        b2.setEnabled(false);  
        b1.setEnabled(false);  
        b3.setEnabled(true);  
    } else {  
        b2.setEnabled(true);  
        b1.setEnabled(true);  
        b3.setEnabled(false);  
    }  
}
```

See the complete code of the `ButtonHtml` class.

Now set up a JavaFX project and run the `SwingNodeSample` application.

To create the `SwingNodeSample` application:

Ensure that JDK 8 is installed on your computer. Then set up a JavaFX project in NetBeans IDE:

1. From the **File** menu, choose **New Project**.
2. In the **JavaFX** application category, choose **JavaFX Application** and click **Next**.
3. Name the project **SwingNodeSample** and select a JavaFX platform based on JDK 8. Click **Finish**.
4. In the **Projects** window, right-click the `swingnodesample` folder under Source Packages. Choose **New** and then choose **Java class**.
5. Name a new class **ButtonHtml** and click **Finish**.
6. Copy the code of the `ButtonHtml` class and paste it in the project.
7. Open the `swingnodesample` folder on your disk and create the `images` folder.
8. Download the images `left.gif`, `middle.gif`, and `right.gif` and save them in the `images` folder.
9. In the `SwingNodeSample` class, remove the code inside the `start` method that was automatically generated by NetBeans.
10. Instead, create a `SwingNode` object and implement the `start` method as shown in [Example 1–7](#).

Example 1–7

```
@Override  
public void start(Stage stage) {  
    final SwingNode swingNode = new SwingNode();  
    createSwingContent(swingNode);  
    StackPane pane = new StackPane();  
    pane.getChildren().add(swingNode);  
  
    Scene scene = new Scene(pane, 450, 100);  
    stage.setScene(scene);  
    stage.setTitle("ButtonHtmlDemo Embedded in JavaFX");  
    stage.show();  
}
```

11. To embed the three buttons produced by the `ButtonHtml` class, set the content of the `SwingNode` object to be an instance of the `ButtonHtml` class as shown in [Example 1–8](#).

Example 1–8

```
private void createSwingContent(final SwingNode swingNode) {  
    SwingUtilities.invokeLater(new Runnable() {  
        @Override  
        public void run() {  
            swingNode.setContent(new ButtonHtml());  
        }  
    });  
}
```

12. Press Ctrl (or Cmd) + Shift + I to correct the import statements.

To download the source code of the `SwingNodeSample` application, click the `SwingNodeSample.zip` link.

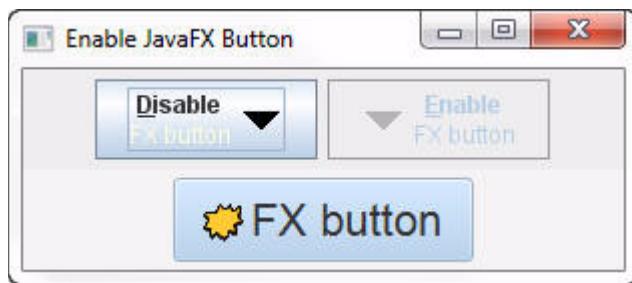
Run the `SwingNodeSample` project and ensure that all means of interactivity provided for the buttons work as they should:

- With the mouse, hover over the buttons and see the tooltips.
- Click the left and right buttons to disable and enable the middle button respectively.
- Press Alt + D and Alt + E keys to disable and enable the middle button respectively.

Adding Interoperability Between Swing and JavaFX Components

You can provide interoperability between JavaFX buttons and Swing buttons. For example, the `EnableFXButton` application shown in [Figure 1–3](#) enables a user to click Swing buttons to disable or enable a JavaFX button. Conversely, the `EnableButtons` application shown in [Figure 1–4](#) enables a user to click a JavaFX button to activate a Swing button.

Figure 1–3 Enable JavaFX Button Sample



Using Swing Buttons to Operate a JavaFX Button

The `EnableFXButton` application is created by modifying the `SwingNodeSample` application and making the middle button an instance of the `javafx.scene.control.Button` class. In the modified application, the Swing buttons (Disable FX button) and (Enable FX button) are used to disable and enable a JavaFX button (FX Button). [Figure 1–3](#) shows the `EnableFXButton` application.

Follow these steps to create the `EnableFXButton` application:

1. From the **File** menu, choose **New Project**.
2. In the **JavaFX** application category, choose **JavaFX Application** and click **Next**.
3. Name the project **EnableFXButton**.
4. In the **Projects** window, right-click the `enablefxbutton` folder under **Source Packages**. Choose **New** and then choose **Java class**.
5. Name the new class **ButtonHtml** and click **Finish**.
6. Copy the code of the `ButtonHtml` class and paste it in the project.
7. Change the package declaration to `enablefxbutton`.
8. Open the `enablefxbutton` folder on your disk and create the `images` folder.

9. Download the images down.gif and middle.gif and save them in the images folder.
10. In the EnableFXButton class, declare a Button object as shown in [Example 1–9](#).

Example 1–9

```
public class EnableFXButton extends Application {  
    public static Button fxbutton;
```

11. Remove the code inside the start method that was automatically generated by NetBeans IDE and implement the start method as shown in [Example 1–10](#).

Example 1–10

```
@Override  
public void start(Stage stage) {  
    final SwingNode swingNode = new SwingNode();  
    createSwingContent(swingNode);  
    BorderPane pane = new BorderPane();  
    fxbutton = new Button("FX button");  
  
    pane.setTop(swingNode);  
    pane.setCenter(fxbutton);  
  
    Scene scene = new Scene(pane, 300, 100);  
    stage.setScene(scene);  
    stage.setTitle("Enable JavaFX Button");  
    stage.show();  
}
```

12. Add the import statement for the SwingNode class as shown in [Example 1–11](#).

Example 1–11

```
import javafx.embed.swing.SwingNode;
```

13. Implement the createSwingContent method to set the content of the SwingNode object as shown in [Example 1–12](#).

Example 1–12

```
private void createSwingContent(final SwingNode swingNode) {  
    SwingUtilities.invokeLater(new Runnable() {  
        @Override  
        public void run() {  
            swingNode.setContent(new ButtonHtml());  
        }  
  
    });  
}
```

14. Press Ctrl (or Cmd) + Shift + I to add an import statement to the javax.swing.SwingUtilities class.
15. Replace the initialization of fxbutton with the code shown in [Example 1–13](#) to add an image and set a tooltip and style for the JavaFX button.

Example 1–13

```
Image fxButtonIcon = new Image(
```

```

getClass().getResourceAsStream("images/middle.gif"));

fxbutton = new Button("FX button", new ImageView(fxButtonIcon));
fxbutton.setTooltip(
new Tooltip("This middle button does nothing when you click it."));
fxbutton.setStyle("-fx-font: 22 arial; -fx-base: #cce6ff;");

```

- 16.** Press Ctrl (or Cmd) + Shift + I to add the import statements shown in [Example 1-14](#).

Example 1-14

```

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.control.Tooltip;

```

- 17.** Open the ButtonHtml class and remove all code related to the middle button b2.
- 18.** Use the down.gif image for b1 (Disable FX button) and b3 (Enable FX button) buttons as shown in [Example 1-15](#).

Example 1-15

```

ImageIcon buttonIcon = createImageIcon("images/down.gif");
b1 = new JButton("<html><center><b><u>D</u>isable</b><br>" +
+ "<font color=#ffffdd>FX button</font>",
buttonIcon);
b3 = new JButton("<html><center><b><u>E</u>nable</b><br>" +
+ "<font color=#ffffdd>FX button</font>",
buttonIcon);

```

- 19.** Modify the actionPerformed method to implement the disabling and enabling of fxbutton as shown in [Example 1-16](#). Note that the disabling and enabling of the JavaFX button must happen on the JavaFX application thread.

Example 1-16

```

@Override
public void actionPerformed(ActionEvent e) {
    if ("disable".equals(e.getActionCommand())) {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                EnableFXButton.fxbutton.setDisable(true);
            }
        });
        b1.setEnabled(false);
        b3.setEnabled(true);
    } else {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                EnableFXButton.fxbutton.setDisable(false);
            }
        });
        b1.setEnabled(true);
        b3.setEnabled(false);
    }
}

```

20. Press Ctrl (or Cmd) + Shift + I to add the import statement shown in [Example 1–17](#).

Example 1–17

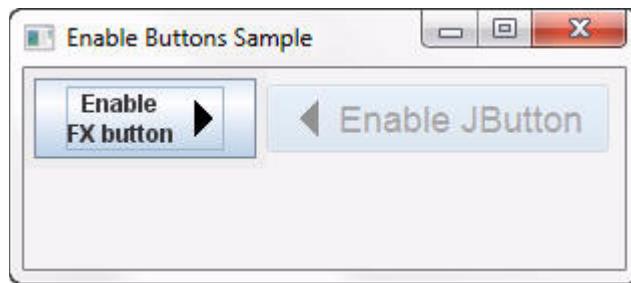
```
import javafx.application.Platform;
```

21. Run the application and click the Swing buttons to disable and enable the JavaFX button, as shown in [Figure 1–3](#).

Using a JavaFX Button to Operate a Swing Button

You can further modify the EnableFXButton application and implement the setOnAction method for the JavaFX button so that clicking the JavaFX button activates the Swing button. The modified application (EnableButtons) is shown in [Figure 1–4](#).

Figure 1–4 Enable Buttons Sample



To create the EnableButtons application:

1. Copy the EnableFXButton project and save it under the EnableButtons name.
2. Rename the EnableFXButton class to EnableButtons and the enablefxbutton package to enablebuttons.
3. Correct the package statement in both the ButtonHtml and EnableButtons classes.
4. Open the EnableButtons class and make the pane an instance of the FlowPane class as shown in [Example 1–18](#).

Example 1–18

```
FlowPane pane = new FlowPane();
```

5. Modify the initialization of the fxButtonIcon variable to use the left.gif image as shown in [Example 1–19](#).

Example 1–19

```
Image fxButtonIcon = new Image(  
getClass().getResourceAsStream("images/left.gif"));
```

6. Change the fxbutton text, tooltip, and font size and set the disableProperty to true as shown in [Example 1–20](#).

Example 1–20

```
fxbutton = new Button("Enable JButton", new ImageView(fxButtonIcon));  
fxbutton.setTooltip(
```

```

new Tooltip("Click this button to enable the Swing button."));
fxbutton.setStyle("-fx-font: 18 arial; -fx-base: #cce6ff;");
fxbutton.setDisable(true);

```

7. Implement the `setOnAction` method as shown in [Example 1–21](#). Note that you must change Swing objects on event dispatch thread only.

Example 1–21

```

fxbutton.setOnAction(new javafx.event.EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                ButtonHtml.b1.setEnabled(true);
            }
        });
        fxbutton.setDisable(true);
    }
});

```

Note: Ignore the error mark that appears on the left of the line that enables `b1`. You will correct this error at step 11.

8. Press Ctrl (or Cmd) + Shift + I to add the import statement to the `javafx.event.ActionEvent` class.
9. Add the `swingNode` and `fxbutton` objects to the layout container as shown in [Example 1–22](#).

Example 1–22

```
pane.getChildren().addAll(swingNode, fxbutton);
```

10. Change the application title to "Enable Buttons Sample" as shown in [Example 1–23](#).

Example 1–23

```
stage.setTitle("Enable Buttons Sample");
```

11. Open the `ButtonHtml` class and change the modifier of the `b1` button to `public static`. Notice that the error mark in the `EnableButtons` class has disappeared.
12. Remove all code related to the `b3` button and the line that sets an action command for `b1`.
13. Modify the `actionPerformed` method as shown in [Example 1–24](#).

Example 1–24

```

@Override
public void actionPerformed(ActionEvent e) {
    Platform.runLater(new Runnable() {
        @Override
        public void run() {
            EnableButtons.fxbutton.setDisable(false);
        }
    });
}

```

```
        }
    });
b1.setEnabled(false);
}
```

Conclusion

In this article you learned how to embed existing Swing components in JavaFX applications and provide interoperability between Swing and JavaFX objects. The ability to embed Swing content into JavaFX applications enables developers to migrate Swing applications that use complex third-party Swing components for which they do not have source code or applications that have legacy modules that exist only in maintenance mode.