JavaFX

Adding HTML Content to JavaFX Applications

Release 2.2

E20487-08

January 2014

ORACLE

JavaFX/Adding HTML Content to JavaFX Applications, Release 2.2

E20487-08

Primary Author: Alla Redko

Contributor: Peter Zhelezniakov

Contents

1 Adding HTML Content to JavaFX Applications

1

Adding HTML Content to JavaFX Applications

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

Supported Features of HTML5

The current implementation of the WebView component supports the following HTML5 features:

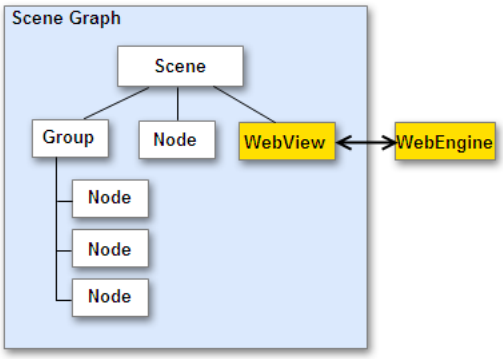
- Canvas
- Media Playback
- Form controls (except for `<input type="color">`)
- Editable content
- History maintenance
- Support for the `<meter>` and `<progress>` tags.
- Support for the `<details>` and `<summary>` tags.
- DOM
- SVG
- Support for domain names written in national languages

Embedded Browser API

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 1–1 shows the architecture of the embedded browser and how it relates to other JavaFX classes.

Figure 1–1 Architecture of the Embedded Browser



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.

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

Example 1–1 Creating WebView and WebEngine Objects

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

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 1–2.

Example 1–2 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.

Adding WebView to the Application Scene

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.

Example 1–3 shows how to add the `WebView` component to the application scene.

Example 1–3 Creating a Browser

```
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(),750,500, 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 750;
    }

    @Override protected double computePrefHeight(double width) {
        return 500;
    }
}
```

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.

The `createSpacer`, `layoutChildren`, `computePrefWidth`, and `computePrefHeight` methods perform layout of the `WebView` object and the control elements in the application toolbar.

When you add, compile, and run this code fragment, it produces the application window shown in Figure 1–2.

Figure 1–2 WebView Object in an Application Scene

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 1–4. 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 1–4 Creating a Toolbar

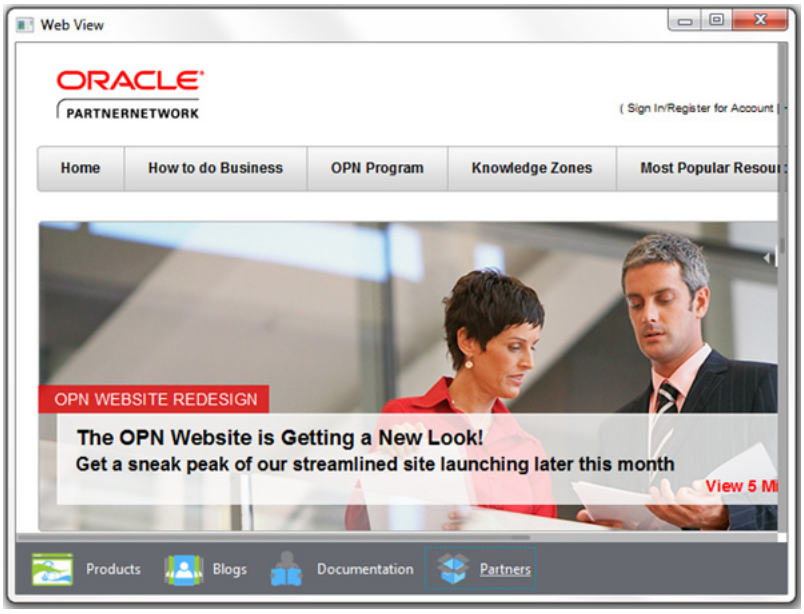
```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Hyperlink;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
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(),750,500, 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;

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

    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 750;
    }

    @Override protected double computePrefHeight(double width) {
        return 500;
    }
}
```

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 1–3.

Figure 1–3 Embedded Browser with the Navigation Toolbar



Processing JavaScript Commands

The `WebEngine` class provides the `executeScript` method to run a particular JavaScript command for the document currently loaded into the browser.

The modified application code in Example 1–5 creates an additional button to hide and show the Java SE documentation for the previous releases. The button is added to the toolbar only when the Documentation page is selected.

Example 1–5 Adding the Toggle Previous Docs button

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.event.EventHandler;
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 {
```

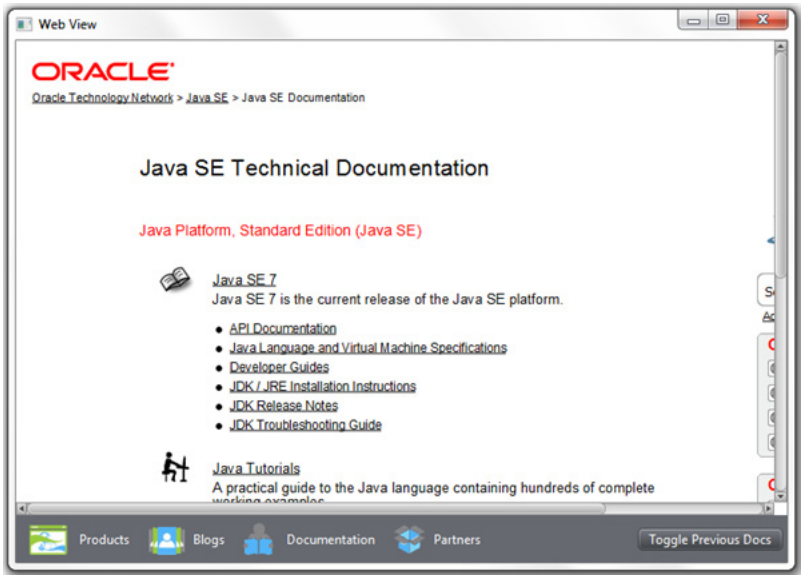
```
    private Scene scene;

    @Override
    public void start(Stage stage) {
        // create scene
        stage.setTitle("Web View");
        scene = new Scene(new Browser(), 750, 500, 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 HBox toolBar;
    private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
    };
    private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
    };
    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();
    final Button showPrevDoc = new Button("Toggle Previous Docs");
    private boolean needDocumentationButton = false;

    public Browser() {
        //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("Documentation"));

            // process event
            hpl.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(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
        showPrevDoc.setOnAction(new EventHandler() {
            @Override
            public void handle(Event t) {
                webEngine.executeScript("toggleDisplay('PrevRel')");
            }
        });

        // process page loading
        webEngine.getLoadWorker().stateProperty().addListener(
            new ChangeListener<State>() {
                @Override
                public void changed(ObservableValue<? extends State> ov,
                    State oldState, State newState) {
                        toolBar.getChildren().remove(showPrevDoc);
                        if (newState == State.SUCCEEDED) {
                            if (needDocumentationButton) {
                                toolBar.getChildren().add(showPrevDoc);
                            }
                        }
                    }
                }
        );

        // 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 750;
    }

    @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 Documentation page is `SUCCEEDED`, the Toggle Previous Docs button is added to the toolbar, as shown in Figure 1–4.

Figure 1–4 Toggle Previous Docs Button



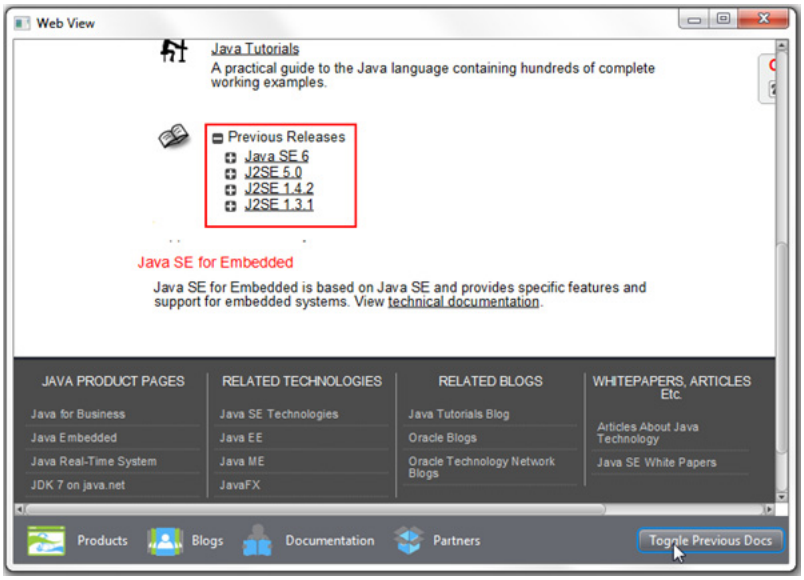
The `setOnAction` method shown in Example 1–6 defines behavior for the Toggle Previous Docs button.

Example 1–6 Executing a JavaScript Command

```
showPrevDoc.setOnAction(new EventHandler() {
    @Override
    public void handle(Event t) {
        webEngine.executeScript("toggleDisplay('PrevRel')");
    }
});
```

When the user clicks the Toggle Previous Doc button, the `executeScript` method runs the `toggleDisplay` JavaScript function for the Documentation page, and the documents for the previous Java releases appear, as shown in Figure 1–5. When the user performs another click, the `toggleDisplay` function hides the lists of the documents.

Figure 1–5 Showing the Java SE Documentation



Making Upcalls from JavaScript to JavaFX

Now that you know how to invoke JavaScript from JavaFX, 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.

For the WebViewSample application, you create the Help toolbar item that leads to the help.html file, where a user can preview reference material about Oracle websites. 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 1–7, to implement this functionality.

Example 1–7 Closing JavaFX Application by Using JavaScript

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.event.EventHandler;
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");
        scene = new Scene(new Browser(), 750, 500, 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 HBox toolBar;
    private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
        "Help"
    };
    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 showPrevDoc = new Button("Toggle Previous Docs");
private boolean needDocumentationButton = false;

public Browser() {
    //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("Documentation"));

        // process event
        hpl.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(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
    showPrevDoc.setOnAction(new EventHandler() {
        @Override
        public void handle(Event t) {
            webEngine.executeScript("toggleDisplay('PrevRel')");
        }
    });

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

```
                    }
                }
            );

            // 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 750;
        }

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

Examine the bold lines in Example 1–7. 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 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()`. See Example 1–8 to evaluate how this call is implemented in the help.html file.

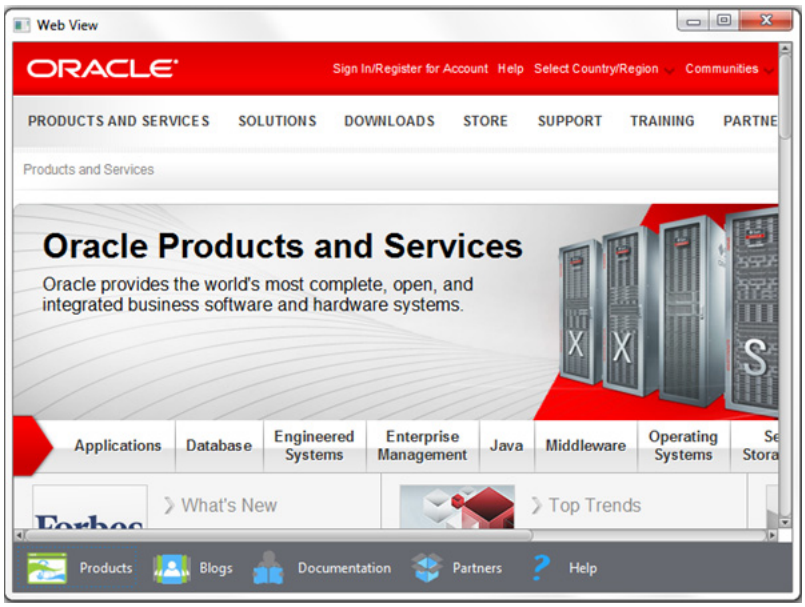
Example 1–8 Making a Call to the WebViewSample Application from the Help File

```
<html lang="en">
<body>
```

```
<p>Help</p>
<ul>
  <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.
      various Oracle products and solution.</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>
```

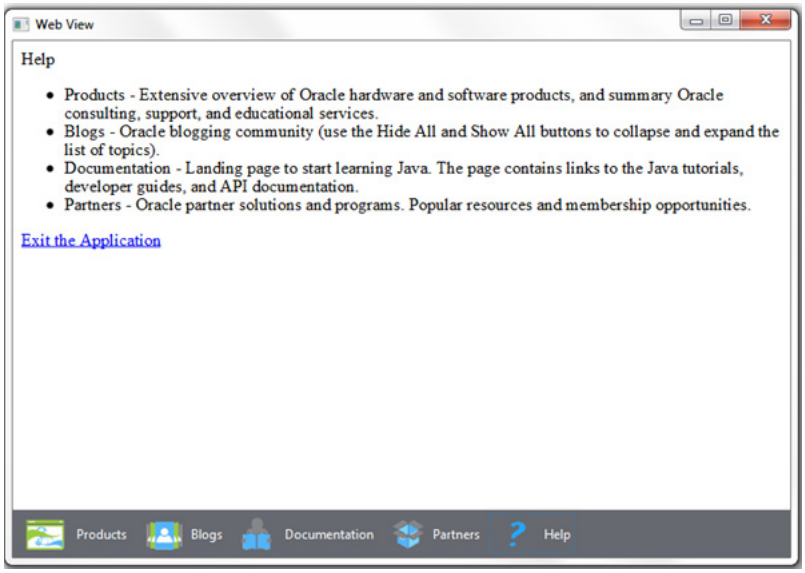
When you compile and run the WebViewSample application, the new icon appears, as shown in Figure 1–6.

Figure 1–6 Help Icon



Click Help to load the help.html file. Examine the content of the file, then click the Exit the Application link, shown in Figure 1–7, to close the WebViewSample application.

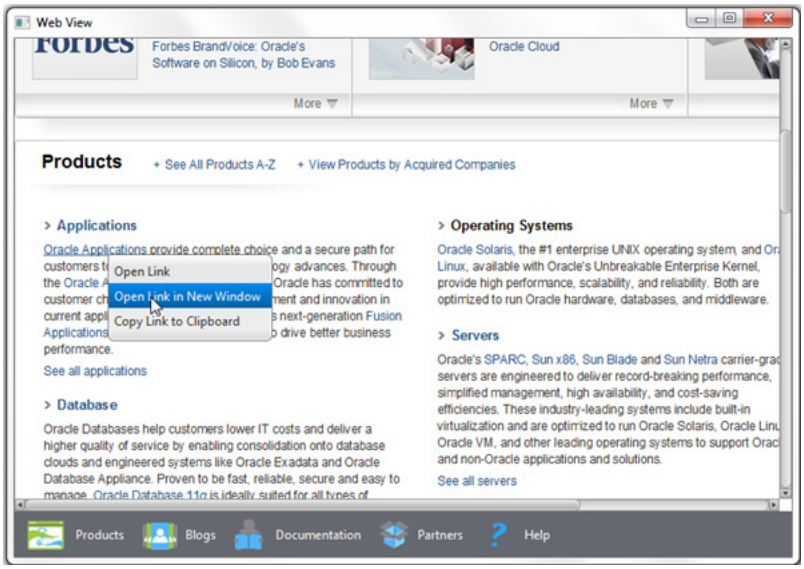
Figure 1–7 Help.html file



Managing Web Pop-Up Windows

You can set an alternative `WebView` object for the documents that will be opened in a separate window. Figure 1–8 shows a context menu a user can open by right-clicking any link.

Figure 1–8 Pop-Up Window



To specify a new browser window for the target document, use the `PopupFeatures` instance as shown in the modified application code in Example 1–9.

Example 1–9 Processing a Command of a Pop-Up Window

```
import javafx.application.Application;
```

```
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.event.EventHandler;
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 javafx.util.Callback;
import netscape.javascript.JSObject;

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(), 750, 500, 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 HBox toolBar;
    private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    private static String[] captions = new String[]{
        "Products",
```

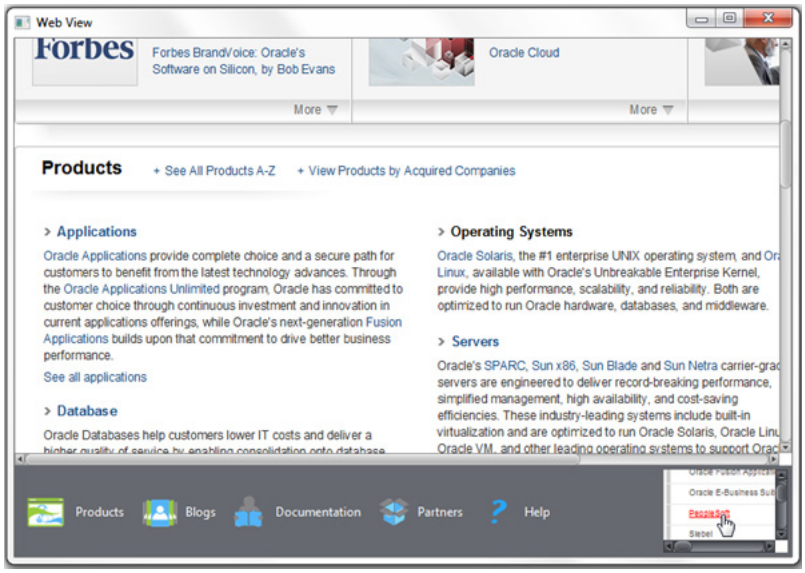
```
    "Blogs",
    "Documentation",
    "Partners",
    "Help"
};
private static String[] urls = new String[]{
    "http://www.oracle.com/products/index.html",
    "http://blogs.oracle.com/",
    "http://docs.oracle.com/javase/index.html",
    "http://www.oracle.com/partners/index.html",
    WebViewSample.class.getResource("help.html").toExternalForm()
};
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 showPrevDoc = new Button("Toggle Previous Docs");
final WebView smallView = new WebView();
private boolean needDocumentationButton = false;

public Browser() {
    //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("Documentation"));

        // process event
        hpl.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(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
    showPrevDoc.setOnAction(new EventHandler() {
        @Override
        public void handle(Event t) {
            webEngine.executeScript("toggleDisplay('PrevRel')");
        }
    });
```

```
        smallView.setPrefSize(120, 80);

        //handle popup windows
        webEngine.setCreatePopupHandler(
            new Callback<PopupFeatures, WebEngine>() {
                @Override public WebEngine call(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(
            new ChangeListener<State>() {
                @Override
                public void changed(ObservableValue<? extends State> ov,
                    State oldState, State newState) {
                    toolBar.getChildren().remove(showPrevDoc);
                    if (newState == State.SUCCEEDED) {
                            JSObject win =
                                (JSObject) webEngine.executeScript("window");
                            win.setMember("app", new JavaApp());
                            if (needDocumentationButton) {
                                toolBar.getChildren().add(showPrevDoc);
                            }
                        }
                    }
                }
        );

        // 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 750;
        }

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

When a user selects the Open option from the pop-up window, 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 1–9.

Figure 1–9 Small Preview Browser



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);`.

Managing Web History

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.

You typically use a standard or custom UI control to display the history list. Example 1–10 shows how to obtain a history items and present them in the `ComboBox` control.

Example 1–10 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 1–11 shows the complete code of the modified application.

Example 1–11 WebViewSample with the History Combo Box

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.ListChangeListener;
import javafx.collections.ListChangeListener.Change;
import javafx.concurrent.Worker.State;
import javafx.event.ActionEvent;
import javafx.event.Event;
```

```
import javafx.event.EventHandler;
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.WebHistory.Entry;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import javafx.util.Callback;
import netscape.javascript.JSObject;


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(), 750, 500, 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 HBox toolBar;
    private static String[] imageFiles = new String[]{
        "product.png",
        "blog.png",
        "documentation.png",
        "partners.png",
        "help.png"
    };
    private static String[] captions = new String[]{
        "Products",
        "Blogs",
        "Documentation",
        "Partners",
```

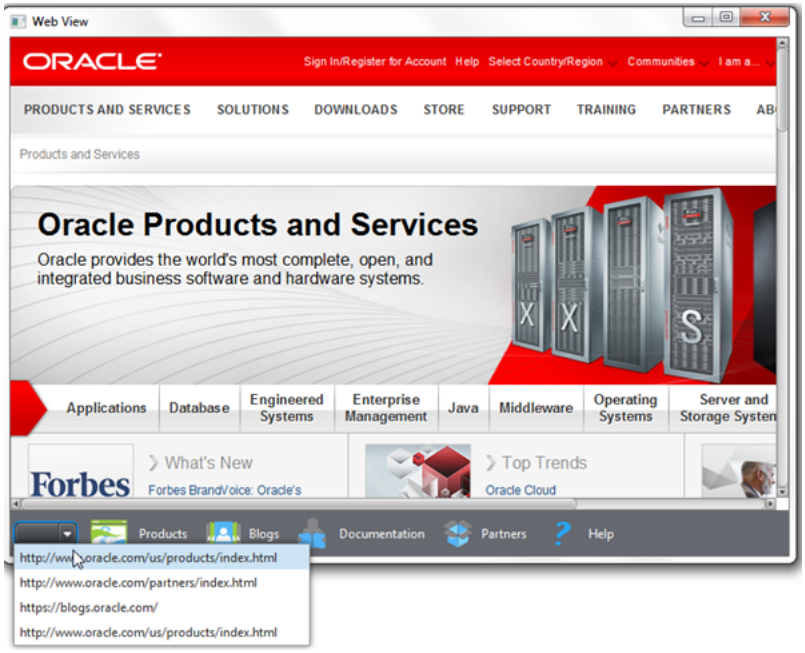
```
    "Help"
};
private static String[] urls = new String[]{
    "http://www.oracle.com/products/index.html",
    "http://blogs.oracle.com/",
    "http://docs.oracle.com/javase/index.html",
    "http://www.oracle.com/partners/index.html",
    WebViewSample.class.getResource("help.html").toExternalForm()
};
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 showPrevDoc = new Button("Toggle Previous Docs");
final WebView smallView = new WebView();
final ComboBox comboBox = new ComboBox();
private boolean needDocumentationButton = false;

public Browser() {
    //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("Documentation"));

        // process event
        hpl.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(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
    showPrevDoc.setOnAction(new EventHandler() {
        @Override
        public void handle(Event t) {
            webEngine.executeScript("toggleDisplay('PrevRel')");
        }
    });
```

```
smallView.setPrefSize(120, 80);

//handle popup windows
webEngine.setCreatePopupHandler(
    new Callback<PopupFeatures, WebEngine>() {
        @Override public WebEngine call(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(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());
            }
        }
});

//set the behavior for the history combobox
comboBox.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent ev) {
        int offset =
                comboBox.getSelectionModel().getSelectedIndex()
                - history.getCurrentIndex();
        history.go(offset);
    }
});


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

```
        );

        // 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 750;
    }

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

When you compile and run the application, it produces the window shown in Figure 1–10.

Figure 1–10 Selecting URL from the History Combo Box



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, and even add more `WebView` instances.

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`