Java FX Tutorials for NetBeans

**Table of Contents**

[JavaFX Setup 2](#_Toc57831097)

[JavaFX Project Setup (Short Method) 3](#_Toc57831098)

[JavaFX Project Setup (Long Method) 7](#_Toc57831099)

[Adding GUI Elements 9](#_Toc57831100)

[Adding Functionality 10](#_Toc57831101)

## JavaFX Setup

1. Before getting started, ensure that the latest version of the Java JDK is installed on your computer. You may run into problems otherwise.
2. Go to [this link](https://gluonhq.com/products/javafx/) and download JavaFX. Make sure to download the latest version (which is 15.0.1 as of now).
3. Extract the .zip file to reveal an SDK folder. Move this folder to the installation directory of the Java JDK (but not inside the JDK folder itself).
4. It is also highly recommended to download Scene Builder from [here](https://gluonhq.com/products/scene-builder/). This is not absolutely necessary, since all the work can be done with code, but Scene Builder makes editing things visually a lot easier, as we will see soon. Again, make sure to download and install the latest version.
5. NetBeans does not recognize the new things we just installed yet. We need to add them. Under the Tools tab at the top, go to the Libraries submenu. This will open the Ant Library Manager. Here,
   1. Make a new library and name it JavaFX 15. The library type should be Class Libraries. Click OK.
   2. Click on Add JAR/Folder and navigate to the SDK folder we extracted a while ago. Inside this folder, there should be a lib subfolder. Inside that, there should be a bunch of .jar files. Select all of those and then select Add JAR/Folder. Make sure to not select the src.zip file.
   3. Click on OK to close the Ant Library Manager.

## JavaFX Project Setup (Short Method)

1. Now we can use JavaFX with NetBeans. However, if we try to create a new project with JavaFX directly, we will run into an error. JavaFX is not technically supported by Oracle’s Open JDK right now. We need to bypass this.
2. Even though the previous step will not work, you may still face an odd problem with being unable to create FXML files later on if you do not attempt it.
   1. Create a new project, and under the Java with Ant submenu, select JavaFX and then select JavaFX FXML Application.
   2. You will face an error, Failed to automatically set up a JavaFX Platform. Once you have faced this error, click on Cancel.
3. To be able to use JavaFX, we now need a workaround. These steps need to be repeated every time we create a new project.
   1. First, we will create a Java with Ant Java Application as we normally do, but in the step where we give the project name, we will uncheck the Create Main Class checkbox.

Since we did not create a main class, we will not have any files under the project. We will only have a default package underneath Source Packages.

* 1. Right-click on the project name from the project tree on the left-hand side menu and go to Properties.
  2. Under the Libraries category, in the Compile tab, click the + symbol next to Classpath and select Add Library. Then add the JavaFX 15 library we created before this.
  3. Under the Libraries category, in the Run tab, click the + symbol next to Modulepath and select Add Library. Again, add the JavaFX 15 library we created.
  4. Under the Run category, go to VM Options and add this text:

--add-modules javafx.controls,javafx.fxml

* 1. We are done working with the project’s properties, so click OK.

1. Back to the left-hand menu, under the Source Packages subfolder in our project’s folder, create a new package. Give it the same name as the project.
2. From the left-hand side submenu, under the package for the project, right-click and select New and create a new Empty FXML. The FXML document is what will hold various GUI elements like checkboxes, buttons, etc. If you do not see this option in this submenu, go to the Other option and search for it. If you still cannot find it, you might have missed steps 1 and 2.
3. In the window for creating a new empty FXML file, give the file the name FXML. In the next step, make sure to check the Use Java Controller checkbox. In the next window, make sure to check the Use Cascading Style Sheets checkbox if you wish to use .css files. We shall not be using .css files for now, but they are super useful to style the components in our FXML document.
4. From the left-hand side submenu, under the package for the project, right-click and select New and create a new JavaFX Main Class file. Give the new file the same name as the project if this will be the main file for our project.
5. Once the file is created, there will be a large amount of code that may be difficult to understand. The only part we need to worry about right now is the start method. The code for that part should look somewhat like this:

public void start(Stage primaryStage)  
{  
 Button btn = new Button();  
 btn.setText("Say 'Hello World'");  
 btn.setOnAction(new EventHandler<ActionEvent>()  
 {  
 @Override  
 public void handle(ActionEvent event)  
 {  
 System.out.println("Hello World!");  
 }  
 });  
  
 StackPane root = new StackPane();  
 root.getChildren().add(btn);  
  
 Scene scene = new Scene(root, 300, 250);  
  
 primaryStage.setTitle("Hello World!");  
 primaryStage.setScene(scene);  
 primaryStage.show();  
}

JAVA

Essentially, a button is being created, an action is being assigned to the button to print some text when it is clicked, a ‘scene’ is being created and it is being given a title. For now, do not worry about what a scene is. Think of like the window on which we will be working.

1. The code in the start method needs to be changed to this:

public void start(Stage primaryStage) throws Exception  
{  
 Parent root = FXMLLoader.load(getClass().getResource("FXML.fxml"));  
 Scene scene = new Scene(root);  
 primaryStage.setTitle("Example Application");  
 primaryStage.setScene(scene);  
 primaryStage.show();  
}

JAVA

Notice that we just got rid of the button and its event, redirected the root so we can work with the FXML file we created and changed the scene so that it no longer has a specified size. Additionally, we had to include the throws Exception part since the FXML file is an external file.

There should be two import errors, one for Parent and one for FXMLLoader. Place the cursor on them and press Alt + Enter to get the option to automatically import these two classes. Note that you will not be able to import the FXMLLoader class without first having created the FXML file.

## JavaFX Project Setup (Long Method)

The first six steps are the same as the short method, so those are being skipped here.

1. From the left-hand side submenu, under the package for the project, right-click and select New and create a new Java Class file. Give the new file the same name as the project if this will be the main file for our project.
2. Whenever we create a new project using the above process, the first thing we have to do is extend the abstract Application class from our project’s default class. This class is what actually allows us to work with JavaFX. This would look something like this:

public class JavaExample extends Application {  
  
}

JAVA

1. When we do this, we will get an error telling us that the Application class is not recognized. Clicking on the error and pressing Alt + Enter should show and option to import javafx.application.Application.
2. Once this is done, we shall face yet another error, this time telling us that we did not override the start method from the Application class. The start method is what runs when we run a JavaFX application.

Again, click on the error and press Alt + Enter to reveal an option Implement all abstract methods. Clicking on this will add the required code that we need to override the start method as well as add an import statement to import the Stage class. What the Stage class is will be explained later.

1. Note that there is a throw statement that is added by default under the start method, UnsupportedOperationException. This exception is used to indicate that a called method is not yet implemented. Consider removing this if this is not needed.
2. We need to add a few more lines of code to the start method. It should look like this:

public void start(Stage primaryStage) throws Exception {  
 Parent root = FXMLLoader.load(getClass().getResource("FXML.fxml"));  
 Scene scene = new Scene(root);  
 primaryStage.setTitle("Example Application");  
 primaryStage.setScene(scene);  
 primaryStage.show();  
}

JAVA

Again, we will face import errors that we need to fix by using Alt + Enter.

1. If you followed the setup from the start, you do not have a main function as of yet. So, create that after the start method. Inside the main function, we just need to add one line:

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

JAVA

Essentially, we are launching the JavaFX application. All our main code for the JavaFX application will be under the start method, not the main method.

## Adding GUI Elements

GUI elements can be added via normal Java code, but here, we will be using Scene Builder. Scene Builder makes the process much easier.

1. Under the Tools tab at the top, go to Options and then to the Java tab. Under the JavaFX tab, there should be an option for Scene Builder Home. Browse to the folder in which Scene Builder was installed. By default, this is under C:\Program Files. This only needs to be done once, not for every project.
2. Now, from the left-hand side menu, if we double-click on the .fxml file, the contents will also open in Scene Builder. If we change anything in Scene Builder and save it, the code in the .fxml file will automatically be updated.

## Adding Functionality

The Controller we created is meant to hold all of the functions that will be called when we interact with the GUI elements we create in the FXML file.

For example, say we go into Scene Builder, and from the left-hand side menu, we add a button. If we save this now, the code for the button will automatically be added to the .fxml file. However, if we run the program and click on the button, obviously, nothing happens. This is because we did not define any behaviour for the button. This is where the Controller comes in.

Say we want to use the button to exit the program.

1. In Scene Builder, on the right-hand side menu, there is a Code tab. In this tab, we can assign an ID to the button under fx:id. Say we give it the ID exitButton.
2. Under On Action, we can define what function will be called when this button is clicked. Say we give the function handleExitButton. We have not created this function yet.
3. In the .fxml file, we should now see an error for the button, since it is meant to call a function that does not yet exist. To fix this, we need to update the Controller. From the left-hand side menu, right-click on the .fxml file and select Make Controller. This will cause the existing controller to be updated to reflect the changes we just made. This includes adding the handleExitButton function.

Of course, the body of this function is still empty, so clicking the button will still do nothing. Once we add the code to exit the application in the body of this function, the button will work as intended.

Note that the handleExitButton method created will be a private method by default. As such, it will not show up in the Javadoc. We can change this to public manually if we wish to.

1. We can run the program to view our work like we would run any other Java project, i.e. by pressing F6. Note that the first time the project is run, we will need to select where the main function is located. Make sure to save the corresponding file, since it will not appear on the list otherwise.