JavaFX

Adding HTML Content to JavaFX Applications

Release 8

E51956-01

March 2014

This tutorial introduces the WebView component and HTML5 features that it supports, teaches how to embed WebView into JavaFX application, and provides instructions to enable basic browsing functionality.

JavaFX Adding HTML Content to JavaFX Applications Release 8

E51956-01

Primary Author: Alla Redko

Contributor: Peter Zhelezniakov

Contents

Preface

This preface describes the document accessibility features and conventions used in this tutorial - *Adding HTML Content to JavaFX Applications*.

About This Tutorial

This tutorial introduces the JavaFX embedded browser, a user interface component that provides a web viewer and full browsing functionality through its API. The document contains the following chapters:

- What Is New

  Describes the new and changed features in the current release.
- Overview of the JavaFX WebView Component

  Lists the basic features of the `WebView` component and introduces classes of the `javafx.scene.web` package.
- Supported Features of HTML5

  Describes the HTML5 features supported by the `WebView` component.
- Adding a WebView Component to the Application Scene

  Provides instructions on how to embed a browser based in the WebView component into the application scene.
- Processing JavaScript Commands

  Explains how to run a particular JavaScript command for the document currently loaded into the browser.
- Making Upcalls from JavaScript to JavaFX

  Provides instructions on how to implement calling from web content to JavaFX application.
- Managing Web Pop-Up Windows

  Teaches how to use the `PopupFeatures` class to set an alternative `WebView` object for the documents that will be opened in a separate window.
- Managing Web History

  Explains how to obtain the list of visited pages by using the `WebHistory` class.
- Printing HTML Content

  Provides a code pattern for printing HTML content of the embedded browser.

This tutorial provides the WebViewSample application so that you better learn the features described in each chapter. By the end of your study, you will have the complete version of the WebViewSample application with all functional code fragment integrated.

You can also find the source files of the application and the corresponding NetBeans project in Appendix A.

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*
- *Working With JavaFX UI Components*
- *Working With Layouts in JavaFX*
- *Mastering FXML*
- *Handling Events*

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.

1 What Is New

This chapter introduces enhancements and additions available in the WebView component with JavaFX 8.

The following changes were made since the previous releases of JavaFX

- Support for additional HTML5 features including Web Sockets, Web Workers, and Web Fonts. See Supported Features of HTML5 for more information.
- Printing capabilities. Lean how to print HTML page loaded in the embedded browser in Printing HTML Content.

2 Overview of the JavaFX WebView Component

This chapter introduces the JavaFX embedded browser, a user interface component that provides a web viewer and full browsing functionality through its API.

The embedded browser component is based on WebKit, an open source web browser engine. It supports Cascading Style Sheets (CSS), JavaScript, Document Object Model (DOM), and HTML5.

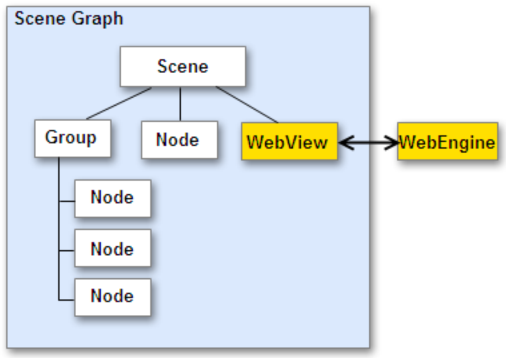
The embedded browser enables you to perform the following tasks in your JavaFX applications:

- Render HTML content from local and remote URLs
- Obtain Web history
- Execute JavaScript commands
- Perform upcalls from JavaScript to JavaFX
- Manage web pop-up windows
- Apply effects to the embedded browser

The embedded browser inherits all fields and methods from the `Node` class, and therefore, it has all its features.

The classes that constitute the embedded browser reside in the `javafx.scene.web` package. Figure 2–1 shows the architecture of the embedded browser and how it relates to other JavaFX classes.

Figure 2–1 Architecture of the Embedded Browser



2.1 WebEngine Class

The `WebEngine` class provides basic web page functionality. It supports user interaction such as navigating links and submitting HTML forms, although it does not interact with users directly. The `WebEngine` class handles one web page at a time. It supports the basic browsing features of loading HTML content and accessing the DOM as well as executing JavaScript commands.

Two constructors enable you to create a `WebEngine` object: an empty constructor and a constructor with the specified URL. If you instantiate an empty constructor, the URL can be passed to a `WebEngine` object through the `load` method.

Starting JavaFX SDK 2.2, developers can enable and disable JavaScript calls for a particular web engine and apply custom style sheets. User style sheets replace the default styles on the pages rendered in this `WebEngine` instance with user-defined ones.

2.2 WebView Class

The `WebView` class is an extension of the `Node` class. It encapsulates a `WebEngine` object, incorporates HTML content into an application's scene, and provides properties and methods to apply effects and transformations. The `getEngine()` method called on a `WebView` object returns a web engine associated with it.

Example 2–1 shows the typical way to create `WebView` and `WebEngine` objects in your application.

Example 2–1 Creating WebView and WebEngine Objects

```
WebView browser = new WebView();
WebEngine webEngine = browser.getEngine();
webEngine.load("http://mySite.com");
```

2.3 PopupFeatures Class

The `PopupFeatures` class describes the features of a web pop-up window as defined by the JavaScript specification. When you need to open a new browser window in your application, the instances of this class are passed into pop-up handlers registered on a `WebEngine` object by using the `setCreatePopupHandler` method as shown in Example 2–2.

Example 2–2 Creating a Pop-Up Handler

```
webEngine.setCreatePopupHandler(new Callback<PopupFeatures, WebEngine>() {
    @Override public WebEngine call(PopupFeatures config) {
        // do something
        // return a web engine for the new browser window
    }
});
```

If the method returns the web engine of the same `WebView` object, the target document is opened in the same browser window. To open the target document in another window, specify the `WebEngine` object of another web view. When you need to block the pop-up windows, return the `null` value.

2.4 Other Features

When working with the `WebView` component, you should remember that it has the default in-memory cache. It means that any cached content is lost once the application containing the `WebView` component is closed. However, developers can implement cache at the application level by means of the `java.net.ResponseCache` class. From WebKit perspectives, the persistent cache is a property of the network layer similar to connection and cookie handlers. Once some of those are installed, the `WebView` component uses them in transparent manner.

3 Supported Features of HTML5

This chapter describes the scope of HTML5 features supported by the JavaFX web component. The majority of the supported functionally is part of the `WebEngine` class and `WebView` class implementations, and this functionality does not have any public APIs.

The current implementation of the JavaFX web component provides support for the following HTML5 features:

- Canvas and SVG
- Media playback
- Form controls
- History maintenance
- Interactive element tags
- DOM
- Web workers
- Web sockets
- Web fonts

3.1 Canvas and SVG

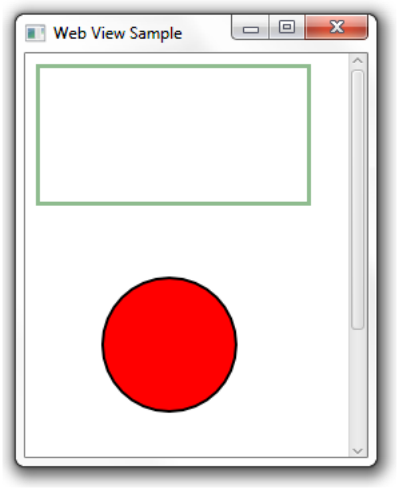
Support for the `canvas` and `svg` element tags enables basic graphical functionality including rendering graphics, building shapes by using Scalable Vector Graphics (SVG) syntax, and applying color settings, visual effects, and animations. Example 3–1 provides a simple test of using the `<canvas>` and `<svg>` tags to render the web component.

Example 3–1 Use of Canvas and SVG Elements

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Canvas and SVG</title>
      <canvas style="border:3px solid darkseagreen;" width="200" height="100">
        </canvas>
        <svg>
            <circle cx="100" cy="100" r="50" stroke="black"
                stroke-width="2" fill="red"/>
        </svg>
    </body>
</html>
```

When you load a page by using the HTML code from Example 3–1 into a WebViewSample application, it renders the graphics shown in Figure 3–1.

Figure 3–1 Rendering Graphics



3.2 Media Playback

The `WebView` component enables you to play video and audio content within a loaded HTML page. The following codecs are supported:

- Audio: AIFF, WAV(PCM), MP3, and AAC
- Video: VP6, H264
- Media containers: FLV, FXM, MP4, and MpegTS (HLS)

For more information about embedding media content, see the HTML5 specification for the `video` and `audio` tags.

3.3 Form Controls

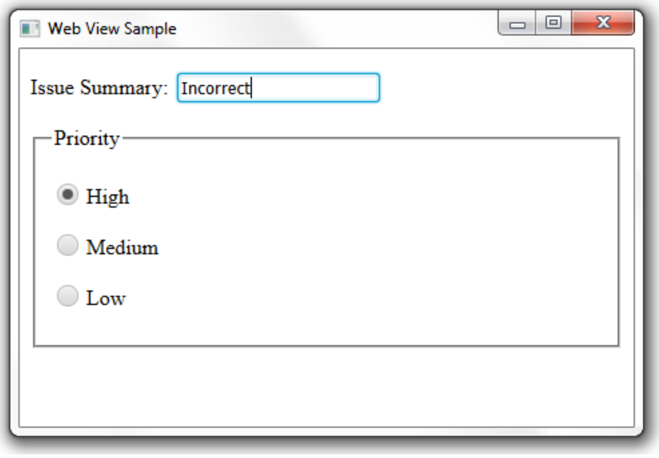
The JavaFX web component enables rendering forms and processing data input. The supported form controls include text fields, buttons, checkboxes, and other available input controls. Example 3–2 provides a simple set of the controls that enable you to enter an issue summary and specify its priority.

Example 3–2 Form Input Controls

```
<!DOCTYPE HTML>
<html>
<form>
 <p><label>Login: <input></label></p>
 <fieldset>
  <legend> Priority </legend>
  <p><label> <input type=radio name=size> High </label></p>
  <p><label> <input type=radio name=size> Medium </label></p>
  <p><label> <input type=radio name=size> Low </label></p>
 </fieldset>
</form>
</html>
```

When the HTML content from Example 3–2 is uploaded in the `WebView` component, it produces the output shown in Figure 3–2.

Figure 3–2 Rendering Form Elements



For more information about how users can submit data and process it using the form controls, see the HTML5 specification.

3.4 History Maintenance

You can obtain a list of visited pages by using the `WebHistory` class available in the `javafx.scene.web` package. The `WebHistory` class represents the session history associated with a `WebEngine` object.

This functionality is enabled in the WebViewSample application that you will use to learn the capabilities of the JavaFX web component. See the Managing Web History chapter for the implementation details.

3.5 Support for Interactive Element Tags

The `WebView` component provides support for interactive HTML5 elements such as `details`, `summary`, `command`, and `menu`. Example 3–3 shows how the `details` and `summary` elements can be rendered in the web component. The sample also uses the `progress` and `meter` control elements.

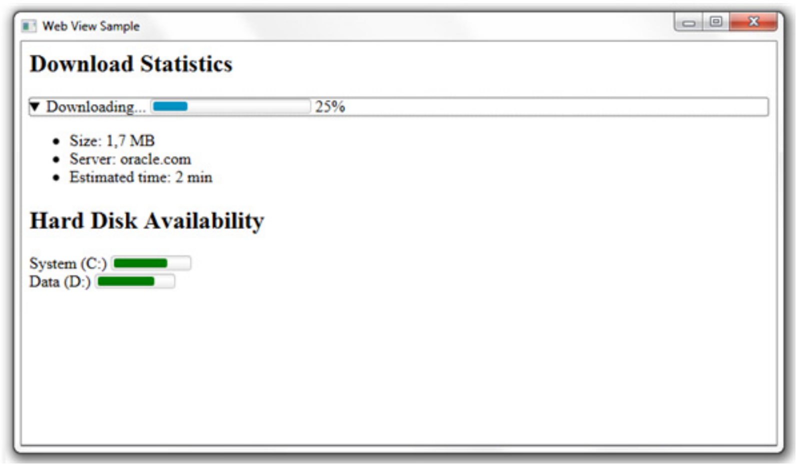
Example 3–3 Use of Details, Summary, Progress, and Meter Elements

```
<!DOCTYPE HTML>
<html>
<h1>Download Statistics</h1>
 <details>
  <summary>Downloading... <progress max="100" value="25"></progress> 25%</summary>
  <ul>
   <li>Size: 1,7 MB</li>
   <li>Server: oracle.com</li>
   <li>Estimated time: 2 min</li>
  </ul>
 </details>
<h1>Hard Disk Availability</h1>
```

```
  System (C:) <meter value=240 max=326></meter> </br>
  Data (D:) <meter value=101 max=130></meter>
</html>
```

When this page is loaded in the web component, the WebViewSample application looks as shown in Figure 3–3.

Figure 3–3 Rendering Interactive HTML5 Elements



Download Statistics

▼ Downloading... 25%

- Size: 1,7 MB
- Server: oracle.com
- Estimated time: 2 min

Hard Disk Availability

System (C:)
Data (D:)

See the HTML5 specification for more information about properties of the interactive elements.

3.6 Document Object Model

A `WebEngine` object, a nonvisual part of the JavaFX web component, can create and provide access to a Document Object Model (DOM) of a web page. The root element of the document model can be accessed by using the `getDocument()` method of the `WebEngine` class. Example 3–4 provides a code fragment to implement some simple tasks: obtaining a URI of the web page and displaying it in the title of the application window.

Example 3–4 Deriving a URI from a DOM

```
WebView browser = new WebView();
WebEngine webEngine = browser.getEngine();
stage.setTitle(webEngine.getDocument().getDocumentURI());
```

Additionally, the document model event specification is supported to define event handlers in JavaFX code. See the specification of the `WebEngine` class for the example that attaches an event listener to an element of a web page.

3.7 Web Sockets

The `WebView` component supports the `WebSocket` interface to enable JavaFX applications to establish bidirectional communication with server processes. The `WebSocket` interface is described in detail in the `WebSocket` API specification. Example 3–5 shows a common model for using web sockets.

Example 3–5 Using Web Sockets in HTML Code

```
<!DOCTYPE HTML>
<html>
<head>
<title>Web Worker</title>
</head>
<body>
<script>
    socketConnection = new WebSocket('ws://example.com:8001');
    socketConnection.onopen = function () {
        // do some stuff
    };
</script>
</body>
</html>
```

3.8 Web Workers

The JavaFX web component supports running web worker scripts in parallel to activities on the loaded web page. This functionality enables long-running scripts to be executed without a need to wait for user interaction.

Example 3–6 shows a web page that uses the `myWorker` script for a long-running task.

Example 3–6 Using a Web Worker Script

```
<!DOCTYPE HTML>
<html>
<head>
<title>Web Worker</title>
</head>
<body>
<script>
   var worker = new Worker('myWorker.js');
   worker.onmessage = function (event) {
     document.getElementById('result').textContent = event.data;
   };
</script>
</body>
</html>
```

Learn more about the web worker script from the HTML5 specification.

3.9 Web Font Support

The JavaFX web component supports web fonts declared with the `@font-face` rule. This rule enables linking fonts that are automatically downloaded when needed. According to the HTML5 specification, this functionality provides the capability to select a font that closely matches the design goals for a given page. The HTML code in Example 3–7 uses the `@font-face` rule to link a font specified by its URL.

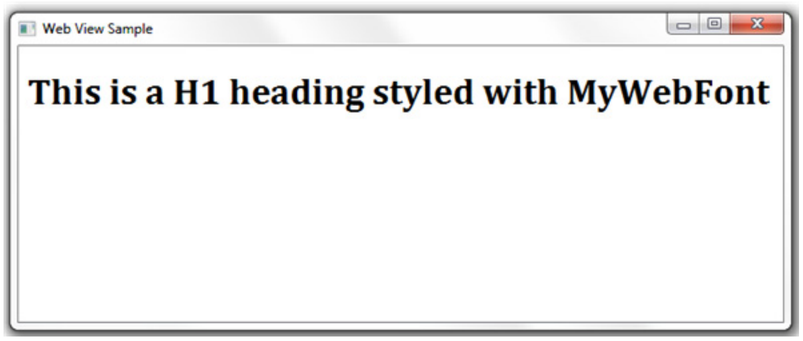
Example 3–7 Using a Web Font

```
<!DOCTYPE HTML>
<html>
  <head>
```

```
    <title>Web Font</title>
    <style>
      @font-face {
        font-family: "MyWebFont";
        src: url("http://example.com/fonts/MyWebFont.ttf")
      }
      h1 { font-family: "MyWebFont", serif;}
    </style>
  </head>
  <body>
    <h1> This is a H1 heading styled with MyWebFont</h1>
  </body>
</html>
```

When this HTML code is loaded into the WebViewSample application, it is rendered as shown in Figure 3–4.

Figure 3–4 Rendering a Web Font



This is a H1 heading styled with MyWebFont

4

Adding a WebView Component to the Application Scene

This chapter introduces the WebViewSample application and explains how to implement the task of adding a `WebView` component to the scene of a JavaFX application.

The WebViewSample application creates the `Browser` class that encapsulates both the `WebView` object and the toolbar with UI controls. The `WebViewSample` class of the application creates the scene and adds a `Browser` object to the scene.

4.1 Creating an Embedded Browser

Example 4–1 shows how to add the `WebView` component to the application scene.

Example 4–1 Creating a Browser by Using the WebView and WebEngine Classes

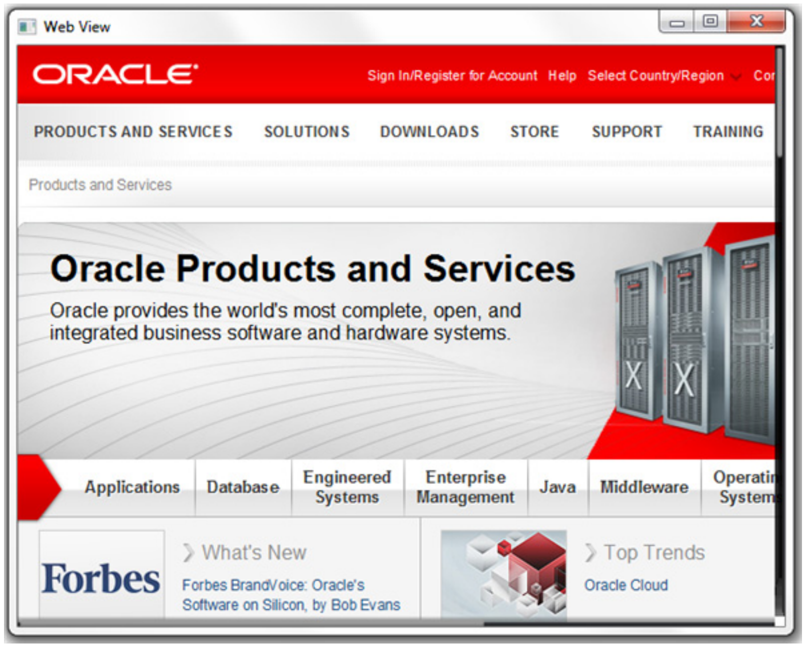
```
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;


public class WebViewSample extends Application {
    private Scene scene;
    @Override public void start(Stage stage) {
        // create the scene
        stage.setTitle("Web View");
        scene = new Scene(new Browser(),900,600, Color.web("#666970"));
        stage.setScene(scene);
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        stage.show();
    }

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

```
class Browser extends Region {

    final WebView browser = new WebView();
    final WebEngine webEngine = browser.getEngine();

    public Browser() {
        //apply the styles
        getStyleClass().add("browser");
        // load the web page
        webEngine.load("http://www.oracle.com/products/index.html");
        //add the web view to the scene
        getChildren().add(browser);

    }
    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        layoutInArea(browser,0,0,w,h,0, HPos.CENTER, VPos.CENTER);
    }

    @Override protected double computePrefWidth(double height) {
        return 900;
    }

    @Override protected double computePrefHeight(double width) {
        return 600;
    }
}
```

In this code, the web engine loads a URL that points to the Oracle corporate web site. The `WebView` object that contains this web engine is added to the application scene by using the `getChildren` and `add` methods.

When you add, compile, and run this code fragment, it produces the application window shown in Figure 4–1.

Figure 4–1 WebView Object in an Application Scene


Web View
ORACLE
Sign In/Register for Account Help Select Country/Region
PRODUCTS AND SERVICES SOLUTIONS DOWNLOADS STORE SUPPORT TRAINING
Products and Services
Applications Database Engineered Systems Enterprise Management Java Middleware
Forbes
What's New
Top Trends
Oracle Cloud

Oracle Products and Services

Oracle provides the world's most complete, open, and integrated business software and hardware systems.

Forbes BrandVoice: Oracle's Software on Silicon, by Bob Evans

4.2 Creating an Application Toolbar

Add a toolbar with four `Hyperlink` objects to switch between different Oracle web resources. Study the modified code of the `Browser` class shown in Example 4–2. It adds URLs for alternative web resources including Oracle products, blogs, Java documentation, and the partner network. The code fragment also creates a toolbar and adds the hyperlinks to it.

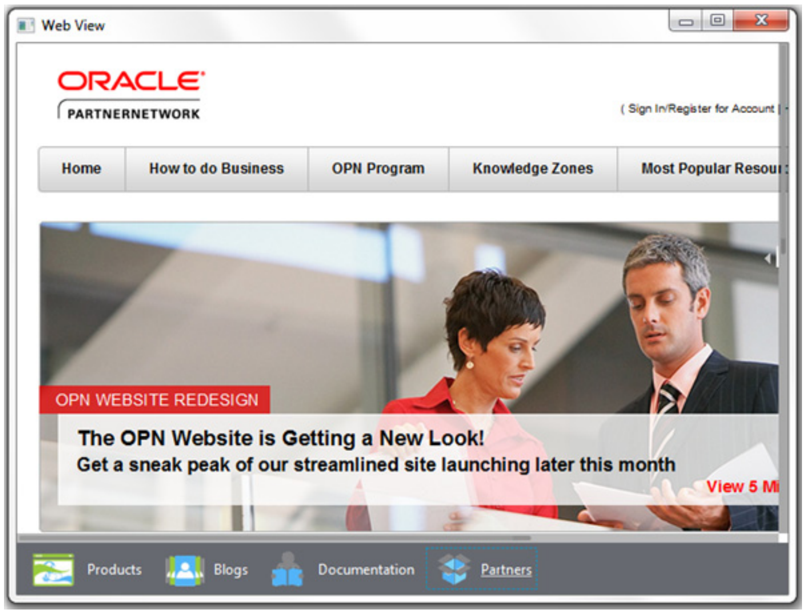
Example 4–2 Creating a Toolbar

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Hyperlink;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

public class WebViewSample extends Application {

    private Scene scene;

    @Override public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View");
        scene = new Scene(new Browser(),900,600, Color.web("#666970"));
        stage.setScene(scene);
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }

    public static void main(String[] args){
        launch(args);
    }
}
class Browser extends Region {
   private HBox toolBar;

   final private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png"
    };
    final private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners"
    };

    final private static String[] urls = new String[]{
        "http://www.oracle.com/products/index.html",
        "http://blogs.oracle.com/",
        "http://docs.oracle.com/javase/index.html",
        "http://www.oracle.com/partners/index.html"
    };

    final ImageView selectedImage = new ImageView();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];
    final WebView browser = new WebView();
    final WebEngine webEngine = browser.getEngine();

    public Browser() {
        //apply the styles
        getStyleClass().add("browser");

        for (int i = 0; i < captions.length; i++) {
            final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            Image image = images[i] =
                new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setGraphic(new ImageView (image));
            final String url = urls[i];

            hpl.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent e) {
                    webEngine.load(url);
                }
            });
        }
```

```
// load the home page
        webEngine.load("http://www.oracle.com/products/index.html");

// create the toolbar
        toolBar = new HBox();
        toolBar.getStyleClass().add("browser-toolbar");
        toolBar.getChildren().addAll(hpls);

        //add components
        getChildren().add(toolBar);
        getChildren().add(browser);
    }

    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        double tbHeight = toolBar.prefHeight(w);
        layoutInArea(browser,0,0,w,h-tbHeight,0, HPos.CENTER, VPos.CENTER);
        layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
    }

    @Override protected double computePrefWidth(double height) {
        return 900;
    }

    @Override protected double computePrefHeight(double width) {
        return 600;
    }
}
```

This code uses a `for` loop to create the hyperlinks. The `setOnAction` method defines the behavior of the hyperlinks. When a user clicks a link, the corresponding URL value is passed to the `load` method of the `webEngine`. When you compile and run the modified application, the application window changes as shown in Figure 4–2.

Figure 4–2 Embedded Browser with the Navigation Toolbar



5

Processing JavaScript Commands

This chapter extends the WebViewSample application and explains how to call JavaScript commands from JavaFX code.

The `WebEngine` class provides API to run a script within the context of the current HTML page.

5.1 Understanding the executeScript method

The `executeScript` method of the `WebEngine` class enables executing any JavaScript commands declared in the loaded HTML page. Use the following string to call this method on a web engine: `webEngine.executeScript("<function name>");`.

The method execution result is converted to a `java.lang.Object` instance by using the following rules:

- JavaScript `Int32` is converted to `java.lang.Integer`
- JavaScript numbers are converted to `java.lang.Double`
- JavaScript string values are converted to `java.lang.String`
- JavaScript boolean values are converted to `java.lang.Boolean`

Refer to the API documentation for the `WebEngine` class for more information about the conversion results.

5.2 Calling JavaScript Commands from JavaFX Code

Extend the WebViewSample application to introduce a help file and execute a JavaScript command that toggles the list of topics in the help file. Create the Help toolbar item that leads to the help.html file, where a user can preview reference material about Oracle web sites.

Add the help.html file, shown in, to the WebViewSample application.

Example 5–1 help.html file

```
<html lang="en">
    <head>
        <!-- Visibility toggle script -->
        <script type="text/javascript">
            <!--
            function toggle_visibility(id) {
                var e = document.getElementById(id);
                if (e.style.display == 'block')
                    e.style.display = 'none';
                else
                    e.style.display = 'block';
            }
//-->
        </script>
    </head>
    <body>
        <h1>Online Help</h1>
        <p class="boxtitle"><a href="#" onclick="toggle_visibility('help_
topics');"
  class="boxtitle">[+] Show/Hide Help Topics</a></p>
        <ul id="help_topics" style='display:none;'>
            <li>Products - Extensive overview of Oracle hardware and software
products,
                and summary Oracle consulting, support, and educational services.
</li>
            <li>Blogs - Oracle blogging community (use the Hide All and Show All
buttons
                to collapse and expand the list of topics).</li>
            <li>Documentation - Landing page to start learning Java. The page
contains
                links to the Java tutorials, developer guides, and API
documentation.</li>
            <li>Partners - Oracle partner solutions and programs. Popular
resources and
                membership opportunities.</li>
        </ul>
    </body>
</html>
```

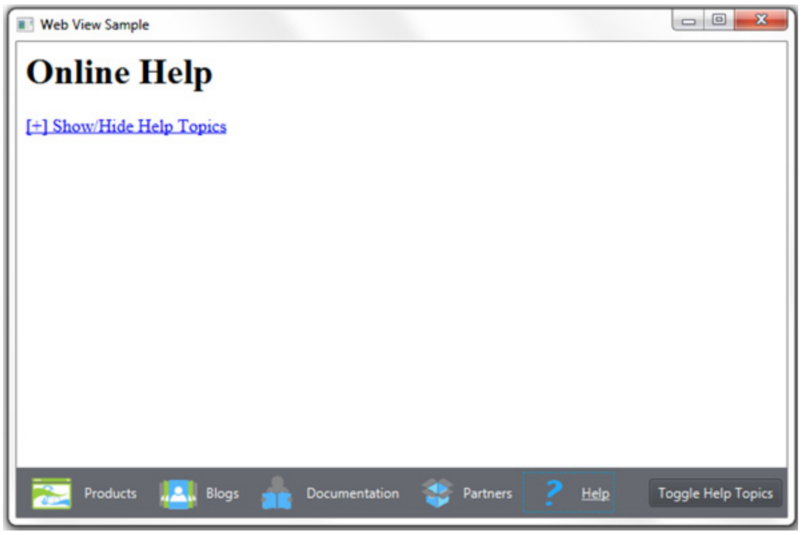
The modified application code shown in Example 5–2 creates the Help toolbar item and an additional button to hide and show help topics. The button is added to the toolbar only when the Help page is selected.

Example 5–2 Adding the Toggle Help Topics Button

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Hyperlink;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

public class WebViewSample extends Application {

    private Scene scene;
    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View Sample");
        scene = new Scene(new Browser(stage), 900, 600, Color.web("#666970"));
        stage.setScene(scene);
        // apply CSS style
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }

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

class Browser extends Region {

    private final HBox toolBar;
    final private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    final private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
        "Help"
    };
    final private static String[] urls = new String[]{
        "http://www.oracle.com/products/index.html",
        "http://blogs.oracle.com/",
        "http://docs.oracle.com/javase/index.html",
        "http://www.oracle.com/partners/index.html",
        WebViewSample.class.getResource("help.html").toExternalForm()
    };
    final ImageView selectedImage = new ImageView();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];
    final WebView browser = new WebView();
    final WebEngine webEngine = browser.getEngine();
    final Button toggleHelpTopics = new Button("Toggle Help Topics");
    private boolean needDocumentationButton = false;


    public Browser(final Stage stage) {
        //apply the styles
        getStyleClass().add("browser");

        for (int i = 0; i < captions.length; i++) {
            // create hyperlinks
            Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            Image image = images[i]
                    = new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setGraphic(new ImageView(image));
            final String url = urls[i];
            final boolean addButton = (hpl.getText().equals("Help"));

            // process event
            hpl.setOnAction((ActionEvent e) -> {
                needDocumentationButton = addButton;
                webEngine.load(url);
            });

        }

        // create the toolbar
        toolBar = new HBox();
        toolBar.setAlignment(Pos.CENTER);
        toolBar.getStyleClass().add("browser-toolbar");
        toolBar.getChildren().addAll(hpls);
        toolBar.getChildren().add(createSpacer());

        //set action for the button
        toggleHelpTopics.setOnAction((ActionEvent t) -> {
            webEngine.executeScript("toggle_visibility('help_topics')");
        });

         // process page loading
        webEngine.getLoadWorker().stateProperty().addListener(
            (ObservableValue<? extends State> ov, State oldState,
                State newState) -> {
                    toolBar.getChildren().remove(toggleHelpTopics);
                    if (newState == State.SUCCEEDED) {
                        if (needDocumentationButton) {
                            toolBar.getChildren().add(toggleHelpTopics);
                        }
                    }
        });

        // load the home page
        webEngine.load("http://www.oracle.com/products/index.html");

        //add components
        getChildren().add(toolBar);
        getChildren().add(browser);
    }

    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override
    protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        double tbHeight = toolBar.prefHeight(w);
        layoutInArea(browser,0,0,w,h-tbHeight,0,HPos.CENTER,VPos.CENTER);
        layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
    }

    @Override
    protected double computePrefWidth(double height) {
        return 900;
    }

    @Override
    protected double computePrefHeight(double width) {
        return 600;
    }
}
```

Loading always happens on a background thread. Methods that initiate loading return immediately after scheduling a background job. The `getLoadWorker()` method provides an instance of the `Worker` interface to track the loading progress. If the progress status of the Help page is `SUCCEEDED`, the Toggle Help Topics button is added to the toolbar, as shown in Figure 5–1.

Figure 5–1 Toggle Help Topics Button



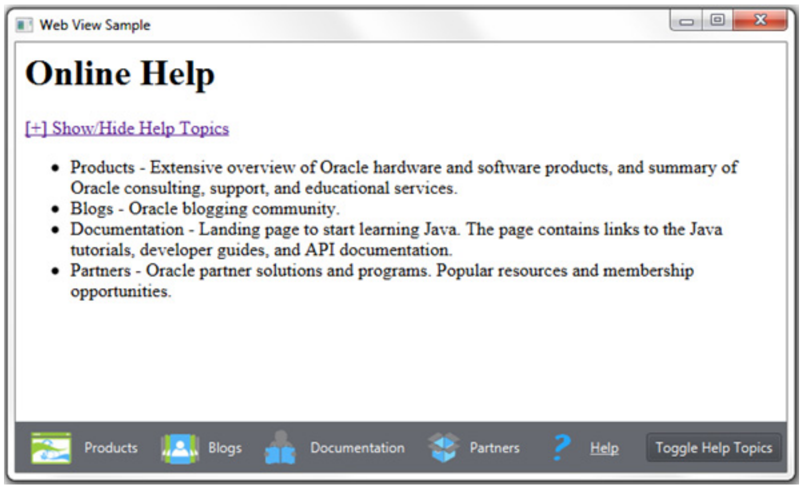
The `setOnAction` method shown in Example 5–3 defines behavior for the Toggle Help Topics button.

Example 5–3 Executing a JavaScript Command

```
//set action for the button
toggleHelpTopics.setOnAction((ActionEvent t) -> {
    webEngine.executeScript("toggle_visibility('help_topics')");
});
```

When the user clicks the Toggle Help Topics button, the `executeScript` method runs the `toggle_visibility` JavaScript function for the help.html page, and the help topics appear, as shown in Figure 5–2. When the user performs another click, the `toggle_visibility` function hides the lists of the topics.

Figure 5–2 Showing the Help Topics



Online Help

[+] Show/Hide Help Topics

- Products - Extensive overview of Oracle hardware and software products, and summary of Oracle consulting, support, and educational services.
- Blogs - Oracle blogging community.
- Documentation - Landing page to start learning Java. The page contains links to the Java tutorials, developer guides, and API documentation.
- Partners - Oracle partner solutions and programs. Popular resources and membership opportunities.

6

Making Upcalls from JavaScript to JavaFX

Now you know how to invoke JavaScript from JavaFX. In this chapter, you can explore the opposite functionality — calling from web content to JavaFX.

The general concept is to create an interface object in the JavaFX application and make it known to JavaScript by calling the `JSObject.setMember()` method. After that, you can call public methods and access public fields of this object from JavaScript.

6.1 Using a JavaScript Command to Exit JavaFX Application

First, add one more line to the help.html file: `<p><a href="about:blank" onclick="app.exit()">Exit the Application</a></p>`. By clicking the Exit the Application link in the help.html file, the user exits the WebViewSample application. Modify the application, as shown in Example 6–1, to implement this functionality.

Example 6–1 Closing JavaFX Application by Using JavaScript

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Hyperlink;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

public class WebViewSample extends Application {

    private Scene scene;

    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View Sample");
        scene = new Scene(new Browser(stage), 900, 600, Color.web("#666970"));
        stage.setScene(scene);
        // apply CSS style
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }

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

class Browser extends Region {

    private final HBox toolBar;
    final private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    final private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
        "Help"
    };
    final private static String[] urls = new String[]{
        "http://www.oracle.com/products/index.html",
        "http://blogs.oracle.com/",
        "http://docs.oracle.com/javase/index.html",
        "http://www.oracle.com/partners/index.html",
        WebViewSample.class.getResource("help.html").toExternalForm()
    };
    final ImageView selectedImage = new ImageView();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];
    final WebView browser = new WebView();
    final WebEngine webEngine = browser.getEngine();
    final Button toggleHelpTopics = new Button("Toggle Help Topics");
    private boolean needDocumentationButton = false;


    public Browser(final Stage stage) {
        //apply the styles
        getStyleClass().add("browser");

        for (int i = 0; i < captions.length; i++) {
            // create hyperlinks
            Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            Image image = images[i]
                    = new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setGraphic(new ImageView(image));
            final String url = urls[i];
            final boolean addButton = (hpl.getText().equals("Help"));
            // process event
            hpl.setOnAction((ActionEvent e) -> {
                needDocumentationButton = addButton;
                webEngine.load(url);
            });

        }

        // create the toolbar
        toolBar = new HBox();
        toolBar.setAlignment(Pos.CENTER);
        toolBar.getStyleClass().add("browser-toolbar");
        toolBar.getChildren().addAll(hpls);
        toolBar.getChildren().add(createSpacer());

        //set action for the button
        toggleHelpTopics.setOnAction((ActionEvent t) -> {
            webEngine.executeScript("toggle_visibility('help_topics')");
        });

         // process page loading
        webEngine.getLoadWorker().stateProperty().addListener(
            (ObservableValue<? extends State> ov, State oldState,
                State newState) -> {
                    toolBar.getChildren().remove(toggleHelpTopics);
                    if (newState == State.SUCCEEDED) {
                        JSObject win
                                = (JSObject) webEngine.executeScript("window");
                        win.setMember("app", new JavaApp());
                        if (needDocumentationButton) {
                            toolBar.getChildren().add(toggleHelpTopics);
                        }
                    }
        });

        // load the home page
        webEngine.load("http://www.oracle.com/products/index.html");

        //add components
        getChildren().add(toolBar);
        getChildren().add(browser);
    }

    // JavaScript interface object
    public class JavaApp {

        public void exit() {
            Platform.exit();
        }
    }

    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override
    protected void layoutChildren() {
```

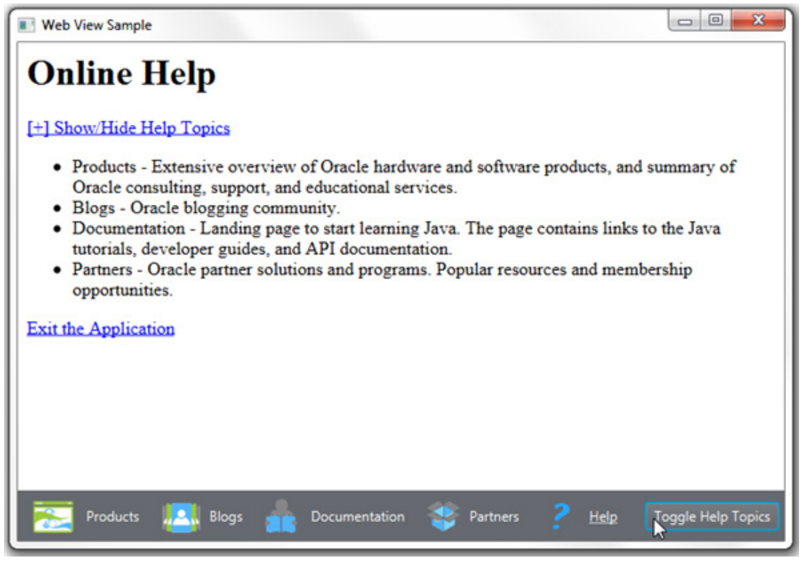
```
            double w = getWidth();
            double h = getHeight();
            double tbHeight = toolBar.prefHeight(w);
            layoutInArea(browser,0,0,w,h-tbHeight,0,HPos.CENTER,VPos.CENTER);
            layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
        }

        @Override
        protected double computePrefWidth(double height) {
            return 900;
        }

        @Override
        protected double computePrefHeight(double width) {
            return 600;
        }
    }
```

Examine the bold lines in Example 6–1. The `exit()` method of the `JavaApp` interface is public; therefore, it can be accessed externally. When this method is called, it causes the JavaFX application to terminate.

The `JavaApp` interface in Example 6–1 is set as a member of the `JSObject` instance, so that JavaScript becomes aware of that interface. It becomes known to JavaScript under the name `window.app`, or just `app`, and its only method can be called from JavaScript as `app.exit()`.

When you compile, run the WebViewSample application, and click the Help icon, the Exit Application link appears at the bottom of the page, as shown in Figure 6–1.

Figure 6–1 Exit Application Link



Online Help

[+] Show/Hide Help Topics

- Products - Extensive overview of Oracle hardware and software products, and summary of Oracle consulting, support, and educational services.
- Blogs - Oracle blogging community.
- Documentation - Landing page to start learning Java. The page contains links to the Java tutorials, developer guides, and API documentation.
- Partners - Oracle partner solutions and programs. Popular resources and membership opportunities.

Exit the Application

Examine the content of the file, then click the Exit the Application link, shown in Figure 6–1, to close the WebViewSample application.

7

Managing Web Pop-Up Windows

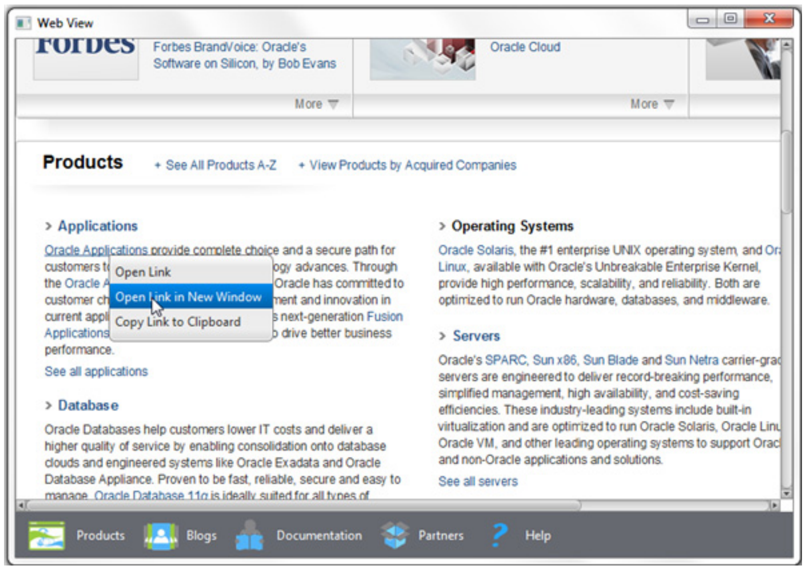
This chapter explains how to work with pop-up windows in the browser created by using the `WebView` component and how to implement this functionality in the WebViewSample application.

When you need to open a new browser window in your application, the instances of the `PopupFeatures` class are passed into pop-up handlers registered on a `WebEngine` object by using the `setCreatePopupHandler` method.

7.1 Using Pop-Up Windows to Set

In the WebViewSample application, you can set an alternative `WebView` object for the documents that will be opened in a separate window. Figure 7–1 shows a context menu a user can open by right-clicking any link.

Figure 7–1 Pop-Up Window



Oracle Applications provide complete choice and a secure path for
customers t ogy advances. Through
the Oracle A Oracle has committed to
customer ch ment and innovation in
current appl s next-generation Fusion
Applications o drive better business
performance.

Oracle Databases help customers lower IT costs and deliver a higher quality of service by enabling consolidation onto database clouds and engineered systems like Oracle Exadata and Oracle Database Appliance. Proven to be fast, reliable, secure and easy to manage, Oracle Database 11g is ideally suited for all types of

Oracle Solaris, the #1 enterprise UNIX operating system, and Or
Linux, available with Oracle's Unbreakable Enterprise Kernel, provide high performance, scalability, and reliability. Both are optimized to run Oracle hardware, databases, and middleware.

Oracle's SPARC, Sun x86, Sun Blade and Sun Netra carrier-grad
servers are engineered to deliver record-breaking performance, simplified management, high availability, and cost-saving efficiencies. These industry-leading systems include built-in virtualization and are optimized to run Oracle Solaris, Oracle Linu
Oracle VM, and other leading operating systems to support Orac
and non-Oracle applications and solutions.

To specify a new browser window for the target document, use the `PopupFeatures` instance as shown in the modified application code in Example 7–1.

Example 7–1 Processing a Command of a Pop-Up Window

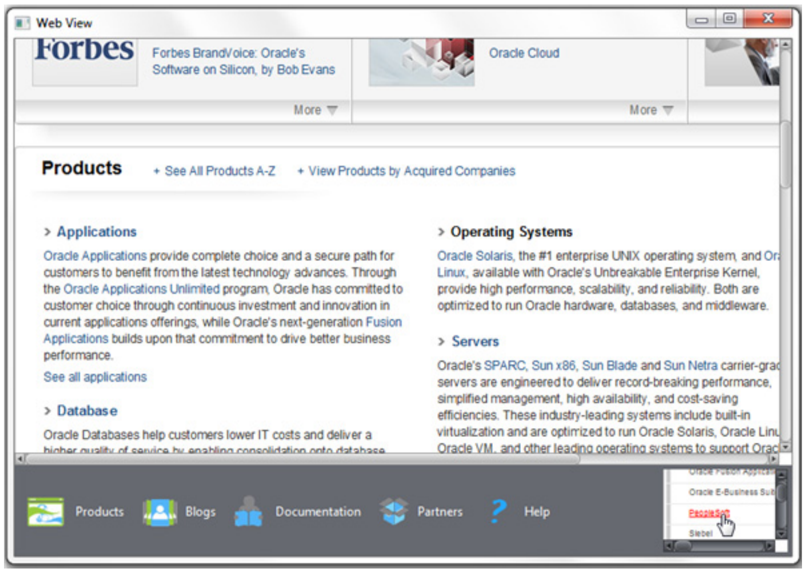
```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Hyperlink;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.PopupFeatures;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

public class WebViewSample extends Application {

    private Scene scene;

    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View Sample");
        scene = new Scene(new Browser(stage), 900, 600, Color.web("#666970"));
        stage.setScene(scene);
        // apply CSS style
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }

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

class Browser extends Region {

    private final HBox toolBar;
    final private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    final private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
        "Help"
    };
```

```
final private static String[] urls = new String[]{
    "http://www.oracle.com/products/index.html",
    "http://blogs.oracle.com/",
    "http://docs.oracle.com/javase/index.html",
    "http://www.oracle.com/partners/index.html",
    WebViewSample.class.getResource("help.html").toExternalForm()
};
final ImageView selectedImage = new ImageView();
final Hyperlink[] hpls = new Hyperlink[captions.length];
final Image[] images = new Image[imageFiles.length];
final WebView browser = new WebView();
final WebEngine webEngine = browser.getEngine();
final Button toggleHelpTopics = new Button("Toggle Help Topics");
private boolean needDocumentationButton = false;
final WebView smallView = new WebView();

public Browser(final Stage stage) {
    //apply the styles
    getStyleClass().add("browser");

    for (int i = 0; i < captions.length; i++) {
        // create hyperlinks
        Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
        Image image = images[i]
                = new Image(getClass().getResourceAsStream(imageFiles[i]));
        hpl.setGraphic(new ImageView(image));
        final String url = urls[i];
        final boolean addButton = (hpl.getText().equals("Help"));

        // process event
        hpl.setOnAction((ActionEvent e) -> {
            needDocumentationButton = addButton;
            webEngine.load(url);
        });

    }

    // create the toolbar
    toolBar = new HBox();
    toolBar.setAlignment(Pos.CENTER);
    toolBar.getStyleClass().add("browser-toolbar");
    toolBar.getChildren().addAll(hpls);
    toolBar.getChildren().add(createSpacer());

    //set action for the button
    toggleHelpTopics.setOnAction((ActionEvent t) -> {
        webEngine.executeScript("toggle_visibility('help_topics')");
    });

    smallView.setPrefSize(120, 80);

    //handle popup windows
    webEngine.setCreatePopupHandler(
            (PopupFeatures config) -> {
                smallView.setFontScale(0.8);
                if (!toolBar.getChildren().contains(smallView)) {
                    toolBar.getChildren().add(smallView);
                }
                return smallView.getEngine();
    });
```

```
         // process page loading
        webEngine.getLoadWorker().stateProperty().addListener(
            (ObservableValue<? extends State> ov, State oldState,
                State newState) -> {
                    toolBar.getChildren().remove(toggleHelpTopics);
                    if (newState == State.SUCCEEDED) {
                        JSObject win
                                = (JSObject) webEngine.executeScript("window");
                        win.setMember("app", new JavaApp());
                        if (needDocumentationButton) {
                            toolBar.getChildren().add(toggleHelpTopics);
                        }
                    }
        });

        // load the home page
        webEngine.load("http://www.oracle.com/products/index.html");

        //add components
        getChildren().add(toolBar);
        getChildren().add(browser);
    }

    // JavaScript interface object
    public class JavaApp {

        public void exit() {
            Platform.exit();
        }
    }

    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override
    protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        double tbHeight = toolBar.prefHeight(w);
        layoutInArea(browser,0,0,w,h-tbHeight,0,HPos.CENTER,VPos.CENTER);
        layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
    }

    @Override
    protected double computePrefWidth(double height) {
        return 900;
    }

    @Override
    protected double computePrefHeight(double width) {
        return 600;
    }
}
```

When a user selects the Open Link in New Window option from the pop-up menu, the `smallView` browser is added to the application toolbar. This behavior is defined by the `setCreatePopupHandler` method, which returns the web engine of an alternative browser to notify the application where to render the target page. The result of compiling and running the modified application is shown in Figure 7–2.

Figure 7–2 Small Preview Browser



Web View

Forbes

Forbes BrandVoice: Oracle's Software on Silicon, by Bob Evans

Oracle Cloud

More

More

Products

+ See All Products A-Z

+ View Products by Acquired Companies

› Applications

Oracle Applications provide complete choice and a secure path for customers to benefit from the latest technology advances. Through the Oracle Applications Unlimited program, Oracle has committed to customer choice through continuous investment and innovation in current applications offerings, while Oracle's next-generation Fusion Applications builds upon that commitment to drive better business performance.

See all applications

› Database

Oracle Databases help customers lower IT costs and deliver a higher quality of service by enabling consolidation onto database

› Operating Systems

Oracle Solaris, the #1 enterprise UNIX operating system, and Or Linux, available with Oracle's Unbreakable Enterprise Kernel, provide high performance, scalability, and reliability. Both are optimized to run Oracle hardware, databases, and middleware.

› Servers

Oracle's SPARC, Sun x86, Sun Blade and Sun Netra carrier-grad servers are engineered to deliver record-breaking performance, simplified management, high availability, and cost-saving efficiencies. These industry-leading systems include built-in virtualization and are optimized to run Oracle Solaris, Oracle Linu Oracle VM, and other leading operating systems to support Orac

Note that context menus are enabled by default for all `WebView` objects. To disable a context menu for a particular `WebView` instance, pass the false value to the `setContextMenuEnabled` method: `browser.setContextMenuEnabled(false);`.

8 Managing Web History

This chapter introduces the WebHistory class and teaches how to obtain and show the URLs of visited pages.

You can obtain the list of visited pages by using the WebHistory class. It represents a session history associated with a WebEngine object. Use the WebEngine.getHistory() method to get the WebHistory instance for a particular web engine, as shown in the following line: `WebHistory history = webEngine.getHistory();`.

The history is basically a list of entries. Each entry represents a visited page and it provides access to relevant page info, such as URL, title, and the date the page was last visited. The list can be obtained by using the getEntries() method.

The history list changes as users navigate through the web. Use the ObservableList API to process the changes.

8.1 Obtaining the List of Visited Pages

You typically use a standard or custom UI control to display the history list. Example 8–1 shows how to obtain a history items and present them in the ComboBox control.

Example 8–1 Obtaining and Processing the List of Web History Items

```
final WebHistory history = webEngine.getHistory();
history.getEntries().addListener(new
    ListChangeListener<WebHistory.Entry>() {
        @Override
        public void onChanged(Change<? extends Entry> c) {
            c.next();
            for (Entry e : c.getRemoved()) {
                comboBox.getItems().remove(e.getUrl());
            }
            for (Entry e : c.getAddedSubList()) {
                comboBox.getItems().add(e.getUrl());
            }
        }
    }
);

comboBox.setPrefWidth(60);
comboBox.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent ev) {
        int offset =
                comboBox.getSelectionModel().getSelectedIndex()
                - history.getCurrentIndex();
        history.go(offset);
    }
});
```

The `ListChangeListener` object tracks the changes of history entries and adds the corresponding URLs to the combo box.

When users select any item in the combo box, the web engine is navigated to the URL defined by the history entry item, which position in the list is defined by the `offset` value. A negative `offset` value specifies the position preceding the current entry, and a positive `offset` value specifies the position following the current entry.

Example 8–2 shows the complete code of the modified application.

Example 8–2 WebViewSample with the History Combo Box

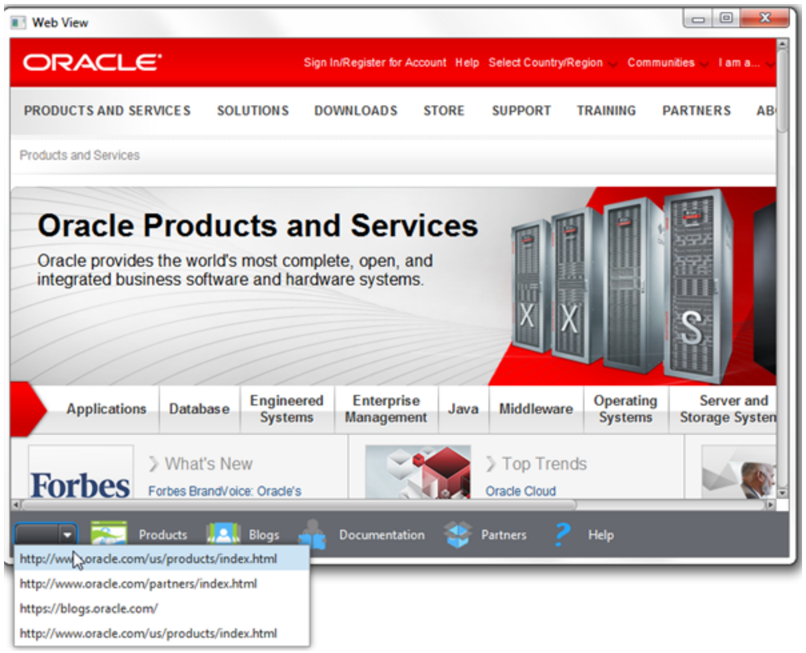
```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.collections.ListChangeListener;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Hyperlink;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.PopupFeatures;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebHistory;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

public class WebViewSample extends Application {

    private Scene scene;

    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View Sample");
        scene = new Scene(new Browser(stage), 900, 600, Color.web("#666970"));
        stage.setScene(scene);
        // apply CSS style
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}

class Browser extends Region {

    private final HBox toolBar;
    final private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    final private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
        "Help"
    };
    final private static String[] urls = new String[]{
        "http://www.oracle.com/products/index.html",
        "http://blogs.oracle.com/",
        "http://docs.oracle.com/javase/index.html",
        "http://www.oracle.com/partners/index.html",
        WebViewSample.class.getResource("help.html").toExternalForm()
    };
    final ImageView selectedImage = new ImageView();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];
    final WebView browser = new WebView();
    final WebEngine webEngine = browser.getEngine();
    final Button toggleHelpTopics = new Button("Toggle Help Topics");
    private boolean needDocumentationButton = false;
    final WebView smallView = new WebView();
    final ComboBox comboBox = new ComboBox();

    public Browser(final Stage stage) {
        //apply the styles
        getStyleClass().add("browser");

        for (int i = 0; i < captions.length; i++) {
            // create hyperlinks
            Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            Image image = images[i]
                    = new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setGraphic(new ImageView(image));
            final String url = urls[i];
            final boolean addButton = (hpl.getText().equals("Help"));

            // process event
            hpl.setOnAction((ActionEvent e) -> {
                needDocumentationButton = addButton;
                webEngine.load(url);
            });

        }
```

```
comboBox.setPrefWidth(60);

// create the toolbar
toolBar = new HBox();
toolBar.setAlignment(Pos.CENTER);
toolBar.getStyleClass().add("browser-toolbar");
toolBar.getChildren().add(comboBox);
toolBar.getChildren().addAll(hpls);
toolBar.getChildren().add(createSpacer());

//set action for the button
toggleHelpTopics.setOnAction((ActionEvent t) -> {
    webEngine.executeScript("toggle_visibility('help_topics')");
});

smallView.setPrefSize(120, 80);

//handle popup windows
webEngine.setCreatePopupHandler(
        (PopupFeatures config) -> {
            smallView.setFontScale(0.8);
            if (!toolBar.getChildren().contains(smallView)) {
                toolBar.getChildren().add(smallView);
            }
            return smallView.getEngine();
});

//process history
final WebHistory history = webEngine.getHistory();
history.getEntries().addListener(
    (ListChangeListener.Change<? extends WebHistory.Entry> c) -> {
        c.next();
        c.getRemoved().stream().forEach((e) -> {
        comboBox.getItems().remove(e.getUrl());
    });
        c.getAddedSubList().stream().forEach((e) -> {
        comboBox.getItems().add(e.getUrl());
    });
});

//set the behavior for the history combobox
comboBox.setOnAction((Event ev) -> {
    int offset
            = comboBox.getSelectionModel().getSelectedIndex()
            - history.getCurrentIndex();
    history.go(offset);
});

 // process page loading
webEngine.getLoadWorker().stateProperty().addListener(
    (ObservableValue<? extends State> ov, State oldState,
        State newState) -> {
            toolBar.getChildren().remove(toggleHelpTopics);
            if (newState == State.SUCCEEDED) {
                JSObject win
                        = (JSObject) webEngine.executeScript("window");
                win.setMember("app", new JavaApp());
                if (needDocumentationButton) {
                    toolBar.getChildren().add(toggleHelpTopics);
                }
```

```
                    }
            });

            // load the home page
            webEngine.load("http://www.oracle.com/products/index.html");

            //add components
            getChildren().add(toolBar);
            getChildren().add(browser);
        }

        // JavaScript interface object
        public class JavaApp {

            public void exit() {
                Platform.exit();
            }
        }

        private Node createSpacer() {
            Region spacer = new Region();
            HBox.setHgrow(spacer, Priority.ALWAYS);
            return spacer;
        }

        @Override
        protected void layoutChildren() {
            double w = getWidth();
            double h = getHeight();
            double tbHeight = toolBar.prefHeight(w);
            layoutInArea(browser,0,0,w,h-tbHeight,0,HPos.CENTER,VPos.CENTER);
            layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
        }

        @Override
        protected double computePrefWidth(double height) {
            return 900;
        }

        @Override
        protected double computePrefHeight(double width) {
            return 600;
        }
    }
```

When you compile and run the application, it produces the window shown in Figure 8–1.

Figure 8–1 Selecting URL from the History Combo Box


Web View
ORACLE
Sign In/Register for Account Help Select Country/Region Communities I am a...
PRODUCTS AND SERVICES SOLUTIONS DOWNLOADS STORE SUPPORT TRAINING PARTNERS
Products and Services
Oracle Products and Services
Oracle provides the world's most complete, open, and integrated business software and hardware systems.
Applications Database Engineered Systems Enterprise Management Java Middleware Operating Systems Server and Storage Systems
What's New
Forbes BrandVoice: Oracle's
Top Trends
Oracle Cloud
Products Blogs Documentation Partners Help

http://www.oracle.com/us/products/index.html
http://www.oracle.com/partners/index.html
https://blogs.oracle.com/
http://www.oracle.com/us/products/index.html

9 Printing HTML Content

This chapter teaches you how to print a web page loaded in the `WebView` component.

With the printing API available in JavaFX 8, you can print graphical content of JavaFX applications. The corresponding classes and enums are located in the `javafx.print` package.

9.1 Using the Printing API

To enable the printing functionality in your JavaFX application, you must use the `PrinterJob` class. This class represents a printer job that is associated with the default system printer. Use the `Printer` class to alter a printer for a particular job. For each print job, you can specify job settings by using the properties of the `JobSettings` class such as `collation`, `copies`, `pageLayout`, `pageRanges`, `paperSource`, `printColor`, `printResolution`, `printQuality`, and `printSides`.

You can print any node of the scene graph including the root node. You can also print nodes that are not added to the scene. Use the `printPage` method to initiate a print job for a particular node: `job.printPage(node)`. See the to JavaFX 8 API specification for more information about printing capabilities.

When working with the JavaFX web component, you typically need to print an HTML page loaded into the browser rather than the application UI itself. That is why the `print` method was added to the `WebEngine` class. This method is geared toward printing an HTML page that is associated with the web engine.

9.2 Adding a Context Menu to Enable Printing

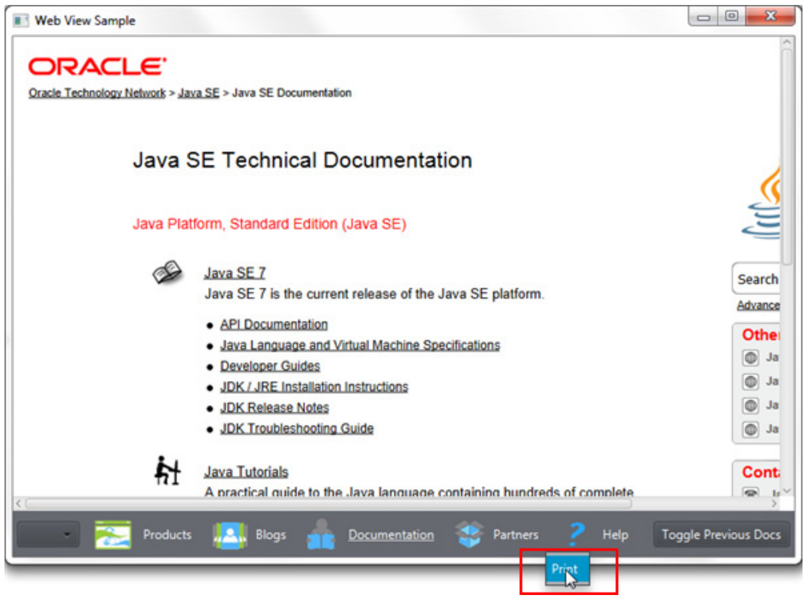
Typically, you add a Print command to an application menu or assign printing to one of the toolbar buttons. In the WebViewSample application, the toolbar is overloaded with controls, which is why you add the Print command to the context menu that is enabled by a right-click. Example 9–1 shows a code fragment that adds a context menu with the Print command to the application toolbar.

Example 9–1   Creating a Toolbar Context Menu

```
//adding context menu
final ContextMenu cm = new ContextMenu();
MenuItem cmItem1 = new MenuItem("Print");
cm.getItems().add(cmItem1);
toolBar.addEventHandler(MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
    if (e.getButton() == MouseButton.SECONDARY) {
        cm.show(toolBar, e.getScreenX(), e.getScreenY());
    }
});
```

When you add the code fragment from Example 9–1 to the WebViewSample application, run it, and right click the toolbar, the Print context menu appears, as shown in Figure 9–1.

Figure 9–1 Print Context Menu


Web View Sample
ORACLE
Oracle Technology Network > Java SE > Java SE Documentation
Java SE Technical Documentation
Java Platform, Standard Edition (Java SE)
Java SE 7
Java SE 7 is the current release of the Java SE platform.
API Documentation
Java Language and Virtual Machine Specifications
Developer Guides
JDK / JRE Installation Instructions
JDK Release Notes
JDK Troubleshooting Guide
Java Tutorials
Search
Products
Blogs
Documentation
Partners
Help
Toggle Previous Docs
Print

9.3 Processing a Print Job

After the Print context menu is added to the application UI, you can define the printing action. First, you must create a `PrinterJob` object. Then, you call the `WebEngine.print` method passing the printer job as a parameter, as shown in Example 9–2.

Example 9–2 Calling the WebEngine.print Method

```
//processing print job
cmItem1.setOnAction((ActionEvent e) -> {
    PrinterJob job = PrinterJob.createPrinterJob();
    if (job != null) {
        webEngine.print(job);
        job.endJob();
    }
});
```

It is important to check for non-null printer jobs, because the `createPrinterJob` method returns `null` if there are no printers available in the system.

Study Example 9–3 to evaluate the complete code of the WebViewSample application with the enabled printing functionality.

Example 9–3 WebViewSample With the Enabled Printing Functionality

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.collections.ListChangeListener.Change;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.print.PrinterJob;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.Hyperlink;
import javafx.scene.control.MenuItem;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.PopupFeatures;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebHistory;
import javafx.scene.web.WebHistory.Entry;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

public class WebViewSample extends Application {

    private Scene scene;

    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View Sample");
        scene = new Scene(new Browser(stage), 900, 600, Color.web("#666970"));
        stage.setScene(scene);
        // apply CSS style
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }

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

class Browser extends Region {

    private final HBox toolBar;
    final private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
```

```
};
final private static String[] captions = new String[]{
    "Products",
    "Blogs",
    "Documentation",
    "Partners",
    "Help"
};
final private static String[] urls = new String[]{
    "http://www.oracle.com/products/index.html",
    "http://blogs.oracle.com/",
    "http://docs.oracle.com/javase/index.html",
    "http://www.oracle.com/partners/index.html",
    WebViewSample.class.getResource("help.html").toExternalForm()
};
final ImageView selectedImage = new ImageView();
final Hyperlink[] hpls = new Hyperlink[captions.length];
final Image[] images = new Image[imageFiles.length];
final WebView browser = new WebView();
final WebEngine webEngine = browser.getEngine();
final Button toggleHelpTopics = new Button("Toggle Help Topics");
final WebView smallView = new WebView();
final ComboBox comboBox = new ComboBox();
private boolean needDocumentationButton = false;

public Browser(final Stage stage) {
    //apply the styles
    getStyleClass().add("browser");

    for (int i = 0; i < captions.length; i++) {
        // create hyperlinks
        Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
        Image image = images[i]
                = new Image(getClass().getResourceAsStream(imageFiles[i]));
        hpl.setGraphic(new ImageView(image));
        final String url = urls[i];
        final boolean addButton = (hpl.getText().equals("Help"));

        // process event
        hpl.setOnAction((ActionEvent e) -> {
            needDocumentationButton = addButton;
            webEngine.load(url);
        });

    }


    comboBox.setPrefWidth(60);

    // create the toolbar
    toolBar = new HBox();
    toolBar.setAlignment(Pos.CENTER);
    toolBar.getStyleClass().add("browser-toolbar");
    toolBar.getChildren().add(comboBox);
    toolBar.getChildren().addAll(hpls);
    toolBar.getChildren().add(createSpacer());

    //set action for the button
    toggleHelpTopics.setOnAction((ActionEvent t) -> {
        webEngine.executeScript("toggle_visibility('help_topics')");
    });

    smallView.setPrefSize(120, 80);

    //handle popup windows
    webEngine.setCreatePopupHandler(
            (PopupFeatures config) -> {
                smallView.setFontScale(0.8);
                if (!toolBar.getChildren().contains(smallView)) {
                    toolBar.getChildren().add(smallView);
                }
                return smallView.getEngine();
    });

    //process history
    final WebHistory history = webEngine.getHistory();
    history.getEntries().addListener(
        (Change<? extends Entry> c) -> {
            c.next();
            c.getRemoved().stream().forEach((e) -> {
            comboBox.getItems().remove(e.getUrl());
        });
            c.getAddedSubList().stream().forEach((e) -> {
            comboBox.getItems().add(e.getUrl());
        });
    });

    //set the behavior for the history combobox
    comboBox.setOnAction((Event ev) -> {
        int offset
                = comboBox.getSelectionModel().getSelectedIndex()
                - history.getCurrentIndex();
        history.go(offset);
    });

    // process page loading
    webEngine.getLoadWorker().stateProperty().addListener(
        (ObservableValue<? extends State> ov, State oldState,
            State newState) -> {
                toolBar.getChildren().remove(toggleHelpTopics);
                if (newState == State.SUCCEEDED) {
                    JSObject win
                            = (JSObject) webEngine.executeScript("window");
                    win.setMember("app", new JavaApp());
                    if (needDocumentationButton) {
                        toolBar.getChildren().add(toggleHelpTopics);
                    }
                }
    });
    //adding context menu
    final ContextMenu cm = new ContextMenu();
    MenuItem cmItem1 = new MenuItem("Print");
    cm.getItems().add(cmItem1);
    toolBar.addEventHandler(MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
        if (e.getButton() == MouseButton.SECONDARY) {
            cm.show(toolBar, e.getScreenX(), e.getScreenY());
        }
    });

    //processing print job
```

```
        cmItem1.setOnAction((ActionEvent e) -> {
            PrinterJob job = PrinterJob.createPrinterJob();
            if (job != null) {
                webEngine.print(job);
                job.endJob();
            }
        });

        // load the home page
        webEngine.load("http://www.oracle.com/products/index.html");

        //add components
        getChildren().add(toolBar);
        getChildren().add(browser);
    }

    // JavaScript interface object
    public class JavaApp {

        public void exit() {
            Platform.exit();
        }
    }

    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override
    protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        double tbHeight = toolBar.prefHeight(w);
        layoutInArea(browser,0,0,w,h-tbHeight,0,HPos.CENTER,VPos.CENTER);
        layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
    }

    @Override
    protected double computePrefWidth(double height) {
        return 900;
    }

    @Override
    protected double computePrefHeight(double width) {
        return 600;
    }
}
```

To extend the printing capabilities of the WebViewSample application, use the classes available in the `javafx.print` package.

In your JavaFX application, you can implement browser tabs by using the `TabPane` class and create a new `WebView` object when a user adds a new tab.

To further enhance this application, you can apply effects, transformations, and animated transitions. You can also add more `WebView` instances to the application scene.

See the JavaFX API documentation and the JavaFX CSS specification for more information about available features. You can also study the JavaFX in Swing tutorial to learn how to add a `WebView` component in your existing Swing application.

Related API Documentation

- `WebView`
- `WebEngine`
- `WebHistory`
- `Region`
- `Hyperlink`
- `Worker`

A Web View Sample

This appendix lists code of the WebViewSample application including the following files:

- WebViewSample.java
- BrowserToolbar.css
- help.html
- Image Resources

The NetBeans project of the WebViewSample application can be found in `WebViewSample.zip`.

WebViewSample.java

```
/*
 * 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.
 */

package webviewsample;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.collections.ListChangeListener.Change;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.print.PrinterJob;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.Hyperlink;
import javafx.scene.control.MenuItem;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.PopupFeatures;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebHistory;
import javafx.scene.web.WebHistory.Entry;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

public class WebViewSample extends Application {

    private Scene scene;

    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View Sample");
        scene = new Scene(new Browser(stage), 900, 600, Color.web("#666970"));
        stage.setScene(scene);
        // apply CSS style
        scene.getStylesheets().add("webviewsample/BrowserToolbar.css");
        // show stage
        stage.show();
    }

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

class Browser extends Region {
```

```
private final HBox toolBar;
final private static String[] imageFiles = new String[]{
    "product.png",
    "blog.png",
    "documentation.png",
    "partners.png",
    "help.png"
};
final private static String[] captions = new String[]{
    "Products",
    "Blogs",
    "Documentation",
    "Partners",
    "Help"
};
final private static String[] urls = new String[]{
    "http://www.oracle.com/products/index.html",
    "http://blogs.oracle.com/",
    "http://docs.oracle.com/javase/index.html",
    "http://www.oracle.com/partners/index.html",
    WebViewSample.class.getResource("help.html").toExternalForm()
};
final ImageView selectedImage = new ImageView();
final Hyperlink[] hpls = new Hyperlink[captions.length];
final Image[] images = new Image[imageFiles.length];
final WebView browser = new WebView();
final WebEngine webEngine = browser.getEngine();
final Button toggleHelpTopics = new Button("Toggle Help Topics");
final WebView smallView = new WebView();
final ComboBox comboBox = new ComboBox();
private boolean needDocumentationButton = false;


public Browser(final Stage stage) {
    //apply the styles
    getStyleClass().add("browser");

    for (int i = 0; i < captions.length; i++) {
        // create hyperlinks
        Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
        Image image = images[i]
                = new Image(getClass().getResourceAsStream(imageFiles[i]));
        hpl.setGraphic(new ImageView(image));
        final String url = urls[i];
        final boolean addButton = (hpl.getText().equals("Help"));

        // process event
        hpl.setOnAction((ActionEvent e) -> {
            needDocumentationButton = addButton;
            webEngine.load(url);
        });

    }


    comboBox.setPrefWidth(60);

    // create the toolbar
    toolBar = new HBox();
```

```
toolBar.setAlignment(Pos.CENTER);
toolBar.getStyleClass().add("browser-toolbar");
toolBar.getChildren().add(comboBox);
toolBar.getChildren().addAll(hpls);
toolBar.getChildren().add(createSpacer());

//set action for the button
toggleHelpTopics.setOnAction((ActionEvent t) -> {
    webEngine.executeScript("toggle_visibility('help_topics')");
});

smallView.setPrefSize(120, 80);

//handle popup windows
webEngine.setCreatePopupHandler(
        (PopupFeatures config) -> {
            smallView.setFontScale(0.8);
            if (!toolBar.getChildren().contains(smallView)) {
                toolBar.getChildren().add(smallView);
            }
            return smallView.getEngine();
});

//process history
final WebHistory history = webEngine.getHistory();
history.getEntries().addListener(
    (Change<? extends Entry> c) -> {
        c.next();
        c.getRemoved().stream().forEach((e) -> {
        comboBox.getItems().remove(e.getUrl());
    });
        c.getAddedSubList().stream().forEach((e) -> {
        comboBox.getItems().add(e.getUrl());
    });
});

//set the behavior for the history combobox
comboBox.setOnAction((Event ev) -> {
    int offset
            = comboBox.getSelectionModel().getSelectedIndex()
            - history.getCurrentIndex();
    history.go(offset);
});

// process page loading
webEngine.getLoadWorker().stateProperty().addListener(
    (ObservableValue<? extends State> ov, State oldState,
        State newState) -> {
            toolBar.getChildren().remove(toggleHelpTopics);
            if (newState == State.SUCCEEDED) {
                JSObject win
                        = (JSObject) webEngine.executeScript("window");
                win.setMember("app", new JavaApp());
                if (needDocumentationButton) {
                    toolBar.getChildren().add(toggleHelpTopics);
                }
            }
});
//adding context menu
final ContextMenu cm = new ContextMenu();
```

```
        MenuItem cmItem1 = new MenuItem("Print");
        cm.getItems().add(cmItem1);
        toolBar.addEventHandler(MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
            if (e.getButton() == MouseButton.SECONDARY) {
                cm.show(toolBar, e.getScreenX(), e.getScreenY());
            }
        });

        //processing print job
        cmItem1.setOnAction((ActionEvent e) -> {
            PrinterJob job = PrinterJob.createPrinterJob();
            if (job != null) {
                webEngine.print(job);
                job.endJob();
            }
        });

        // load the home page
        webEngine.load("http://www.oracle.com/products/index.html");

        //add components
        getChildren().add(toolBar);
        getChildren().add(browser);
    }

    // JavaScript interface object
    public class JavaApp {

        public void exit() {
            Platform.exit();
        }
    }

    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }

    @Override
    protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        double tbHeight = toolBar.prefHeight(w);
        layoutInArea(browser,0,0,w,h-tbHeight,0,HPos.CENTER,VPos.CENTER);
        layoutInArea(toolBar,0,h-tbHeight,w,tbHeight,0,HPos.CENTER,VPos.CENTER);
    }

    @Override
    protected double computePrefWidth(double height) {
        return 900;
    }

    @Override
    protected double computePrefHeight(double width) {
        return 600;
    }
}
```

BrowserToolbar.css

```
/*
 * 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.
 */
.browser{
    -fx-background-color: #666970;
}
.browser-toolbar .hyperlink, .browser-toolbar .button, .browser-toolbar{
    -fx-text-fill: white;
}
.browser-toolbar{
    -fx-base: #505359;
    -fx-background: #505359;
    -fx-shadow-highlight-color: transparent;
    -fx-spacing: 5;
    -fx-padding: 4 4 4 4;
}
```

help.html

```
<!--
 * 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.
-->
<html lang="en">
    <head>
        <!-- Visibility toggle script -->
        <script type="text/javascript">
            <!--
            function toggle_visibility(id) {
                var e = document.getElementById(id);
                if (e.style.display == 'block')
                    e.style.display = 'none';
                else
                    e.style.display = 'block';
            }
//-->
        </script>
    </head>
    <body>
        <h1>Online Help</h1>
        <p class="boxtitle"><a href="#" onclick="toggle_visibility('help_
topics');"
  class="boxtitle">[+] Show/Hide Help Topics</a></p>
        <ul id="help_topics" style='display:none;'>
            <li>Products - Extensive overview of Oracle hardware and software
products,
                and summary of Oracle consulting, support, and educational
services. </li>
            <li>Blogs - Oracle blogging community.</li>
            <li>Documentation - Landing page to start learning Java. The page
contains
                links to the Java tutorials, developer guides, and API
documentation.</li>
            <li>Partners - Oracle partner solutions and programs. Popular
resources and
                membership opportunities.</li>
        </ul>
     <p><a href="about:blank" onclick="app.exit()">Exit the Application</a></p>
    </body>
</html>
```

Image Resources

The following images can be used to build the toolbar icons in the WebViewSample application.

Table A–1

File Name	Image
product.png	
blog.png	
documentation.png	
partners.png	
help.png	