



The interactive UI design tool originally developed by Oracle continues to advance in the open source community.

About Scene Builder

In practice, many applications contain both JavaFX code and FXML code. The JavaFX APIs and the FXML constructs are designed to work together.

On the JavaFX code side, the [FXMLLoader](#) class loads an FXML file from the JAR containing the application file or from the classpath:

The connection between your Java code and the UI elements that are declared in the FXML file is made by a controller class. This is a regular Java class that may contain annotations linking the UI elements to Java classes. This approach separates the UI declaration from the behavior of the application, while still allowing the application to access the UI elements directly.

Although the FXML file can be edited within any IDE as a regular XML file, this practice is not recommended, because the IDE provides only basic syntax checking and autocomple-

tion but not visual guidance. A better approach is to open the FXML file with Scene Builder.

Installing Scene Builder

You can install Scene Builder by [downloading it](#). Make sure to download the right version for your operating system. Then follow the platform-specific instructions for installing it in the default location, or select a custom location if you are

installing it on Windows. (This is a new feature available in Scene Builder 8.3.0.) Once you have installed Scene Builder, open your IDE so you can set its location and you can open any FXML file from your IDE:

- On NetBeans, select Tools -> Options (or Preferences on Mac) -> Java -> JavaFX, and click Browse to find the main Scene Builder folder.
- On IntelliJ, select File -> Settings (or Preferences on Mac)

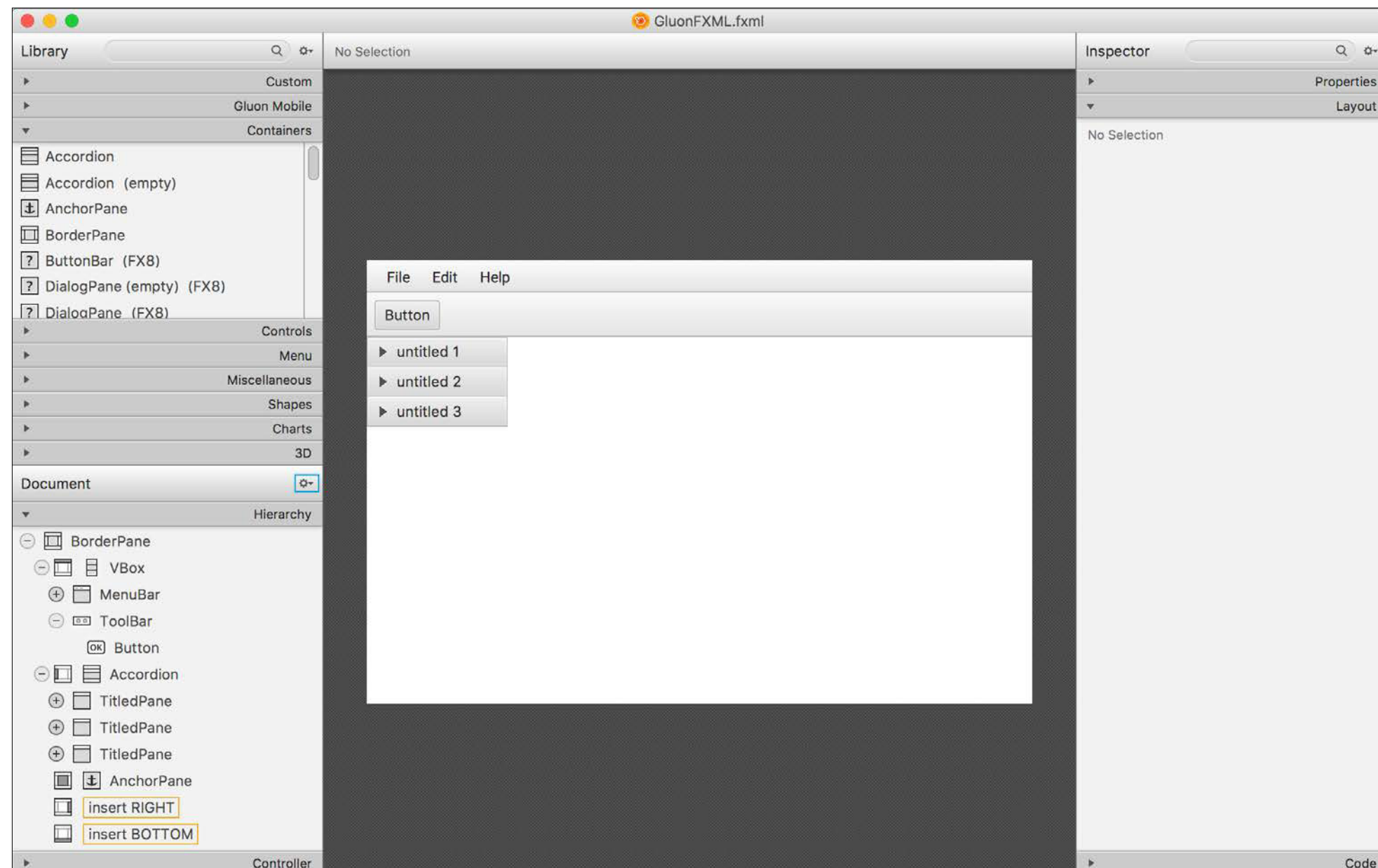


Figure 1. Adding containers and controls

-> Languages & Frameworks -> JavaFX, and browse for the application path.

- On Eclipse, select Window -> Preferences (or Preferences on Mac) -> JavaFX, and browse for the application path.

Whenever you have an FXML file, you will be able to edit it with Scene Builder simply by right-clicking it and selecting **Open with Scene Builder**.

Creating a Basic Interface

You can open an existing FXML file with Scene Builder, or you can open the Scene Builder application and create a new FXML file.

Creating a UI with Scene Builder is easy. You can drag and drop containers and controls to your view. Let's step through an example.

To begin with, open Scene Builder, and select Start New Project from the Welcome menu. For mobile projects, a built-in Gluon Mobile theme is set. If you want to create a regular JavaFX project, you can do this by selecting the Modena theme from Preview -> JavaFX theme -> Modena (FX8).

Add a main container for your scene. In this example, I will add a `BorderPane`. You can drag a `BorderPane` from the Containers left panel to the middle of the screen or to the hierarchy panel.

In a similar way, you can drag and drop any JavaFX built-in container or control (see **Figure 1**, previous page). You can use the Library Manager to include libraries containing custom controls. The [online documentation](#) can help with anything that's not intuitive.

A Hierarchy panel is available. It shows the hierarchy of containers and controls. If you want to use containers and controls in your Java code, you need to tag them with `fx:id`. This can be done in the Code right panel. This `fx:id` tag is a very important concept, because it bridges the world of the designer using Scene Builder with the developer code in an IDE.

If you want to have interactions between the FXML file and your Java code, you need to specify the name of the controller class. This name should be added to the FXML file in the Controller panel. You open the Controller panel by clicking the widget in the lower left corner.

If you want to access UI elements from your Java code (via the controller class), you need to make sure the value provided in the `fx:id` tag is exactly the same as the value of the `@FXML` annotation for the corresponding field in the controller class.

To make this easier for the developer, and to avoid typos, Scene Builder can generate a sample controller skeleton for you. This sample controller is auto-generated Java code that contains FXML-annotated fields for all UI elements that are tagged with `fx:id`.

You can easily copy the different nodes to the controller by selecting View -> Show Sample Controller Skeleton. Click the Copy button, and on your IDE paste the content into the controller class.

Typical code will look like **Listing 1**.

■ Listing 1. Sample Skeleton

```
import com.gluonhq.charm.glisten.control.AppBar;
import com.gluonhq.charm.glisten.control.Avatar;
import javafx.fxml.FXML;
import javafx.scene.control.ScrollPane;
import javafx.scene.layout.StackPane;
```

```
public class GluonFXMLSampe {
```

```
@FXML
private AppBar appBar;
```

```
@FXML
private StackPane stackPane;
```

@FXML

```
private ScrollPane scrollPane;  
  
@FXML  
private Avatar avatar;  
  
}
```

You can easily add CSS to the scene by providing a CSS file that can be included using the Stylesheets option in the Properties panel on the right. You can add inline styling also, by providing style rules to any node on the scene. You can apply new or existing style classes to any node as well.

Features related to layout, such as position, dimensions, margin, padding, and transforms (translation, rotation,

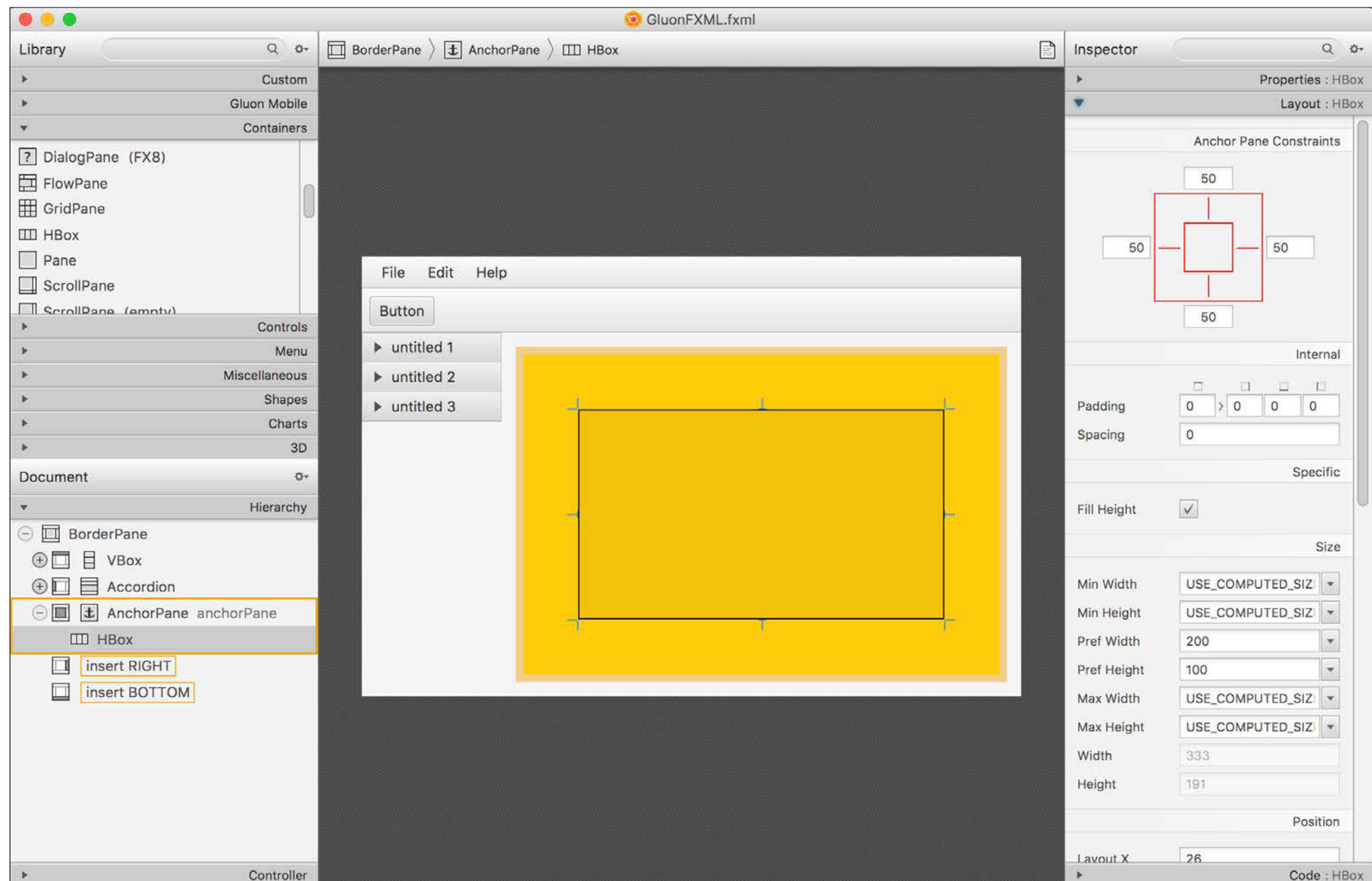


Figure 2. Defining anchor pane constraints in the Layout panel

scaling, and so forth), can be set in the Layout right panel, as shown in Figure 2.

At any moment, you can preview the created scene by clicking Preview -> Show Preview in Window. A resizable dialog box with the designed scene will be shown. By resizing it, you can make sure every node behaves as expected.

Integrate the Basic Interface in a Java App and Show It

Once the FXML is ready, you can integrate it into your Java application by calling `FXMLLoader` to load it. Here's how:

```
public class GluonSceneBuilder extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass()
            .getResource("GluonFXML.fxml"));

        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
}
```

In the controller class, you can add the required action handlers and the response to the user interaction. You can create new controls as well, and combine them with those injected by the `FXMLLoader`. Notice that for the controls you add programmatically, you need to create new instances. This is not needed for controls that are declared in the FXML file, because the `FXMLLoader` already creates them for you. The code snippet below shows a piece of a controller class that works with two controls: an `HBox` control defined in the FXML file, and a `Label` that is not created in the FXML file. The `HBox` instance does not need to be created in the controller class, but the `Label` instance does.

```
@FXML
private HBox hBox;

private Label label;

public void initialize() {
    label = new Label();
    hBox.getChildren().add(label);
    titledPane1.expandedProperty()
        .addListener((obs, ov, nv) -> {
            if (nv) {
                label.setText("TitledPane1");
            }
        });
    . . .
}
```

In a controller class, you can annotate not only fields representing controls with the `@FXML` annotation, but also methods. As an example, I define the following event handler in the controller:

```
@FXML
void buttonClicked(ActionEvent event) {
    label.setText("Button");
}
```

This event handler simply sets the text of the `Label` to `"Button"`. Because the event handler annotated it with `@FXML`, Scene Builder will find it and can assign it to a corresponding action for a node.

Integrate the Interface with Your Favorite IDE

Scene Builder is a standalone application. When FXML is being edited outside Scene Builder (by directly modifying it in your IDE, for example), Scene Builder reacts to the changes

