JavaFX/FXML CookBook (Nutshell Examples)

Manuel Schwarzer

December 17, 2020

I hereby declare that:

1. I have written this paper myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.
2. This paper has not been previously presented as an examination paper in this or any other form in Austria or abroad.
3. This paper is identical with the thesis assessed by the examiner.

The individual contribution of each writer as well as the co-written passages have been indicated.

17.12.20

Date

Unterschrift

Contents

Abstract

JavaFX and SceneBuilder companied are a powerful approach to set up simple or complex GUIs, which can be used unchanged on Windows, Mac or Linux. BSF4ooRexx acts therefore like a bridge between two worlds of different programming languages, ooRexx and Java. Standalone nutshell examples will demonstrate how a GUI can be created and how JavaFX can be used by ooRexx.

1 Introduction

This paper will give a brief overview about different components about JavaFX and FXML. At the beginning, more specific in Chapter 1 it will be given some information about the various software components that will be used. Furthermore, in Chapter 2, there is an installation guide to show how these components shall be installed. Moreover, in Chapter 3, there are some serious nutshell examples to reveal the true strength about JavaFX and FXML.

1.1 BSF4ooRexx

BSF4ooRexx (stands for: BeanScriptingFramework for ooRexx) is on of the major parts to use JavaFX and FXML in ooRexx. BSF4ooRexx acts like a translator who translates from on language into another one. So basically it translates Java code to make it usable in ooRexx and vice versa. Furthermore, with BSF4ooRexx the whole Java-Class-Collection will be usable. How to do so will be shown later.
BSF4Rexx was developed to act like a bridge between different kinds of software. Mainly to cover Java methods so that these methods could be used platform independently. This goal was achieved in January 2000 by a proof-of-concept study at the University of Essen [7]. After years of development BSF4ooRexx with its ooRexx package "BSF.CLS" was published. With this package it is possible to use Java classes like they were ooRexx classes. [7]

1.2 JavaFX/FXML

JavaFX is a next generation client application platform for desktop, mobile and embedded systems built on Java. It is a collaborative effort by many individuals and companies with the goal of producing a modern, efficient, and fully featured toolkit for developing rich client application. [1]
FXML in addition, is an XML-based user interface markup language created by Oracle for defining the user interface of a JavaFX application. FXML presents an alternative to designing user interfaces using procedural code, and allows for abstracting program design from program logic. [2] Furthermore, with JavaFX and especially with FXML, it is possible do use a an

interface platform independently. So, if an interface is designed on a Microsoft based platform, it is possible to use the same Graphical User Interface, short GUI, written in FXML, on an other platform, which contains JavaFX as well [5].
These two, JavaFX and FXML, combined, will led to an easier way to design and create Graphical User Interfaces. In combination with the application "SceneBuilder" ,which will be explained in the next section of this Chapter, it is getting even easier.

1.3 SceneBuilder

SceneBuilder is an open source application which is supported by Gluon [3]. With SceneBuilder it is a lot easier to design and create many different, complex or simple, Graphical User Interfaces. Within SceneBuilder there are three different ares.
First, there is an area where all kinds of usable stuff, which can be placed on a GUI, is located. Second, in the middle of the screen, between the first and the third area, there is the working area, where you can add something like a button. This can be done simply via drag and drop. Third, on the right side there is an area where you can gather information about the properties of the previously added component. All kinds of information will be shown there, for example the font size of a button, or the color of the text within the button. These three areas can be seen in figure 1 underneath.

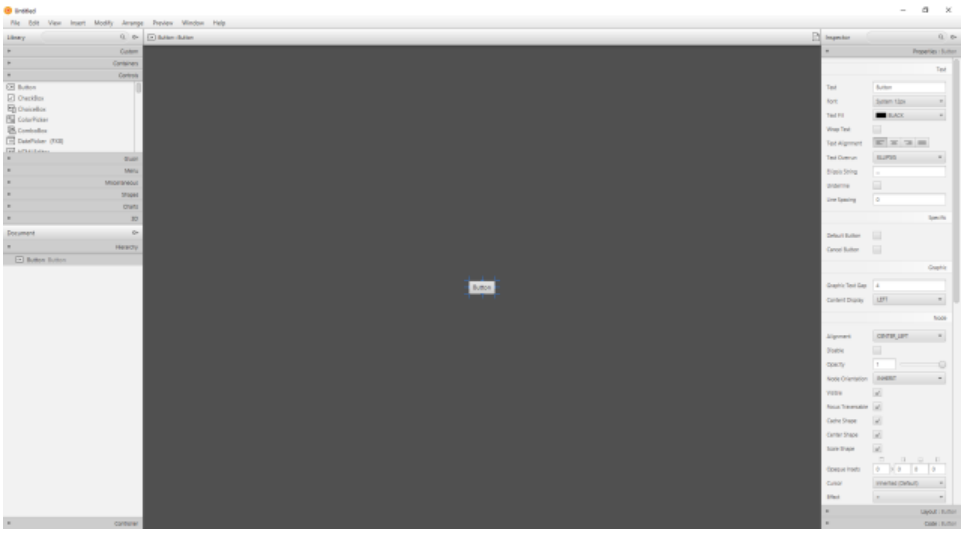

Figure 1: Button added

With SceneBuilder it is the most efficient way to design and create a GUI, because you can add/drop components of a GUI via drag and drop,

furthermore, you can use the created GUI immediately within your ooRexx source code. More details about this in Chapter 3 where a short example of a GUI creation will be shown step by step.

2 Installation

In this section, it will be shown what kind of software components will be needed for this paper. In general, there are four major components that are required and two optional components.
First, it is required to make sure that Java itself is installed on the operating system. Second, ooRexx must be installed. After that, BSF4ooRexx can be installed. Furthermore, JavaFX and SceneBuilder are required. Last, an optional software component, which will help with the interaction between ooRexx/Java and a database will be shown.

2.1 Java

First of all, Java itself has to be downloaded and installed. If it is not already installed on the platform, please download it. Make sure that it matches the same bit rate as ooRexx. In this paper Java version "1.8.0_191 64-bit" was used.
It can be found here:
$https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html$. Within this archive download JDK 8u191 with the required bit rate, mostly 64-bit.

2.2 ooRexx

If it is not already installed on your operating system, ooRexx must be downloaded and installed too. For this paper, ooRexx beta version 5.0.0 r11982 was used. The required data can be downloaded here:
$https://sourceforge.net/projects/oorexx/files/oorexx/$. It is highly recommended to install the same bit rate (32-bit or 64-bit) as the operating system has and especially as the version of Java has, otherwise some error can occur.

2.3 BSF4ooRexx

After the successful installation of Java and ooRexx, the associated Bean Scripting Framework has to be downloaded and installed. The required files can be found here: https://sourceforge.net/projects/bsf4oorexx/files/.
In this paper, BSF4ooRexx version "bsf4ooRexx-v641-20200130-bin" was used. After following these steps, it will be possible to use Java in ooRexx and vice versa:

1. Visit https://sourceforge.net/projects/bsf4oorexx/files/
2. Download the latest version of BSF4ooRexx
3. Unzip the downloaded archive
4. Navigate into the unzipped subdirectory "bsf4oorexx/install/windows"
5. Double click install.cmd

Now, it is possible to use Java classes within ooRexx. Furthermore, with BSF4ooRexx it is possible to automate business process and to interact with other applications like Microsoft Word and so on [7]. But, that is not the focus of this paper.

2.4 JavaFX

JavaFX is one of the main parts to generate GUIs. Nonetheless, nothing additional must be downloaded, stored or implemented. JavaFX comes with JDK 8. So, if a different version of JDK will be used, take care of the fact that, JavaFX is not always implemented by default. Nowadays, many different JavaFX approaches take place, since Oracle does not fully support JavaFX within their JDK packages any more. Since Oracle and JavaFX split up, OpenJFX and its open-source community take care of JavaFX and its latest versions [6].

2.5 SceneBuilder

The last required software component is "SceneBuilder". It is a program that will help with the designing process of a GUI application, because the designing process will get more interactive via drag and drop functions. Moreover, not a single line of code must be written to create a GUI by yourself. To get SceneBuilder follow these steps:

1. Visit https://gluonhq.com/products/scene-builder/
2. Download the latest version of SceneBuilder (In this scenario, version 11.0.0 is used)
3. Double click the downloaded executable file
4. Follow the instractions

As can be seen at the homepage of Gluon, it is recommended to download version 8.5 of SceneBuilder if Java 8 is used [3]. But there are some troubles between modern styling classes of SceneBuilder, like the later explained class "JFoenix", and the older version of SceneBuilder. In this paper SceneBuilder version 11 was used, without any complications.

2.6 DB Browser (SQLite)

In this subsection, an application will be explained, which helps with creating a database. If it is needed, that apart from the used database in Chapter 3, a new database should be created, DB Browser for SQLlite is one of the easiest ways to do so. A quick guide of how to use this application will be explained underneath.

1. Visit https://sqlitebrowser.org/dl/
2. Download the latest version of DB Browser for SQLite (In this scenario, version 3.12.0-win64 is used)
3. Double click the downloaded .exe - file
4. Follow the instractions

After the installation process was successful, open the application. It should look like figure 2.

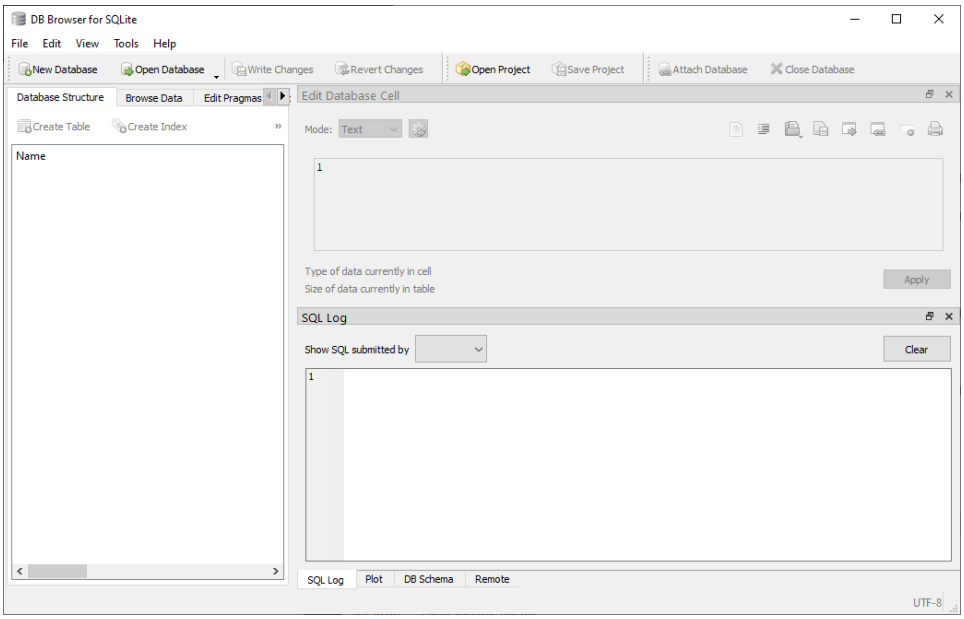

Figure 2: DB Browser for SQLite

Now, it is possible to create a new database or to open an existing one. For testing purpose, the included database within this paper called "Nutshell-DB.db" can be opened and looked at. To do so click on "Open Database" and navigate to the example database. Double click the database that should be opened. After that, it is possible to browse through the

database's data, tables and properties. But, for this paper, DB Browser for SQLite is not necessary, it is only an optional component, which may help to understand the nutshell example within Chapter 3 a bit more easier.

3 Nutshell Examples

In this section, it will be shown how versatile JavaFX and especially FXML in combination with ooRexx can be.
First of all, a simple GUI will be created. This should point out how easy it is to create a GUI without any knowledge about Java. Second, this example of a GUI will be extended with a styling sheet, called CSS. Third, a useful example of how to use a database in combination with a GUI and ooRexx will be shown. Fourth, a comparison between default components of SceneBuilder and an additionally added class with components will be shown. Last, a combination of multiple windows and its handling will be explained.

3.1 My first GUI

In this subsection it will be shown how a short example of a GUI will be created step by step with SceneBuilder.
First, SceneBuilder has to be started. After the successful start of SceneBuilder, three working areas will be shown, like in figure 3 underneath.

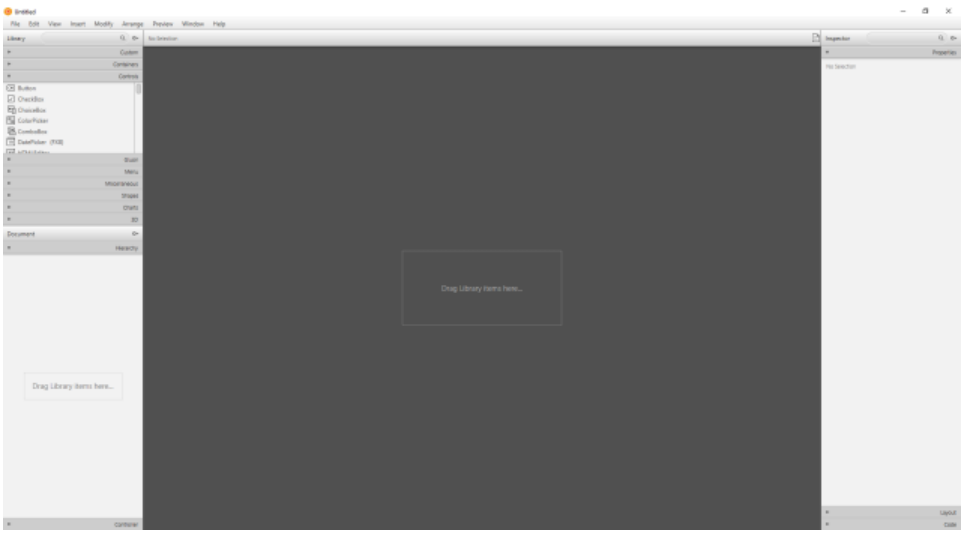

Figure 3: SceneBuilder starting screen

As can be seen, the first area mentioned, is the left area with all different kinds of controls and containers, which will be used to design a GUI.

Followed by the second area, the drag and drop area, where components like buttons be placed on. Last but not least, on the right hand side there is area number three, the properties area, which adjusts every time you click on a different component within the drag and drop area.
First of all, to start a container has to be added to the working area in the middle. This can be an AnchorPane, a VBox or something else. One of these containers has to be added before other controls can be placed on. This containers are used to define a program's "window". If, for example, an AnchorPane is placed on, the GUI window has the same size as the AnchorPane displayed on the properties area. Furthermore, this AnchorPane serves as a container for all components that will be placed on later.

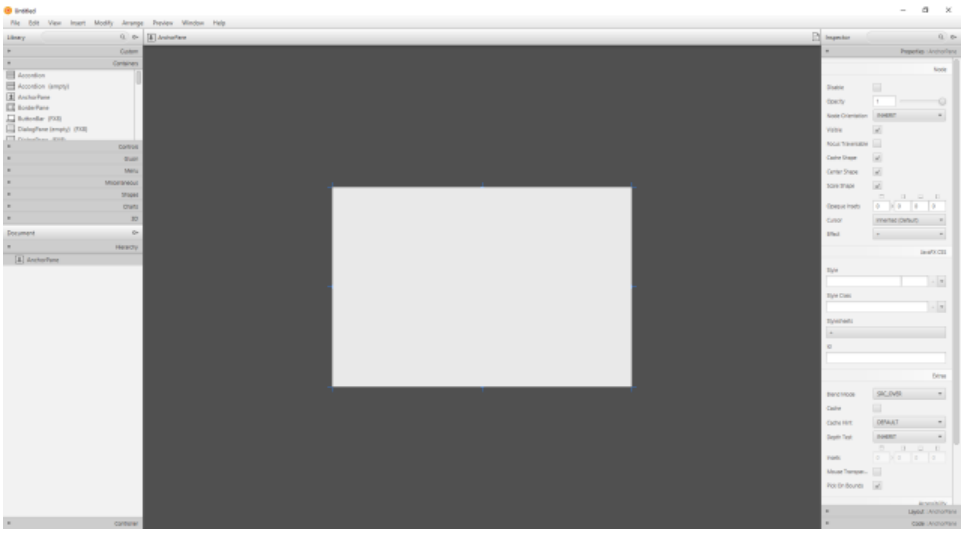

Figure 4: AnchorPane

As can be seen in figure 4, an AnchorPane was added to the drag and drop area. After this, the creative design of the GUI can be started. Several buttons, textfields, labels and so on can be added. A short example is shown in figure 5.

This GUI is one of the simplest that can be created. It has one Label which displays the headline "My first GUI", one TextField where something can be written in and a button that can be clicked. As can be seen, it is possible to adjust all these kinds of components. Furthermore, no component was selected during the snapshot, so the properties area is empty.

Every time something is added or dropped to the GUI, code will be generated or deleted in the background. The source code of the GUI can be seen if the saved SceneBuilder file will be reopened via an editor. This could be a simple notepad or a more complex application like IntelliJ or Eclipse.

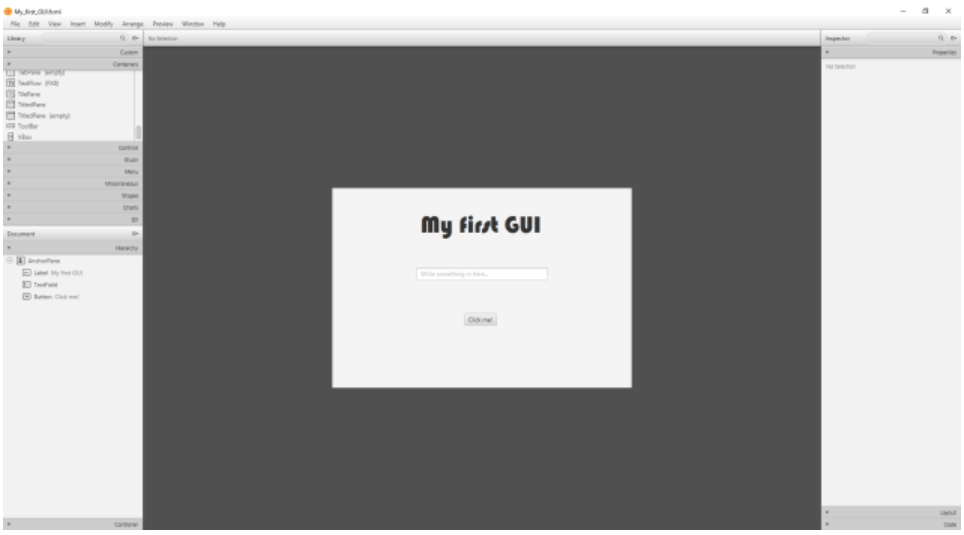

Figure 5: My first GUI

After reopening the file, its source code will be displayed like the source code underneath.

In SceneBuilder everything is based on drag and drop and nicely shaped objects, in the text-based editor everything is written in text, in particular, its written in XML.
XML stands for Extensible Markup Language and is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable [4]. At this point JavaFX and XML get together to FXML, which is an XML-based user interface markup language created by Oracle [2].

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.control.Button?>
4  <?import javafx.scene.control.Label?>
5  <?import javafx.scene.control.TextField?>
6  <?import javafx.scene.layout.AnchorPane?>
7  <?import javafx.scene.text.Font?>
8
9
10 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
       minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http
      ://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">
11    <children>
12       <Label alignment="CENTER" layoutX="63.0" layoutY="46.0" prefHeight="
             44.0" prefWidth="469.0" text="My first GUI" textAlignment="CENTER"
             >
13          <font>
14             <Font name="Bauhaus 93" size="48.0" />
```

```
15          </font>
16        </Label>
17        <TextField layoutX="169.0" layoutY="159.0" prefHeight="14.0" prefWidth
              ="263.0" promptText="Write something in here..." />
18        <Button layoutX="265.0" layoutY="251.0" mnemonicParsing="false" text="
              Click me!" />
19    </children>
20 </AnchorPane>
```

If these two representations of the GUI will be compared to each other, the SceneBuilder approach is more convenient and faster than writing source code by hand. The source code of the main and FXML file can be found within the Appendix.

3.2 My first GUI with CSS

In the previous subsection, a short example was explained. Now, this example will be extended. Every now and then it is needed that a GUI in general looks more entertaining and fancy. There are many different ways to do so. One of them is to rewrite the source code, or to redesign the whole GUI in SceneBuilder. Another way, a more efficient way, is to create a so called CSS. Cascading Style Sheet, was actually invented for HTML, which is the base of the World Wide Web [8]. With CSS it is possible to redesign a web side without a change in the source code.
Nowadays, CSS is used in many different aspects. For example in this case, with CSS it is possible to redesign a GUI without major changes within the source code. To show this circumstance, only a simple CSS, wich is shown in the code section underneath, was added to “My first GUI”.

```
1
2 .root{
3     -fx-background-image: url("background.jpeg");
4 }
5
6 .button {
7     -fx-text-fill: white;
8     -fx-font-family: "Arial Narrow";
9     -fx-font-weight: bold;
10    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
11    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 ,
          1 );
12 }
13
14 .label {
15    -fx-font-size: 36px;
16    -fx-font-weight: bold;
17    -fx-text-fill: #333333;
18    -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
19 }
```

```
.textfield{
-fx-prompt-text-fill: rgba(255,255,255,0.5);
}
```

Additionally, it is required to add the line of code "*stylesheet* = @*n_vs_j_css.css*" in every component that should be changed. The "new" GUI with CSS looks like figure 6.

```

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?language rexx?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
     minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" styleClass="
    root" stylesheets="@n_vs_j_css.css" xmlns="http://javafx.com/javafx
    /11.0.1" xmlns:fx="http://javafx.com/fxml/1">
  <!-- call Rexx program, its public routine "buttonClicked" is known
       afterwards -->
  <fx:script source="My_first_GUI_controller.rexx" />
  <children>
    <Label alignment="CENTER" layoutX="63.0" layoutY="46.0" prefHeight="
        44.0" prefWidth="469.0" stylesheets="@n_vs_j_css.css" text="My
        first GUI with CSS" textAlignment="CENTER">
      <font>
        <Font name="Bauhaus 93" size="48.0" />
      </font>
    </Label>
    <TextField fx:id="textField1" layoutX="169.0" layoutY="159.0"
        prefHeight="14.0" prefWidth="263.0" stylesheets="@n_vs_j_css.css"
        promptText="Write something in here..." />
    <Button fx:id="button1" layoutX="265.0" layoutY="251.0"
        mnemonicParsing="false" onAction="slotDir=arg(arg()); call
        buttonClicked slotDir;" stylesheets="@n_vs_j_css.css" text="Click
        me!" />
  </children>
</AnchorPane>
```

As it shown in the source code above, every component starting with the AnchorPane and ending with the Button, every component got the additional line of code mentioned above. As a result, the GUI looks more state of the art. As can be seen in figure 6. The main file can be found within the Appendix.

Figure 6: CSS added

3.3 SQLite - JDBC

In this subsection, a useful way of a GUI in cooperation with a database will be shown. This database sample combines four different files.

- DB-terminal.fxml (Appendix)
- DB-terminal_controller.rexx
- DB-terminal_main.rexx
- DatabaseHandler.CLS

To get to the point were a database can be used by a Graphical User Interface, a so called JDBC-Driver has to be installed. For this example SQLite-JDBC was used and can be downloaded here:
https://github.com/xerial/sqlite-jdbc/releases. In this example,
$sqlite - jdbc - 3.32.3.2.jar$ was used. Furthermore, this variable must be declared in the class-path of your operating system.

To do so, follow the following steps:

1. Press the Windows key on your keyboard
2. Type the word “env”
3. Press enter
4. Click on “Environment Variable”, like in figure 7
5. Double click on CLASSPATH, like in figure 8
6. The “Edit environment variable” window will be opened
7. Click “New” and enter the path were
   $sqlite - jdbc - 3.32.3.2.jar$ is saved, like figure 9

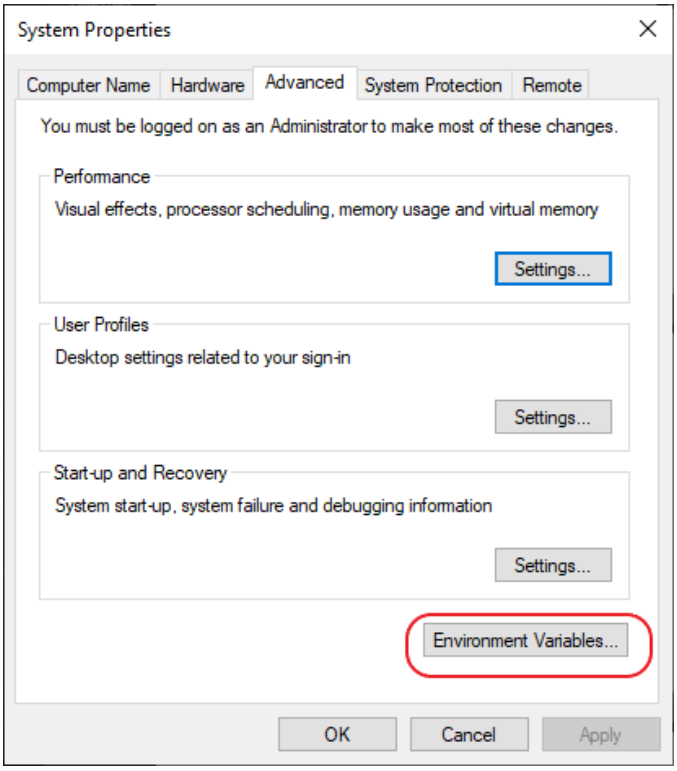

Figure 7: Environment Variable

This jar file is needed to translate between Java and the database language. In this case between Java and SQLite. Now, it is possible to use the GUI, shown in figure 10. Furthermore, a test database is need, which will be given as an extra file included in this paper, called “Nutshell-DB.db”. This

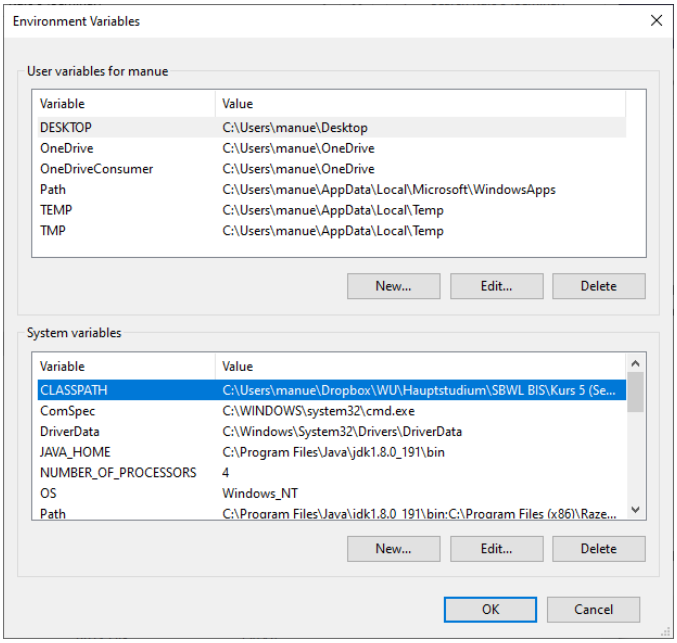

Figure 8: CLASSPATH

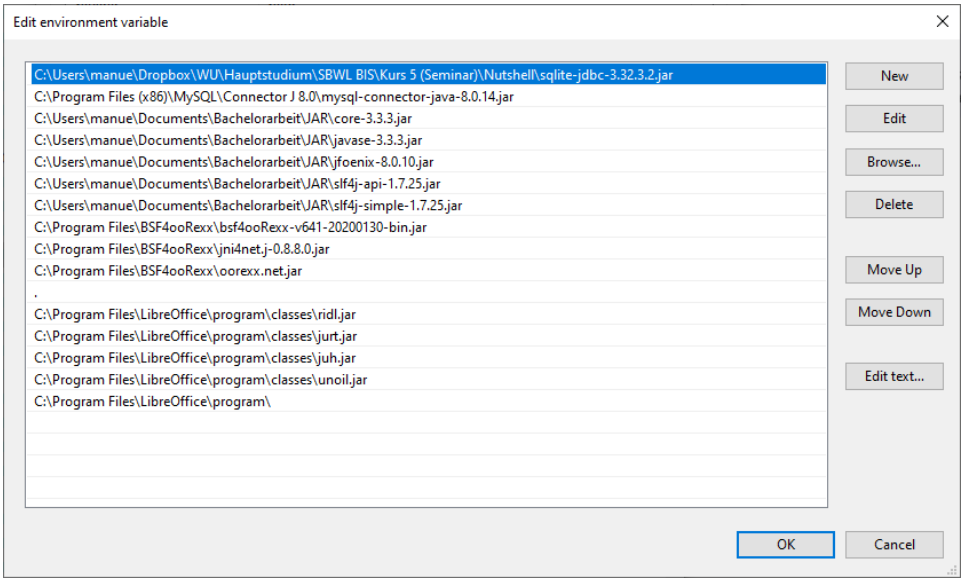

Figure 9: Edit environment variable

database must also be implemented within the code of the file “Database-Handler.CLS” This will be explained later in detail, for now, this is the class which contains the program’s logic of the database.

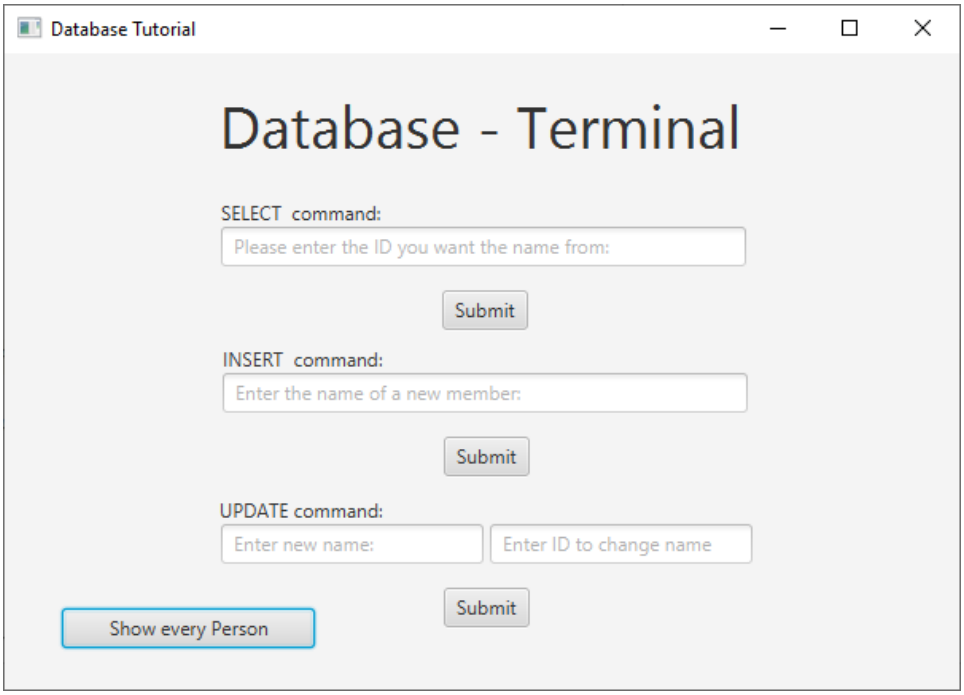

Figure 10: Database GUI

This nutshell example should point out, how powerful and useful a GUI can be. For testing purpose, this interface shows a few database operations which will be addressed by the components of the GUI. It is recommended that after the program started, the button “Show every Person” should be clicked. This will led to that every person, which was previously added to the database, will be shown in the command line where you started the program. For example, if you start the program via IntelliJ, names will be displayed in the output section, like in figure 11.

```
REXXout>[1 Hermann Maier]
REXXout>[2 Marcel Hirscher]
REXXout>[3 Hannes Reichelt]
REXXout>[4 Benjamin Reich]
```

Figure 11: Output: Person

After all names are displayed, it is much easier to experiment with the additional functions. The first function, which will be explained, is the SELECT function. With this function it is possible to gather information from a database. So, if a number, more specific the ID of a person is written into the textfield and the button is clicked, the associated name will be displayed in the textfield. For example, if the number 1 is entered, the name "Hermann Maier" will be displayed.

The middle section, the INSERT command, of the GUI is used to implement a person into the database. So, if a new person or member should be added, only the name must be written into the textfield. The associated ID will be added automatically. So, if a name is already written down into the textfield and after that the button is clicked, the previously written name will be transmitted to the database. For example, if a new name like "Armin Assinger" should be saved in the database, the name has to be written in the text field. After that, the submit button has to be clicked. Now, the database will display six names, if the button "Show every person" is clicked.

The last section, the UPDATE command, is used to change a person's name. Therefore, it must be entered a new name and the associated ID from that person, which its name should be changed. So, if the name of Hermann Maier should be changed, enter the new name for example "Herminator" into the first of the two textfield. Into the second one, the associated ID must be entered, in this case the number 1. After clicking the submit button, the name of Hermann Maier changed to Herminator.

By now it should be clear what purpose "DB-terminal.fxml" has. This file is used for the design of the GUI and can be found within the Appendix. The next file which will be explained is the main file of the interface. After the explanation of "DB-terminal_main.rexx" its associated controller will be explained. The main file contains the main program, which must be started to see the GUI, the source code can be found underneath. And last but not least, the class "DatabaseHandler". This Class is needed to connect to a database and to interact with it. Without this file a communication would not work. The source code will be explained later.

```
1
2  ####### DB_terminal_main.rexx #######
3
4  #!/usr/bin/env rexx
5     -- change directory to program location such that relatively addressed
          resources can be found
6  parse source . . pgm
7  call directory filespec('L', pgm) -- change to the directory where the
       program resides
8
9  .environment~setEntry("my.app", .directory~new)
10 .my.app~homeDir = filespec('Location',fullPath)
11 .my.app~dbh = .DatabaseHandler~new
12
```

```
13 .my.app~dbh~initSettings
14
15 success = .my.app~dbh~connect
16 IF \success THEN CALL connectionError
17 else say "SUCCESS: Connected to DB"
18
19 rxApp=.RexxApplication~new -- create Rexx object that will control the FXML
       set up
20 jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
21 jrxApp~launch(jrxApp~getClass, .nil) -- launch the application, invokes "
       start"
22
23 EXIT 0
24
25 connectionError:
26 say "Not connected to DB"
27
28 ------------------------------------------------------------------------
29 ::REQUIRES "DatabaseHandler.CLS"
30 ::REQUIRES "BSF.CLS"
31
32 ------------------------------------------------------------------------
33 ::class RexxApplication -- implements the abstract class "javafx.application
       .Application"
34
35 ::method start -- Rexx method "start" implements the abstract method
36   use arg primaryStage -- fetch the primary stage (window)
37   primaryStage~setTitle("Database Tutorial")
38
39    -- create an URL for the FMXLDocument.fxml file (hence the protocol "file
         :")
40   fxmlUrl=.bsf~new("java.net.URL", "file:DB_terminal.fxml")
41    -- use FXMLLoader to load the FXML and create the GUI graph from its
         definitions:
42   rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)
43
44   scene=.bsf~new("javafx.scene.Scene", rootNode) -- create a scene for our
       document
45   primaryStage~setScene(scene) -- set the stage to our scene
46   primaryStage~show -- show the stage (and thereby our scene)
47 ------------------------------------------------------------------------
```

DB-terminal_main.rexx starts the GUI, more specific its FXML file, shown in line 41 and 43. But more important, before the GUI is displayed a new environmental variable will be created, which is shown in line 9 and 10. Furthermore, a new "DatabaseHandler" is generated in line 11, which triggers the init-method from the DatabaseHandler.CLS-file. After that, the initSettings-method is called, which creates a true path to the database. After that a connection is been initialized, shown in line 12.
Should an error occur, "connectionError:" will be triggered and a message will be displayed in the output section of your program.

```
1
2  ####### DB-terminal_controller.rexx #######
3
4  ::routine selectUserName public
5    use arg slotDir
6    scriptContext=slotDir~scriptContext -- get the slotDir entry
7
8    /* @get(textFieldSelect) */
9    inputID = textFieldSelect~text
10   say inputID
11   name = .my.app~dbh~selectUserName(inputID)
12   say name
13   textFieldSelect~setText(name)
14
15 ------------------------------------------------------------------------------
16 ::routine insertUserName public
17   use arg slotDir
18   scriptContext=slotDir~scriptContext -- get the slotDir entry
19
20   /* @get(textFieldInsert) */
21   name = textFieldInsert~text
22   say name
23   .my.app~dbh~insertUserName(name)
24
25 ------------------------------------------------------------------------------
26 ::routine updateUserNameByID public
27   use arg slotDir
28   scriptContext=slotDir~scriptContext -- get the slotDir entry
29
30   /* @get(textFieldUpdateName textFieldUpdateID) */
31   name = textFieldUpdateName~text
32   id = textFieldUpdateID~text
33   .my.app~dbh~updateUserNameByID(name, id)
34
35 ------------------------------------------------------------------------------
36 ::routine listAllUserNameByID public
37   use arg slotDir
38   scriptContext=slotDir~scriptContext -- get the slotDir entry
39
40   .my.app~dbh~getlistAllUser
41
42 ------------------------------------------------------------------------------
43    ::REQUIRES "DatabaseHandler.CLS"
44    ::REQUIRES "BSF.CLS"
```

A controller file is handling all interactions between the GUI and its behavior. As already said, every method, every work order has to be declared here. Within this controller it is possible to address other files, like in this case a database class. For testing purpose, some general database function were implemented.
So, as shown in the source code above, the first routine in this scenario is

the SELECT function of the database terminal. So, if the submit button is clicked a routine called “selectUserName” will be called. This routine first fetches the information within the textfield. To do so “/* get(textFieldSelect) */ ” will be used. This line of code will gather information from the fx:id of a GUI component. In this case, the first textfield is called “textFieldSelect” within the FXML file. So, as already said, with the /*get (xyz) */ command, it is possible to get access to a GUI-component and its information stored.
In line 9 of the source code, it is shown that this information of the textfield will be stored in a ooRexx variable, for later use. In line 11, the real database interaction happens. With the previously created variable “inputID” a function called “selectUserName(inputID)” will be called. This method is stored in DatabaseHandler.CLS and uses the inputID as an input variable. Furthermore, is uses its environment variable, that was previously initialed by the main program. All kinds of database function will be described underneath, when the DatabaseHandler will be explained. Last, the gathered information from the database will be displayed in the same textfield.

The second routine within this file is the “insertUserName” routine. This routine gathers information from a textfield like the first one, but instead of displaying some names it will store new names in the database. For this case, In line 20, the textfield’s information will be gathered. The entered name will also be an input variable for another database interaction function, which will also be explained later.

The third routine, the “updateUserNameByID”, is a bit more complex because we need to gather information from two different textfields, that can be seen in line 30. After this, the database function “updateUserNameByID” with its two input variables is called.

The last routine in this file will be called by the button “Show every Person”, which also calls a function within the database handler underneath. So as can be seen in this scenario, many different files combined will led to a much more powerful GUI.

```
1
2  ####### DatabaseHandler.CLS #######
3
4  ::CLASS DatabaseHandler PUBLIC
5  ::METHOD conn ATTRIBUTE
6  ::METHOD DB_URL ATTRIBUTE
7  ::METHOD DriverManager ATTRIBUTE
8  ::METHOD init
9    EXPOSE DriverManager
10   DriverManager = bsf.import("java.sql.DriverManager")
11   ------------------------------------------------------------------------
12 ::METHOD initSettings PUBLIC
13   EXPOSE DB_URL
14     DB_URL = "jdbc:sqlite:C:/users/manue/Dropbox/WU/Hauptstudium/SBWL BIS/
         Kurs 5 (Seminar)/Nutshell/Nutshell-DB.db"
```

```
15 ------------------------------------------------------------------------------
16 ::METHOD connect PUBLIC
17   EXPOSE DriverManager DB_URL conn
18   SIGNAL ON SYNTAX NAME connectionError
19   conn = DriverManager~getConnection(DB_URL)
20   SIGNAL OFF SYNTAX
21   RETURN .true
22 ------------------------------------------------------------------------------
23 ::METHOD selectUserName PUBLIC
24   EXPOSE conn
25   USE ARG ID
26   query = "SELECT name FROM Person WHERE ID = ?"
27   prepStatement = conn~prepareStatement(query)
28   prepStatement~setString(1, ID)
29   queryResults = prepStatement~executeQuery
30   IF queryResults~next THEN DO
31       name = queryResults~getString("name")
32    END
33    ELSE
34    name = "This ID does not exist here!"
35    return name
36 ------------------------------------------------------------------------------
37 ::METHOD insertUserName PUBLIC
38   EXPOSE conn
39   USE ARG name
40   query = "INSERT INTO Person (name) VALUES (?)"
41   prepStatement = conn~prepareStatement(query)
42   prepStatement~setString(1, name)
43   prepStatement~execute
44 ------------------------------------------------------------------------------
45 ::METHOD updateUserNameByID PUBLIC
46   EXPOSE conn
47   USE ARG name, id
48   query = "UPDATE Person SET name = ? Where ID = ?"
49   prepStatement = conn~prepareStatement(query)
50   prepStatement~setString(1, name)
51   prepStatement~setString(2, id)
52   prepStatement~execute
53 ------------------------------------------------------------------------------
54 ::METHOD getlistAllUser PUBLIC
55   EXPOSE conn
56   query = "SELECT * FROM Person"
57   prepStatement = conn~createStatement
58   queryResults = prepStatement~executeQuery(query)
59   index = 1;
60
61     DO WHILE queryResults~next
62       id = queryResults~getString("ID")
63       name = queryResults~getString("Name")
64       nameList.index = name;
65       idList.index = id;
66       say pp(idList.index nameList.index);
67       index = index + 1
68     END
```

```
69
70 ------------------------------------------------------------------------
71 ::REQUIRES "BSF.CLS"
```

As already mentioned, “DatabaseHandler.CLS” contains all methods, which interact with a database.
This class consists of three different attributes and seven different methods. The first attribute is the conn attribute which stores the information about the database’s connection. The second attribute is the database’s url. This url contains the information, more specific the path where the database is stored. This path can be changed and adapted. Last, the “DriverManager” attribute saves the information, gathered from the Java class “java.sql.DiverManager”. This driver manager helps to interact with the database and it is written in Java. But, as it is shown in line 11, it can be used by ooRexx. Every time a class is newly initialized, first the constructor will be called. In this case, the init method is called, which initializes the driver manager.

This initialization process can be seen in DB-terminal_main.rexx . The first method, which is shown in line 12, is therefore to set the right path for the database. This method is called by the main program.To be able to test the nutshell program it is necessary that the path of the URL will be adapted. Write the true path, where the “Nutshell-DB.db” is stored, down there in line 14. The shown URL was generated for testing purpose.

The third method shown in line 16, is the method which handles the connection to the database and stores the information about it.
The fourth method is the first one, which is called by a GUI component itself. In line 24 the connection will be exposed and used. In line 25 the input variable for this method is declared. Line 27 and 28 create the query statement for the database, in this case a SELECT query is generated where the name of an associated ID will be given in return. The “setString(1,ID)” part is used to replace the question mark in the query with the entered ID. After that, the query will be executed. If there is a name associated with this ID the name will be given in return, if not “This ID does not exist here!” will be displayed.

The fifth method, “insertUserName”, is therefore to enter a new person in the database. The associated ID will be generated automatically. The sixth method, as shown in line 44 is used for the UPDATE-function. The last method is therefore to list all previously entered IDs and names.

These four files listed above will be enough to interact with a database via JavaFX/FXML and ooRexx.

3.4 JFoenix - Class

In this subsection it will be shown, how versatile SceneBuilder and its components are. Figure 12 demonstrates the appearance of default components of SceneBuilder and newly added components of a class called "JFoenix". This class contains many different components mostly with a modern design in comparison to the default ones. The two files needed, main and .fxml, can be be found within the Appendix, because there is not much which differs from the previous ones. Furthermore, to create components in JFoenix style, the class must be downloaded and stored at first. In this subsection it will be shown how it is done. Follow these few steps:

1. Download JFoenix here: https://github.com/jfoenixadmin/JFoenix (In this scenario jfoenix-9.0.10.jar is used)
2. Open SceneBuilder and Press the gear symbole, like in figure 13. Then click on "JAR/FXML Manager". The Library Manager will be openend.
3. Click on "Add Library/FXML from file system"
4. Go to the subdirectory where jfoenix-9.0.10.jar was downloaded and double click it
5. A window like figure 14 will be opened
6. Click "Import Components"

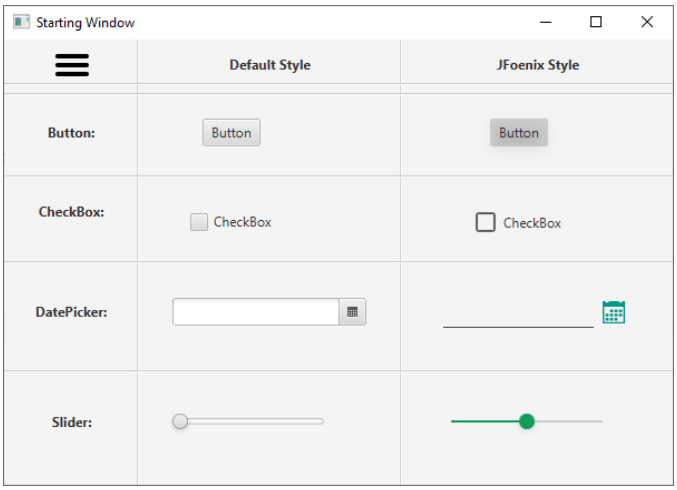

Figure 12: Comparison between components

Now, SceneBuilder contains many new components under tab “Custom” on the left side. With these components it is possible to build a GUI in a more modern way. As it is shown in figure 12, four different kinds of components, which are often used in graphical user interfaces, where contrasted. The first comparison is between two different kinds of buttons. The left one, the default styled one, looks very old fashioned and blank. The right one, the button within the JFoenix class, looks a bit more modern and fancy. It also displays a black colored wave within the button, at that moment it is clicked. Also the colored wave changes its direction. For example, if the cursor clicks on left side of the button, the wave comes also from the left side, if it clicks on the right side, the wave will come from the right side of the button.

The second component, which will be compared, are CheckBoxes. A CheckBox is often used within online surveys or online shopping carts. Based on the web side or survey, some of them uses more neutral CheckBoxes like the default one, but also some of them uses the more fancy and colored style, like the right CheckBox.

Furthermore, DatePicker are very common. As collecting data from costumers is very important for most companies, it is often seen that a web side asks for a date of birth. Like the previously mentioned components, this could be done in a more colorful way or just plain and simple.

Last but not least, the starting window in figure 12 shows two different kinds of sliders. The left one only can be moved from the left to the right and the other way round. The right one can also be moved like this, additionally it displays the value of the slider.

The main file and its associated FXML file can be found within the Appendix.

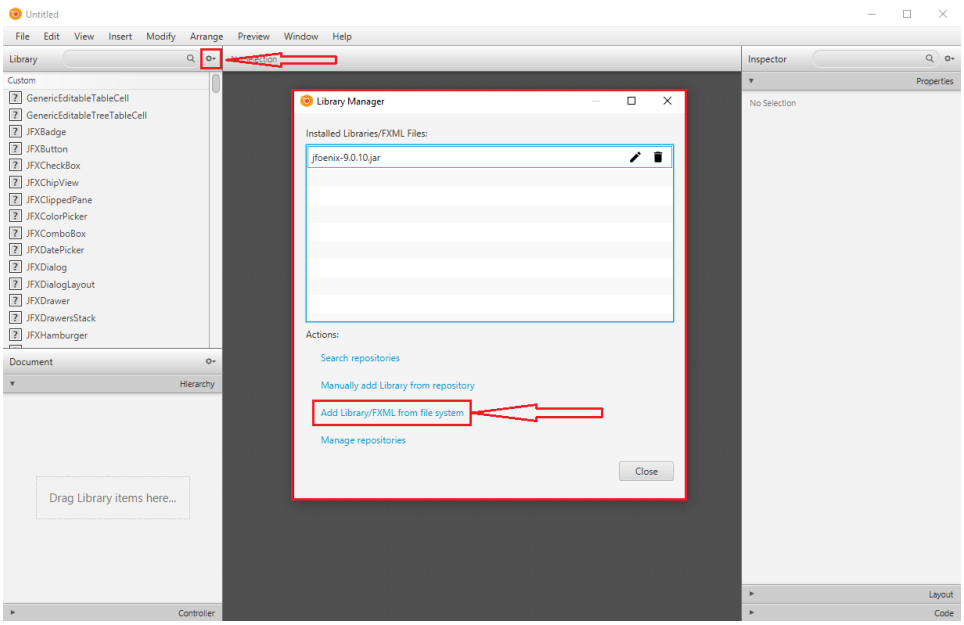

Figure 13: SceneBuilder - Library Manager

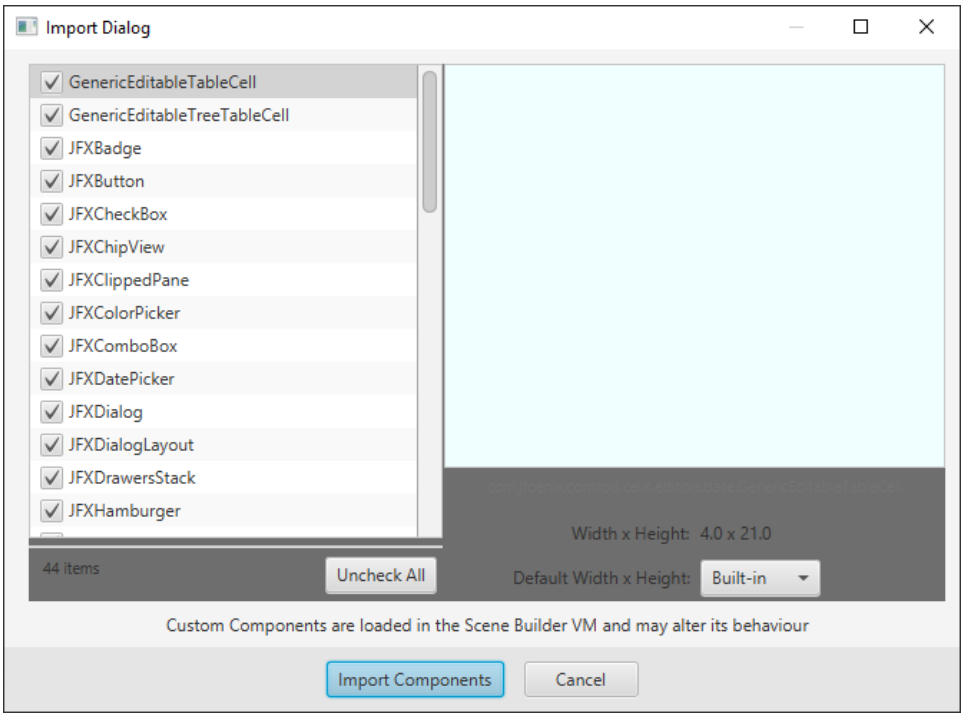

Figure 14: Import Dialog

3.5 Multiple Windows

In this subsection it will be shown, how to manage different windows within one GUI. The main window looks like figure 15. Within this interface it can be opened two different kinds of windows. One, figure 16, enlarges a picture and the second one, figure 17, will display a bar chart. These two types of windows were chosen to show various kinds of operation areas of GUIs itself. This nutshell example contains five different files, wich will be explained step by step underneath. Files containing:

- Windows_main.rexx
- first_window.fxml (Appendix)
- enlarge_barChart_controller.rexx
- enlargePicture.fxml (Appendix)
- openStats.fxml (Appendix)

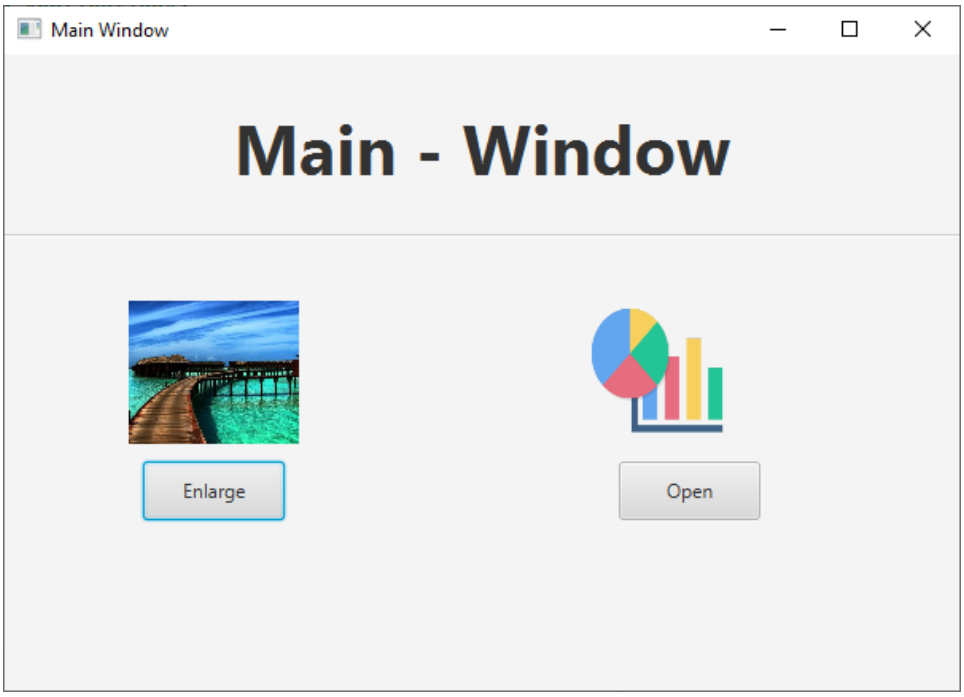

Figure 15: Main Window

Underneath the source code of Windows_main.rexx is shown. This main file differs compared to other nutshell examples because it is a bit more complex. To manage different windows within a GUI a so called “Stagehandler” is needed.

```
1
2  ####### Windows_main.rexx #######
3
4  PARSE SOURCE . . fullPath
5  CALL directory filespec('L', fullPath)
6
7  .environment~setEntry("my.app", .directory~new)
8  .my.app~homeDir = filespec('Location',fullPath)
9  stageHandler = .StageHandler~new
10 .my.app~stageHandler = stageHandler
11
12 stageHandlerProxy = BsfCreateRexxProxy(stageHandler,,"javafx.application.
       Application")
13 stageHandlerProxy~launch(stageHandlerProxy~getClass, .nil)
14 EXIT 0
15
16 ------------------------------------------------------------------------
17 ::CLASS StageHandler
18 ::METHOD stage ATTRIBUTE
19 ::METHOD scene ATTRIBUTE
20 ::METHOD windowStage ATTRIBUTE
21 ::METHOD FXMLLoader
22 ::METHOD init
23   EXPOSE FXMLLoader
24   FXMLLoader = bsf.import("javafx.fxml.FXMLLoader")
25
26 ------------------------------------------------------------------------
27 ::METHOD start
28   EXPOSE stage scene FXMLLoader
29   USE ARG stage
30
31   stage~setTitle("Main Window")
32   url=.bsf~new("java.net.URL", "file:first_window.fxml")
33   fxml = FXMLLoader~load(url)
34   scene = .bsf~new("javafx.scene.Scene", fxml)
35   stage~setScene(scene)
36   stage~show
37
38 ------------------------------------------------------------------------
39 ::METHOD newWindow
40   EXPOSE stage windowStage FXMLLoader
41   USE ARG title, fileName
42   windowStage = .bsf~new("javafx.stage.Stage")
43   windowStage~setTitle(title)
44   url =.bsf~new("java.net.URL", fileName)
45   fxml = FXMLLoader~load(url)
46   scene = .bsf~new("javafx.scene.Scene", fxml)
47   windowStage~setScene(scene)
48   --windowStage~initOwner(stage)
49   windowStage~show
50 ------------------------------------------------------------------------
51 ::REQUIRES "BSF.CLS"
```

The complexity exists because some kind of logic must be implemented, which manages the coordination between several different windows. In this case, it is the class “Stagehandler”. A “StageHandler” does nothing more than to coordinate windows. In line 7 to 10 can be seen that an environment variable will be created, after the main program was started. After this, a new stage handler will be initialized, which invokes the method init. This newly initialized stage handler will be stored in the environment variable “.my.app˜stageHandler”, which helps with future coordination. In line 12 and 13 a proxy of the class “StageHandler” will be initialized and launched, which will invoke the start method. These two steps, creating an environment variable and launching a proxy, mainly the latter, can also be found in all other main files of this paper. Only the class StageHandler differs.
If only a GUI with one window is requirded, it is not needed to manage different windows, so, one primary stage is enough. This difference can be seen if other main files will be compared to this one. Within the invocation of the method start, the title of the GUI will be set, the URL of the FXML will be loaded and the scene, first_window.fxml, will be displayed.

Within this interface it is possible to invoke two additional GUIs. So, if the button “Enlarge” is clicked, a newly created window, enlargePicture.fxml, will be opened. To do so, the routine “enlargePicture” within enlarge_barChart_controller.rexx will be called. The specific function of this routine will be explained later, when the controller file will be explained. The used method to generate additional windows is called “newWindow” in line 39. Therefore, two input variables will be required. One, the title of the new window. Second, the file name of the window which should be opened, for this example enlargePicture.fxml. In line 42, a new stage will be initialized. This will led to the fact, that a new GUI will be created, which does not replace the main window. So, if the button is clicked, figure 16 will be displayed.

The previously opened GUI, can be moved around, so the enlarged picture is not on top of the main window. With the line of code in line 48, which is commented out now, it is possible to apply the ability that a new window can only be opened, if the previous opened was closed. So, if this line of code is not implemented, many different windows can be opened, but, if this line of code is active, every additional window apart from the main window has to be closed to open a new one.

Furthermore, if the button “Open” is clicked, the statistics interface will be opened. This interface contains a bar chart, which is also a JavaFX component within SceneBuilder. It is also possible to generate test values, after clicking the “See Stats” button. This circumstance is shown in figure 18 underneath.

After explaining the main file, it is now time to explain the controller of the main window, more specific the file: enlarge_barChart_controller.rexx, which its source code is underneath.

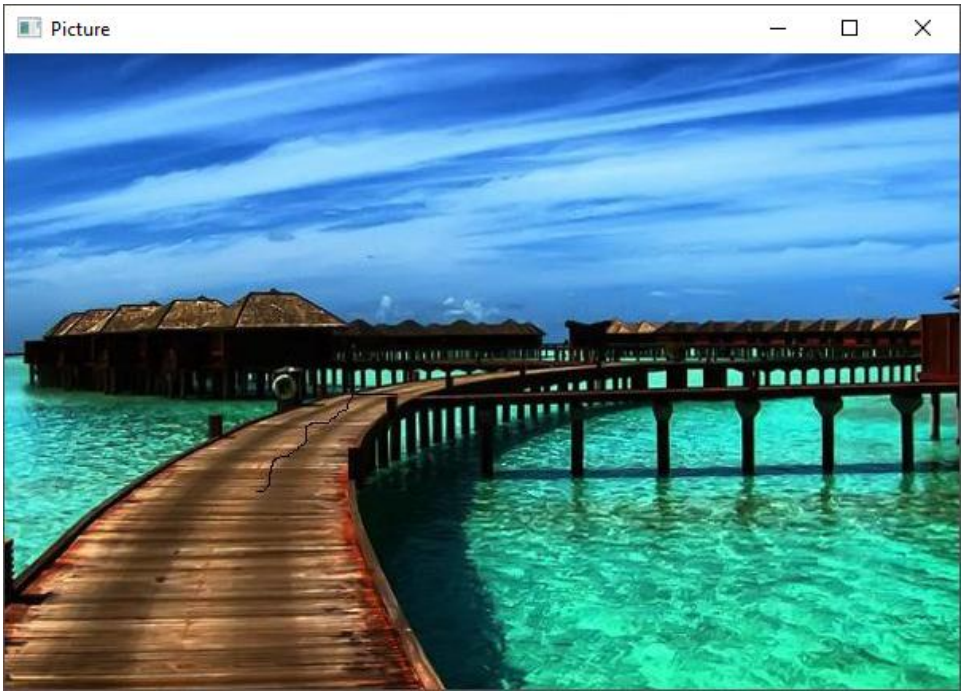

Figure 16: Enlarged Picture

```
1
2  ####### enlarge_barChart_controller.rexx #######
3
4  #!/usr/bin/env rexx
5
6  ::routine openStat public
7    use arg slotDir
8    scriptContext=slotDir~scriptContext
9    URL = "file:openStats.fxml"
10   .my.app~stageHandler~newWindow("Statistics", URL)
11
12 ------------------------------------------------------------------------
13 ::routine enlargePicture public
14   use arg slotDir
15   scriptContext=slotDir~scriptContext -- get the slotDir entry
16   URL = "file:enlargePicture.fxml"
17   .my.app~stageHandler~newWindow("Picture", URL)
18
19 ------------------------------------------------------------------------
20 ::routine openSt public
21   use arg slotDir
22   scriptContext=slotDir~scriptContext -- get the slotDir entry
23   /*@get(barChart yAxis xAxis)*/
24   barChart~getData~clear
25
```

```
26   XYChartData = bsf.import("javafx.scene.chart.XYChart$Data")
27   XYChartSeries = bsf.import("javafx.scene.chart.XYChart$Series")
28
29   dataSeries = XYChartSeries~new
30   dataSeries2 = XYChartSeries~new
31   dataSeries3 = XYChartSeries~new
32
33   yAxis~setLabel("yAxis")
34   xAxis~setLabel("xAxis")
35
36   temp1 = box("String", "July")
37   tempfloat1 = box("float", 5)
38   dataSeries~getData~add(XYChartData~new(temp1, tempfloat1))
39   barChart~getData~add(dataSeries)
40
41   temp2 = box("String", "August")
42   tempfloat2 = box("float", 200)
43   dataSeries2~getData~add(XYChartData~new(temp2, tempfloat2))
44   barChart~getData~add(dataSeries2)
45
46   temp3 = box("String", "September")
47   tempfloat3 = box("float", 50)
48   dataSeries3~getData~add(XYChartData~new(temp3, tempfloat3))
49   barChart~getData~add(dataSeries3)
50 ------------------------------------------------------------------------
51 ::REQUIRES "BSF.CLS"
52 ------------------------------------------------------------------------
```

The first routine within this file is the routine called “openStat”. This routine will be be called , whenever the button “Open” in the main window is clicked. Within this function a URL is generated, which will be forwarded to the stagehandler’s method called “newWindow”. As explained earlier, newWindow needs a title and a filename or a true path to the FXML file, which should be opened. More specific, the “.my.app~stageHandler” environment variable, which stores the information about the stage handler of the GUI, will be addressed with its method “newWindow” and the associated input variables, in this case, the title “Picture” and the filename of openStats.fxml.

The second routine within this source code is “enlargePicture” and is apart from the name of the file and the title of the window identical with the first routine.

The third routine, the “openSt” routine, creates the values for the bar chart. If the routine to open the openStats.fxml is called, the bar chart remains blank until the button “See Stats” is clicked. The button will call the routine which generates the values shown between line 18 and 48.
First of all, there has to be some kind of access to the bar chart. This will be done by the line of code in line 21. There it is shown, that every fx:id mentioned in the associated FXML file can be addressed.

After that, the bar chart gets cleared. This has to be done, because otherwise some layout errors can occur. After these two lines, two Java classes have to be imported for later use. The first one is needed to generate data for the bar chart, the second one to generate bundles or series of data. For every colored bar, a new data series has to be created. If every bar of this bar chart is of the same data series, only create one series and simply add the data to that one. For only one series of data use the same "dataSeries" every time.
But, in this scenario there are three different data series to handle with. They are getting initialized between line 28 and 30. Further, the label names of the bar chart are getting added as well.

To be able to generate series of data for Java within ooRexx, line 34 is needed. As it is shown there, XYChartData needs two input variables. One, is a string variable and the other one is, in this case, a float variable. To get these ooRexx variables into Java, they have to be boxed, as it is shown in the source code above. The remaining series of data work the same. The result of this can be seen in figure 18.

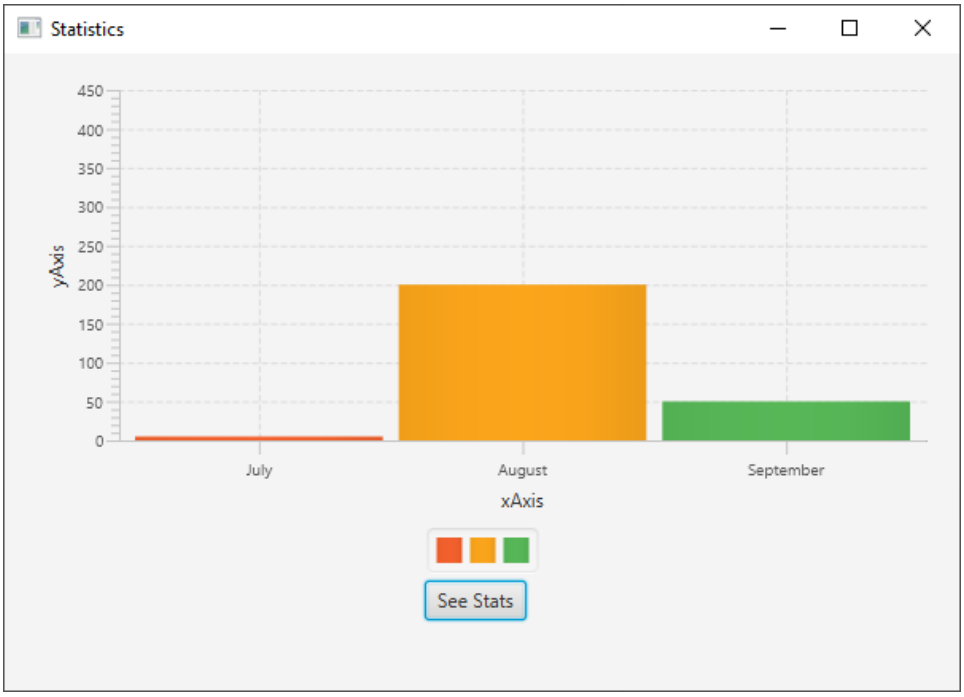

Figure 17: Open BarChart

4 Conclusion

This paper generates a brief overview about JavaFX combined with FXML. To show their versatile operation areas five different nutshell examples were generated. To be able to test these examples a few software components are required. First of all, Java has to be installed. Second, ooRexx and its Bean Scripting Framework BSF4ooRexx have to be implemented as well. Additionally, an application called "SceneBuilder" is required, which helps with the designing process of GUI. Optionally, "DB Browser for SQLite" can be downloaded and installed to get access to the database without any source code. The database can simply be opened within the application.

Moreover, every source code or file, which may a bit complex are explained within this paper. Files, which repeat themselves can be found within the Appendix.

5 Outlook

This paper can not refer to all kinds of aspects of JavaFX. This paper contains only a peek of JavaFX, to be able to interact and to get in touch with JavaFX. Furthermore, there is a ton of operation areas, in which JavaFX can be used on. For this circumstance visit the homepage of OpenJFX and take a look at the documentary of JavaFX. As an example, there are 3D shaped GUI applications or some application with a sound componend. Furthermore, animations can be generated with JavaFX [6].

References

[1] OpenJFX. openjfx.io. Accessed: 2020-11-26.

[2] JavaFX. https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html. Accessed: 2020-11-27.

[3] GluonHQ. https://gluonhq.com/products/scene-builder/. Accessed: 2020-11-27.

[4] w3 XML. https://www.w3.org/TR/REC-xml/. Accessed: 2020-12-12.

[5] Java SE doc. https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html. Accessed: 2020-11-12.

[6] Open Source Community of OpenJFX. https://wiki.openjdk.java.net/display/OpenJFX. Accessed: 2020-11-12.

[7] "Rony G. Flatscher". Automatisierung mit ooRexx und BSF4ooRexx. https://subs.emis.de/LNI/Proceedings/Proceedings208/307.pdf. Accessed: 2020-11-13.

[8] "Hakon Lie, Bert Bos, and Chris Lilley ". The text/css Media Type. RFC 2318, RFC Editor, March 1998.

List of Figures

A
My first GUI

```
1
2 ####### My_first_GUI.rexx (Main - File) #######
3
4 #!/usr/bin/env rexx
5
6 parse source . . pgm
7 call directory filespec('L', pgm) -- change to the directory where the
      program resides
8
9 rxApp=.RexxApplication~new -- create Rexx object that will control the FXML
      set up
10 jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
11 jrxApp~launch(jrxApp~getClass, .nil) -- launch the application, invokes "
      start"
12
13 ------------------------------------------------------------------------
14
15 ::requires "BSF.CLS" -- get Java support
16
17 ------------------------------------------------------------------------
18
19 ::class RexxApplication -- implements the abstract class "javafx.application
      .Application"
20
21 ::method start -- Rexx method "start" implements the abstract method
22   use arg primaryStage -- fetch the primary stage (window)
23   primaryStage~setTitle("My first GUI!")
24
25    -- create an URL for the FMXLDocument.fxml file (hence the protocol "file
         :")
26   fxmlUrl=.bsf~new("java.net.URL", "file:My_first_GUI.fxml")
27    -- use FXMLLoader to load the FXML and create the GUI graph from its
         definitions:
28   rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)
29
30   scene=.bsf~new("javafx.scene.Scene", rootNode) -- create a scene for our
       document
31   primaryStage~setScene(scene) -- set the stage to our scene
32   primaryStage~show -- show the stage (and thereby our scene)
33
34 ------------------------------------------------------------------------
```

Within this source code it is shown, how to start a GUI with only one interface to look at. This is a simple way to display a GUI.

```
1
2  <!-- My_first_GUI.fxml (FXML - File) -->
3
4  <?xml version="1.0" encoding="UTF-8"?>
5
6  <?import javafx.scene.control.Button?>
7  <?import javafx.scene.control.Label?>
8  <?import javafx.scene.control.TextField?>
9  <?import javafx.scene.layout.AnchorPane?>
10 <?import javafx.scene.text.Font?>
11
12 <?language rexx?>
13
14 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
       minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http
      ://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">
15    <fx:script source="My_first_GUI_controller.rexx" />
16    <children>
17       <Label alignment="CENTER" layoutX="63.0" layoutY="46.0" prefHeight="
             44.0" prefWidth="469.0" text="My first GUI" textAlignment="CENTER"
             >
18          <font>
19             <Font name="Bauhaus 93" size="48.0" />
20          </font>
21       </Label>
22       <TextField fx:id="textField1" layoutX="169.0" layoutY="159.0"
             prefHeight="14.0" prefWidth="263.0" promptText="Write something in
              here..." />
23       <Button fx:id="button1" layoutX="265.0" layoutY="251.0" onAction="
             slotDir=arg(arg()); call buttonClicked slotDir;" mnemonicParsing="
             false" text="Click me!" />
24    </children>
25 </AnchorPane>
```

The source code above shows the content within the FXML file. This FXML file will be called by the associated main application to display the "My first GUI" interface.

B
My first GUI with CSS

```
1
2 ####### My_first_CSS_main.rexx #######
3
4 #!/usr/bin/env rexx
5
6 parse source . . pgm
7 call directory filespec('L', pgm) -- change to the directory where the
      program resides
8
9 rxApp=.RexxApplication~new -- create Rexx object that will control the FXML
      set up
10 jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
11 jrxApp~launch(jrxApp~getClass, .nil) -- launch the application, invokes "
      start"
12
13 ------------------------------------------------------------------------
14 ::requires "BSF.CLS" -- get Java support
15 ------------------------------------------------------------------------
16
17 -- Rexx class defines "javafx.application.Application" abstract method "
      start"
18 ::class RexxApplication -- implements the abstract class "javafx.application
      .Application"
19
20 ::method start -- Rexx method "start" implements the abstract method
21   use arg primaryStage -- fetch the primary stage (window)
22   primaryStage~setTitle("My first GUI with CSS!")
23
24    -- create an URL for the FMXLDocument.fxml file (hence the protocol "file
         :")
25   fxmlUrl=.bsf~new("java.net.URL", "file:My_first_GUI_CSS.fxml")
26    -- use FXMLLoader to load the FXML and create the GUI graph from its
         definitions:
27   rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)
28
29   scene=.bsf~new("javafx.scene.Scene", rootNode) -- create a scene for our
       document
30   primaryStage~setScene(scene) -- set the stage to our scene
31   primaryStage~show -- show the stage (and thereby our scene)
```

C
SQLite - JDBC

```
1
2  <!-- DB_terminal.fxml -->
3
4  <?xml version="1.0" encoding="UTF-8"?>
5
6  <?import javafx.scene.control.Button?>
7  <?import javafx.scene.control.Label?>
8  <?import javafx.scene.control.TextField?>
9  <?import javafx.scene.layout.AnchorPane?>
10 <?import javafx.scene.text.Font?>
11 <?language rexx?>
12
13 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
       minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http
      ://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">
14 <fx:script source="DB_terminal_controller.rexx" />
15 <children>
16       <Label layoutX="135.0" layoutY="20.0" prefHeight="54.0" prefWidth="
             330.0" text="Database - Terminal">
17          <font>
18             <Font size="37.0" />
19          </font>
20       </Label>
21       <TextField fx:id="textFieldSelect" layoutX="136.0" layoutY="109.0"
             prefHeight="25.0" prefWidth="330.0" promptText="Please enter the
             ID you want the name from:" />
22       <Button fx:id="buttonSelect" layoutX="275.0" layoutY="149.0"
             mnemonicParsing="false" onAction="slotDir=arg(arg()); call
             selectUserName slotDir;" text="Submit" />
23       <TextField fx:id="textFieldInsert" layoutX="137.0" layoutY="201.0"
             prefHeight="25.0" prefWidth="330.0" promptText="Enter the name of
             a new member:" />
24       <Button fx:id="buttonInstert" layoutX="276.0" layoutY="241.0"
             mnemonicParsing="false" onAction="slotDir=arg(arg()); call
             insertUserName slotDir;" text="Submit" />
25       <TextField fx:id="textFieldUpdateName" layoutX="136.0" layoutY="296.0"
              prefHeight="25.0" prefWidth="165.0" promptText="Enter new name:"
             />
26       <Button fx:id="buttonUpdate" layoutX="276.0" layoutY="336.0"
             mnemonicParsing="false" onAction="slotDir=arg(arg()); call
             updateUserNameByID slotDir;" text="Submit" />
27       <TextField fx:id="textFieldUpdateID" layoutX="305.0" layoutY="296.0"
             prefHeight="25.0" prefWidth="165.0" promptText="Enter ID to change
              name" />
28       <Label layoutX="136.0" layoutY="92.0" prefHeight="17.0" prefWidth="
             330.0" text="SELECT command:" />
29       <Label layoutX="137.0" layoutY="184.0" prefHeight="17.0" prefWidth="
             330.0" text="INSERT command:" />
30       <Label layoutX="135.0" layoutY="279.0" prefHeight="17.0" prefWidth="
             330.0" text="UPDATE command:" />
```

```
31      <Button layoutX="36.0" layoutY="349.0" mnemonicParsing="false"
            prefHeight="25.0" prefWidth="159.0" onAction="slotDir=arg(arg());
            call listAllUserNameByID slotDir;" text="Show every Person" />
32    </children>
33 </AnchorPane>
```

The XML code above is used to generate a simple terminal window for a database.

D JFoenix - Class

```
1
2  <!-- normal_vs_jfoenix.fxml -->
3
4  <?xml version="1.0" encoding="UTF-8"?>
5
6  <?import com.jfoenix.controls.JFXButton?>
7  <?import com.jfoenix.controls.JFXCheckBox?>
8  <?import com.jfoenix.controls.JFXDatePicker?>
9  <?import com.jfoenix.controls.JFXHamburger?>
10 <?import com.jfoenix.controls.JFXSlider?>
11 <?import javafx.scene.control.Button?>
12 <?import javafx.scene.control.CheckBox?>
13 <?import javafx.scene.control.DatePicker?>
14 <?import javafx.scene.control.Label?>
15 <?import javafx.scene.control.Separator?>
16 <?import javafx.scene.control.Slider?>
17 <?import javafx.scene.layout.AnchorPane?>
18 <?import javafx.scene.text.Font?>
19
20
21 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
       minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http
      ://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1">
22    <children>
23       <Label alignment="CENTER" layoutX="120.0" layoutY="14.0" prefHeight="
             17.0" prefWidth="236.0" text="Default Style">
24          <font>
25             <Font name="System Bold" size="12.0" />
26          </font>
27       </Label>
28       <Separator layoutX="353.0" orientation="VERTICAL" prefHeight="400.0"
             prefWidth="7.0" />
29       <Label alignment="CENTER" layoutX="361.0" layoutY="14.0" prefHeight="
             17.0" prefWidth="236.0" text="JFoenix Style">
30          <font>
31             <Font name="System Bold" size="12.0" />
32          </font>
33       </Label>
```

```
34      <Separator layoutX="117.0" orientation="VERTICAL" prefHeight="400.0"
            prefWidth="7.0" />
35      <Separator layoutY="39.0" prefHeight="2.0" prefWidth="600.0" />
36      <Label alignment="CENTER" layoutX="7.0" layoutY="69.0" prefHeight="
            28.0" prefWidth="107.0" text="Button:">
37        <font>
38          <Font name="System Bold" size="12.0" />
39        </font>
40      </Label>
41      <Label alignment="CENTER" layoutX="7.0" layoutY="140.0" prefHeight="
            28.0" prefWidth="107.0" text="CheckBox:">
42        <font>
43          <Font name="System Bold" size="12.0" />
44        </font>
45      </Label>
46      <Label alignment="CENTER" layoutX="7.0" layoutY="230.0" prefHeight="
            28.0" prefWidth="107.0" text="DatePicker:">
47        <font>
48          <Font name="System Bold" size="12.0" />
49        </font>
50      </Label>
51      <Label alignment="CENTER" layoutX="7.0" layoutY="329.0" prefHeight="
            28.0" prefWidth="107.0" text="Slider:">
52        <font>
53          <Font name="System Bold" size="12.0" />
54        </font>
55      </Label>
56      <Separator layoutY="122.0" prefHeight="2.0" prefWidth="600.0" />
57      <Separator layoutY="199.0" prefHeight="2.0" prefWidth="600.0" />
58      <Separator layoutY="297.0" prefHeight="2.0" prefWidth="600.0" />
59      <Separator layoutY="48.0" prefHeight="2.0" prefWidth="600.0" />
60      <JFXButton buttonType="RAISED" layoutX="436.0" layoutY="71.0"
            ripplerFill="#171717" text="Button" />
61      <JFXCheckBox layoutX="423.0" layoutY="155.0" text="CheckBox" />
62      <JFXDatePicker layoutX="394.0" layoutY="232.0" />
63      <JFXSlider layoutX="399.0" layoutY="336.0" />
64      <JFXHamburger layoutX="46.0" layoutY="13.0" />
65
66      <DatePicker layoutX="151.0" layoutY="232.0" />
67      <Slider layoutX="151.0" layoutY="336.0" />
68      <CheckBox layoutX="167.0" layoutY="155.0" mnemonicParsing="false" text
            ="CheckBox" />
69      <Button layoutX="178.0" layoutY="71.0" mnemonicParsing="false" text="
            Button" />
70
71    </children>
72  </AnchorPane>
```

```
1  ####### normal_vs_jfoenix_main.rexx #######
2
3  PARSE SOURCE . . fullPath
4  CALL directory filespec('L', fullPath)
5
6  .environment~setEntry("my.app", .directory~new)
7  .my.app~homeDir = filespec('Location',fullPath)
8  stageHandler = .StageHandler~new
9  .my.app~stageHandler = stageHandler
10
11 stageHandlerProxy = BsfCreateRexxProxy(stageHandler,,"javafx.application.
       Application")
12 stageHandlerProxy~launch(stageHandlerProxy~getClass, .nil)
13 EXIT 0
14 ------------------------------------------------------------------------
15
16 ::CLASS StageHandler
17 ::METHOD stage ATTRIBUTE
18 ::METHOD scene ATTRIBUTE
19 ::METHOD windowStage ATTRIBUTE
20 ::METHOD FXMLLoader
21 ::METHOD init
22   EXPOSE FXMLLoader
23   FXMLLoader = bsf.import("javafx.fxml.FXMLLoader")
24
25 ::METHOD start
26   EXPOSE stage scene FXMLLoader
27   USE ARG stage
28   stage~setTitle("Starting Window")
29   url=.bsf~new("java.net.URL", "file:normal_vs_jfoenix.fxml")
30   fxml = FXMLLoader~load(url)
31   scene = .bsf~new("javafx.scene.Scene", fxml)
32   stage~setScene(scene)
33   stage~show
34 ------------------------------------------------------------------------
35 ::REQUIRES "BSF.CLS"
36 ------------------------------------------------------------------------
```

As can be seen here, a "StageHandler" is used even though only one window is displayed. This is shown for testing purpose. Both approaches, the primary stage or stage handler, can be used.

E
Multiple Windows

```
1
2 <!-- first_window.fxml -->
3
4 <?xml version="1.0" encoding="UTF-8"?>
5
6 <?import javafx.scene.control.Button?>
7 <?import javafx.scene.control.Label?>
8 <?import javafx.scene.control.Separator?>
9 <?import javafx.scene.image.Image?>
10 <?import javafx.scene.image.ImageView?>
11 <?import javafx.scene.layout.AnchorPane?>
12 <?import javafx.scene.text.Font?>
13 <?language rexx?>
14
15 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
      minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http
     ://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">
16    <fx:script source="enlarge_barChart_controller.rexx" />
17    <children>
18       <Label alignment="CENTER" layoutX="51.0" layoutY="29.0" prefHeight="
             38.0" prefWidth="498.0" text="Main - Window">
19          <font>
20             <Font name="System Bold" size="43.0" />
21          </font>
22       </Label>
23       <Button layoutX="87.0" layoutY="256.0" mnemonicParsing="false"
             prefHeight="37.0" prefWidth="89.0" onAction="slotDir=arg(arg());
             call enlargePicture slotDir;" text="Enlarge" />
24       <ImageView fitHeight="90.0" fitWidth="107.0" layoutX="78.0" layoutY="
             155.0">
25          <image>
26             <Image url="@background.jpeg" />
27          </image>
28       </ImageView>
29       <Separator layoutY="113.0" prefHeight="0.0" prefWidth="600.0" />
30       <ImageView fitHeight="106.0" fitWidth="121.0" layoutX="354.0" layoutY=
             "147.0">
31          <image>
32             <Image url="@chart-icon-color.png" />
33          </image>
34       </ImageView>
35       <Button layoutX="386.0" layoutY="256.0" mnemonicParsing="false"
             prefHeight="37.0" prefWidth="89.0" onAction="slotDir=arg(arg());
             call openStat slotDir;" text="Open" />
36    </children>
37 </AnchorPane>
```

This is the main GUI within the nutshell example “Multiple Windows”.

```
1
2 <!-- enlargePicture.fxml -->
3
4 <?xml version="1.0" encoding="UTF-8"?>
5
6 <?import javafx.scene.image.Image?>
7 <?import javafx.scene.image.ImageView?>
8 <?import javafx.scene.layout.AnchorPane?>
9
10
11 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
      minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http
     ://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1">
12    <children>
13       <ImageView fitHeight="400.0" fitWidth="600.0">
14          <image>
15             <Image url="@background.jpeg" />
16          </image>
17       </ImageView>
18    </children>
19 </AnchorPane>
```

This is the GUI which will be displayed if the button “Enlarge” is clicked within the nutshell example “Multiple Windows”.

```
1
2 <!-- barChart.fxml -->
3
4 <?xml version="1.0" encoding="UTF-8"?>
5
6 <?import javafx.scene.chart.CategoryAxis?>
7 <?import javafx.scene.chart.NumberAxis?>
8 <?import javafx.scene.chart.StackedBarChart?>
9 <?import javafx.scene.control.Slider?>
10 <?import javafx.scene.layout.AnchorPane?>
11 <?import javafx.scene.layout.VBox?>
12 <?language rexx?>
13
14 <VBox xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/
     fxml/1">
15    <children>
16       <AnchorPane maxHeight="-1.0" maxWidth="-1.0" prefHeight="-1.0"
           prefWidth="-1.0" VBox.vgrow="ALWAYS">
17          <children>
18             <StackedBarChart fx:id="barChart" layoutX="14.0" layoutY="
                   72.0" prefHeight="434.0" prefWidth="800.0">
19                <xAxis>
20                   <CategoryAxis side="BOTTOM" />
21                </xAxis>
22                <yAxis>
23                   <NumberAxis side="LEFT" />
24                </yAxis>
25             </StackedBarChart>
```

```
26            <Slider id="updateButtonSmall" fx:id="slider" blockIncrement="1.0
                  " layoutX="32.0" layoutY="35.0" majorTickUnit="0.05"
                  prefWidth="300.0" showTickLabels="true" showTickMarks="true"
                  snapToTicks="true" />
27            </children>
28        </AnchorPane>
29    </children>
30    <fx:script source="barChartController.rex" />
31 </VBox>
```

That is the interface of the bar chart, which will be created.