Release: JavaFX 2.2
Last Updated: September 2013
Download as PDF

Profiles

Alla Redko
Technical Writer, Oracle

Alla is a technical writer for Oracle. She lives in St. Petersburg, Russia, and develops tutorials and technical articles for Java and JavaFX technologies. Prior to her assignment at Oracle, she worked as a technical writer in different IT companies.

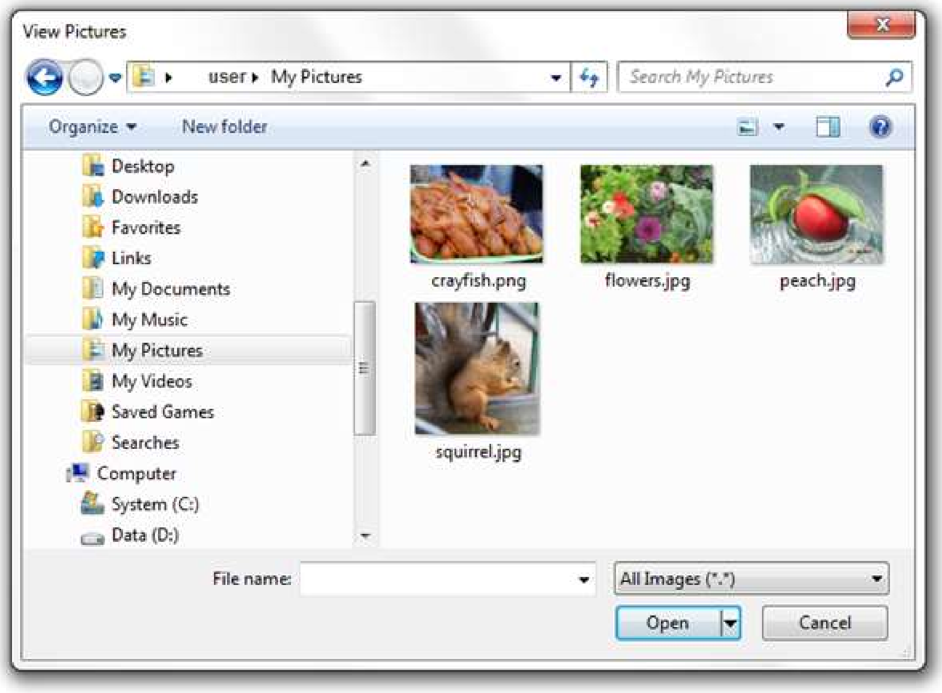
26 File Chooser

This chapter explains how to use the `FileChooser` class to enable users to navigate the file system. The samples provided in this chapter explain how to open one or several files, configure a file chooser dialog window, and save the application content.

Unlike other user interface component classes, the `FileChooser` class does not belong to the `javafx.scene.controls` package. However, this class deserves to be mentioned in the JavaFX UI Controls tutorial, because it supports one of the typical GUI application functions: file system navigation.

The `FileChooser` class is located in the `javafx.stage` package along with the other basic root graphical elements, such as `Stage`, `Window`, and `Popup`. The View Pictures window in Figure 26-1 is an example of the file chooser dialog in Windows.

Figure 26-1 Example of a File Chooser Window



Description of "Figure 26-1 Example of a File Chooser Window"

Opening Files

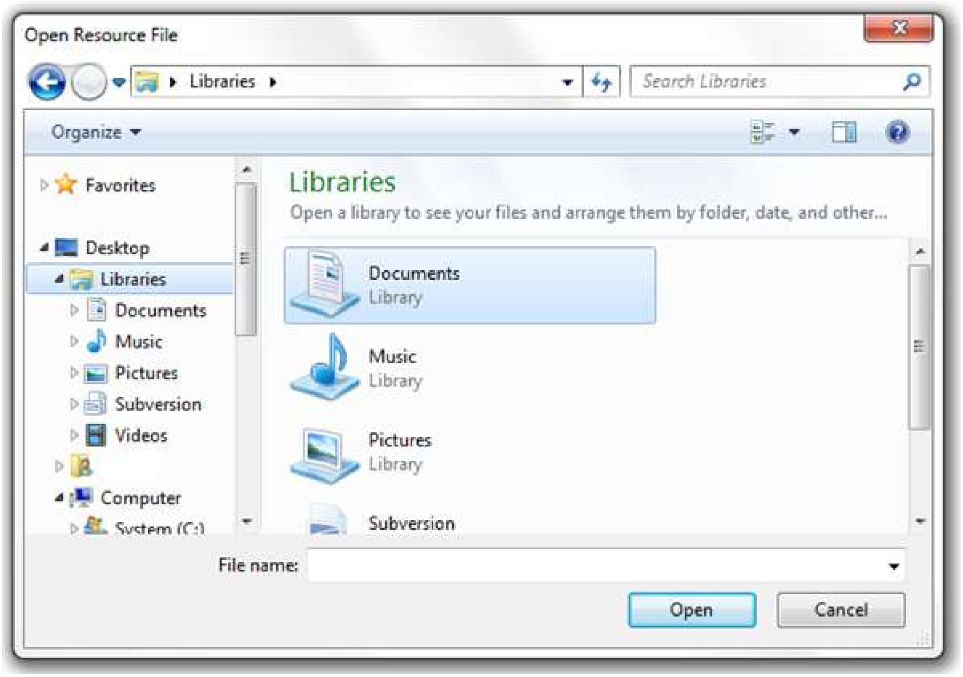
A file chooser can be used to invoke an open dialog window for selecting either a single file or multiple files, and to enable a file save dialog window. To display a file chooser, you typically use the `FileChooser` class. Example 26-1 provides the simplest way to enable a file chooser in your application.

Example 26-1 Showing a File Chooser

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Open Resource File");
fileChooser.showOpenDialog(stage);
```

After the code from Example 26-1 is added to a JavaFX application, the file chooser dialog window appears immediately when the application starts, as shown in Figure 26-2.

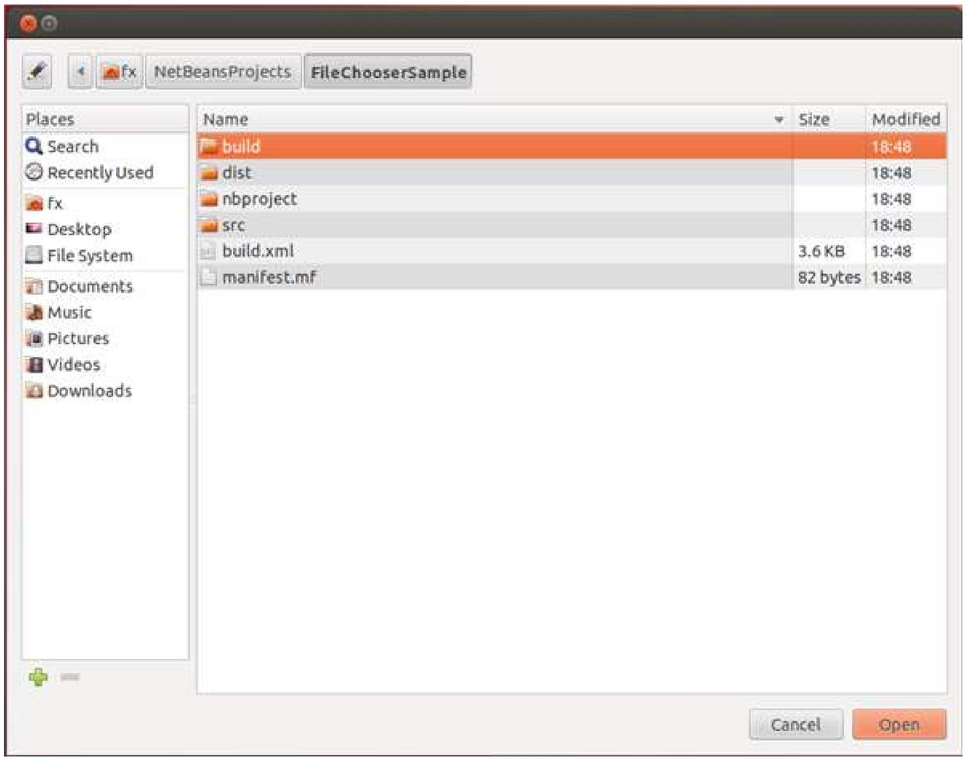
Figure 26-2 Simple File Chooser



Description of "Figure 26-2 Simple File Chooser"

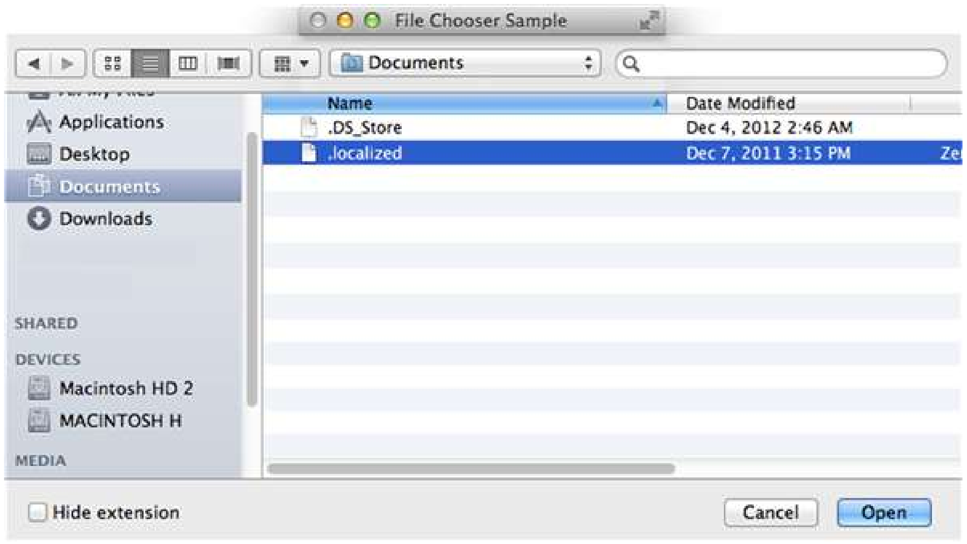
Note:
Figure 26-2 shows the file chooser in Windows. When you open file choosers in other operating systems that support this functionality, you receive alternative windows. Figure 26-3 and Figure 26-4 show examples of file chooser windows in Linux and Mac OS.

Figure 26-3 File Chooser Window in Linux



Description of "Figure 26-3 File Chooser Window in Linux"

Figure 26-4 File Chooser Window in Mac OS



Description of "Figure 26-4 File Chooser Window in Mac OS"

Although in the previous example the file chooser appears automatically when the application starts, a more typical approach is to invoke a file chooser by selecting the corresponding menu item or clicking a dedicated button. In this tutorial, you create an application that enables a user to click a button and open a one or more pictures located in the file system. Example 26-2 shows the code of the FileChooserSample application that implements this task.

Example 26-2 Opening File Chooser for Single and Multiple Selection

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();

        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(final ActionEvent e) {
                    File file = fileChooser.showOpenDialog(stage);
                    if (file != null) {
                        openFile(file);
                    }
                }
            });

        openMultipleButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(final ActionEvent e) {
                    List<File> list =
                        fileChooser.showOpenMultipleDialog(stage);
                    if (list != null) {
                        for (File file : list) {
                            openFile(file);
                        }
                    }
                }
            });


        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 0);
        GridPane.setConstraints(openMultipleButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

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

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

    private void openFile(File file) {
        try {
            desktop.open(file);
        } catch (IOException ex) {
            Logger.getLogger(
                FileChooserSample.class.getName()).log(
                    Level.SEVERE, null, ex
                );
        }
    }
}
```

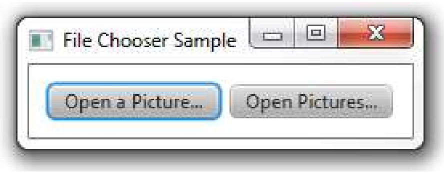
In Example 26-2, the Open a Picture button enables the user to open a file chooser for a single selection, and the Open Pictures button enables the user to open a file chooser for multiple selections. The `setOnAction` methods for these buttons are almost identical. The only difference is in the method that is used to invoke a `FileChooser`.

- The `showOpenDialog` method shows a new file open dialog in which one file can be selected. The method returns the value that specifies the file chosen by the user or `null` if no selection has been made.
- The `showOpenMultipleDialog` method shows a new file open dialog in which multiple files can be selected. The method returns the value that specifies the list of files chosen by the user or `null` if no selection has been made. The returned list cannot be modified and throws `UnsupportedOperationException` on each modification attempt.

Both methods do not return results until the displayed open dialog window is dismissed (in other words, until a user commits or cancels the selection).

When you compile and run the FileChooserSample application, it produces the window shown in Figure 26-5.

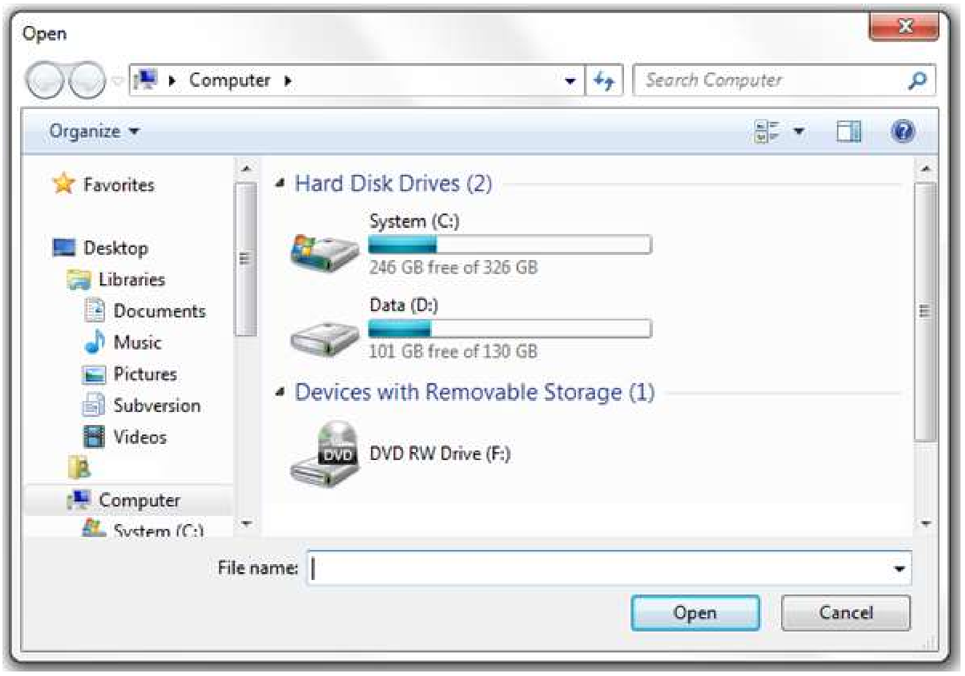
Figure 26-5 FileChooserSample with Two Buttons



Description of "Figure 26-5 FileChooserSample with Two Buttons"

When you click either of the buttons, the dialog window shown in Figure 26-6 appears. The opened file chooser dialog window shows the location that is the default for your operating system.

Figure 26-6 Default File Chooser Window



Description of "Figure 26-6 Default File Chooser Window"

Users of the FileChooserSample application may navigate to a directory containing pictures and select a picture. After a file is selected, it is opened with the associated application. The example code implements this by using the `open` method of the `java.awt.Desktop` class: `desktop.open(file);`.

Note:
Availability of the `Desktop` class is platform dependent. Refer to API documentation for more information on the `Desktop` class. You can also use the `isDesktopSupported()` method to check if it is supported in your system.

You can improve the user experience for this application by setting the file chooser directory to the specific directory that contains the pictures.

Configuring a File Chooser

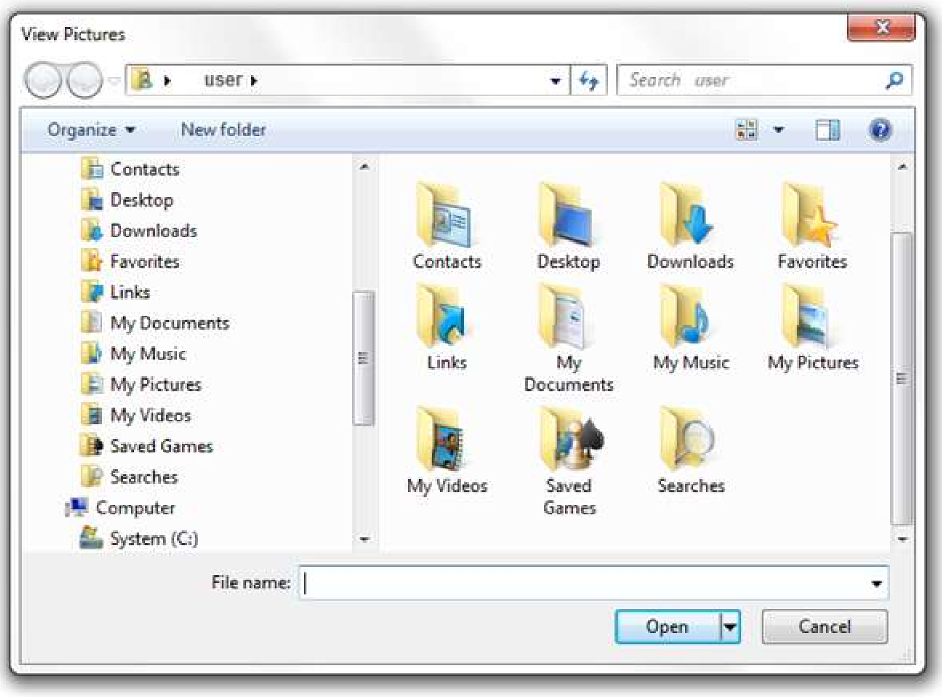
You can configure the file chooser dialog window by setting the `initialDirectory` and `title` properties of a `FileChooser` object. Example 26-3 shows how to specify the initial directory and a suitable title to preview and open pictures.

Example 26-3 Setting the Initial Directory and Window Title

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();

        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(final ActionEvent e) {
                    configureFileChooser(fileChooser);
                    File file = fileChooser.showOpenDialog(stage);
                    if (file != null) {
                        openFile(file);
                    }
                }
            });

        openMultipleButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(final ActionEvent e) {
                    configureFileChooser(fileChooser);
                    List<File> list =
                        fileChooser.showOpenMultipleDialog(stage);
                    if (list != null) {
                        for (File file : list) {
                            openFile(file);
                        }
                    }
                }
            });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 0);
        GridPane.setConstraints(openMultipleButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

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

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

        private static void configureFileChooser(final FileChooser fileChooser){
        fileChooser.setTitle("View Pictures");
        fileChooser.setInitialDirectory(
            new File(System.getProperty("user.home"))
        );
    }


    private void openFile(File file) {
        try {
            desktop.open(file);
        } catch (IOException ex) {
            Logger.getLogger(
                FileChooserSample.class.getName()).log(
                    Level.SEVERE, null, ex
                );
        }
    }
}
```

The `configureFileChooser` method sets the View Pictures title and the path to the user home directory with the My Pictures sub-directory. When you compile and run the FileChooserSample and click one of the buttons, the file chooser shown in Figure 26-7 appears.

Figure 26-7 Opening a Pictures Library



Description of "Figure 26-7 Opening a Pictures Library"

You can also let the users specify the target directory by using the `DirectoryChooser` class. In the code fragment shown in Example 26-4, the click of the `browseButton` invokes the `directoryChooser.showDialog` method.

Example 26-4 Use of the DirectoryChooser Class

```
final Button browseButton = new Button("...");
browseButton.setOnAction(
    new EventHandler<ActionEvent>() {
        @Override
        public void handle(final ActionEvent e) {
            final DirectoryChooser directoryChooser =
                new DirectoryChooser();
            final File selectedDirectory =
                    directoryChooser.showDialog(stage);
            if (selectedDirectory != null) {
                selectedDirectory.getAbsolutePath();
            }
        }
    }
);
```

After the selection is performed, you can assign the corresponding value to the file chooser as follows: `fileChooser.setInitialDirectory(selectedDirectory);`.

Setting Extension Filters

As the next configuration option, you can set the extension filter to determine which files to open in a file chooser, as shown in Example 26-5.

Example 26-5 Setting an Image Type Filter

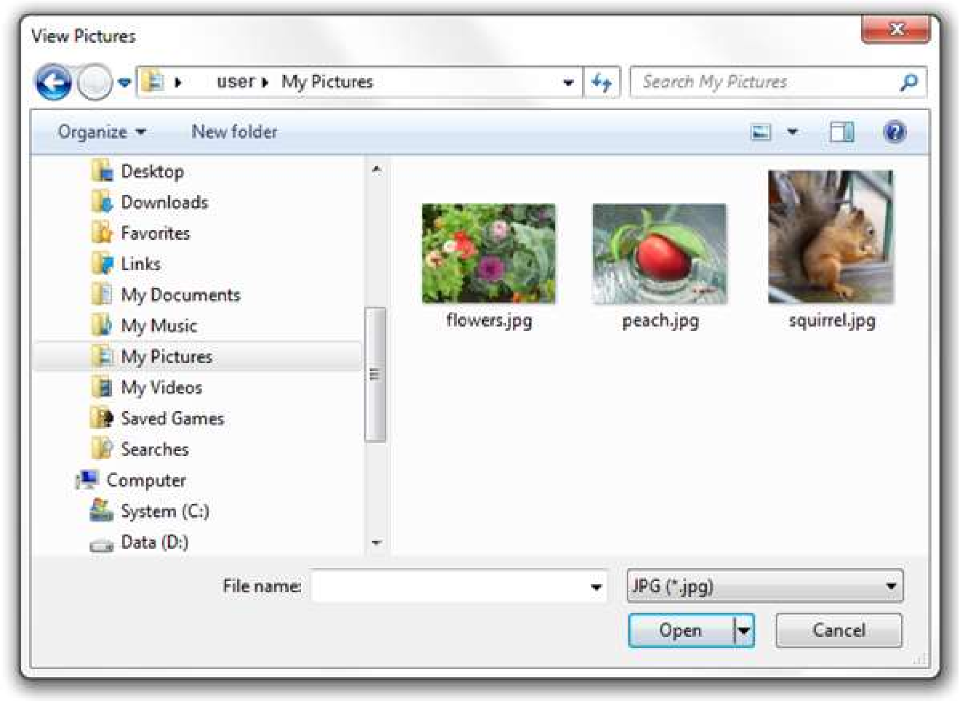
```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();
        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(final ActionEvent e) {
                    configureFileChooser(fileChooser);
                    File file = fileChooser.showOpenDialog(stage);
                    if (file != null) {
                        openFile(file);
                    }
                }
            });

        openMultipleButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(final ActionEvent e) {
                    configureFileChooser(fileChooser);
                    List<File> list =
                        fileChooser.showOpenMultipleDialog(stage);
                    if (list != null) {
                        for (File file : list) {
                            openFile(file);
                        }
                    }
                }
            });


        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 1);
        GridPane.setConstraints(openMultipleButton, 1, 1);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

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

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

    private static void configureFileChooser(
        final FileChooser fileChooser) {
            fileChooser.setTitle("View Pictures");
            fileChooser.setInitialDirectory(
                new File(System.getProperty("user.home"))
            );
            fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("All Images", "*.*"),
                new FileChooser.ExtensionFilter("JPG", "*.jpg"),
                new FileChooser.ExtensionFilter("PNG", "*.png")
            );
    }

    private void openFile(File file) {
        try {
            desktop.open(file);
        } catch (IOException ex) {
            Logger.getLogger(FileChooserSample.class.getName()).log(
                Level.SEVERE, null, ex
            );
        }
    }
}
```

In Example 26-5, you set an extension filter by using `FileChooser.ExtensionFilter` to define the following options for file selection: All images, JPG, and PNG.

When you compile, run the FileChooserSample code from Example 26-5, and click one of the buttons, the extension filters appear in the file chooser window. If a user selects JPG, then the file chooser displays only pictures of the JPG type.

Figure 26-8 captures the moment of selection JPG images in the My Pictures directory.

Figure 26-8 Filtering JPG Files in File Chooser



Description of "Figure 26-8 Filtering JPG Files in File Chooser"

Saving Files

In addition to opening and filtering files, the `FileChooser` API provides a capability to let a user specify a file name (and its location in the file system) for a file to be saved by the application. The `showSaveDialog` method of the `FileChooser` class opens a save dialog window. Like other show dialog methods, the `showSaveDialog` method returns the file chosen by the user or `null` if no selection has been performed.

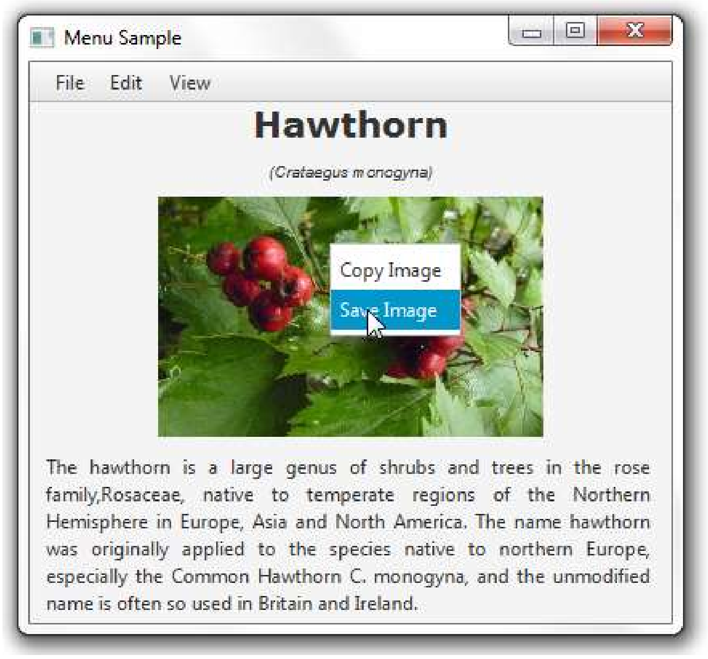
The code fragment shown in Example 26-6 is an addition to the Menu sample. It implements one more item of the context menu that saves the displayed image in the file system.

Example 26-6 Saving an Image with the FileChooser Class

```
MenuItem cmItem2 = new MenuItem("Save Image");
    cmItem2.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save Image");
            System.out.println(pic.getId());
            File file = fileChooser.showSaveDialog(stage);
            if (file != null) {
                try {
                    ImageIO.write(SwingFXUtils.fromFXImage(pic.getImage(),
                        null), "png", file);
                } catch (IOException ex) {
                    System.out.println(ex.getMessage());
                }
            }
            }
        }
);
```

When you add Example 26-6 to the MenuSample application (find the source code in the Application Files), compile and run it, you enable the Save Image menu item, as shown in Figure 26-9.

Figure 26-9 Saving Image

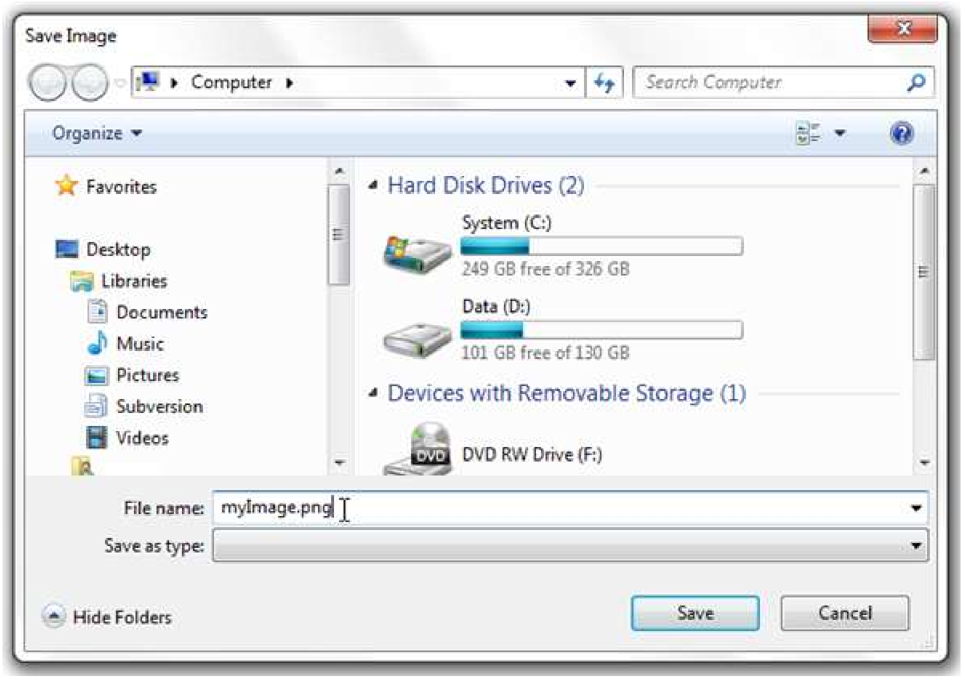


The hawthorn is a large genus of shrubs and trees in the rose family,Rosaceae, native to temperate regions of the Northern Hemisphere in Europe, Asia and North America. The name hawthorn was originally applied to the species native to northern Europe, especially the Common Hawthorn C. monogyna, and the unmodified name is often so used in Britain and Ireland.

Description of "Figure 26-9 Saving Image"

After a user selects the Save Image item, the Save Image window shown in Figure 26-10 appears.

Figure 26-10 The Save Image Window



File name: myImage.png
Save as type:

Description of "Figure 26-10 The Save Image Window"

The Save Image window corresponds to the typical user experience for the save dialog windows: the user needs to select the target directory, type in the name of the saving file, and click Save.

Related API Documentation

- `FileChooser`
- `DirectoryChooser`

Previous Page Next Page