JavaFX

Interoperability

Release 8

E50477-01

March 2014

This tutorial describes the capabilities provided by the javafx.concurrent package to create multithreaded applications. You find out how to integrate JavaFX content into Swing applications and how to use Swing components in JavaFX applications. You learn how to add JavaFX scene graph to a Standard Widget Toolkit (SWT) application, and how to make SWT and JavaFX controls interoperate.

ORACLE®

JavaFX Interoperability, Release 8

E50477-01

Primary Author: Irina Fedortsova, Nancy Hilderbrandt, Steve Northover

Contributor: Artem Ananiev, Anton Tarasov, Alexander Zvegintsev, Alexander Kouznetsov

Contents

About This Document vii
Audience vii
Documentation Accessibility vii
Related Documents vii
Conventions vii

Part I Concurrency in JavaFX

1 Concurrency in JavaFX

Why Use the javafx.concurrent Package? 1-1
Overview of the javafx.concurrent Package 1-1
The Worker Interface 1-2
The Task Class 1-2
Cancelling the Task 1-3
Showing the Progress of a Background Task 1-4
The Service Class 1-4
The WorkerStateEvent Class and State Transitions 1-6
The ScheduledService Class 1-7
Conclusion 1-8

Part II JavaFX-Swing Interoperability

2 The JavaFX Advantage for Swing Developers

Using FXML 2-1
JavaFX Scene Builder 2-1
CSS Support 2-1
JavaFX Media Support 2-2
Animation 2-2
HTML Content 2-2

3 Integrating JavaFX into Swing Applications

Adding JavaFX Content to a Swing Component 3-1
Swing–JavaFX Interoperability and Threads 3-2
Changing JavaFX Data in Response to a Change in Swing Data 3-2
Changing Swing Data in Response to a Change in JavaFX Data 3-3

Introducing the SimpleSwingBrowser Application ... 3-3
Initializing Swing Data ... 3-3
Loading JavaFX Content ... 3-5
Updating Swing Data ... 3-6
Application Files ... 3-7

4 Enriching Swing Applications with JavaFX Functionality

Sample Swing Application ... 4-1
Integrating JavaFX Bar Chart ... 4-2
Application Files ... 4-5

5 Leveraging Applications with Media Features

About Media Integration ... 5-1
Building the Media Player Application ... 5-1
Skinning the Application with CSS ... 5-2
Adding a New Control to the Control Bar ... 5-3
Application Files ... 5-4

6 Implementing a Swing Application in JavaFX

Analyzing the Converter Application Developed in Swing ... 6-1
Planning the Converter Application in JavaFX ... 6-2
Creating the Converter Application in JavaFX ... 6-2
Standard JavaFX Pattern to Create the GUI ... 6-2
Containers and Layouts ... 6-3
UI Controls ... 6-3
Mechanism of Getting Notifications on User Actions and Binding ... 6-4
Creating the ConversionPanel Class ... 6-4
Creating Instance Variables for UI Controls ... 6-4
Creating DoubleProperty and NumberFormat Objects ... 6-5
Laying Out the Components ... 6-5
Creating InvalidationListener Objects ... 6-6
Adding Change Listeners to Controls and Ensuring Synchronization ... 6-6
Creating the Converter Class ... 6-6
Defining Instance Variables ... 6-6
Creating the Constructor for the Converter Class ... 6-7
Creating the Graphical Scene ... 6-7
Application Files ... 6-8

7 Embedding Swing Content in JavaFX Applications

SwingNode Class ... 7-1
Embedding Swing Content and Handling Events ... 7-2
Adding Interoperability Between Swing and JavaFX Components ... 7-5
Conclusion ... 7-9
Application Files ... 7-9

Part III Interoperability with SWT

8 JavaFX Interoperability with SWT

Introduction 8-1
Adding JavaFX Content to an SWT Component 8-2
Creating SWT-JavaFX Applications in an IDE 8-4
Packaging SWT-JavaFX Applications 8-4
Packaging the Application when JavaFX is Bundled with the JDK 8-4
Packaging the Application with a Standalone JavaFX Installation 8-4
Application Files 8-4

Part IV Source Code for the Interoperability Tutorial

A SimpleSwingBrowser.java

B SwingInterop.java

C SampleTableModel.java

D MediaPlayer.java

E MediaControl.java

F mediaplayer.css

G Converter.java

H ConversionPanel.java

I SwingNodeSample.java

J ButtonHtmlDemo.java

K EnableFXButton.java

L EnableButtons.java

M Image Source Files

left.gif M-1
right.gif M-2
down.gif M-2
middle.gif M-2

Preface

This preface describes the document accessibility features and conventions used in this tutorial - *JavaFX Interoperability Tutorial*.

About This Document

This tutorial describes the capabilities provided by the `javafx.concurrent` package to create multithreaded applications. You find out how to integrate JavaFX content into Swing applications and vice versa, how to use Swing components in JavaFX applications. You also learn how to add a JavaFX scene graph to a Standard Widget Toolkit (SWT) application and how to make SWT and JavaFX controls interoperate.

Audience

This document is intended for JavaFX developers.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the JavaFX documentation set:

- *Getting Started with JavaFX*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Concurrency in JavaFX

In this tutorial, you learn about the capabilities provided by the `javafx.concurrent` package to create multithreaded applications.

This tutorial contains the following topics:

- The Worker Interface
- The Task Class
- The Service Class
- The WorkerStateEvent Class and State Transitions
- The ScheduledService Class

1

Concurrency in JavaFX

This chapter describes the capabilities provided by the `javafx.concurrent` package to create multithreaded applications.

You learn how to keep your JavaFX application user interface (UI) responsive by delegating time-consuming task execution to background threads.

Why Use the javafx.concurrent Package?

The JavaFX scene graph, which represents the graphical user interface of a JavaFX application, is not thread-safe and can only be accessed and modified from the UI thread also known as the JavaFX Application thread. Implementing long-running tasks on the JavaFX Application thread inevitably makes an application UI unresponsive. A best practice is to do these tasks on one or more background threads and let the JavaFX Application thread process user events.

If you have special requirements or need extra power over the code, implementing a background worker by creating a Runnable object and a new thread is an appropriate way to go. Note that at some point you must communicate with the JavaFX Application thread, either with a result or with the progress of the background task.

For the most cases and for the majority of developers the recommended way is to use the JavaFX APIs provided by the `javafx.concurrent` package, which takes care of multithreaded code that interacts with the UI and ensures that this interaction happens on the correct thread.

Overview of the javafx.concurrent Package

The Java platform provides a complete set of concurrency libraries available through the `java.util.concurrent` package. The `javafx.concurrent` package leverages the existing API by considering the JavaFX Application thread and other constraints faced by GUI developers.

The `javafx.concurrent` package consists of the `Worker` interface and two concrete implementations, `Task` and `Service` classes. The `Worker` interface provides APIs that are useful for a background worker to communicate with the UI. The `Task` class is a fully observable implementation of the `java.util.concurrent.FutureTask` class. The `Task` class enables developers to implement asynchronous tasks in JavaFX applications. The `Service` class executes tasks.

The `WorkerStateEvent` class specifies an event that occurs whenever the state of a Worker implementation changes. Both the `Task` and `Service` classes implement the `EventTarget` interface and thus support listening to the state events.

The Worker Interface

The `Worker` interface defines an object that performs some work on one or more background threads. The state of the Worker object is observable and usable from the JavaFX Application thread.

The lifecycle of the Worker object is defined as follows. When created, the Worker object is in the `READY` state. Upon being scheduled for work, the Worker object transitions to the `SCHEDULED` state. After that, when the Worker object is performing the work, its state becomes `RUNNING`. Note that even when the Worker object is immediately started without being scheduled, it first transitions to the `SCHEDULED` state and then to the `RUNNING` state. The state of a Worker object that completes successfully is `SUCCEEDED`, and the `value` property is set to the result of this Worker object. Otherwise, if any exceptions are thrown during the execution of the Worker object, its state becomes `FAILED` and the `exception` property is set to the type of the exception that occurred. At any time before the end of the Worker object the developer can interrupt it by invoking the `cancel` method, which puts the Worker object into the `CANCELLED` state.

Distinctions in the lifecycle of a `ScheduledService` object can be found in the The ScheduledService Class section.

The progress of the work being done by the Worker object can be obtained through three different properties such as `totalWork`, `workDone`, and `progress`.

For more information on the range of the parameter values, see the API documentation.

The Task Class

Tasks are used to implement the logic of work that needs to be done on a background thread. First, you need to extend the `Task` class. Your implementation of the `Task` class must override the `call` method to do the background work and return the result.

The `call` method is invoked on the background thread, therefore this method can only manipulate states that are safe to read and write from a background thread. For example, manipulating an active scene graph from the `call` method throws runtime exceptions. On the other hand, the `Task` class is designed to be used with JavaFX GUI applications, and it ensures that any changes to public properties, change notifications for errors or cancellation, event handlers, and states occur on the JavaFX Application thread. Inside the `call` method, you can use the `updateProgress`, `updateMessage`, `updateTitle` methods, which update the values of the corresponding properties on the JavaFX Application thread. However, if the task was canceled, a return value from the `call` method is ignored.

Note that the `Task` class fits into the Java concurrency libraries because it inherits from the `java.utils.concurrent.FutureTask` class, which implements the `Runnable` interface. For this reason, a Task object can be used within the Java concurrency `Executor` API and also can be passed to a thread as a parameter. You can call the Task object directly by using the `FutureTask.run()` method, which enables calling this task from another background thread. Having a good understanding of the Java concurrency API will help you understand concurrency in JavaFX.

A task can be started in one of the following ways:

- By starting a thread with the given task as a parameter:

```
Thread th = new Thread(task);
th.setDaemon(true);
```

```
th.start();
```

- By using the `ExecutorService` API:

```
ExecutorService.submit(task);
```

The `Task` class defines a one-time object that cannot be reused. If you need a reusable Worker object, use the `Service` class.

Cancelling the Task

There is no reliable way in Java to stop a thread in process. However, the task must stop processing whenever `cancel` is called on the task. The task is supposed to check periodically during its work whether it was cancelled by using the `isCancelled` method within the body of the `call` method. Example 1–1 shows a correct implementation of the `Task` class that checks for cancellation.

Example 1–1

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations;
        for (iterations = 0; iterations < 100000; iterations++) {
            if (isCancelled()) {
               break;
            }
            System.out.println("Iteration " + iterations);
        }
        return iterations;
    }
};
```

If the task implementation has blocking calls such as `Thread.sleep` and the task is cancelled while in a blocking call, an `InterruptedException` is thrown. For these tasks, an interrupted thread may be the signal for a cancelled task. Therefore, tasks that have blocking calls must double-check the `isCancelled` method to ensure that the `InterruptedException` was thrown due to the cancellation of the task as shown in Example 1–2.

Example 1–2

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations;
        for (iterations = 0; iterations < 1000; iterations++) {
            if (isCancelled()) {
                updateMessage("Cancelled");
                break;
            }
            updateMessage("Iteration " + iterations);
            updateProgress(iterations, 1000);

            //Block the thread for a short time, but be sure
            //to check the InterruptedException for cancellation
            try {
                Thread.sleep(100);
            } catch (InterruptedException interrupted) {
```

```
                if (isCancelled()) {
                    updateMessage("Cancelled");
                    break;
                }
            }
        }
        return iterations;
    }
};
```

Showing the Progress of a Background Task

A typical use case in multithreaded applications is showing the progress of a background task. Suppose you have a background task that counts from one to one million and a progress bar, and you must update the progress on this progress bar as the counter runs in the background. Example 1–3 shows how to update a progress bar.

Example 1–3

```
import javafx.concurrent.Task;

Task task = new Task<Void>() {
    @Override public Void call() {
        static final int max = 1000000;
        for (int i=1; i<=max; i++) {
            if (isCancelled()) {
               break;
            }
            updateProgress(i, max);
        }
        return null;
    }
};
ProgressBar bar = new ProgressBar();
bar.progressProperty().bind(task.progressProperty());
new Thread(task).start();
```

First, you create the task by overriding the `call` method where you implement the logic of the work to be done and invoke the `updateProgress` method, which updates the `progress`, `totalWork`, and `workDone` properties of the task. This is important because you can now use the `progressProperty` method to retrieve the progress of the task and bind the progress of the bar to the progress of the task.

The Service Class

The `Service` class is designed to execute a Task object on one or several background threads. The `Service` class methods and states must only be accessed on the JavaFX Application thread. The purpose of this class is to help the developer to implement the correct interaction between the background threads and the JavaFX Application thread.

You have the following control over the Service object: you can start, cancel and restart it as you need. To start the Service object, use the `Service.start()` method.

Using the `Service` class, you can observe the state of the background work and optionally cancel it. Later, you can reset the service and restart it. Thus, the service can be defined declaratively and restarted on demand.

For a service that needs to be automatically restarted, see The ScheduledService Class section.

When implementing the subclasses of the `Service` class, be sure to expose the input parameters to the Task object as properties of the subclass.

The service can be executed in one of the following ways:

- By an Executor object, if it is specified for the given service
- By a daemon thread, if no executor is specified
- By a custom executor such as a ThreadPoolExecutor

Example 1–4 shows an implementation of the `Service` class which reads the first line from any URL and returns it as a string.

Example 1–4

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.concurrent.Service;
import javafx.concurrent.Task;
import javafx.concurrent.WorkerStateEvent;
import javafx.event.EventHandler;
import javafx.stage.Stage;

public class FirstLineServiceApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        FirstLineService service = new FirstLineService();
        service.setUrl("http://google.com");
        service.setOnSucceeded(new EventHandler<WorkerStateEvent>() {

            @Override
            public void handle(WorkerStateEvent t) {
                System.out.println("done:" + t.getSource().getValue());
            }
        });
        service.start();
    }

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

    private static class FirstLineService extends Service<String> {
        private StringProperty url = new SimpleStringProperty();

        public final void setUrl(String value) {
            url.set(value);
        }

        public final String getUrl() {
            return url.get();
```

```
        }

        public final StringProperty urlProperty() {
           return url;
        }


        @Override
        protected Task<String> createTask() {
            return new Task<String>() {
                @Override
                protected String call()
                        throws IOException, MalformedURLException {
                    try ( BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                    new URL(getUrl()).openStream;
                            in = new BufferedReader(
                                new InputStreamReader(u.openStream()))) {
                        return in.readLine();
                    }
                }
        };
    }
}
```

The WorkerStateEvent Class and State Transitions

Whenever the state of the Worker implementation changes, an appropriate event, defined by the `WorkerStateEvent` class, occurs. For example, when the Task object transitions to the `SUCCEEDED` state, the `WORKER_STATE_SUCCEEDED` event occurs, the `onSucceeded` event handler is called, after which the protected convenience method `succeeded` is invoked on the JavaFX Application thread.

There are several protected convenience methods such as `cancelled`, `failed`, `running`, `scheduled`, and `succeeded`, which are invoked when the Worker implementation transitions to the corresponding state. These methods can be overridden by subclasses of the `Task` and `Service` classes when the state is changed to implement the logic of your application. Example 1–5 shows a Task implementation that updates the status message on the task's success, cancellation, and failure.

Example 1–5

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations = 0;
        for (iterations = 0; iterations < 100000; iterations++) {
            if (isCancelled()) {
                break;
            }
            System.out.println("Iteration " + iterations);
        }
        return iterations;
    }

    @Override protected void succeeded() {
        super.succeeded();
        updateMessage("Done!");
```

```
    }

    @Override protected void cancelled() {
        super.cancelled();
        updateMessage("Cancelled!");
    }

@Override protected void failed() {
    super.failed();
    updateMessage("Failed!");
    }
};
```

The ScheduledService Class

Many use cases that involve polling require a service that restarts automatically. To meet those needs, the `Service` class was extended to produce the `ScheduledService` class. The `ScheduledService` class represents a service that automatically restarts after a successful execution and, under special conditions, upon its failure.

When created, the `ScheduledService` object is in the `READY` state.

After calling the `ScheduledService.start()` or `ScheduledService.restart()` method, the `ScheduledService` object transitions to the `SCHEDULED` state for the duration specified by the `delay` property.

In the `RUNNING` state, the `ScheduledService` object executes its task.

Task completes successfully

After the task completes, the `ScheduledService` object transitions to the `SUCCEEDED` state, then to the `READY` state, and then back to the `SCHEDULED` state. The duration of being in the `SCHEDULED` state depends on the time when the last transition to the `RUNNING` state occurred, the current time, and the value of the `period` property, which defines the minimum amount of time between two consequent runs. If the previous execution completed before the period expired, then the `ScheduledService` object stays in the `SCHEDULED` state until the period expires. Otherwise, if the previous execution took longer than the specified period, then the `ScheduledService` object instantly transitions to the `RUNNING` state.

Task fails

In the case when the task terminates in the `FAILED` state, the `ScheduledService` object either restarts or quits, depending on the values for the `restartOnFailure`, `backoffStrategy`, and `maximumFailureCount` properties.

If the `restartOnFailure` property is `false`, then the `ScheduledService` object transitions to the `FAILED` state and quits. In this case, you can restart the failed `ScheduledService` object manually.

If the `restartOnFailure` property is `true`, then the `ScheduledService` object transitions to the `SCHEDULED` state and remains in this state for the duration of `cumulativePeriod` property, which is obtained as a result of calling the `backoffStrategy` property. Using the `cumulativePeriod` property, you can force the failed `ScheduledService` object to wait longer before the next run. After the `ScheduledService` completes successfully, the `cumulativePeriod` property is reset to the value of the `period` property. When the amount of consequent failures reaches the

value of the `maximumFailureCount` property, the `ScheduledService` object transitions to the `FAILED` state and quits.

Any changes that happen to the `delay` and `period` properties while the `ScheduledService` object is running will be taken into account on the next iteration. The default values for the `delay` and `period` properties are set to 0.

Conclusion

In this chapter, you learned the basic capabilities provided by the `javafx.concurrent` package and became familiar with several examples of the `Task` and `Service` classes implementation. For more examples of how to create the Task implementation correctly, see the API documentation for the `Task` class.

Part II

JavaFX-Swing Interoperability

This tutorial provides an overview of JavaFX benefits available to GUI developers, illustrates the JavaFX–Swing interoperability, shows how to enrich an existing Swing application by taking advantage of JavaFX functionality, and how to implement a typical Swing application in JavaFX.

The tutorial contains the following chapters:

- The JavaFX Advantage for Swing Developers
- Integrating JavaFX into Swing Applications
- Enriching Swing Applications with JavaFX Functionality
- Leveraging Applications with Media Features
- Implementing a Swing Application in JavaFX
- Embedding Swing Content in JavaFX Applications

2

The JavaFX Advantage for Swing Developers

JavaFX is designed to provide applications with such sophisticated GUI features as smooth animation, web views, audio and video playback, and styles based on Cascading Style Sheets (CSS).

For more than 10 years, application developers have found Swing to be a highly effective toolkit for building graphical user interfaces (GUIs) and adding interactivity to Java applications. However, some of today's most popular GUI features cannot be easily implemented by using Swing. These features and others described in the following sections can help application developers to meet the full range of modern requirements. Later chapters in this document explain how to use Swing and JavaFX together.

Using FXML

FXML is an XML-based markup language that enables developers to create a user interface (UI) in a JavaFX application separately from implementing the application logic. Swing has never offered a declarative approach to building a user interface. The declarative method for creating a UI is particularly suitable for the scene graph, because the scene graph is more transparent in FXML. Using FXML enables developers to more easily maintain complex user interfaces.

To learn more about the benefits of using FXML, see Mastering FXML.

JavaFX Scene Builder

To help developers build the layout of their applications, JavaFX provides a design tool called the JavaFX Scene Builder. You drag and drop UI components to a JavaFX Content pane, and the tool generates the FXML code that can be used in an IDE such as NetBeans or Eclipse.

For more information, see the Scene Builder documentation.

CSS Support

Cascading style sheets contain style definitions that control the look of UI elements. The usage of CSS in JavaFX applications is similar to the usage of CSS in HTML. With CSS, you can easily customize and develop themes for JavaFX controls and scene graph objects.

Using CSS as opposed to setting inline styles enables you to separate the logic of the application from setting its visual appearance. Using CSS also simplifies further maintenance of how your application looks and provides some performance benefits.

For more information about CSS, see Skinning JavaFX Applications with CSS and JavaFX CSS Reference Guide.

JavaFX Media Support

With the media support provided by the JavaFX platform, you can leverage your desktop application by adding media functionality such as playback of audio and video files. Media functionality is available on all platforms where JavaFX is supported. For the list of supported media codecs, see Introduction to JavaFX Media.

For more details, see the Leveraging Applications with Media Features chapter.

Animation

Animation brings dynamics and a modern look to the interface of your applications. Animating objects in a Swing application is possible but is not straightforward. In the Swing rendering model, painting happens on a double buffer. All alterations of object properties and positions with time are rendered on a double buffer. Only when the painting is completed, is the final result actually painted onto the screen. To show time-based alterations of objects requires significant efforts from a developer using Swing. In contrast, JavaFX enables developers to animate graphical objects in their applications more easily because of the scene graph underlying the platform and the particular APIs that are specifically created for that purpose.

For more details about animation in JavaFX, see Creating Transitions and Timeline Animations. Be sure to check the Tree animation example.

HTML Content

For a long time, Swing developers have wanted the ability to render HTML content in Java applications. JavaFX brought this feature to life by providing a user interface component that has web view and full browsing functionality.

For more details, see Adding HTML Content to JavaFX Applications.

3

Integrating JavaFX into Swing Applications

This chapter describes how to add JavaFX content into a Swing application and how to use threads correctly when both Swing and JavaFX content operate within a single application.

JavaFX SDK provides the `JFXPanel` class, which is located in the `javafx.embed.swing` package and enables you to embed JavaFX content into Swing applications.

Adding JavaFX Content to a Swing Component

For the purpose of this chapter, you create a `JFrame` component, add a `JFXPanel` object to it, and set the graphical scene of the JFXPanel component that contains JavaFX content.

As in any Swing application, you create the graphical user interface (GUI) on an event dispatch thread (EDT). Example 3–1 shows the `initAndShowGUI` method, which creates a `JFrame` component and adds a `JFXPanel` object to it. Creating an instance of the `JFXPanel` class implicitly starts the JavaFX runtime. After the GUI is created, call the `initFX` method to create the JavaFX scene on the JavaFX application thread.

Example 3–1

```
import javafx.application.Platform;
import javafx.embed.swing.JFXPanel;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class Test {

    private static void initAndShowGUI() {
        // This method is invoked on the EDT thread
        JFrame frame = new JFrame("Swing and JavaFX");
        final JFXPanel fxPanel = new JFXPanel();
        frame.add(fxPanel);
        frame.setSize(300, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Platform.runLater(new Runnable() {
            @Override
            public void run() {
```

```
                initFX(fxPanel);
            }
        });
    }

    private static void initFX(JFXPanel fxPanel) {
        // This method is invoked on the JavaFX thread
        Scene scene = createScene();
        fxPanel.setScene(scene);
    }

    private static Scene createScene() {
        Group  root  =  new  Group();
        Scene  scene  =  new  Scene(root, Color.ALICEBLUE);
        Text  text  =  new  Text();

        text.setX(40);
        text.setY(100);
        text.setFont(new Font(25));
        text.setText("Welcome JavaFX!");

        root.getChildren().add(text);

        return (scene);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                initAndShowGUI();
            }
        });
    }
}
```

Swing–JavaFX Interoperability and Threads

With JavaFX and Swing data coexisting in a single application, you may encounter the following interoperability situations:

- A JavaFX data change is triggered by a change in Swing data.
- A Swing data change is triggered by a change in JavaFX data.

Changing JavaFX Data in Response to a Change in Swing Data

JavaFX data should be accessed only on the JavaFX User thread. Whenever you must change JavaFX data, wrap your code into a Runnable object and call the `Platform.runLater` method as shown in Example 3–2.

Example 3–2

```
jbutton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                fxlabel.setText("Swing button clicked!");
```

```
            }
        });
    }
});
```

Changing Swing Data in Response to a Change in JavaFX Data

Swing data should be changed only on the EDT. To ensure that your code is implemented on the EDT, wrap it into a Runnable object and call the `SwingUtilities.invokeLater` method as shown in Example 3–3.

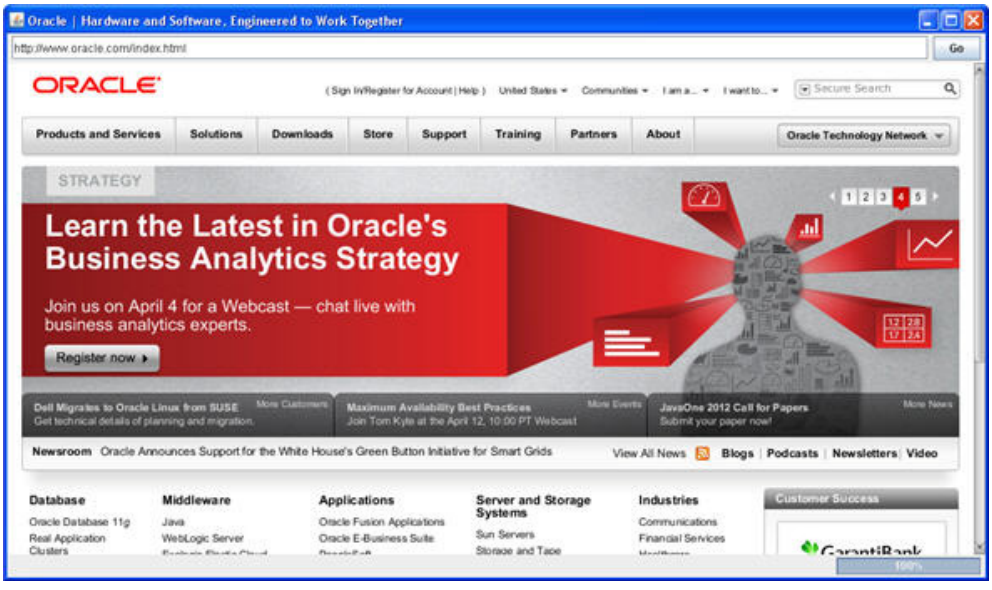
Example 3–3

```
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        //Code to change Swing data.
    }
});
```

Introducing the SimpleSwingBrowser Application

To see how Swing–JavaFX interoperability works, consider the `SimpleSwingBrowser` application. This is a Swing application with an integrated JavaFX component intended to view Web pages. You can type a URL in an address bar and view the page loaded in the application window. The `SimpleSwingBrowser` application window is shown in Figure 3–1.

Figure 3–1 The SimpleSwingBrowser Application Window



Initializing Swing Data

You can view the `SimpleSwingBrowser.java` file or download the `SimpleSwingBrowser.zip` file with a NetBeans project. Extract files from the zip file to a directory on your local file system and run the project in your Netbeans IDE.

As of version 7.2, the NetBeans IDE provides support for Swing applications with the embedded JavaFX content. When creating a new project, in the **JavaFX** category choose **JavaFX in Swing Application**.

Note: To run this application from behind a firewall, you must specify proxy settings in order for the application to access a remote resource.

In the NetBeans IDE, right-click the **SimpleSwingBrowser** project in the Projects window, select **Properties**, and in the Projects Properties dialog, select **Run**.

In the VM Options field, set the proxy in the following format:

-Dhttp.proxyHost=webcache.mydomain.com -Dhttp.proxyPort=8080

The GUI of the `SimpleSwingBrowser` application is created on the EDT when the application starts. The `main` method is implemented as shown in Example 3–4.

Example 3–4

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {

        @Override
        public void run() {
            SimpleSwingBrowser browser = new SimpleSwingBrowser();
            browser.setVisible(true);
            browser.loadURL("http://oracle.com");
        }
    });
}
```

The `SimpleSwingBrowser` class initializes Swing objects and calls the `initComponents` method to create the GUI as shown in Example 3–5.

Example 3–5

```
public class SimpleSwingBrowser extends JFrame {

    private final JFXPanel jfxPanel = new JFXPanel();
    private WebEngine engine;

    private final JPanel panel = new JPanel(new BorderLayout());
    private final JLabel lblStatus = new JLabel();

    private final JButton btnGo = new JButton("Go");
    private final JTextField txtURL = new JTextField();
    private final JProgressBar progressBar = new JProgressBar();

    public SimpleSwingBrowser() {
        super();
        initComponents();
    }

    private void initComponents() {
        createScene();

        ActionListener al = new ActionListener() {
```

```
            @Override
            public void actionPerformed(ActionEvent e) {
                loadURL(txtURL.getText());
            }
        };

        btnGo.addActionListener(al);
        txtURL.addActionListener(al);

        progressBar.setPreferredSize(new Dimension(150, 18));
        progressBar.setStringPainted(true);

        JPanel topBar = new JPanel(new BorderLayout(5, 0));
        topBar.setBorder(BorderFactory.createEmptyBorder(3, 5, 3, 5));
        topBar.add(txtURL, BorderLayout.CENTER);
        topBar.add(btnGo, BorderLayout.EAST);

        JPanel statusBar = new JPanel(new BorderLayout(5, 0));
        statusBar.setBorder(BorderFactory.createEmptyBorder(3, 5, 3, 5));
        statusBar.add(lblStatus, BorderLayout.CENTER);
        statusBar.add(progressBar, BorderLayout.EAST);

        panel.add(topBar, BorderLayout.NORTH);
        panel.add(jfxPanel, BorderLayout.CENTER);
        panel.add(statusBar, BorderLayout.SOUTH);

        getContentPane().add(panel);

        setPreferredSize(new Dimension(1024, 600));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
    }
}
```

The topmost window of this application is a JFrame object, which contains various Swing components such as a text field, a button, a progress bar, and a JFX panel intended to display JavaFX content.

Loading JavaFX Content

On the first run, the web page at http://oracle.com is loaded into a WebView object. As a new URL is entered in the address bar, the action listener, which is attached to the `txtURL` text field in the `initComponents` method, initiates the loading of a page as shown in Example 3–6.

Example 3–6

```
ActionListener al = new ActionListener() {
    @Override public void actionPerformed(ActionEvent e) {
        loadURL(txtURL.getText());
    }
};
```

JavaFX data should only be accessed on the JavaFX application thread. The `loadURL` method wraps the code into a Runnable object and calls the `Platform.runLater` method as shown in Example 3–7.

Example 3–7

```
public void loadURL(final String url) {
    Platform.runLater(new Runnable() {
        @Override public void run() {
            String tmp = toURL(url);

            if (url == null) {
            tmp = toURL("http://" + url);
            }

           engine.load(tmp);
        }
    });
}

private static String toURL(String str) {
    try {
        return new URL(str).toExternalForm();
    } catch (MalformedURLException exception) {
        return null;
    }
}
```

Updating Swing Data

As a new page is loaded into the WebView component, the title of the page is retrieved from the JavaFX data and passed to the Swing GUI to be placed on the application window as a title. This behavior is implemented in the `createScene` method as shown in Example 3–8.

Example 3–8

```
private void createScene() {

    Platform.runLater(new Runnable() {
        @Override
        public void run() {

            WebView view = new WebView();
            engine = view.getEngine();

            engine.titleProperty().addListener(new ChangeListener<String>() {
                @Override
                public void changed(ObservableValue<? extends String> observable,
String oldValue, final String newValue) {
                    SwingUtilities.invokeLater(new Runnable() {
                        @Override
                        public void run() {
                            SimpleSwingBrowser.this.setTitle(newValue);
                        }
                    });
                }
            });
        }
    });
}
```

Application Files

Source Code

- SimpleSwingBrowser.java

NetBeans Projects

SimpleSwingBrowser.zip

4

Enriching Swing Applications with JavaFX Functionality

In this chapter you learn how to intermix a Swing table and JavaFX bar chart in a single application.

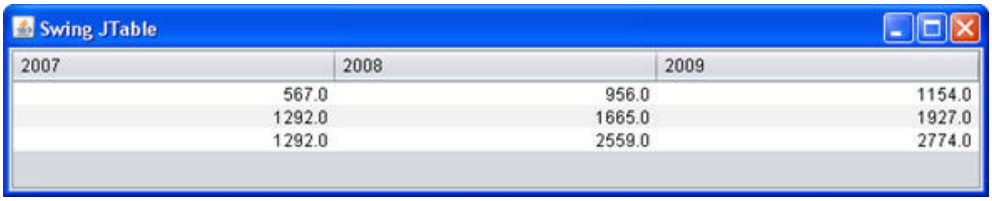
This chapter starts with a Swing application and provides an example of how to enrich the Swing application by adding JavaFX functionality.

Sample Swing Application

Many real-world projects employ Swing applications that deal with tables. You can continue using the existing code and still take an advantage of JavaFX APIs. For example, you can add a JavaFX bar chart to provide a colorful illustration of the tabular data. This chapter provides the `SwingInterop` example that handles a Swing table and a JavaFX bar chart. As you change the data in a table cell, the bar chart immediately updates.

Start with the sample application that has only the Swing table shown in Figure 4–1.

Figure 4–1 Swing JTable Application Window



This application consists of two classes:

- `SampleTableModel.java`
- `SwingInterop.java`

The `SampleTableModel` class inherits from the `AbstractTableModel` class and defines the table.

The `SwingInterop` class inherits from the `JApplet` class and is the basic class of the application. Its `main` method calls the `run` method on the Event Dispatch Thread (EDT) to create the graphical user interface (GUI). The `run` method creates a JFrame object and a JApplet object, and initializes the JApplet object with an instance of the `SwingInterop` class. Then it calls the `init` method, which creates the table and adds the table to the content pane of the applet.

You can see the implementation of both classes by clicking the links above.

Integrating JavaFX Bar Chart

To provide data for a bar chart, modify the `SampleTableModel` class by adding a new class variable (`bcData`) and a method that retrieves data from the table and returns the data in the format appropriate for the bar chart. The implementation of the `getBarChartData` method is shown in Example 4–1.

Example 4–1

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.chart.BarChart;

public class SampleTableModel extends AbstractTableModel {
    private static ObservableList<BarChart.Series> bcData;

    public ObservableList<BarChart.Series> getBarChartData() {
        if (bcData == null) {
            bcData = FXCollections.<BarChart.Series>observableArrayList();
            for (int row = 0; row < getRowCount(); row++) {
                ObservableList<BarChart.Data> series =
FXCollections.<BarChart.Data>observableArrayList();
                for (int column = 0; column < getColumnCount(); column++) {
                    series.add(new BarChart.Data(getColumnName(column),
getValueAt(row, column)));
                }
                bcData.add(new BarChart.Series(series));
            }
        }
        return bcData;
    }
//rest of the SampleTableModel class code
}
```

The `SwingInterop` class overrides the `JApplet.init` method to create the content pane of the application. Modify the `init` method to create a JFXPanel object to hold the JavaFX bar chart and a JSplitPane object to hold both the JavaFX chart and the table. The required changes to the `init` method are shown in bold in Example 4–2.

Example 4–2

```
@Override
public void init() {
    tableModel = new SampleTableModel();
    // create javafx panel for charts
    chartFxPanel = new JFXPanel();
    chartFxPanel.setPreferredSize(new Dimension(PANEL_WIDTH_INT, PANEL_HEIGHT_
INT));

    //create JTable
    JTable table = new JTable(tableModel);
    table.setAutoCreateRowSorter(true);
    table.setGridColor(Color.DARK_GRAY);
    SwingInterop.DecimalFormatRenderer renderer =
new SwingInterop.DecimalFormatRenderer();
    renderer.setHorizontalAlignment(JLabel.RIGHT);
    for (int i = 0; i < table.getColumnCount(); i++) {
        table.getColumnModel().getColumn(i).setCellRenderer(renderer);
```

```
    }
    JScrollPane tablePanel = new JScrollPane(table);
    tablePanel.setPreferredSize(new Dimension(PANEL_WIDTH_INT,
TABLE_PANEL_HEIGHT_INT));
    JPanel chartTablePanel = new JPanel();
    chartTablePanel.setLayout(new BorderLayout());

    //Create split pane that holds both the bar chart and table
    JSplitPane jsplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
    jsplitPane.setTopComponent(chartTablePanel);
    jsplitPane.setBottomComponent(tablePanel);
    jsplitPane.setDividerLocation(410);
    chartTablePanel.add(chartFxPanel, BorderLayout.CENTER);

    //Add the split pane to the content pane of the application
    add(jsplitPane, BorderLayout.CENTER);
}
```

To get rid of a syntax error, add import statements and the definition of the `chartFxPanel` class variable to the `SwingInterop` class as shown in Example 4–3.

Example 4–3

```
import javafx.embed.swing.JFXPanel;
import javax.swing.*;

public class SwingInterop extends JApplet {
    private static JFXPanel chartFxPanel;
// rest of the SwingInterop class code here
}
```

You prepared the UI of your Swing application to render JavaFX data. The next step is creating the JavaFX scene. Because the JavaFX scene must be created on the JavaFX Application thread, wrap your code into a Runnable object as shown in Example 4–4. Add this code at the end of the `init` method.

Example 4–4

```
Platform.runLater(new Runnable() {
    @Override
    public void run() {
        createScene();
    }
});
```

Add the import statement shown in Example 4–5 to the `SwingInterop` class.

Example 4–5

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

Implement the `createScene` method of the `SwingInterop` class as shown in Example 4–6. Add the import statements and define the instance variable `chart`.

Example 4–6

```
import javafx.scene.Scene;
import javafx.scene.chart.Chart;

private void createScene() {
```

```
    chart = createBarChart();
    chartFxPanel.setScene(new Scene(chart));
}
```

The `createBarChart` method creates the chart diagram and adds a change listener to the table. Note that any change of JavaFX data must happen on the JavaFX thread. For this reason, wrap the code in the event handler, which updates the JavaFX chart, into a Runnable object and pass it to the `Platform.runLater` method. The implementation of the `createBarChart` method is shown in Example 4–7.

Example 4–7

```
private BarChart createBarChart() {
    CategoryAxis xAxis = new CategoryAxis();
xAxis.setCategories(FXCollections.<String>observableArrayList(tableModel.
getColumnNames()));
    xAxis.setLabel("Year");
    double tickUnit = tableModel.getTickUnit();

    NumberAxis yAxis = new NumberAxis();
    yAxis.setTickUnit(tickUnit);
    yAxis.setLabel("Units Sold");

    final BarChart chart = new BarChart(xAxis, yAxis,
tableModel.getBarChartData());
    tableModel.addTableModelListener(new TableModelListener() {

        public void tableChanged(TableModelEvent e) {
            if (e.getType() == TableModelEvent.UPDATE) {
                final int row = e.getFirstRow();
                final int column = e.getColumn();
                final Object value =
((SampleTableModel) e.getSource()).getValueAt(row, column);

                Platform.runLater(new Runnable() {
                    public void run() {
                        XYChart.Series<String, Number> s =
(XYChart.Series<String, Number>) chart.getData().get(row);
                        BarChart.Data data = s.getData().get(column);
                        data.setYValue(value);
                    }
                });
            }
        }
    });
    return chart;
}
```

Add the import statements shown in Example 4–8.

Example 4–8

```
import javafx.collections.FXCollections;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
```

Rename the title of the frame to "Swing JTable and Bar Chart" and run the `SwingInterop` application.

The application window is shown in Figure 4–2.

Figure 4–2 SwingInterop Application Window

Application Files

Source Code

- `SwingInterop.java`
- `SampleTableModel.java`

NetBeans Projects

- `SwingInterop.zip`

5

Leveraging Applications with Media Features

In this chapter you review a Media Player application that plays a video file and has controls typical for a video player such as a start/pause button, sliders to show playback progress and adjust volume, and a check box that turns repeat on.

For the purpose of this chapter, get familiar with the `javafx.scene.media` package that enables developers to create media applications.

About Media Integration

Any JavaFX media application can be built using the following key classes:

- `Media` class: Represents a media resource
- `MediaPlayer` class: Provides the controls for playing the specified resource
- `MediaView` class: Provides a view of the media resource played by a MediaPlayer object

Because the `MediaView` class is a subclass of the `Node` class, the MediaView object can be added to a JavaFX scene. This is the principal factor that provides a foundation for integration of the JavaFX media functionality into desktop and web applications. Now that you know how to embed the JavaFX scene into Swing applications, you can further leverage your applications by integrating the Media Player component. You can animate the MediaView object, transform it, and apply effects to it, just as you can with any other node. In this way, you can support numerous creative tasks.

Building the Media Player Application

The Incorporating Media Assets Into JavaFX Applications document provides step-by-step instructions on how to create the `EmbeddedMediaPlayer` application. It also provides the Netbeans project source. Follow the detailed instructions to build the application or download the source project using the link on the sidebar.

The `MediaPlayer` application discussed in this chapter is based on the `EmbeddedMediaPlayer` application but is slightly improved as follows:

- As a best programming practice, the application uses an external CSS file.
- The control bar contains the `Loop` check box to turn repeat on.

The application window is shown in Figure 5–1.

Figure 5–1 Media Player Application Window

You can modify the EmbeddeMediaPlayer project or save its copy with a different name and modify the new project.

Skinning the Application with CSS

To skin the application with CSS, first create the `mediaplayer.css` file and save it in the folder with the source files of your application. Add the style rules shown in Example 5–1.

Example 5–1

```
#mediaControl {
    -fx-background-color: #bfc2c7;
}
#mediaViewPane {
    -fx-background-color: black;;
}
```

Next, open the `MediaControl.java` file and remove from the `MediaControl` constructor the lines shown in Example 5–2.

Example 5–2

```
setStyle("-fx-background-color: #bfc2c7;");
mvPane.setStyle("-fx-background-color: black;");
```

Then modify the `MediaControl` constructor by adding the lines shown in bold in Example 5–3.

Example 5–3

```
public MediaControl(final MediaPlayer mp) {
    this.mp = mp;
    setId("mediaControl");

    mediaView = new MediaView(mp);
    Pane mvPane = new Pane();
    mvPane.getChildren().add(mediaView);
```

```
mvPane.setId("mediaViewPane");
setCenter(mvPane);
```

Adding a New Control to the Control Bar

Adding a new control to the control bar requires only a few steps. In the section where you define the `MediaControl` class instance variables, remove the definition of the `repeat` variable shown in Example 5–4.

Example 5–4

```
private final boolean repeat = false;
```

In the `MediaControl` class, remove the code that used the `repeat` instance variable shown in Example 5–5.

Example 5–5

```
mp.setCycleCount(repeat ? MediaPlayer.INDEFINITE : 1);
```

Now add the class variable `repeatBox` as shown in Example 5–6.

Example 5–6

```
private CheckBox repeatBox;
```

Add a label and the check box to the control bar of your Media Player. Place the following code in the `MediaControl` constructor after the lines that added the `volumeSlider` to the bar, as shown in Example 5–7.

Example 5–7

```
mediaBar.getChildren().add(volumeSlider);

Label repeatLabel = new Label("  Loop: ");
repeatLabel.setPrefWidth(50);
repeatLabel.setMinWidth(25);
mediaBar.getChildren().add(repeatLabel);

repeatBox = new CheckBox();
repeatBox.setSelected(true);
mediaBar.getChildren().add(repeatBox);

setBottom(mediaBar);
```

Implement the logic of using the check box in the `setOnEndOfMedia` method, as shown in Example 5–8.

Example 5–8

```
mp.setOnEndOfMedia(new Runnable() {

    public void run() {
        if (repeatBox.isSelected()) {
            mp.seek(mp.getStartTime());
        } else {
            playButton.setText(">");
            stopRequested = true;
            atEndOfMedia = true;
```

```
        }
    }
});
```

To enable the Media Player access a remote media resource when running from behind a firewall, provide the proxy settings in the following format: `-Dhttp.proxyHost=yourproxyhost.com -Dhttp.proxyPort=portNumber`. In this example, `yourproxyhost.com` is your proxy and `portNumber` is a port number to use.

Application Files

Source Code

- `MediaPlayer.java`
- `MediaControl.java`

NetBeans Projects

- `MediaPlayer.zip`

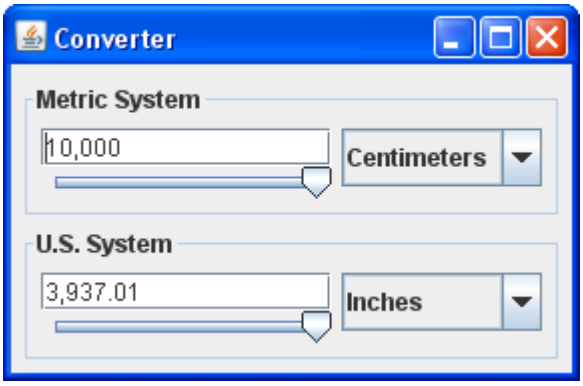
6

Implementing a Swing Application in JavaFX

In this chapter, you consider a Swing application and learn how to implement it in JavaFX.

For the purpose of this chapter, get familiar with the `Converter` application shown in Figure 6–1. This application converts distance measurements between metric and U.S. units.

Figure 6–1 Converter Application in Java



Analyzing the Converter Application Developed in Swing

For more information about the implementation of this example in the Java programming language, see How to Use Panels and Using Models trails in the Swing tutorial. In particular, the graphical user interface (GUI) is discussed in the trail about the panels.

To learn the code of the Converter application, download its NetBeans project or the source files available at the example index.

Swing components use models. If you look at the contents of the project, you notice the `ConverterRangeModel` and `FollowerRangeModel` classes that define models for the Converter application.

The Converter application consists of the following files:

- `ConversionPanel.java` — contains a custom `JPanel` subclass to hold components
- `Converter.java` — contains the main application class
- `ConverterRangeModel.java` — defines the top slider's model

- `FollowerRangeModel.java` — defines the bottom slider's model
- `Units.java` — creates `Unit` objects

Note that the synchronization between each text field and its slider is implemented by event handlers that listen for changes in values.

Planning the Converter Application in JavaFX

The `Converter` application contains two similar panels that hold components such as a text field, slider, and combo box. The panels have titles. The `TitlePane` class from the javafx.scene.control package ideally suits the GUI of the `Converter` application.

In what follows, you will implement the `ConversionPanel` class and add two instances of this class to the graphical scene of the `Converter` application.

First, note that the components within a single `ConversionPanel` object should be synchronized as follows. Whenever you move the knob on the slider, you must update the value in the text field and vice versa: Whenever you change the value in the text field, you must adjust the position of the knob on the slider.

As soon as you choose another value from the combo box, you must update the value of the text field and, hence, the position of the knob on the slider.

Second, note that both `ConversionPanel` objects should be synchronized. As soon as changes happen on one panel, the corresponding components on another panel must be updated.

It is suggested that you implement synchronization between the panels using the `DoubleProperty` object, called `meters`, and listen to changes in the properties of the text fields and combo boxes by creating and registering two `InvalidationListener` objects: `fromMeters` and `toMeters`. Whenever the property of the text field on one panel changes, the `invalidated` method of the attached `InvalidationListener` object is called, which updates the `meters` property. Because the `meters` property changes, the `invalidated` method of the `InvalidationListener` object, attached to the `meters` property, is called, which updates the corresponding text field on another panel.

Similarly, whenever the property of the combo box on one panel changes, the `invalidated` method of the attached `InvalidationListener` object is called, which updates the text field on this panel.

To provide synchronization between the value of the slider and the value of the `meters` object, use bidirectional binding.

For more information about JavaFX properties and binding, see Using JavaFX Properties and Binding.

Creating the Converter Application in JavaFX

Create a new JavaFX project in NetBeans IDE and name it Converter. Copy the `Unit.java` file from the Swing application to the Converter project. Add a new java class to this project and name it `ConversionPanel.java`.

Standard JavaFX Pattern to Create the GUI

Before you start creating the GUI of the `Converter` application in JavaFX, see the standard pattern of GUI creation in Swing applications, as shown in Example 6–1.

Example 6–1

```
public class Converter {
    private void initAndShowGUI() {
        ...
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                initAndShowGUI();
            }
        });
    }
}
```

To map this pattern to JavaFX, you extend the `javafx.application.Application` class, override the `start` method, and call the `main` method, as shown in Example 6–2.

Example 6–2

```
import javafx.application.Application;
import javafx.stage.Stage;

public class Converter extends Application {
    @Override
    public void start(Stage t) {
        ...
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

When you create a new JavaFX project in the NetBeans IDE, this pattern is automatically generated for you. However, it is important that you understand the basic approach to GUI creation in JavaFX, especially if you use a text editor.

Containers and Layouts

In Swing, containers and layout managers are different entities. You create a container, such as a `JPanel` or `JComponent` object, and set a layout manager for this container. You can assign a specific layout manager and write `.add()` in your code or assign none of the layout managers.

In JavaFX, the container itself takes care of laying out its child nodes. You create a specific layout pane, such as a `Vbox`, `FlowPane`, or `TitledPane` object, and then add content to the list of its child nodes using the `.getChildren().add()` methods.

There are several layout container classes in JavaFX, called panes, some of which have their counterparts in Swing, such as the `FlowPane` class in JavaFX and `FlowLayout` class in Swing.

For more information, see Working With Layouts in JavaFX.

UI Controls

JavaFX SDK provides a set of standard UI controls. Some of the UI controls have their counterparts in Swing such as the `Button` class in JavaFX and `JButton` in Swing;

`Slider` in JavaFX and `JSlider` in Swing; and `TextField` in JavaFX and `JTextField` in Swing.

To implement the `Converter` application in JavaFX, you can use the standard UI controls provided by the `TextField`, `Slider`, and `ComboBox` classes.

For more information, see Using JavaFX UI Controls.

Mechanism of Getting Notifications on User Actions and Binding

In Swing, you can register a listener on any component and listen for changes in the component properties, such as size, position, or visibility; or listen for events, such as whether the component gained or lost the keyboard focus; or whether the mouse was clicked, pressed, or released over the component.

In JavaFX, each object has a set of properties for which you can register a listener. The listener is called whenever a value of the property changes.

Note that an object can be registered as a listener for changes in another object's properties. Thus, you can use the binding mechanism to synchronize some properties of two objects.

Creating the ConversionPanel Class

The `ConversionPanel` class is used to hold components: a text field, a slider, and a combo box. When creating the graphical scene of the `Converter` application, you add two instances of the `ConversionPanel` class to the graphical scene. Add the import statement for the `TitledPane` class and extend the `ConversionPanel` class as shown in Example 6–3.

Example 6–3

```
import javafx.scene.control.TitledPane;

public class ConversionPanel extends TitledPane {

}
```

Creating Instance Variables for UI Controls

Add import statements for the `TextField`, `Slider`, `ComboBox` controls and define instance variables for the components as shown in Example 6–4.

Example 6–4

```
import java.text.NumberFormat;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Slider;
import javafx.scene.control.TextField;

private ComboBox<Unit> comboBox;
private Slider slider;
private TextField textField;
```

Creating DoubleProperty and NumberFormat Objects

Add the import statement for the `DoubleProperty` and `NumberFormat` classes and create a `DoubleProperty` object named `meters` as shown in Example 6–5. The `meters` object is used to ensure the synchronization between two `ConversionPanel` objects.

Example 6–5

```
import javafx.beans.property.DoubleProperty;

private DoubleProperty meters;
provate numberFormat;
```

Laying Out the Components

To lay out the text field and the slider, use the `VBox` class. To lay out both of these components and a combo box, use the `HBox` class. Add the import statements for the `ObservableList` class and implement the constructor of the `ConversionPanel` class as shown in Example 6–6.

Example 6–6

```
import javafx.collections.ObservableList;


public ConversionPanel(String title, ObservableList<Unit> units,
DoubleProperty meters) {
    setText(title);
    setCollapsible(false);

    numberFormat = NumberFormat.getNumberInstance();
    numberFormat.setMaximumFractionDigits(2);

    textField = new TextField();
    slider = new Slider(0, MAX, 0);
    comboBox = new ComboBox(units);
    comboBox.setConverter(new StringConverter<Unit>() {

        @Override
        public String toString(Unit t) {
            return t.description;
        }

        @Override
        public Unit fromString(String string) {
            throw new UnsupportedOperationException("Not supported yet.");
        }
    })
    VBox vbox = new VBox(textField, slider);
    HBox hbox = new HBox(vbox, comboBox);
    setContent(hbox);
    this.meters = meters;

    comboBox.getSelectionModel().select(0);
}
```

The last line of code selects a value in the `ComboBox` object.

Creating InvalidationListener Objects

To listen to changes in the properties of the text fields and combo boxes, create the `InvalidationListener` objects `fromMeters` and `toMeters` as shown in Example 6–7.

Example 6–7

```
import javafx.beans.InvalidationListener;

private InvalidationListener fromMeters = t -> {
    if (!textField.isFocused()) {
        textField.setText(numberFormat.format(meters.get() / getMultiplier()));
    }
};

private InvalidationListener toMeters = t -> {
    if (!textField.isFocused()) {
        return;
    try {
        meters.set(numberFormat.parse(textField.getText()).doubleValue() *
getMultiplier());
    } catch (ParseException | Error | RuntimeException ignored) {
    }
};
```

Adding Change Listeners to Controls and Ensuring Synchronization

To provide the synchronization between the text fields and combo boxes, add change listeners as shown in Example 6–8.

Example 6–8

```
meters.addListener(fromMeters);
comboBox.valueProperty().addListener(fromMeters);
textField.textProperty().addListener(toMeters);
fromMeters.invalidated(null);
```

Create a bidirectional binding between the value of the slider and the value of the `meters` object as shown in Example 6–9.

Example 6–9

```
slider.valueProperty().bindBidirectional(meters);
```

When a new value is typed in the text field, the `invalidated` method of the `toMeters` listener is called, which updates the value of the `meters` object.

Creating the Converter Class

Open the `Converter.java` file that was automatically generated by the NetBeans IDE and remove all of the code except for the `main` method. Then, press Ctrl (or Cmd)+Shift+I to correct the import statements.

Defining Instance Variables

Add import statements for the `ObservableList`, `DoubleProperty`, and `SimpleDoubleProperty` classes and create `metricDistances`, `usaDistances`, and `meters` variables of the appropriate types as shown in Example 6–10.

Example 6–10

```
import javafx.beans.property.DoubleProperty;
import javafx.collections.ObservableList;
import javafx.beans.property.SimpleDoubleProperty;

private ObservableList<Unit> metricDistances;
private ObservableList<Unit> usaDistances;
private DoubleProperty meters = new SimpleDoubleProperty(1);
```

Creating the Constructor for the Converter Class

In the constructor for the `Converter` class, create `Unit` objects for the metric and the U.S. distances as shown in Example 6–11. Add the import statement for the `FXCollections` class. Later, you will instantiate two `ConversionPanel` objects with these units.

Example 6–11

```
import javafx.collections.FXCollections;

public Converter() {
metricDistances = FXCollections.observableArrayList(
        new Unit("Centimeters", 0.01),
        new Unit("Meters", 1.0),
        new Unit("Kilometers", 1000.0));

usaDistances = FXCollections.observableArrayList(
        new Unit("Inches", 0.0254),
        new Unit("Feet", 0.305),
        new Unit("Yards", 0.914),
        new Unit("Miles", 1613.0));
}
```

Creating the Graphical Scene

Override the `start` method to create the graphical scene for your `Converter` application. Add two `ConversionPanel` objects to the graphical scene and lay out them vertically. Note that two `ConversionPanel` objects are instantiated with the same `meters` object. Use the `VBox` class as a root container for the graphical scene. Instantiate two `ConversionPanel` objects as shown in Example 6–12.

Example 6–12

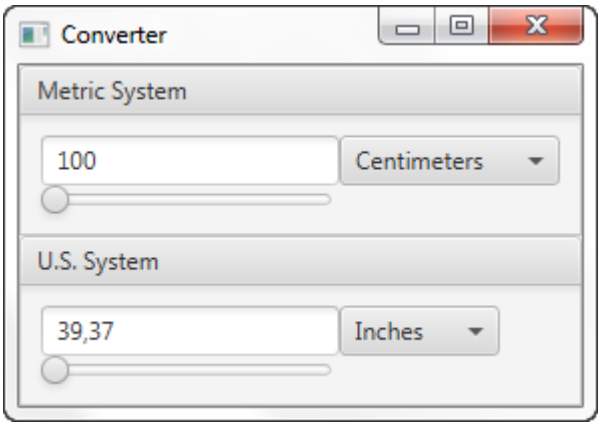
```
@Override
public void start(Stage stage) {
    VBox vbox = new VBox(
            new ConversionPanel(
                    "Metric System", metricDistances, meters),
            new ConversionPanel(
                    "U.S. System", usaDistances, meters));
    Scene scene = new Scene(vbox);

    stage.setTitle("Converter");
    stage.setScene(scene);
    stage.show();
}
```

You can view the source code and download the NetBeans project of the `Converter` application in JavaFX using the links at the bottom of this document.

The Converter application in JavaFX is shown in Figure 6–2.

Figure 6–2 Converter Application in JavaFX



Compare the two applications with the same functionality implemented using the Swing library and JavaFX.

Not only does the application in JavaFX contain three files as compared with five files of the Swing application, but the code in JavaFX is cleaner. The applications also differ in look and feel.

Application Files

Source Code

- `Converter.java`
- `ConversionPanel.java`

NetBeans Projects

- `Converter.zip`

7

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 7–1 illustrates the general pattern of using the `SwingNode` class.

Example 7–1

```
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);
```

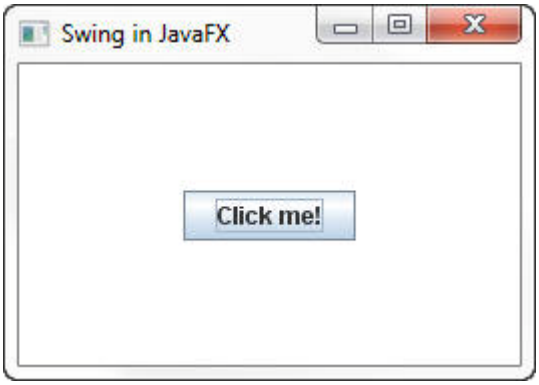
```
        StackPane pane = new StackPane();
        pane.getChildren().add(swingNode);

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

    private void createSwingContent(final SwingNode swingNode) {
        SwingUtilities.invokeLater(() -> {
            swingNode.setContent(new JButton("Click me!"));
        });
    }
}
```

When run, this code produces the output shown in Figure 7–1.

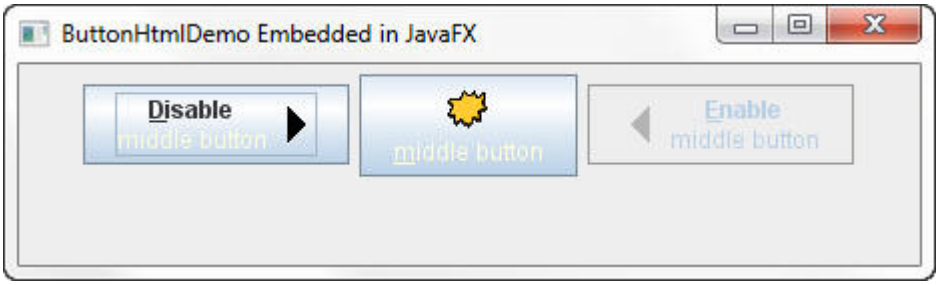
Figure 7–1 Swing JButton Embedded in a JavaFX Application



Embedding Swing Content and Handling Events

The `ButtonHtmlDemo` in the Swing tutorial adds font, color, and other formatting to three buttons shown in Example 7–2 and Example 7–3. The buttons respond to mouse and keyboard events as shown in Example 7–5 and Example 7–6. Figure 7–2 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 7–2 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 7–2.

Example 7–2

```
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 7–3.

Example 7–3

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

All three buttons have the tooltips and mnemonic characters as shown in Example 7–4.

Example 7–4

```
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 7–5.

Example 7–5

```
b1.addActionListener(this);
b3.addActionListener(this);

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

Implement the `actionPerformed` method shown in Example 7–6. This method is called when the user clicks the left or right button.

Example 7–6

```
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 `ButtonHtmlDemo.java` 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 `ButtonHtmlDemo.java` 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` by right clicking the image and selecting Save Image As, 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 7–7.

Example 7–7

```
@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 7–8.

Example 7–8

```
private void createSwingContent(final SwingNode swingNode) {
    SwingUtilities.invokeLater(() -> {
        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 7–3 enables a user to click Swing buttons to disable or enable a JavaFX button. Conversely, the `EnableButtons` application shown in Figure 7–4 enables a user to click a JavaFX button to activate a Swing button.

Figure 7–3 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 7–3 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 `ButtonHtmlDemo.java` 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` by right clicking the image and selecting Save Image As, and save them in the images folder.
10. In the `EnableFXButton` class, declare a `Button` object as shown in Example 7–9.

Example 7–9

```
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 7–10.

Example 7–10

```
@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 7–11.

Example 7–11

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

13. Implement the `createSwingContent` method to set the content of the SwingNode object as shown in Example 7–12.

Example 7–12

```
private void createSwingContent(final SwingNode swingNode) {
    SwingUtilities.invokeLater(() -> {
        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 7–13 to add an image and set a tooltip and style for the JavaFX button.

Example 7–13

```
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 7–14.

Example 7–14

```
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 7–15.

Example 7–15

```
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 7–16. Note that the disabling and enabling of the JavaFX button must happen on the JavaFX application thread.

Example 7–16

```
@Override
public void actionPerformed(ActionEvent e) {
    if ("disable".equals(e.getActionCommand())) {
        Platform.runLater(() -> {
            EnableFXButton.fxbutton.setDisable(true);
        });
        b1.setEnabled(false);
        b3.setEnabled(true);
    } else {
        Platform.runLater(() -> {
            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 7–17.

Example 7–17

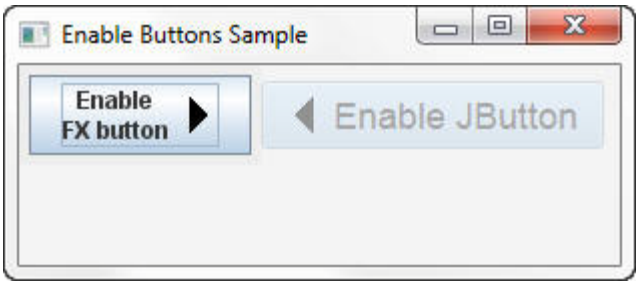
```
import javafx.application.Platform;
```

21. Run the application and click the Swing buttons to disable and enable the JavaFX button, as shown in Figure 7–3.

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 7–4.

Figure 7–4 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 7–18.

Example 7–18

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

5. Modify the initialization of the `fxButtonIcon` variable to use the `left.gif` image as shown in Example 7–19.

Example 7–19

```
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 7–20.

Example 7–20

```
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 by using a lambda expression as shown in Example 7–21. Note that you must change Swing objects on event dispatch thread only.

Example 7–21

```
fxbutton.setOnAction(ActionEvent e) {
    SwingUtilities.invokeLater(() -> {
        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 7–22.

Example 7–22

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

10. Change the application title to "Enable Buttons Sample" as shown in Example 7–23.

Example 7–23

```
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 by using a lambda expression as shown in Example 7–24.

Example 7–24

```
@Override
public void actionPerformed(ActionEvent e) {
    Platform.runLater(() -> {
        EnableButtons.fxbutton.setDisable(false);
    });
    b1.setEnabled(false);
}
```

Conclusion

In this chapter 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.

Application Files

Source Code

- `SwingNodeSample.java`
- `ButtonHtmlDemo.java`

- EnableButtons.java
- EnableFXButton.java

NetBeans Projects

- SwingNodeSample.zip
- EnableButtons.zip
- EnableFXButton.zip

Part III

Interoperability with SWT

This document shows how to add a JavaFX scene graph to a Standard Widget Toolkit (SWT) application, and how to make SWT and JavaFX controls interoperate.

- "Introduction"
- "Adding JavaFX Content to an SWT Component"
- "Creating SWT-JavaFX Applications in an IDE"
- "Packaging SWT-JavaFX Applications"

8

JavaFX Interoperability with SWT

This article shows how to add a JavaFX scene graph to a Standard Widget Toolkit (SWT) application, and how to make SWT and JavaFX controls interoperate.

- "Introduction"
- "Adding JavaFX Content to an SWT Component"
- "Creating SWT-JavaFX Applications in an IDE"
- "Packaging SWT-JavaFX Applications"
- "Application Files"

Introduction

If you develop SWT applications, you know that SWT uses the native operating system controls and cannot easily be configured to use advanced GUI features, such as animation. You can quickly add sparkle to an SWT application by integrating JavaFX with SWT. All you need is the `FXCanvas` class from the `javafx.embed.swt` package. The `javafx.embed.swt` package can be found in `jfxswt.jar`, which is located in the `JDK_Home/jre/lib/` directory. FXCanvas is a regular SWT canvas that can be used anywhere that an SWT canvas can appear. It's that simple.

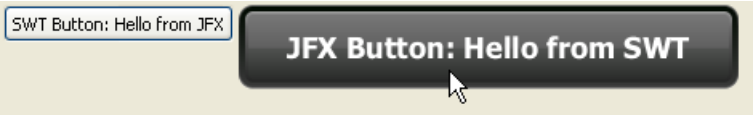
In this article, you will see how to create an interactive SWT button and JavaFX button, shown in Figure 8–1.

Figure 8–1 SWT Button on Left, JavaFX Button on Right

When the user clicks either button, the text is changed in the other button, as shown in Figure 8–2 and Figure 8–3. This example shows how the SWT code and JavaFX code can interoperate.

Figure 8–2 Clicking the SWT Button Changes the JavaFX Button Label

Figure 8–3 Clicking the JavaFX Button Changes the SWT Button Label



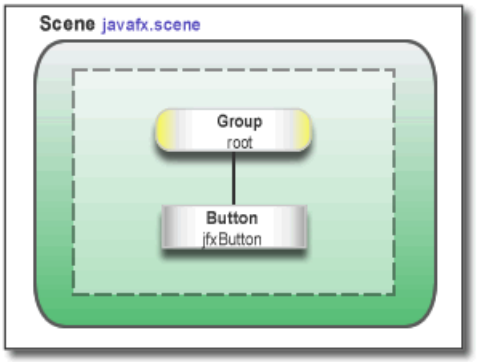
Adding JavaFX Content to an SWT Component

In JavaFX, the Java code that creates and manipulates JavaFX classes runs in the JavaFX User thread. In SWT, code that creates and manipulates SWT widgets runs in the event loop thread. When JavaFX is embedded in SWT, these two threads are the same. This means that there are no restrictions when calling methods defined in one toolkit from the other.

Example 8–1 shows the code to create the SWT button and JavaFX button shown in Figure 8–1. As shown in the code, you set JavaFX content into an FXCanvas with the `setScene()` method in the `FXCanvas` class. To force SWT to lay out the canvas based on the new JavaFX content, resize the JavaFX content first. To do this, get the JavaFX Window that contains the JavaFX content and call `sizeToScene()`. When JavaFX is embedded in SWT, a new preferred size is set for `FXCanvas`, enabling SWT to resize the embedded JFX content in the same manner as other SWT controls.

JavaFX constructs content in terms of a hierarchical scene graph, placed inside a scene. The code in Example 8–1 places the JavaFX button into a scene with the scene graph shown in Figure 8–4 and described in comments in the code example.

Figure 8–4 JavaFX Scene Graph in SWT Application



Example 8–1 Java Code for Plain SWT and JavaFX Buttons

```
import javafx.embed.swt.FXCanvas;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.layout.RowLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
```

```
import org.eclipse.swt.widgets.Shell;

public class TwoButtons {

    public static void main(String[] args) {
        final Display display = new Display();
        final Shell shell = new Shell(display);
        final RowLayout layout = new RowLayout();
        shell.setLayout(layout);

        /* Create the SWT button */
        final org.eclipse.swt.widgets.Button swtButton =
                new org.eclipse.swt.widgets.Button(shell, SWT.PUSH);
        swtButton.setText("SWT Button");

        /* Create an FXCanvas */
        final FXCanvas fxCanvas = new FXCanvas(shell, SWT.NONE) {

                @Override
                public Point computeSize(int wHint, int hHint, boolean changed) {
                getScene().getWindow().sizeToScene();
                int width = (int) getScene().getWidth();
                int height = (int) getScene().getHeight();
                return new Point(width, height);
            }
        };
        /* Create a JavaFX Group node */
        Group group = new Group();
        /* Create a JavaFX button */
        final Button jfxButton = new Button("JFX Button");
        /* Assign the CSS ID ipad-dark-grey */
        jfxButton.setId("ipad-dark-grey");
        /* Add the button as a child of the Group node */
        group.getChildren().add(jfxButton);
        /* Create the Scene instance and set the group node as root */
        Scene scene = new Scene(group, Color.rgb(
                shell.getBackground().getRed(),
                shell.getBackground().getGreen(),
                shell.getBackground().getBlue()));
        /* Attach an external stylesheet */
        scene.getStylesheets().add("twobuttons/Buttons.css");
        fxCanvas.setScene(scene);

        /* Add Listeners */
        swtButton.addListener(SWT.Selection, new Listener() {

            @Override
            public void handleEvent(Event event) {
                jfxButton.setText("JFX Button: Hello from SWT");
                shell.layout();
            }
        });
        jfxButton.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                swtButton.setText("SWT Button: Hello from JFX");
                shell.layout();
            }
        });
```

```
            shell.open();
            while (!shell.isDisposed()) {
                if (!display.readAndDispatch()) {
                    display.sleep();
                }
            }
            display.dispose();
        }
    }
```

The button style is based on a blog by Jasper Potts at the following location:
http://fxexperience.com/2011/12/styling-fx-buttons-with-css/

Creating SWT-JavaFX Applications in an IDE

Creating an SWT-JavaFX application in an IDE is simply a matter of adding the following libraries to your project:

- `swt.jar`, from an SWT zip download, available at http://eclipse.org/swt
- `jfxswt.jar`, from the `JDK_HOME`/jre/lib directory:
  - For example, for a default JDK installation on Windows, the full path is: C:\Program Files\Java\jdk1.8.0\jre\lib

---

Note: Ensure that all JAR files are either 32 bit or 64 bit, as required for your environment.

Packaging SWT-JavaFX Applications

How you package your SWT-JavaFX application depends on whether JavaFX is bundled with the JDK (7u6 and later) or installed in a different location (for releases prior to JDK 7u6).

Packaging the Application when JavaFX is Bundled with the JDK

If you use NetBeans IDE 7.2 or later, no special handling is required to package your application, provided you have added the libraries as described in Creating SWT-JavaFX Applications in an IDE. You can simply do a Clean and Build, which produces a double-clickable JAR file in the /dist directory of the project.

Packaging the Application with a Standalone JavaFX Installation

When an SWT-JavaFX application is built, the JAR file must be packaged as a JavaFX application so the application on startup will look for the standalone JavaFX Runtime on the user's system. The SWT library (swt.jar) must be included as a resource (32-bit or 64-bit to match the target system).

Application Files

NetBeans Projects

- `TwoButtons.zip`

Part IV

Source Code for the Interoperability Tutorial

The following table lists the demo applications in this document with their associated source code files.

Tutorial	Source Code	NetBeans Project File
Integrating JavaFX into Swing	SimpleSwingBrowser.java	SimpleSwingBrowser.zip
Enriching Swing Applications with JavaFX Functionality	SwingInterop.java SampleTableModel.java	SwingInterop.zip
Leveraging Applications with Media Features	MediaControl.java MediaPlayer.java mediaplayer.css	MediaPlayer.zip
Implementing a Swing Application in JavaFX	Converter.java ConversionPanel.java	Converter.zip
Embedding Swing Content in JavaFX	SwingNodeSample.java	SwingNodeSample.zip
	ButtonHtmlDemo.java	EnableFXButton.zip
	EnableFXButton.java	EnableButtons.zip
	EnableButtons.java	
Interoperability with SWT		TwoButtons.zip

A

SimpleSwingBrowser.java

For a description, see Integrating JavaFX into Swing Applications.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package simpleswingbrowser;

import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.embed.swing.JFXPanel;
import javafx.event.EventHandler;
import javafx.scene.Scene;
```

```
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebEvent;
import javafx.scene.web.WebView;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.MalformedURLException;
import java.net.URL;

import static javafx.concurrent.Worker.State.FAILED;

public class SimpleSwingBrowser extends JFrame {

    private final JFXPanel jfxPanel = new JFXPanel();
    private WebEngine engine;

    private final JPanel panel = new JPanel(new BorderLayout());
    private final JLabel lblStatus = new JLabel();

    private final JButton btnGo = new JButton("Go");
    private final JTextField txtURL = new JTextField();
    private final JProgressBar progressBar = new JProgressBar();

    public SimpleSwingBrowser() {
        super();
        initComponents();
    }

    private void initComponents() {
        createScene();

        ActionListener al = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                loadURL(txtURL.getText());
            }
        };

        btnGo.addActionListener(al);
        txtURL.addActionListener(al);

        progressBar.setPreferredSize(new Dimension(150, 18));
        progressBar.setStringPainted(true);

        JPanel topBar = new JPanel(new BorderLayout(5, 0));
        topBar.setBorder(BorderFactory.createEmptyBorder(3, 5, 3, 5));
        topBar.add(txtURL, BorderLayout.CENTER);
        topBar.add(btnGo, BorderLayout.EAST);

        JPanel statusBar = new JPanel(new BorderLayout(5, 0));
        statusBar.setBorder(BorderFactory.createEmptyBorder(3, 5, 3, 5));
        statusBar.add(lblStatus, BorderLayout.CENTER);
        statusBar.add(progressBar, BorderLayout.EAST);

        panel.add(topBar, BorderLayout.NORTH);
        panel.add(jfxPanel, BorderLayout.CENTER);
        panel.add(statusBar, BorderLayout.SOUTH);

        getContentPane().add(panel);
```

```
        setPreferredSize(new Dimension(1024, 600));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
    }

    private void createScene() {

        Platform.runLater(new Runnable() {
            @Override
            public void run() {

                WebView view = new WebView();
                engine = view.getEngine();

                engine.titleProperty().addListener(new ChangeListener<String>() {
                    @Override
                    public void changed(ObservableValue<? extends String>
observable, String oldValue, final String newValue) {
                        SwingUtilities.invokeLater(new Runnable() {
                            @Override
                            public void run() {
                                SimpleSwingBrowser.this.setTitle(newValue);
                            }
                        });
                    }
                });

                engine.setOnStatusChanged(new EventHandler<WebEvent<String>>() {
                    @Override
                    public void handle(final WebEvent<String> event) {
                        SwingUtilities.invokeLater(new Runnable() {
                            @Override
                            public void run() {
                                lblStatus.setText(event.getData());
                            }
                        });
                    }
                });

                engine.locationProperty().addListener(new ChangeListener<String>()
{
                    @Override
                    public void changed(ObservableValue<? extends String> ov,
String oldValue, final String newValue) {
                        SwingUtilities.invokeLater(new Runnable() {
                            @Override
                            public void run() {
                                txtURL.setText(newValue);
                            }
                        });
                    }
                });

                engine.getLoadWorker().workDoneProperty().addListener(new
ChangeListener<Number>() {
                    @Override
                    public void changed(ObservableValue<? extends Number>
observableValue, Number oldValue, final Number newValue) {
                        SwingUtilities.invokeLater(new Runnable() {
```

```
                            @Override
                            public void run() {
                                progressBar.setValue(newValue.intValue());
                            }
                        });
                    }
                });

                engine.getLoadWorker()
                        .exceptionProperty()
                        .addListener(new ChangeListener<Throwable>() {

                            @Override
                            public void changed(ObservableValue<? extends
Throwable> o, Throwable old, final Throwable value) {
                                if (engine.getLoadWorker().getState() == FAILED) {
                                    SwingUtilities.invokeLater(new Runnable() {
                                        @Override
                                        public void run() {
                                            JOptionPane.showMessageDialog(
                                            panel,
                                            (value != null)
                                            ? engine.getLocation() + "\n" +
value.getMessage()
                                            : engine.getLocation() + "\nUnexpected
error.",
                                            "Loading error...",
                                            JOptionPane.ERROR_MESSAGE);
                                        }
                                    });
                                }
                            }
                        });

                jfxPanel.setScene(new Scene(view));
            }
        });
    }

    public void loadURL(final String url) {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                String tmp = toURL(url);

                if (tmp == null) {
                    tmp = toURL("http://" + url);
                }

                engine.load(tmp);
            }
        });
    }

    private static String toURL(String str) {
        try {
            return new URL(str).toExternalForm();
        } catch (MalformedURLException exception) {
            return null;
        }
```

```
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {

            @Override
            public void run() {
                SimpleSwingBrowser browser = new SimpleSwingBrowser();
                browser.setVisible(true);
                browser.loadURL("http://oracle.com");
            }
        });
    }
}
```

B

SwingInterop.java

For a description, see Enriching Swing Applications with JavaFX Functionality.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package swinginterop;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.text.DecimalFormat;
```

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import javafx.embed.swing.JFXPanel;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.Chart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javax.swing.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableCellRenderer;

public class SwingInterop extends JApplet {

    private static final int PANEL_WIDTH_INT = 600;
    private static final int PANEL_HEIGHT_INT = 400;
    private static final int TABLE_PANEL_HEIGHT_INT = 100;
    private static JFXPanel chartFxPanel;
    private static SampleTableModel tableModel;
    private Chart chart;


    @Override
    public void init() {
        tableModel = new SampleTableModel();
        // create javafx panel for charts
        chartFxPanel = new JFXPanel();
        chartFxPanel.setPreferredSize(new Dimension(PANEL_WIDTH_INT, PANEL_HEIGHT_
INT));


        //JTable
        JTable table = new JTable(tableModel);
        table.setAutoCreateRowSorter(true);
        table.setGridColor(Color.DARK_GRAY);
        SwingInterop.DecimalFormatRenderer renderer = new
SwingInterop.DecimalFormatRenderer();
        renderer.setHorizontalAlignment(JLabel.RIGHT);
        for (int i = 0; i < table.getColumnCount(); i++) {
            table.getColumnModel().getColumn(i).setCellRenderer(renderer);
        }
        JScrollPane tablePanel = new JScrollPane(table);
        tablePanel.setPreferredSize(new Dimension(PANEL_WIDTH_INT, TABLE_PANEL_
HEIGHT_INT));

        JPanel chartTablePanel = new JPanel();
        chartTablePanel.setLayout(new BorderLayout());

        //Split pane that holds both chart and table
        JSplitPane jsplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        jsplitPane.setTopComponent(chartTablePanel);
        jsplitPane.setBottomComponent(tablePanel);
        jsplitPane.setDividerLocation(410);
        chartTablePanel.add(chartFxPanel, BorderLayout.CENTER);
```

```
        add(jsplitPane, BorderLayout.CENTER);

        // create JavaFX scene
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                createScene();
            }
        });
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {

            @Override
            public void run() {
                try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
                } catch (Exception e) {}

                JFrame frame = new JFrame("Swing JTable");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                JApplet applet = new SwingInterop();
                applet.init();

                frame.setContentPane(applet.getContentPane());

                frame.pack();
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);

                applet.start();
            }
        });
    }

    private void createScene() {
        chart = createBarChart();
        chartFxPanel.setScene(new Scene(chart));
    }

    private BarChart createBarChart() {
        CategoryAxis xAxis = new CategoryAxis();

xAxis.setCategories(FXCollections.<String>observableArrayList(tableModel.getColumn
Names()));
        xAxis.setLabel("Year");

        double tickUnit = tableModel.getTickUnit();

        NumberAxis yAxis = new NumberAxis();
        yAxis.setTickUnit(tickUnit);
        yAxis.setLabel("Units Sold");

        final BarChart chart = new BarChart(xAxis, yAxis,
tableModel.getBarChartData());
        tableModel.addTableModelListener(new TableModelListener() {
```

```
            @Override
            public void tableChanged(TableModelEvent e) {
                if (e.getType() == TableModelEvent.UPDATE) {
                    final int row = e.getFirstRow();
                    final int column = e.getColumn();
                    final Object value = ((SampleTableModel)
e.getSource()).getValueAt(row, column);

                    Platform.runLater(new Runnable() {
                        @Override
                        public void run() {
                            XYChart.Series<String, Number> s =
(XYChart.Series<String, Number>) chart.getData().get(row);
                            BarChart.Data data = s.getData().get(column);
                            data.setYValue(value);
                        }
                    });
                }
            }
        });
        return chart;
    }


    private static class DecimalFormatRenderer extends DefaultTableCellRenderer {
        private static final DecimalFormat formatter = new DecimalFormat("#.0");

        @Override
        public Component getTableCellRendererComponent(JTable table, Object value,
boolean isSelected, boolean hasFocus, int row, int column) {
            value = formatter.format((Number) value);
            return super.getTableCellRendererComponent(table, value, isSelected,
hasFocus, row, column);
        }
    }
}
```

C

SampleTableModel.java

For a description, see Enriching Swing Applications with JavaFX Functionality.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package swinginterop;

import java.util.Arrays;
import java.util.List;

import javax.swing.table.AbstractTableModel;

import javafx.collections.FXCollections;
```

```
import javafx.collections.ObservableList;
import javafx.scene.chart.BarChart;

/**
 * SampleTableModel
 */
public class SampleTableModel extends AbstractTableModel {
    private static ObservableList<BarChart.Series> bcData;

    private final String[] names = {"2007", "2008", "2009"};

    private final Object[][] data = {
    {new Double(567), new Double(956), new Double(1154)},
            {new Double(1292), new Double(1665), new Double(1927)},
            {new Double(1292), new Double(2559), new Double(2774)}
        };

    public double getTickUnit() {
        return 1000;
    }
    public List<String> getColumnNames() {
        return Arrays.asList(names);
    }

    @Override
    public int getRowCount() {
        return data.length;
    }

    @Override
    public int getColumnCount() {
        return names.length;
    }

    @Override
    public Object getValueAt(int row, int column) {
        return data[row][column];
    }

    @Override
    public String getColumnName(int column) {
        return names[column];
    }

    @Override
    public Class getColumnClass(int column) {
        return getValueAt(0, column).getClass();
    }

    @Override
    public boolean isCellEditable(int row, int column) {
        return true;
    }

    @Override
    public void setValueAt(Object value, int row, int column) {
        if (value instanceof Double) {
            data[row][column] = (Double)value;
        }
```

```
        fireTableCellUpdated(row, column);
    }

    public ObservableList<BarChart.Series> getBarChartData() {
        if (bcData == null) {
            bcData = FXCollections.<BarChart.Series>observableArrayList();
            for (int row = 0; row < getRowCount(); row++) {
                ObservableList<BarChart.Data> series =
FXCollections.<BarChart.Data>observableArrayList();
                for (int column = 0; column < getColumnCount(); column++) {
                    series.add(new BarChart.Data(getColumnName(column),
getValueAt(row, column)));
                }
                bcData.add(new BarChart.Series(series));
            }
        }
        return bcData;
    }
}
```

D

MediaPlayer.java

For a description, see Leveraging Applications with Media Features.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package mediaplayer;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.media.Media;
import javafx.stage.Stage;
```

```
public class MediaPlayer extends Application {
    private static final String MEDIA_URL =
"http://download.oracle.com/otndocs/products/javafx/oow2010-2.flv";
    private static String arg1;

    @Override public void start(Stage stage) {
        stage.setTitle("Media Player");
        Group root = new Group();
        Scene scene = new Scene(root,600,265);
        // create media player
        Media media = new Media((arg1 != null) ? arg1 : MEDIA_URL);
        javafx.scene.media.MediaPlayer mediaPlayer = new
javafx.scene.media.MediaPlayer(media);
        mediaPlayer.setAutoPlay(true);
        MediaControl mediaControl = new MediaControl(mediaPlayer);
        scene.setRoot(mediaControl);

scene.getStylesheets().add(MediaPlayer.class.getResource("mediaplayer.css").toExte
rnalForm());
        // show stage
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        if (args.length > 0) {
            arg1 = args[0];
        }
        Application.launch(args);
    }
}
```

E

MediaControl.java

For a description, see Leveraging Applications with Media Features.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package mediaplayer;

import javafx.application.Platform;
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
```

```
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaPlayer.Status;
import javafx.scene.media.MediaView;
import javafx.util.Duration;

public class MediaControl extends BorderPane {

    private MediaPlayer mp;
    private MediaView mediaView;
    private boolean stopRequested = false;
    private boolean atEndOfMedia = false;
    private Duration duration;
    private Slider timeSlider;
    private final Label playTime;
    private CheckBox repeatBox;
    private Slider volumeSlider;
    private HBox mediaBar;

    @Override
    protected void layoutChildren() {

        if (mediaView != null && getBottom() != null) {
            mediaView.setFitWidth(getWidth());
            mediaView.setFitHeight(getHeight() - getBottom().prefHeight(-1));
        }
        super.layoutChildren();
        if (mediaView != null) {
            mediaView.setTranslateX((((Pane) getCenter()).getWidth() -
mediaView.prefWidth(-1)) / 2);
            mediaView.setTranslateY((((Pane) getCenter()).getHeight() -
mediaView.prefHeight(-1)) / 2);
        }
    }

    @Override
    protected double computeMinWidth(double height) {
        return mediaBar.prefWidth(-1);
    }

    @Override
    protected double computeMinHeight(double width) {
        return 200;
    }

    @Override
    protected double computePrefWidth(double height) {
        return Math.max(mp.getMedia().getWidth(), mediaBar.prefWidth(height));
    }

    @Override
    protected double computePrefHeight(double width) {
```

```
            return mp.getMedia().getHeight() + mediaBar.prefHeight(width);
        }

        @Override
        protected double computeMaxWidth(double height) {
            return Double.MAX_VALUE;
        }

        @Override
        protected double computeMaxHeight(double width) {
            return Double.MAX_VALUE;
        }

        public MediaControl(final MediaPlayer mp) {
            this.mp = mp;
            setId("mediaControl");

            mediaView = new MediaView(mp);
            Pane mvPane = new Pane();
            mvPane.getChildren().add(mediaView);
            mvPane.setId("mediaViewPane");
            setCenter(mvPane);

            mediaBar = new HBox();
            mediaBar.setAlignment(Pos.CENTER);
            mediaBar.setPadding(new Insets(5, 10, 5, 10));
            BorderPane.setAlignment(mediaBar, Pos.CENTER);

            final Button playButton = new Button(">");
            playButton.setOnAction(new EventHandler<ActionEvent>() {

                @Override
                public void handle(ActionEvent e) {
                    updateValues();
                    Status status = mp.getStatus();

                    if (status == Status.UNKNOWN
                            || status == Status.HALTED) {
                        System.out.println("Player is in a bad or unknown state, can't
play.");
                        return;
                    }

                    if (status == Status.PAUSED
                            || status == Status.READY
                            || status == Status.STOPPED) {
                        // rewind the movie if we're sitting at the end
                        if (atEndOfMedia) {
                            mp.seek(mp.getStartTime());
                            atEndOfMedia = false;
                            playButton.setText(">");
                            updateValues();
                        }
                        mp.play();
                        playButton.setText("||");
                    } else {
                        mp.pause();
                    }
                }
            });
```

```
mp.currentTimeProperty().addListener(new InvalidationListener() {

    @Override
    public void invalidated(Observable ov) {
        updateValues();
    }
});

mp.setOnPlaying(new Runnable() {

    @Override
    public void run() {
        if (stopRequested) {
            mp.pause();
            stopRequested = false;
        } else {
            playButton.setText("||");
        }
    }
});

mp.setOnPaused(new Runnable() {

    @Override
    public void run() {
        playButton.setText(">");
    }
});

mp.setOnReady(new Runnable() {

    @Override
    public void run() {
        duration = mp.getMedia().getDuration();
        updateValues();
    }
});

mp.setOnEndOfMedia(new Runnable() {

    @Override
    public void run() {
       if (repeatBox.isSelected()) {
            mp.seek(mp.getStartTime());
        } else {
            playButton.setText(">");
            stopRequested = true;
            atEndOfMedia = true;


        }
    }
});

mediaBar.getChildren().add(playButton);
// Add spacer
Label spacer = new Label("   ");
spacer.setMaxWidth(Double.MAX_VALUE);
mediaBar.getChildren().add(spacer);
```

```
        Label timeLabel = new Label("Time: ");
        timeLabel.setMinWidth(Control.USE_PREF_SIZE);
        mediaBar.getChildren().add(timeLabel);

        timeSlider = new Slider();
        timeSlider.setMaxWidth(Double.MAX_VALUE);
        timeSlider.setMinWidth(50);
        HBox.setHgrow(timeSlider, Priority.ALWAYS);
        timeSlider.valueProperty().addListener(new InvalidationListener() {

            @Override
            public void invalidated(Observable ov) {
                if (timeSlider.isValueChanging()) {
                    // multiply duration by percentage calculated by slider
position
                    if (duration != null) {
                        mp.seek(duration.multiply(timeSlider.getValue() / 100.0));
                    }
                    updateValues();
                }
            }
        });
        mediaBar.getChildren().add(timeSlider);

        playTime = new Label();
        playTime.setPrefWidth(130);
        playTime.setMinWidth(50);
        mediaBar.getChildren().add(playTime);

        Label volumeLabel = new Label("Vol: ");
        volumeLabel.setMinWidth(Control.USE_PREF_SIZE);
        mediaBar.getChildren().add(volumeLabel);

        volumeSlider = new Slider();
        volumeSlider.setPrefWidth(70);
        volumeSlider.setMaxWidth(Region.USE_PREF_SIZE);
        volumeSlider.setMinWidth(30);
        volumeSlider.valueProperty().addListener(new ChangeListener<Number>() {

            @Override
            public void changed(ObservableValue<? extends Number> observable,
Number oldValue, Number newValue) {
                if (volumeSlider.isValueChanging()) {
                    mp.setVolume(volumeSlider.getValue() / 100.0);
                }
            }
        });
        mediaBar.getChildren().add(volumeSlider);

        Label repeatLabel = new Label("  Loop: ");
        repeatLabel.setPrefWidth(50);
        repeatLabel.setMinWidth(25);
        mediaBar.getChildren().add(repeatLabel);

        repeatBox = new CheckBox();
        repeatBox.setSelected(true);
        mediaBar.getChildren().add(repeatBox);

        setBottom(mediaBar);
```

```
    }

    protected void updateValues() {
        if (playTime != null && timeSlider != null && volumeSlider != null) {
            Platform.runLater(new Runnable() {

                @Override
                public void run() {
                    Duration currentTime = mp.getCurrentTime();
                    playTime.setText(formatTime(currentTime, duration));
                    timeSlider.setDisable(duration.isUnknown());
                    if (!timeSlider.isDisabled() &&
duration.greaterThan(Duration.ZERO) && !timeSlider.isValueChanging()) {

timeSlider.setValue(currentTime.divide(duration.toMillis()).toMillis() * 100.0);
                    }
                    if (!volumeSlider.isValueChanging()) {
                        volumeSlider.setValue((int) Math.round(mp.getVolume() *
100));
                    }
                }
            });
        }
    }

    private static String formatTime(Duration elapsed, Duration duration) {
        int intElapsed = (int) Math.floor(elapsed.toSeconds());
        int elapsedHours = intElapsed / (60 * 60);
        if (elapsedHours > 0) {
            intElapsed -= elapsedHours * 60 * 60;
        }
        int elapsedMinutes = intElapsed / 60;
        int elapsedSeconds = intElapsed - elapsedHours * 60 * 60 - elapsedMinutes
* 60;

        if (duration.greaterThan(Duration.ZERO)) {
            int intDuration = (int) Math.floor(duration.toSeconds());
            int durationHours = intDuration / (60 * 60);
            if (durationHours > 0) {
                intDuration -= durationHours * 60 * 60;
            }
            int durationMinutes = intDuration / 60;
            int durationSeconds = intDuration - durationHours * 60 * 60 -
durationMinutes * 60;

            if (durationHours > 0) {
                return String.format("%d:%02d:%02d/%d:%02d:%02d",
                        elapsedHours, elapsedMinutes, elapsedSeconds,
                        durationHours, durationMinutes, durationSeconds);
            } else {
                return String.format("%02d:%02d/%02d:%02d",
                        elapsedMinutes, elapsedSeconds,
                        durationMinutes, durationSeconds);
            }
        } else {
            if (elapsedHours > 0) {
                return String.format("%d:%02d:%02d",
                        elapsedHours, elapsedMinutes, elapsedSeconds);
            } else {
                return String.format("%02d:%02d",
```

```
                                elapsedMinutes, elapsedSeconds);
                    }
                }
            }
        }
```

F

mediaplayer.css

For a description, see Leveraging Applications with Media Features.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
#mediaControl {
    -fx-background-color: #bfc2c7;
}
#mediaViewPane {
    -fx-background-color: black;;
}
```

G

Converter.java

For a description, see Implementing a Swing Application in JavaFX.

Legal Terms and Copyright Notice

Code

```
package converter;


import javafx.application.Application;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
```

```
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;



public class Converter extends Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

    private final ObservableList<Unit> metricDistances;
    private final ObservableList<Unit> usaDistances;
    private final DoubleProperty meters = new SimpleDoubleProperty(1);


    public Converter() {
        //Create Unit objects for metric distances, and then
        //instantiate a ConversionPanel with these Units.
        metricDistances = FXCollections.observableArrayList(
                new Unit("Centimeters", 0.01),
                new Unit("Meters", 1.0),
                new Unit("Kilometers", 1000.0));

        //Create Unit objects for U.S. distances, and then
        //instantiate a ConversionPanel with these Units.
        usaDistances = FXCollections.observableArrayList(
                new Unit("Inches", 0.0254),
                new Unit("Feet", 0.305),
                new Unit("Yards", 0.914),
                new Unit("Miles", 1613.0));
    }

    @Override
    public void start(Stage stage) {
        VBox vbox = new VBox(
                new ConversionPanel(
                        "Metric System", metricDistances, meters),
                new ConversionPanel(
                        "U.S. System", usaDistances, meters));
        Scene scene = new Scene(vbox);

        stage.setTitle("Converter");
        stage.setScene(scene);
        stage.show();
    }
}
```

H

ConversionPanel.java

For a description, see Implementing a Swing Application in JavaFX.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package converter;


import java.text.NumberFormat;
import java.text.ParseException;
import javafx.beans.InvalidationListener;

import javafx.beans.property.DoubleProperty;
```

```
import javafx.collections.ObservableList;
import javafx.scene.control.ComboBox;

import javafx.scene.control.Slider;

import javafx.scene.control.TextField;

import javafx.scene.control.TitledPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

import javafx.util.StringConverter;


public class ConversionPanel extends TitledPane {

    final static int MAX = 10000;

    private ComboBox<Unit> comboBox;
    private Slider slider;
    private TextField textField;
    private DoubleProperty meters;
    private NumberFormat numberFormat;

    private InvalidationListener fromMeters = t -> {
        if (!textField.isFocused()) {
            textField.setText(numberFormat.format(meters.get() /
getMultiplier()));
        }
    };

    private InvalidationListener toMeters = t -> {
        if (!textField.isFocused()) {
            return;
        }
        try {
            meters.set(numberFormat.parse(textField.getText()).doubleValue() *
getMultiplier());
            } catch (ParseException | Error | RuntimeException ignored) {
            }
    };

    public ConversionPanel(String title, ObservableList<Unit> units,
DoubleProperty meters) {
        setText(title);
        setCollapsible(false);

        numberFormat = NumberFormat.getNumberInstance();
        numberFormat.setMaximumFractionDigits(2);

        textField = new TextField();
        slider = new Slider(0, MAX, 0);
        comboBox = new ComboBox(units);
        comboBox.setConverter(new StringConverter<Unit>() {

            @Override
            public String toString(Unit t) {
                return t.description;
            }
```

```
            @Override
            public Unit fromString(String string) {
                throw new UnsupportedOperationException("Not supported yet.");
            }
        });
        VBox vbox = new VBox(textField, slider);
        HBox hbox = new HBox(vbox, comboBox);
        setContent(hbox);
        this.meters = meters;

        comboBox.getSelectionModel().select(0);
        meters.addListener(fromMeters);
        comboBox.valueProperty().addListener(fromMeters);
        textField.textProperty().addListener(toMeters);
        fromMeters.invalidated(null);

        slider.valueProperty().bindBidirectional(meters);
    }

    /**
     * Returns the multiplier (units/meter) for the currently
     * selected unit of measurement.
     * @return
     */
    public double getMultiplier() {
        return comboBox.getValue().multiplier;
    }
}
```

SwingNodeSample.java

For a description, see Embedding Swing Content in JavaFX Applications.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package swingnodesample;

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.SwingUtilities;
```

```
public class SwingNodeSample extends Application {


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

    @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();
    }

    private void createSwingContent(final SwingNode swingNode) {
        SwingUtilities.invokeLater(() -> {
            swingNode.setContent(new ButtonHtml());
        });
    }
}
```

J

ButtonHtmlDemo.java

For a description, see Embedding Swing Content in JavaFX Applications.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package swingnodesample;

import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.AbstractButton;
```

```
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.ImageIcon;

/*
 * ButtonHtmlDemo.java uses the following files:
 *   images/right.gif
 *   images/middle.gif
 *   images/left.gif
 */
public class ButtonHtml extends JPanel
                         implements ActionListener {
    protected JButton b1, b2, b3;

    public ButtonHtml() {
        ImageIcon leftButtonIcon = createImageIcon("images/right.gif");
        ImageIcon middleButtonIcon = createImageIcon("images/middle.gif");
        ImageIcon rightButtonIcon = createImageIcon("images/left.gif");

        b1 = new JButton("<html><center><b><u>D</u>isable</b><br>"
                         + "<font color=#ffffdd>middle button</font>",
                         leftButtonIcon);
        Font font = b1.getFont().deriveFont(Font.PLAIN);
        b1.setFont(font);
        b1.setVerticalTextPosition(AbstractButton.CENTER);
        b1.setHorizontalTextPosition(AbstractButton.LEADING); //aka LEFT, for
left-to-right locales
        b1.setMnemonic(KeyEvent.VK_D);
        b1.setActionCommand("disable");

        b2 = new JButton("middle button", middleButtonIcon);
        b2.setFont(font);
        b2.setForeground(new Color(0xffffdd));
        b2.setVerticalTextPosition(AbstractButton.BOTTOM);
        b2.setHorizontalTextPosition(AbstractButton.CENTER);
        b2.setMnemonic(KeyEvent.VK_M);

        b3 = new JButton("<html><center><b><u>E</u>nable</b><br>"
                         + "<font color=#ffffdd>middle button</font>",
                         rightButtonIcon);
        b3.setFont(font);
        //Use the default text position of CENTER, TRAILING (RIGHT).
        b3.setMnemonic(KeyEvent.VK_E);
        b3.setActionCommand("enable");
        b3.setEnabled(false);

        //Listen for actions on buttons 1 and 3.
        b1.addActionListener(this);
        b3.addActionListener(this);

        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.");

        //Add Components to this container, using the default FlowLayout.
        add(b1);
        add(b2);
        add(b3);
    }
```

```
    @Override
    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);
        }
    }

    /** Returns an ImageIcon, or null if the path was invalid.
     * @param path
     * @return  */
    protected static ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = ButtonHtml.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }

}
```

K

EnableFXButton.java

For a description, see Embedding Swing Content in JavaFX Applications.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package enablefxbutton;

import javafx.application.Application;
import javafx.embed.swing.SwingNode;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Tooltip;
import javafx.scene.image.Image;
```

```
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import javax.swing.SwingUtilities;


public class EnableFXButton extends Application {
    public static Button fxbutton;

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


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

        createSwingContent(swingNode);
        BorderPane pane = new BorderPane();

        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;");
        pane.setTop(swingNode);
        pane.setCenter(fxbutton);


        Scene scene = new Scene(pane, 300, 100);
        stage.setScene(scene);
        stage.setTitle("Enable FX Button");
        stage.show();
    }

    private void createSwingContent(final SwingNode swingNode) {
        SwingUtilities.invokeLater(() -> {
            swingNode.setContent(new ButtonHtml());
        });
    }
}
```

L

EnableButtons.java

For a description, see Embedding Swing Content in JavaFX Applications.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

Code

```
package enablebuttons;

import javafx.application.Application;
import javafx.embed.swing.SwingNode;
import javafx.event.ActionEvent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Tooltip;
```

```
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
import javax.swing.SwingUtilities;


public class EnableButtons extends Application {
    public static Button fxbutton;

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


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

        FlowPane pane = new FlowPane();

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

        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);

        fxbutton.setOnAction((ActionEvent e) -> {
            SwingUtilities.invokeLater(() -> {
                ButtonHtml.b1.setEnabled(true);
            });
            fxbutton.setDisable(true);
        });

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

        Scene scene = new Scene(pane, 300, 100);

        stage.setScene(scene);
        stage.setTitle("Enable Buttons Sample");
        stage.show();
    }

    private void createSwingContent(final SwingNode swingNode) {
        SwingUtilities.invokeLater(() -> {
            swingNode.setContent(new ButtonHtml());
        });
    }
}
```

M

Image Source Files

This appendix provides graphical images used in the Embedding Swing Content in JavaFX Applications.

Legal Terms and Copyright Notice

```
/*
 * Copyright (c) 1995, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

left.gif

right.gif

down.gif

middle.gif