source : <http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm>

**AGENDA** :

1. What is FXML?

2. Why Use FXML?

3. Benefits of FXML.

4. Components of FXML.

5. FXML CSS

6. Layout of FXML.

7. Prerequisites(how to install and configure)

8. Simple example of FXML.

9. FXML and Scene Builder.

**1. What is FXML?**

FXML is an XML-based language that provides the structure for building a user interface separate from the application logic of your code for **Desktop Application / Web based Desktop application**.

FXML does not have a schema, but it does have a basic predefined structure. What you can express in FXML, and how it applies to constructing a scene graph, depends on the API of the objects you are constructing.

Because FXML maps directly to Java, you can use the API documentation to understand what elements and attributes are allowed.

While you can use FXML to create any user interface, FXML is particularly useful for user interfaces that have large, complex scene graphs, forms, data entry, or complex animation. FXML is also well-suited for defining static layouts such as forms, controls, and tables. In addition, you can use FXML to construct dynamic layouts by including scripts.

**2. Why Use FXML?**

Earlier we used java swing for the desktop application (standalone application) but that is not that much rich in gui and many more drawback so to overcome all these things Java introduce JavaFX with FXML with lots of new feature and rich GUI.

In swing we need to design each and every component in java code format not in XML format so there is no separation of UI and Java code logic, both mixed in a same place.

Now in JavaFx they separate java logic code and GUI.

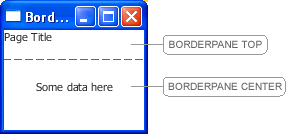
From a Model View Controller (MVC) perspective, the FXML file that contains the description of the user interface is the view.

The controller is a Java class, optionally implementing theInitializable class, which is declared as the controller for the FXML file.

The model consists of domain objects, defined on the Java side, that you connect to the view through the controller.

## 2.1. Simple Example of FXML

The easiest way to show the advantages of FXML is with an example. Take a look at Figure 1-1, which shows a user interface that includes a border pane layout that has a top and center region, each of which contains a label.



*Figure 1-1 Border Pane Simple Example*

First, look at how the user interface is constructed and built directly in the source code, as shown in Example 1-1.

***Example 1-1 Java Code for a User Interface***

BorderPane border = new BorderPane();  
Label toppanetext = new Label("Page Title");  
border.setTop(toppanetext);  
Label centerpanetext = new Label ("Some data here");  
border.setCenter(centerpanetext);

Next, look at [Example 1-2](http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDBCBEG), which shows the same user interface, but in FXML markup. You can see the hierarchical structure of the user interface, which in turn makes it easier to add components and build upon the user interface.

***Example 1-2 FXML Markup for a User Interface***

<BorderPane>  
 <top>  
 <Label text="Page Title"/>  
 </top>  
 <center>  
 <Label text="Some data here"/>  
 </center>  
</BorderPane>

**3. Benefits of FXML.**

In addition to providing web developers a familiar approach to designing user interfaces, FXML offers these benefits:

* Because the scene graph is more transparent in FXML, it is easy for a development team to create and maintain a testable user interface.
* FXML is not a compiled language; you do not need to recompile the code to see the changes. just run and see
* You can use FXML with any Java Virtual Machine (JVM) language, such as Java, Scala, or Clojure.
* FXML is not limited to the view portion of the MVC interface. You can construct services or tasks or domain objects, and you can use JavaScript or other scripting languages in FXML. For an example of using JavaScript.
* [Use a Scripting Language to Handle Events](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#JFXGS213) in the FXML tutorial of the Getting Started guide, go for this link : <http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#JFXGS213>
* The content of an FXML file can be localized as the file is read. For example, if an FXML file is loaded using the en\_US locale, then it produces the string "First Name" for a label based on the following resource string:

<Label text="%firstName"/>

If the locale is changed to fr\_FR and the FXML file is reloaded, then the label shows "Prénom."

The same is not true for Java code, because you must manually update the content of every element of your user interface by obtaining a reference to it and calling the appropriate setter (such as setText()).

for more : <http://docs.oracle.com/javafx/2/swing/overview.htm>

**4. Components of FXML:**

Here we are having a huge amount of component but we will talk about few ones as follows,

HBox : As name says Horizontal Box, so It contains all element in horizontal manner.

eg : <HBox **spacing="10" alignment="bottom\_right" styleClass=”style.css”**>

// content reside here in horizontal

</HBox>

VBox : As name says Vertical Box, so it contains all elements in vertical manner.

eg : <VBox **spacing="10" alignment="bottom\_right" styleClass=”style.css”**>

// content reside here

</VBox>

Label : For text writing purpose

<Label fx:id= “myWord” text=”Hello world” />

Button :

<Button fx:id= “myAction” text="My Button"/>

TableView :

<TableView fx:id="tableView" GridPane.columnIndex="0" GridPane.rowIndex="1">  
 **<columns>**  
 **<TableColumn text="First Name">**  
 **</TableColumn>**  
 **<TableColumn text="Last Name">**  
 **</TableColumn>**  
 **<TableColumn text="Email Address">**  
 **</TableColumn>**  
 **</columns>**   
 </TableView>

Image : For refering an image from FXML end

<ImageView>

<image>

<fx:reference source="myImage"/>

</image>

</ImageView>

**5. FXML CSS :**

JavaFX Cascading Style Sheets (CSS) is based on the W3C CSS version 2.1 with some additions from current work on version 3.

The goal for JavaFX CSS is to allow web developers already familiar with CSS for HTML to use CSS to customize and develop themes for JavaFX controls and scene graph objects in a natural way.

This enables the mixing of CSS styles for JavaFX and for other purposes (such as for HTML pages) into a single style sheet. To this end, all JavaFX property names have been prefixed with a vendor extension of "-fx-". Even properties that might seem to be compatible with standard HTML CSS have been prefixed, because JavaFX has somewhat different semantics for their values.

JavaFX CSS does not support CSS layout properties such as *float*, *position*, *overflow*, and *width*.

However, the CSS *padding* and *margins* properties are supported on some JavaFX scene graph objects.

All other aspects of layout are handled programmatically in JavaFX code. In addition, CSS support for HTML-specific elements such as Tables are not supported since there is no equivalent construct in JavaFX.

eg:

.button { -fx-background-color: abc }

**CSS Limitations :**

While the JavaFX CSS parser will parse valid CSS syntax, it is not a fully compliant CSS parser.

- @-keyword statements are ignored.

- The ":first-child" and ":lang" pseudo-classes are not supported.

- The ":first-line", ":first-letter", ":after", and ":before" pseudo-elements are not supported.

- The ":active" and ":focus" dynamic pseudo-classes are not supported. However, [Nodes](http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html#node) do support the ":pressed" and ":focused" pseudo-classes, which are similar.

- The ":link" and ":visited" pseudo-classes are not supported in general. However, [Hyperlink](http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html#hyperlink) objects can be styled, and they support the ":visited" pseudo-class.

- JavaFX CSS does not support comma-separated series of font family names in the -fx-font-family property. The optional line height parameter when specifying fonts is not supported. There is no equivalent for the font-variant property.

- JavaFX CSS uses the HSB color model instead of the HSL color model.

<http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html#anchorpane>

**6. Layout of FXML.**

FXML have rich number of layouts as follows,

#### AnchorPane

#### BorderPane

#### FlowPane

#### GridPane

#### HBox

#### Pane

#### Region

#### StackPane

#### TilePane

#### VBox

**6. Prerequisites.**

* **Latest JDK with JavaFX support**
* **You can download it here:** [**http://www.oracle.com/technetwork/java/javase/downloads/jdk7u9-downloads-1859576.html**](http://www.oracle.com/technetwork/java/javase/downloads/jdk7u9-downloads-1859576.html)
* **Created JavaFX project with one Main FXML form.**
* **You can download or create one by previous post:** [**How to create Apache Maven project with JavaFX and FXML support**](http://www.korecky.org/?p=718)
* **Your favorite IDE**
* **(e.g: Eclipse or NetBeans)**

**7. Simple example of FXML.**

Here, you use FXML to create the same login user interface, separating the application design from the application logic, thereby making the code easier to maintain. The login user interface you build in this tutorial is shown in [Figure 4-1](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDICHAE).

***Figure 4-1 Login User Interface***



uses NetBeans IDE. Ensure that the version of NetBeans IDE that you are using supports JavaFX 2.2. See the [System Requirements](http://docs.oracle.com/javafx/release-documentation.html) for details.

## Set Up the Project

Your first task is to set up a JavaFX FXML project in NetBeans IDE:

1. From the **File** menu, choose **New Project**.
2. In the **JavaFX** application category, choose **JavaFX FXML Application**. Click **Next**.
3. Name the project **FXMLExample** and click **Finish**.
4. NetBeans IDE opens an FXML project that includes the code for a basic Hello World application. The application includes three files:
   1. **FXMLExample.java.** This file takes care of the standard Java code required for an FXML application.
   2. **Sample.fxml**. This is the FXML source file in which you define the user interface.
   3. **SampleController.java**. This is the controller file for handling the mouse and keyboard input.
5. Rename SampleController.java to FXMLExampleController.java so that the name is more meaningful for this application.
   1. In the Projects window, right-click **SampleController.java** and choose **Refactor** then **Rename**.
   2. Enter **FXMLExampleController**, and click **Refactor**.
6. Rename Sample.fxml to fxml\_example.fxml.
   1. Right-click **Sample.fxml** and choose **Rename**.
   2. Enter **fxml\_example** and click **OK**.

## Load the FXML Source File

The first file you edit is the FXMLExample.java file. This file includes the code for setting up the application main class and for defining the stage and scene. More specific to FXML, the file uses the FXMLLoader class, which is responsible for loading the FXML source file and returning the resulting object graph.

Make the changes shown in bold in [Example 4-1](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDDBEGD).

***Example 4-1 FXMLExample.java***

@Override  
 public void start(Stage stage) throws Exception {  
 Parent root = FXMLLoader.load(getClass().getResource("**fxml\_example.fxml**"));  
   
 Scene scene = new Scene(root**, 300, 275**);  
   
 **stage.setTitle("FXML Welcome");**  
 stage.setScene(scene);  
 stage.show();  
 }

A good practice is to set the height and width of the scene when you create it, in this case 300 by 275; otherwise the scene defaults to the minimum size needed to display its contents.

## Modify the Import Statements

Next, edit the fxml\_example.fxml file. This file specifies the user interface that is displayed when the application starts. The first task is to modify the import statements so your code looks like [Example 4-2](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDFJDAE).

***Example 4-2 XML Declaration and Import Statements***

<?xml version="1.0" encoding="UTF-8"?>  
  
**<?import java.net.\*?>**  
**<?import javafx.geometry.\*?>**  
**<?import javafx.scene.control.\*?>**  
**<?import javafx.scene.layout.\*?>**  
**<?import javafx.scene.text.\*?>**

As in Java, class names can be fully qualified (including the package name), or they can be imported using the import statement, as shown in [Example 4-2](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDFJDAE). If you prefer, you can use specific import statements that refer to classes.

## Create a GridPane Layout

The Hello World application generated by NetBeans uses an AnchorPane layout. For the login form, you will use a GridPane layout because it enables you to create a flexible grid of rows and columns in which to lay out controls.

Remove the AnchorPane layout and its children and replace it with the GridPane layout in[Example 4-3](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDIFIAC).

***Example 4-3 GridPane Layout***

<GridPane fx:controller="fxmlexample.FXMLExampleController"   
 xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">  
<padding><Insets top="25" right="25" bottom="10" left="25"/></padding>  
  
</GridPane>

In this application, the GridPane layout is the root element of the FXML document and as such has two attributes. The fx:controller attribute is required when you specify controller-based event handlers in your markup. The xmlns:fx attribute is always required and specifies the fxnamespace.

The remainder of the code controls the alignment and spacing of the grid pane. The alignment property changes the default position of the grid from the top left of the scene to the center. The gap properties manage the spacing between the rows and columns, while the paddingproperty manages the space around the edges of the grid pane.

As the window is resized, the nodes within the grid pane are resized according to their layout constraints. In this example, the grid remains in the center when you grow or shrink the window. The padding properties ensure there is a padding around the grid when you make the window smaller.

## Add Text and Password Fields

Looking back at [Figure 4-1](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDICHAE), you can see that the login form requires the title “Welcome” and text and password fields for gathering information from the user. The code in [Example 4-4](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDJAJCI) is part of the GridPane layout and must be placed above the </GridPane> statement.

***Example 4-4 Text, Label, TextField, and Password Field Controls***

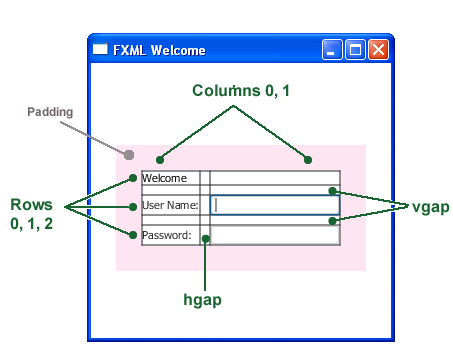
<Text text="Welcome"   
 GridPane.columnIndex="0" GridPane.rowIndex="0"  
 GridPane.columnSpan="2"/>  
   
 <Label text="User Name:"  
 GridPane.columnIndex="0" GridPane.rowIndex="1"/>  
   
 <TextField   
 GridPane.columnIndex="1" GridPane.rowIndex="1"/>  
   
 <Label text="Password:"  
 GridPane.columnIndex="0" GridPane.rowIndex="2"/>  
   
 <PasswordField fx:id="passwordField"   
 GridPane.columnIndex="1" GridPane.rowIndex="2"/>

The first line creates a Text object and sets its text value to Welcome. TheGridPane.columnIndex and GridPane.rowIndex attributes correspond to the placement of theText control in the grid. The numbering for rows and columns in the grid starts at zero, and the location of the Text control is set to (0,0), meaning it is in the first column of the first row. TheGridPane.columnSpan attribute is set to 2, making the Welcome title span two columns in the grid. You will need this extra width later in the tutorial when you add a style sheet to increase the font size of the text to 32 points.

The next lines create a Label object with text User Name at column 0, row 1 and a TextFieldobject to the right of it at column 1, row 1. Another Label and PasswordField object are created and added to the grid in a similar fashion.

When working with a grid layout, you can display the grid lines, which is useful for debugging purposes. In this case, set the gridLinesVisible property to true by adding the statement<gridLinesVisible>true</gridLinesVisible> right after the <padding></padding> statement. Then, when you run the application, you see the lines for the grid columns and rows as well as the gap properties, as shown in [Figure 4-2](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDEIAAJ).

***Figure 4-2 Login Form with Grid Lines***



## Add a Button and Text

The final two controls required for the application are a Button control for submitting the data and a Text control for displaying a message when the user presses the button. The code is in[Example 4-5](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDCCBEC). Add this code before </GridPane>.

Example 4-5 HBox, Button, and Text

<HBox spacing="10" alignment="bottom\_right"   
 GridPane.columnIndex="1" GridPane.rowIndex="4">  
 <Button text="Sign In"   
 onAction="#handleSubmitButtonAction"/>  
</HBox>  
  
<Text fx:id="actiontarget"  
 GridPane.columnIndex="1" GridPane.rowIndex="6"/>

An HBox pane is needed to set an alignment for the button that is different from the default alignment applied to the other controls in the GridPane layout. The alignment property is set tobottom\_right, which positions a node at the bottom of the space vertically and at the right edge of the space horizontally. The HBox pane is added to the grid in column 1, row 4.

The HBox pane has one child, a Button with text property set to Sign in and an onActionproperty set to handleSubmitButtonAction(). While FXML is a convenient way to define the structure of an application's user interface, it does not provide a way to implement an application's behavior. You implement the behavior for the handleSubmitButtonAction()method in Java code in the next section of this tutorial, [Add Code to Handle an Event](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDJDGCE).

Assigning an fx:id value to an element, as shown in the code for the Text control, creates a variable in the document's namespace, which you can refer to from elsewhere in the code. While not required, defining a controller field helps clarify how the controller and markup are associated.

## Add Code to Handle an Event

Now make the Text control display a message when the user presses the button. You do this in the FXMLExampleController.java file. Delete the code that NetBeans IDE generated and replace it with the code in [Example 4-6](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDJIDHE).

***Example 4-6 FXMLExampleController.java***

package fxmlexample;  
   
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.text.Text;  
   
public class FXMLExampleController {  
 @FXML private Text actiontarget;  
   
 @FXML protected void handleSubmitButtonAction(ActionEvent event) {  
 actiontarget.setText("Sign in button pressed");  
 }  
  
}

The @FXML annotation is used to tag nonpublic controller member fields and handler methods for use by FXML markup. The handleSubmtButtonAction method sets the actiontarget variable to Sign in button pressed when the user presses the button.

You can run the application now to see the complete user interface. [Figure 4-3](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDHHJDI) shows the results when you type text in the two fields and click the Sign in button. If you have any problems, then you can compare your code against the FXMLLogin example.

***Figure 4-3 FXML Login Window***



## Use a Scripting Language to Handle Events

As an alternative to using Java code to create an event handler, you can create the handler with any language that provides a JSR 223-compatible scripting engine. Such languages include JavaScript, Groovy, Jython, and Clojure.

Optionally, you can try using JavaScript now.

1. In the file fxml\_example.fxml, add the JavaScript declaration after the XML doctype declaration.
2. <?language javascript?>
3. In the Button markup, change the name of the function so the call looks as follows:
4. onAction="handleSubmitButtonAction(event);"
5. Remove the fx:controller attribute from the GridPane markup and add the JavaScript function in a <script> tag directly under it, as shown in [Example 4-7](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDBEIHJ).
6. ***Example 4-7 JavaScript in FXML***
7. <GridPane xmlns:fx="http://javafx.com/fxml"   
    alignment="center" hgap="10" vgap="10">  
    <fx:script>  
    function handleSubmitButtonAction() {  
    actiontarget.setText("Calling the JavaScript");  
    }  
    </fx:script>
8. Alternatively, you can put the JavaScript functions in an external file (such asfxml\_example.js) and include the script like this:
9. <fx:script source="fxml\_example.js"/>

The result is in [Figure 4-4](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDCGEEE).

***Figure 4-4 Login Application Using JavaScript***



If you are considering using a scripting language with FXML, then note that an IDE might not support stepping through script code during debugging.

## Style the Application with CSS

The final task is to make the login application look attractive by adding a Cascading Style Sheet (CSS).

1. Create a style sheet.
   1. In the Project window, right-click the login folder under Source Packages and choose **New**, then **Other**.
   2. In the New File dialog box, choose **Other**, then **Cascading Style Sheet** and click**Next**.
   3. Enter **Login** and click **Finish**.
   4. Copy the contents of the [Login.css](http://docs.oracle.com/javafx/2/get_started/Login.css.html) file attached to this document into your CSS file. For a description of the classes in the CSS file, see [Fancy Forms with JavaFX CSS](http://docs.oracle.com/javafx/2/get_started/css.htm#BEIBBBCI).
2. Download the gray, linen-like image for the background in the [background.jpg](http://docs.oracle.com/javafx/2/get_started/background.jpg.html) file and add it to the fxmlexample folder.
3. Open the fxml\_example.fxml file and add a stylesheets element before the end of the markup for the GridPane layout as shown in [Example 4-8](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDIHIAC).
4. ***Example 4-8 Style Sheet***
5. <stylesheets>  
    <URL value="@Login.css" />  
    </stylesheets>  
     
   </GridPane>
6. The @ symbol before the name of the style sheet in the URL indicates that the style sheet is in the same directory as the FXML file.
7. To use the root style for the grid pane, add a style class to the markup for the GridPanelayout as shown in [Example 4-9](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#BJEEHFJJ).
8. ***Example 4-9 Style the GridPane***
9. <GridPane fx:controller="fxmlexample.FXMLExampleController"   
    xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10"   
    **styleClass="root"**>
10. Create a welcome-text ID for the Welcome Text object so it uses the style #welcome-textdefined in the CSS file, as shown in [Example 4-10](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDEEAAA).
11. ***Example 4-10 Text ID***
12. <Text **id="welcome-text"** text="Welcome"   
     GridPane.columnIndex="0" GridPane.rowIndex="0"   
     GridPane.columnSpan="2"/>
13. Run the application. [Figure 4-5](http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#CHDFBIIB) shows the stylized application.

***Figure 4-5 Stylized Login Application***



For information about how to run your application outside NetBeans IDE, see [Deploying Your First JavaFX Application](http://docs.oracle.com/javafx/2/get_started/basic_deployment.htm#BABIDHGA).

**8. FXML and Scene Builder.**

You can also try out the JavaFX Scene Builder tool by opening the fxml\_example.fxml file in Scene Builder and making modifications. This tool provides a visual layout environment for designing the UI for JavaFX applications and automatically generates the FXML code for the layout. Note that the FXML file might be reformatted when saved. See [Getting Started with JavaFX Scene Builder](http://docs.oracle.com/javafx/scenebuilder/1/get_started/jsbpub-get_started.htm) for more information on this tool. The [Skinning with CSS and CSS Analyzer](http://docs.oracle.com/javafx/scenebuilder/1/user_guide/stylesheet-support.htm#CHDCGIBJ)section of the [JavaFX Scene Builder User Guide](http://docs.oracle.com/javafx/scenebuilder/1/user_guide/jsbpub-user_guide.htm) also give you information on how you can skin your FXML layout.

**9. Resources :**

<http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm>

<http://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html>

<http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm>

http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html