Connecting SceneBuilder edited FXML to Java code

By daniel on [Jun 25, 2012](https://blogs.oracle.com/jmxetc/entry/connecting_scenebuilder_edited_fxml_to)

Recently I had to answer several questions regarding how to connect an UI built with the JavaFX SceneBuilder 1.0 Developer Preview to Java Code. So I figured out that a short overview might be helpful. But first, let me state the obvious.

**What is FXML?**

To make it short, FXML is an XML based declaration format for JavaFX. JavaFX provides an FXML loader which will parse FXML files and from that construct a graph of Java object. It may sound complex when stated like that but it is actually quite simple. Here is an example of FXML file, which instantiate a StackPane and puts a Button inside it:

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.\*?>

<?import java.util.\*?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.paint.\*?>

<StackPane prefHeight="150.0" prefWidth="200.0" xmlns:fx="http://javafx.com/fxml">

<children>

<Button mnemonicParsing="false" text="Button" />

</children>

</StackPane>

... and here is the code I would have had to write if I had chosen to do the same thing programatically:

**import** **javafx.scene.control.\***;

**import** **javafx.scene.layout.\***;

...

**final** Button button = **new** Button("Button");

button.setMnemonicParsing(**false**);

**final** StackPane stackPane = **new** StackPane();

stackPane.setPrefWidth(**200.0**);

stackPane.setPrefHeight(**150.0**);

stacPane.getChildren().add(button);

As you can see - FXML is rather simple to understand - as it is quite close to the JavaFX API. So OK FXML is simple, but why would I use it?

Well, there are several answers to that - but my own favorite is: because you can make it with SceneBuilder.

**What is SceneBuilder?**

In short SceneBuilder is a layout tool that will let you graphically build JavaFX user interfaces by dragging and dropping JavaFX components from a library, and save it as an FXML file. SceneBuilder can also be used to load and modify JavaFX scenegraphs declared in FXML. Here is how I made the small FXML file above:

* Start the [JavaFX SceneBuilder 1.0 Developer Preview](http://www.oracle.com/technetwork/java/javafx/downloads/devpreview-1429449.html)
* In the Library on the left hand side, click on 'StackPane' and drag it on the content view (the white rectangle)
* In the Library, select a Button and drag it onto the StackPane on the content view.
* In the Hierarchy Panel on the left hand side - select the StackPane component, then invoke 'Edit > Trim To Selected' from the menubar

That's it - you can now save, and you will obtain the small FXML file shown above. Of course this is only a trivial sample, made for the sake of the example - and SceneBuilder will let you create much more complex UIs.

So, I have now an FXML file. But what do I do with it? How do I include it in my program? How do I write my main class?

**Loading an FXML file with JavaFX**

Well, that's the easy part - because the piece of code you need to write never changes. You can download and look at the [SceneBuilder samples](http://www.oracle.com/technetwork/java/javafx/downloads/devpreview-1429449.html) if you need to get convinced, but here is the short version:

* Create a Java class (let's call it 'Main.java') which extends javafx.application.Application
* In the same directory copy/save the FXML file you just created using SceneBuilder. Let's name it"simple.fxml"

Now here is the Java code for the Main class, which simply loads the FXML file and puts it as root in a stage's scene.

/\*

\* Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved.

\*/

**package** simple;

**import** **java.util.logging.Level**;

**import** **java.util.logging.Logger**;

**import** **javafx.application.Application**;

**import** **javafx.fxml.FXMLLoader**;

**import** **javafx.scene.Scene**;

**import** **javafx.scene.layout.StackPane**;

**import** **javafx.stage.Stage**;

**public** **class** **Main** **extends** Application {

/\*\*

\* @param args the command line arguments

\*/

**public** **static** **void** **main**(String[] args) {

Application.launch(Main.class, (java.lang.String[])**null**);

}

**@Override**

**public** **void** **start**(Stage primaryStage) {

**try** {

StackPane page = (StackPane) FXMLLoader.load(Main.class.getResource("simple.fxml"));

Scene scene = **new** Scene(page);

primaryStage.setScene(scene);

primaryStage.setTitle("FXML is Simple");

primaryStage.show();

} **catch** (Exception ex) {

Logger.getLogger(Main.class.getName()).log(Level.SEVERE, **null**, ex);

}

}

}

Great! Now I only have to use my favorite IDE to compile the class and run it. But... wait... what does it do? Well nothing. It just displays a button in the middle of a window. There's no logic attached to it. So how do we do that? How can I connect this button to my application logic? Here is how:

**Connection to code**

First let's define our application logic. Since this post is only intended to give a very brief overview - let's keep things simple. Let's say that the only thing I want to do is print a message on System.out when the user clicks on my button. To do that, I'll need to register an action handler with my button. And to do that, I'll need to somehow get a handle on my button. I'll need some kind of controller logic that will get my button and add my action handler to it. So how do I get a handle to my button and pass it to my controller? Once again - this is easy: I just need to write a controller class for my FXML.

With each FXML file, it is possible to associate a controller class defined for that FXML. That controller class will make the link between the UI (the objects defined in the FXML) and the application logic. To each object defined in FXML we can associate an fx:id. The value of the id must be unique within the scope of the FXML, and is the name of an instance variable inside the controller class, in which the object will be injected. Since I want to have access to my button, I will need to add an fx:id to my button in FXML, and declare an @FXML variable in my controller class with the same name. In other words - I will need to add fx:id="myButton" to my button in FXML:

<Button fx:id="myButton" mnemonicParsing="false" text="Button" />

and declare @FXML private Button myButton in my controller class

**@FXML**

**private** Button myButton; // value will be injected by the FXMLLoader

Let's see how to do this.

**Add an fx:id to the Button object**

* Load "simple.fxml" in SceneBuilder - if not already done
* In the hierarchy panel (bottom left), or directly on the content view, select the Button object.
* Open the Properties sections of the inspector (right panel) for the button object
* At the top of the section, you will see a text field labelled fx:id. Enter myButton in that field and validate.

**Associate a controller class with the FXML file**

Still in SceneBuilder, select the top root object (in our case, that's the StackPane), and open the Code section of the inspector (right hand side)

At the top of the section you should see a text field labelled Controller Class. In the field, typesimple.SimpleController. This is the name of the class we're going to create manually.

If you save at this point, the FXML will look like this:

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.\*?>

<?import java.util.\*?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.paint.\*?>

<StackPane prefHeight="150.0" prefWidth="200.0" xmlns:fx="http://javafx.com/fxml" fx:controller="simple.SimpleController">

<children>

<Button fx:id="myButton" mnemonicParsing="false" text="Button" />

</children>

</StackPane>

As you can see, the name of the controller class has been added to the root object:fx:controller="simple.SimpleController"

**Coding the controller class**

In your favorite IDE, create an empty SimpleController.java class. Now what does a controller class looks like? What should we put inside? Well - SceneBuilder will help you there: it will show you an example of controller skeleton tailored for your FXML. In the menu bar, invoke View > Show Sample Controller Skeleton. A popup appears, displaying a suggestion for the controller skeleton: copy the code displayed there, and paste it into your SimpleController.java:

/\*\*

\* Sample Skeleton for "simple.fxml" Controller Class

\* Use copy/paste to copy paste this code into your favorite IDE

\*\*/

**package** simple;

**import** **java.net.URL**;

**import** **java.util.ResourceBundle**;

**import** **javafx.fxml.FXML**;

**import** **javafx.fxml.Initializable**;

**import** **javafx.scene.control.Button**;

**public** **class** **SimpleController**

**implements** Initializable {

**@FXML** // fx:id="myButton"

**private** Button myButton; // Value injected by FXMLLoader

**@Override** // This method is called by the FXMLLoader when initialization is complete

**public** **void** **initialize**(URL fxmlFileLocation, ResourceBundle resources) {

**assert** myButton != **null** : "fx:id=\"myButton\" was not injected: check your FXML file 'simple.fxml'.";

// initialize your logic here: all @FXML variables will have been injected

}

}

Note that the code displayed by SceneBuilder is there only for educational purpose: SceneBuilder does not create and does not modify Java files. This is simply a hint of what you can use, given the fx:id present in your FXML file. You are free to copy all or part of the displayed code and paste it into your own Java class.

Now at this point, there only remains to add our logic to the controller class. Quite easy: in the initialize method, I will register an action handler with my button:

...

// initialize your logic here: all @FXML variables will have been injected

myButton.setOnAction(**new** EventHandler<ActionEvent>() {

**@Override**

**public** **void** **handle**(ActionEvent event) {

System.out.println("That was easy, wasn't it?");

}

});

...

That's it - if you now compile everything in your IDE, and run your application, clicking on the button should print a message on the console!

**Summary**

What happens is that in Main.java, the FXMLLoader will load simple.fxml from the jar/classpath, as specified by'FXMLLoader.load(Main.class.getResource("simple.fxml"))'. When loading simple.fxml, the loader will find the name of the controller class, as specified by 'fx:controller="simple.SimpleController"' in the FXML. Upon finding the name of the controller class, the loader will create an instance of that class, in which it will try to inject all the objects that have an fx:id in the FXML.   
Thus, after having created '<Button fx:id="myButton" ... />', the FXMLLoader will inject the button instance into the '@FXML private Button myButton;' instance variable found on the controller instance. This is because

* The instance variable has an @FXML annotation,
* The name of the variable **exactly matches** the value of the fx:id

Finally, when the whole FXML has been loaded, the FXMLLoader will call the controller's initialize method, and our code that registers an action handler with the button will be executed.

For a complete example, take a look at the HelloWorld [SceneBuilder sample](http://www.oracle.com/technetwork/java/javafx/downloads/devpreview-1429449.html). Also make sure to follow the[SceneBuilder Get Started guide](http://docs.oracle.com/javafx/), which will guide you through a much more complete example.

Of course, there are more elegant ways to set up an Event Handler using FXML and SceneBuilder. There are also many different ways to work with the FXMLLoader. But since it's starting to be very late here, I think it will have to wait for another post.

I hope you have enjoyed the tour!

--daniel

Ref: https://blogs.oracle.com/jmxetc/entry/connecting\_scenebuilder\_edited\_fxml\_to