This tutorial describes the JavaFX media functionality available through the Java APIs for JavaFX, including the formats of media files that are currently supported.
## Contents

### Preface

- About This Tutorial .......................................................... v
- Audience ............................................................................ v
- Documentation Accessibility ............................................. v
- Related Documents .......................................................... v
- Conventions ........................................................................ vi

### 1 Introduction to JavaFX Media

- Supported Media Codecs .................................................. 1-1
- HTTP Live Streaming Support ........................................... 1-2
- Creating a Media Player ................................................... 1-3

### 2 Embedding Media Into a Web Page

- To Get Started .................................................................... 2-1
- Create the Application ..................................................... 2-1

### 3 Controlling Media Playback

- Creating Controls ............................................................. 3-1
- Add the Functional Logic Code .......................................... 3-4
- Modify the EmbeddedMediaPlayer.java .......................... 3-7
- Compile and Run the EmbeddedMedia ............................... 3-7
Preface

This preface describes the document accessibility features and conventions used in this tutorial - *Incorporating Media Assets Into JavaFX Applications*.

**About This Tutorial**

This tutorial describes the JavaFX media functionality available through the Java APIs for JavaFX, including the formats of media files that are currently supported.

The document contains the following chapters:

- Introduction to JavaFX Media
- Embedding Media Into a Web Page
- Controlling Media Playback

**Audience**

This document is intended for JavaFX developers.

**Documentation Accessibility**


**Access to Oracle Support**


**Related Documents**

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

- *What Is JavaFX?*
- *Getting Started with JavaFX*
The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Introduction to JavaFX Media

The active growth of media content on the web has made video and audio an essential part of rich Internet applications. The idea of broadening the horizons of traditional media usage led to the creation of the JavaFX media functionality that is available through a set of Java APIs. The `javafx.scene.media` package enables developers to create media applications that provide media playback in the desktop window or within a web page on supported platforms.

Figure 1–1 demonstrates a variety of possible media usages in JavaFX applications.

**Figure 1–1 Samples of Media Usages**

The operating systems and Java Runtime Environments (JREs) supported by JavaFX media features are the same as those listed in the Certified System Configurations page, which is linked from the Java SE download page at [http://www.oracle.com/technetwork/java/javase/downloads/](http://www.oracle.com/technetwork/java/javase/downloads/).

**Supported Media Codecs**

The formats currently supported are the following:

- **Audio:** MP3; AIFF containing uncompressed PCM; WAV containing uncompressed PCM; MPEG-4 multimedia container with Advanced Audio Coding (AAC) audio
- **Video:** FLV containing VP6 video and MP3 audio; MPEG-4 multimedia container with H.264/AVC (Advanced Video Coding) video compression
HTTP Live Streaming Support

The FLV container is supported by the media stack on the platforms supported by the JavaFX SDK. A single movie encoded in this format works seamlessly on supported platforms. Standard FLV MIME settings are required on the server side to enable media streaming.

The MPEG-4 multimedia container is also supported on all operating systems supported by the JavaFX SDK. On the Mac OS X and Windows 7 platforms, playback will be functional without requiring additional software. However, the Linux operating system and versions of Windows older than Windows 7 require the installation of readily available third party software packages, as documented in the Certified System Configurations page, which is linked from the Java SE download page at [http://www.oracle.com/technetwork/java/javase/downloads/](http://www.oracle.com/technetwork/java/javase/downloads/). AAC and H.264/AVC decoding have certain platform-dependent limitations, as described in the Release Notes available at [http://www.oracle.com/technetwork/java/javase/downloads/](http://www.oracle.com/technetwork/java/javase/downloads/).

Decoding of some audio and video compression types relies on operating system-specific media engines. The JavaFX media framework does not attempt to handle all multimedia container formats and media encodings supported by these native engines. Instead, the framework attempts to provide equivalent and well-tested functionality across all platforms on which JavaFX is supported.

Some of the features supported by the JavaFX media stack include the following:

- FLV container with MP3 and VP6
- MP3 audio
- MPEG-4 container with either AAC, H.264, or both
- HTTP, FILE protocol support
- Progressive download
- Seeking
- Buffer progress
- Playback functions (Play, Pause, Stop, Volume, Mute, Balance, Equalizer)

HTTP Live Streaming Support

With the addition of HTTP live streaming support, you can now download the playlist file and playback video or audio segments using JavaFX Media. Media players are now able to switch to alternate streams, as specified in the playlist file and based on network conditions. For a given stream, there is a playlist file and a set of segments into which the stream is broken. The stream can be either an MP3 raw stream or an MPEG-TS containing multiplexed AAC audio and H.264 video. The stream can be played on demand when the stream is a static file or played live when the stream is actually a live feed. In both cases, the stream can adjust its bit rate and for video, its resolution can be adjusted.

**Note:** You may not integrate the On2 VP6 video decoder in the design of a semiconductor or register transfer level (RTL) or any other similar level necessary for development of semiconductor implementation of the On2 VP6 video decoder.
Creating a Media Player

The JavaFX media concept is based on the following entities.

- **Media** – A media resource, containing information about the media, such as its source, resolution, and metadata
- **MediaPlayer** – The key component providing the controls for playing media
- **MediaView** – A Node object to support animation, translucency, and effects

Each element of the media functionality is available through the JavaFX API. Figure 1–2 shows the classes that reside in the javafx.scene.media package. These classes are interdependent and are used in combination to create an embedded media player.

![Figure 1–2 Classes in the javafx.scene.media Package](image)

The MediaPlayer class provides all the attributes and functions needed to control media playback. You can either set the AUTO_PLAY mode, call the play() function directly, or explicitly specify the number of times that the media should play. The VOLUME variable and the BALANCE variable can be used to adjust the volume level and left-right settings, respectively. The volume level range is from 0 to 1.0 (the maximum value). The balance range is continuous from -1.0 on the far left, 0 at the center, and 1.0 at the right.

The play(), stop(), and pause() functions control media playback. Additionally, a bundle of functions handles specific events when the player does one of the following:

- Buffers data
- Reaches the end of media
- Stalls because it has not received data fast enough to continue playing
- Encounters any of the errors defined in the MediaErrorEvent class

The MediaView class extends the Node class and provides a view of the media being played by the media player. It is responsible mostly for effects and transformations. Its mediaPlayer instance variable specifies the MediaPlayer object by which the
media is being played. Other Boolean attributes serve to apply the particular effect provided by the `Node` class, for example, to enable the media player to be rotated. For more information about the `javafx.scene.media` package, see the API documentation.
Embedding Media Into a Web Page

In this section, you’ll explore how to add animated media content to your web page by creating a simple media panel. To create a media player you need to implement the structure of the three nested objects that is shown in Figure 2–1.

**Figure 2–1  Structure of the Embedded Media Player**

To Get Started

You can build a JavaFX application using any development tool designed for creating a Java application. The tool used in this document is the NetBeans IDE. Do the following steps before continuing to build this document’s sample application that uses the JavaFX Media features:

2. If necessary, see the Getting Started with JavaFX document to get an overview of the JavaFX features and create simple JavaFX applications.

Create the Application

1. From the NetBeans IDE, set up your JavaFX project as follows:
   a. From the File menu, choose New Project.
   b. In the JavaFX application category, choose JavaFX Application. Click Next.
   c. Name the project EmbeddedMediaPlayer and ensure the Create Application Class field has the value of embeddedmediaplayer.EmbeddedMediaPlayer. Click Finish.
2. Copy the import statements in Example 2–1 and paste them in the EmbeddedMediaPlayer.java file, replacing all of the import statements that were automatically generated by the NetBeans IDE.

**Example 2–1  Replace Default Import Statements**

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

For now, ignore the warnings on the margin since the lines of code that use the Media classes will be added in the next few steps.

3. Specify the media file source to be used and the String variable by adding the lines in bold in Example 2–2. For this example, use the animated video located at download.oracle.com or specify your own file. Add the lines after the public class EmbeddedMediaPlayer line.

**Example 2–2  Specify the Media File Source**

```java
public class EmbeddedMediaPlayer extends Application {
```

4. Modify the start method so that it looks like Example 2–3. This is will create an empty scene with a group root node and dimension of 540 wide by 210 height.

**Example 2–3  Modify the start Method**

```java
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Embedded Media Player");
    Group root = new Group();
    Scene scene = new Scene(root, 540, 210);

    primaryStage.setScene(scene);
    primaryStage.sizeToScene();
    primaryStage.show();
}
```

5. Now, define the Media and the MediaPlayer objects by adding the code in Example 2–4 before the primaryStage.setScene(scene) line. Set the autoPlay variable to true so that the video can start immediately.

**Example 2–4  Add media and mediaPlayer Objects**

```java
// create media player
Media media = new Media(MEDIA_URL);
MediaPlayer mediaPlayer = new MediaPlayer(media);
mediaPlayer.setAutoPlay(true);
```

6. Define the MediaView object and add the media player to the Node-based viewer by copying the comment and two lines of code in Example 2–5 and pasting it right after the mediaPlayer.setAutoPlay(true) line.
Example 2–5 Define MediaView Object

```
// create mediaView and add media player to the viewer
MediaView mediaView = new MediaView(mediaPlayer);
((Group)scene.getRoot()).getChildren().add(mediaView);
```

7. Right-click on any whitespace and select Format to fix the line formatting after adding the lines of code.

8. Right-click the EmbeddedMediaPlayer project node in the Projects pane and select Clean and Build.

9. After a successful build, run your application by right-clicking the project node and selecting Run.

---

**Note:** If you are using the media file source used in this tutorial and you are running the application behind a firewall, you might need to set the application’s proxy in order for the application to be able to access the media source file. Right-click the EmbeddedMediaPlayer project node in the Project window, select Properties, and in the Project Properties dialog, select Run. Set the VM Options field with something similar to `-Dhttp.proxyHost=yourproxyhost.com -Dhttp.proxyPort=somePort#`, where `yourproxyhost.com` is your company’s proxy server and `somePort#` is a port number you are assigned to use.
In this section you create a full-functional media player with graphical UI elements that control the playback.

To create a media player you need to implement the structure of the three nested media objects, encode graphical controls, and add some logic for playback functions, as illustrated in the Figure 3–1 below.

Figure 3–1 Structure of Media Player with Playback Controls

You step through adding playback controls to the media player that you created in Chapter 2, “Embedding Media Into a Web Page”. If you haven’t already done so, complete that media player application before proceeding with the rest of this chapter. The media control panel you add consists of three elements: playButton, progress, and volumeControl.

Creating Controls

In this section you create a new JavaFX source file, MediaControl.java, that will contain the pane and UI controls for the play/pause, progress, and volume features.

1. With the EmbeddedMediaPlayer opened as the main project in the NetBeans IDE, create a new JavaFX file to add to the project.
   a. Use Ctrl+N or select File > New File from the IDE’s main menu.
   b. Select Category JavaFX and file type JavaFX Main class. Click Next.
Creating Controls

3-2

JavaFX Incorporating Media Assets with JavaFX Applications

1. Create a new JavaFX window project named "embeddedmediaplayer," and select the "Media Control" template. Give the project a class name of "MediaControl".

2. In the Name and Location dialog, type MediaControl in the Class Name field.

3. In the Package field, select embeddedmediaplayer from the drop-down list and click Finish.

4. In the MediaControl.java source file, delete all the lines after the package embeddedmediaplayer line.

5. Add the import statements that are shown in Example 3–1 to the top of the file.

Example 3–1 Import Statements to Add

```java
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.util.Duration;
```

6. Copy and paste the lines of code in Example 3–2 to create the BorderPane that will hold the controls.

Example 3–2 Add MediaControl Class Code

```java
public class MediaControl extends BorderPane {
    private MediaPlayer mp;
    private MediaView mediaView;
    private final boolean repeat = false;
    private boolean stopRequested = false;
    private boolean atEndOfMedia = false;
    private Duration duration;
    private Slider timeSlider;
    private Label playTime;
    private Slider volumeSlider;
    private HBox mediaBar;

    public MediaControl(final MediaPlayer mp) {
        this.mp = mp;
        setStyle("-fx-background-color: #bfc2c7; ");
        mediaView = new MediaView(mp);
        Pane mvPane = new Pane() {
            @Override
            public void layoutChildren() {
                super.layoutChildren();
                mediaView.getRegion().add(mediaView);
                mvPane.setStyle("-fx-background-color: black; ");
                mvPane.setCenter(mvPane);
            }
        }
        mvPane.getChildren().add(mediaView);
        mvPane.setStyle("-fx-background-color: black; ");
        mvPane.setCenter(mvPane);
    }
}
```

7. Copy the lines of code in Example 3–3 and paste them immediately after the line that says setCenter(mvPane). This code adds the Media toolbar and the Play button.

Example 3–3 Add Media Toolbar and Play Button

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

final Button playButton = new Button(">");
mediaBar.getChildren().add(playButton);
setBottom(mediaBar);
}

6. Add the import statements shown in Example 3–4 to the top of the list of import statements.

Example 3–4  Add More Import Statements

import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Button;

7. Add the remainder of the UI controls to the control pane. Put the lines of code in Example 3–5 after the mediaBar.getChildren().add(playButton); line and before the setBottom(mediaBar) line.

Example 3–5  Add the Rest of the UI Controls

// Add spacer
Label spacer = new Label("   ");
mediaBar.getChildren().add(spacer);

// Add Time label
Label timeLabel = new Label("Time: ");
mediaBar.getChildren().add(timeLabel);

// Add time slider
timeSlider = new Slider();
HBox.setHgrow(timeSlider, Priority.ALWAYS);
timeSlider.setMinWidth(50);
timeSlider.setMaxWidth(Double.MAX_VALUE);
mediaBar.getChildren().add(timeSlider);

// Add Play label
playTime = new Label();
playTime.setPrefWidth(130);
playTime.setMinWidth(50);
mediaBar.getChildren().add(playTime);

// Add the volume label
Label volumeLabel = new Label("Vol: ");
mediaBar.getChildren().add(volumeLabel);

// Add Volume slider
volumeSlider = new Slider();
volumeSlider.setPrefWidth(70);
volumeSlider.setMaxWidth(Region.USE_PREF_SIZE);
volumeSlider.setMinWidth(30);
mediaBar.getChildren().add(volumeSlider);

8. Add more import statements shown in Example 3–6 to the top of the file.
Add the Functional Logic Code

After all the controls have been created and added to the control panel, add the functional logic to manage the media playback and make your application interactive.

1. Add the event handler and listener for the Play button. Copy and paste the lines of code in Example 3–7 after the final Button playButton = new Button(">") line and before the mediaBar.getChildren().add(playButton) line.

Example 3–7 Add Play Button’s Event Handler and Listener

playButton.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        Status status = mp.getStatus();
        if (status == Status.UNKNOWN || status == Status.HALTED)
            return;
        if (status == Status.PAUSED || status == Status.READY || status == Status.STOPPED)
            // rewind the movie if we’re sitting at the end
            if (atEndOfMedia) {
                mp.seek(mp.getStartTime());
                atEndOfMedia = false;
            }
            mp.play();
        else
            mp.pause();
    }
});

2. The import statements used by the code you just added from Example 3–7 could have been added beforehand to avoid getting errors. But this time, to eliminate all of marked errors, press Ctrl+Shift+I or right-click anywhere and select Fix Imports. From the Fix All Imports dialog, select javafx.scene.media.MediaPlayer.Status, javafx.event.ActionEvent, and javafx.event.EventHandler from the drop-down menus. Click OK.

3. Add the following lines of code immediately after the lines of code you added from Example 3–7 and before the line that says mediaBar.getChildren().add(playButton). This code will handle the listener.

Example 3–8 Add Listener Code

    mp.currentTimeProperty().addListener(new InvalidationListener() {
        
    });
Add the Functional Logic Code

Controlling Media Playback

```java
public void invalidated(Observable ov) {
    updateValues();
}
}

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

mp.setOnPaused(new Runnable() {
    public void run() {
        System.out.println("onPaused");
        playButton.setText(">");
    }
});

mp.setOnReady(new Runnable() {
    public void run() {
        duration = mp.getMedia().getDuration();
        updateValues();
    }
});

mp.setCycleCount(repeat ? MediaPlayer.INDEFINITE : 1);
mp.setOnEndOfMedia(new Runnable() {
    public void run() {
        if (!repeat) {
            playButton.setText("*");
            stopRequested = true;
            atEndOfMedia = true;
        }
    }
});

Note that the errors that appear will be fixed by adding more code in the next steps.

4. Add listener for the time slider by adding the following code snippet after the line that says `timeSlider.setMaxWidth(Double.MAX_VALUE)` and before the line that says `mediaBar.getChildren().add(timeSlider).

**Example 3–9 Add Listener for Time Slider**

```java
timeSlider.valueProperty().addListener(new InvalidationListener() {
    public void invalidated(Observable ov) {
        if (timeSlider.isValueChanging()) {
            // multiply duration by percentage calculated by slider position
            mp.seek(duration.multiply(timeSlider.getValue() / 100.0));
        }
    }
});
```

5. Add listener for the volume slider control by adding the following code snippet after the line that says `volumeSlider.setMinWidth(30)` and before the `mediabar.getChildren().add(volumeSlider).`
Example 3–10 Add Listener for the Volume Control

```java
volumeSlider.valueProperty().addListener(new InvalidationListener() {
    public void invalidated(Observable ov) {
        if (volumeSlider.isValueChanging()) {
            mp.setVolume(volumeSlider.getValue() / 100.0);
        }
    }
});
```

6. Create Method `updateValues` used by the playback controls. Add it after the `public MediaControl()` method.

Example 3–11 Add `updateValues` Method

```java
protected void updateValues() {
    if (playTime != null && timeSlider != null && volumeSlider != null) {
        Platform.runLater(new Runnable() {
            public void run() {
                Duration currentTime = mp.getCurrentTime();
                playTime.setText(formatTime(currentTime, duration));
                timeSlider.setDisable(duration.isUnknown());
                if (!timeSlider.isDisabled() 
                    && duration.greaterThan(Duration.ZERO) 
                    && !timeSlider.isValueChanging()) {
                    timeSlider.setValue(currentTime.divide(duration).toMillis() * 100.0);
                }
                if (!volumeSlider.isValueChanging()) {
                    volumeSlider.setValue((int)Math.round(mp.getVolume() * 100));
                }
            }
        });
    }
}
```

7. Add the private method `formatTime()` after the `updateValues()` method. The `formatTime()` method calculates the elapsed time the media has been playing and formats it to be displayed on the control toolbar.

Example 3–12 Add Method for Calculating Elapsed Time

```java
private static String formatTime(Duration elapsed, Duration duration) {
    int intElapsed = (int)Math.floor(elapsed.toSeconds());
    int elapsedHours = intElapsed / (60 * 60);
    if (elapsedHours > 0) {
        intElapsed -= elapsedHours * 60 * 60;
    }
    int elapsedMinutes = intElapsed / 60;
    int elapsedSeconds = intElapsed - elapsedHours * 60 * 60
        - elapsedMinutes * 60;
    if (duration.greaterThan(Duration.ZERO)) {
        int intDuration = (int)Math.floor(duration.toSeconds());
        int durationHours = intDuration / (60 * 60);
        if (durationHours > 0) {
            intDuration -= durationHours * 60 * 60;
        }
        int durationMinutes = intDuration / 60;
        int durationSeconds = intDuration - durationHours * 60 * 60
            - durationMinutes * 60;
        if (durationHours > 0) {
```
```java
return String.format("%d:%02d:%02d/%d:%02d:%02d", 
elapsedHours, elapsedMinutes, elapsedSeconds, 
durationHours, durationMinutes, durationSeconds);
} else {
    return String.format("%02d:%02d/%02d:%02d", 
elapsedMinutes, elapsedSeconds, durationMinutes, 
durationSeconds);
}
) else {
    if (elapsedHours > 0) {
        return String.format("%d:%02d:%02d", elapsedHours, 
elapsedMinutes, elapsedSeconds);
    } else {
        return String.format("%02d:%02d", elapsedMinutes, 
elapsedSeconds);
    }
}
}

8. Lastly, fix the imports. Right-click in any white space and select 
javaFX.application.Platform and javaFX.beans.Observable from the 
Fix All Imports dialog. Click OK.

Modify the EmbeddedMediaPlayer.java

To add the control, modify the EmbeddedMediaPlayer.java file that you created in 
the previous chapter and add the code to add the MediaControl object.

1. Copy the lines of code in Example 3–13 and paste them right after the 
mediaPlayer.setAutoPlay(true) line.

Example 3–13 Add the Source Code to Create MediaControl Object

   MediaControl mediaControl = new MediaControl(mediaPlayer);
   scene.setRoot(mediaControl);

2. Delete the three lines shown in Example 3–14, which previously created the 
mediaView and mediaPlayer objects.

Example 3–14 Delete Lines of Code

   // create mediaView and add media player to the viewer
   MediaView mediaView = new MediaView(mediaPlayer);
   ((Group)scene.getRoot()).getChildren().add(mediaView);

3. Delete the import statement for the MediaView:
   import javaFX.scene.media.MediaView;

4. Adjust the size of the scene’s height to accommodate the addition of the media 
controls.

Example 3–15 Change the Scene’s Height

   Scene scene = new Scene(root, 540, 241);

Compile and Run the EmbeddedMedia

Now build the application you just created in the previous section and run it.

1. Right-click the EmbeddedMediaPlayer project node and select Clean and Build.
2. If there are no build errors, right-click the node again and select Run. The media player with control appears, similar to Figure 3–2 and begins to play.

*Figure 3–2  Media Player with Playback Controls*

3. Stop and resume the video using the play/pause control button. Move forwards or backwards through the video using the progress bar. Adjust the volume using the volume control button.

Find the complete application code in the EmbeddedMediaPlayer.zip file.