Employing Portable JavaFX GUIs with Scripting Languages

Flatscher, Rony G.; Müller, Günter

Published in:
Central European Conference on Information and Intelligent Systems

Published: 01/01/2021

Document Version
Publisher's PDF, also known as Version of record

Employing Portable JavaFX GUIs with Scripting Languages

Rony G. Flatscher
WU (Wirtschaftsuniversität Wien)
Institut für Wirtschaftsinformatik und Gesellschaft
Welthandelsplatz 1, 1020 Wien, Austria
rony.flatscher@wu.ac.at

Günter Müller
Albert-Ludwigs-Universität Freiburg
Institut für Informatik und Gesellschaft (IIG)
Emmy-Noetherstraße 2, 79011 Freiburg, Germany
mueller@iig.uni-freiburg.de

Abstract. *When creating standalone applications with scripting languages it may become a challenge to devise powerful graphical user interfaces (GUI) for them that can be deployed and run on all the major operating system platforms, notably Windows, MacOS and Linux. This article introduces a platform independent solution for scripting languages that makes it feasible to quickly implement GUIs, ranging from simple to the most complex needs, by exploiting JavaFX (OpenJFX) and taking advantage of the Java (OpenJDK) scripting framework (javax.script). This way it becomes possible to create standalone, portable GUI applications with scripting languages that support the Java scripting framework.*

Keywords. End-user programming, Java, Java Scripting Languages, Java Scripting Framework, portable, GUI, JavaFX, SceneBuilder, open-source, ooRexx, Groovy, JRuby, Nashorn (JavaScript)

1 Introduction

ooRexx (Cowlishaw 1990, Flatscher 2013, ooRexx 2021) is an easy to learn, message based, dynamically typed and caseless scripting language that has been successfully employed for teaching Business administration students programming from scratch in a single semester with a teaching load of only four hours per week (Flatscher & Müller 2021).

Like many (scripting) programming languages ooRexx[1] does not come with the necessary function or class libraries that would allow for creating portable graphical user interfaces (GUI). The programming language Java (OpenJDK 2021) on the other hand has been devised with portability and includes GUI classes that are therefore available on all platforms where Java is installed. As a matter of fact, standard Java comes with the portable "abstract window toolkit (awt)" and the "swing" GUI classes. Java 8 includes in addition an extremely powerful GUI library named "JavaFX" (OpenFX 2021, JavaFX 2021a, JavaFX 2021b) that constitutes an independent, powerful portable GUI environment, which got separated as four proper JavaFX modules starting with the introduction of Java 11.

As it was important for Business administration students to learn and experiment with GUI applications it was seen to be beneficial, if they were able to exploit JavaFX with their freshly learned ooRexx skills only. So the challenge then was to combine ooRexx to interact with Java and its various GUI classes, to create portable GUIs. This is possible with the external function and class library BSF4ooRexx ("Bean Scripting Framework for ooRexx") (Flatscher 2010, BSF4ooRexx 2021) which actually is a bidirectional Java bridge that allows one to fully exploit Java from ooRexx. The most powerful Java GUI library to create (the most complex) portable GUI applications is "JavaFX". As "JavaFX" supports the Java scripting framework (Java package *javax.script*, Oracle 2021a) this endeavor can be facilitated considerably, if there is an appropriate Java script engine available for the desired programming language, which is the case with the BSF4ooRexx bridge for the ooRexx language.

Interestingly, there are almost no resources or tutorials that would explain and demonstrate how any programming language that can be used from Java as a scripting language could be employed for stand-alone portable JavaFX GUIs.

This article will therefore introduce JavaFX from a bird eyes view, followed by sketching the most important features of the Java scripting framework, followed by one of its most important usability features, the defining of GUIs in the form of XML marked up text files that may include references to non-Java programming languages for running programs explicitly or implicitly as JavaFX event handlers and controllers.

[1] This article is based on the beta version of ooRexx 5.0 as of summer 2021 which has achieved release quality.

A simple, bare-bone JavaFX GUI will be devised and explained that combines the different introduced concepts to create an ooRexx nutshell program that creates a portable JavaFX GUI that runs unchanged on Windows, Linux and MacOS without a need to code anything in Java and which is intended to serve as a role model for other "Java scripting languages", i.e. scripting languages that can be addressed with an appropriate implementation of the Java *javax.script.ScriptEngine* interface.

The concepts being introduced and demonstrated with ooRexx can be applied to any programming language with a Java *ScriptEngine* implementation. The JavaFX GUI example introduced in this article and explained and demonstrated with the ooRexx scripting language will also be demonstrated with the equivalent implementations in Groovy (Groovy 2021), JRuby (JRuby 2021) and the Nashorn JavaScript (Nashorn 2021) scripting languages for event handlers and controllers.

2 Related Work

With the introduction of JavaFX 1.0 in 2008 (JavaFX 2021a, 2021b) a new scripting language, JavaFX Script (JavaFX Script 2021), got introduced as well, which was supposed to be used for creating controller code. "JavaFX Script" got dropped from JavaFX 2.0, but support for Java scripting languages using the Java scripting framework (Java package *javax.script*, a.k.a. JSR-223) (JSR-223 2021) has been kept. Therefore the ability to deploy non-Java programming languages has been kept and can still be exploited to this day. Since then, new JavaFX tutorials, books and articles have concentrated on using Java only for application development with JavaFX, e.g. (Eden-Rump 2021, Epple 2015, Fedortsova 2021, SceneBuilder 2021b).

A few resources pointed at the possibility of employing Java scripting languages for creating event handlers and controller code for JavaFX GUIs like (Jenkov 2021, Pomarolli 2021), however exclusively using JavaScript as the scripting language. None of such resources have come up with standalone applications written in a Java scripting language only that takes advantage of JavaFX, but remains self-contained, i.e. is not dependent on using a Java application as its scripting and JavaFX GUI host.

To demonstrate the functionality and basic architecture of JavaFX in small nutshell programs it has become a custom pattern to use a simple window with a push button and a label field that gets updated, each time the button gets pressed (e.g. Eden-Rump 2021, Ruzicka 2019). Such small "nutshell" examples among other things allow to explain and demonstrate how event handlers and controllers can be written that update the JavaFX GUI.

Detailed information about using the ooRexx scripting language with portable JavaFX GUIs can be found in (Flatscher 2017a, 2017b, 2018) which explain in detail how the Java script engine for ooRexx implementation got conceived (Flatscher 2017a) and how to apply it for creating JavaFX GUIs in detail (Flatscher 2017b) using the standard "push button updates label" pattern. The JavaFX samples introduce and demonstrate many more features of FXML GUIs, like rendering the GUI with CSS (cascading stylesheets), resource bundles, automatically updating GUI elements with values from the *javax.script.ScriptContext* Bindings. (Flatscher 2018) introduces the two distinct Java GUI event dispatch threads for Java "awt/swing" and the "JavaFX Application thread" and introduces an easy way to instrumentate them reliably using an ooRexx message based solution for updating the GUI from non-GUI event dispatcher threads.

3 Making Portable GUIS with JavaFX

The JavaFX GUI classes use the top level package name "*javafx*" instead of "*java*" or "*javax*" which makes it easy to determine whether the Java classes in programs originate from it.

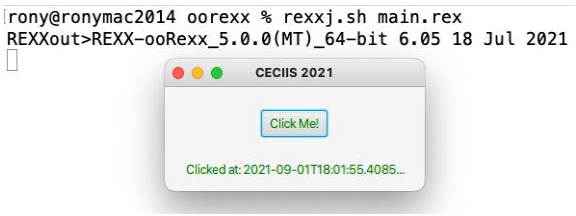

Figure 1. A GUI created with JavaFX FXML

3.1 A JavaFX Application in a Nutshell

Life-cycle of a JavaFX application in a nutshell:

- To start up the JavaFX application one needs to invoke one of its *launch(...)* methods which are responsible for creating the JavaFX event dispatching thread called "JavaFX Application Thread" to manage the JavaFX GUI and invoking the method *start(javafx.stage.Stage)*.
- A GUI in JavaFX is defined as a tree of JavaFX nodes representing a *javafx.scene.Scene* (the GUI) that gets displayed on a *Stage* (a window). This tree corresponds to a DOM tree. The JavaFX nodes of a GUI (*Scene*) can be optionally defined in an XML marked up text file of type FXML that gets loaded using one of the static load(…) methods of the Java class *javafx.fxml.FXMLLoader*. FXML files can be created and maintained with a GUI tool, named "SceneBuilder" for which a Java tutorial (SceneBuilder, 2021) exists that uses a simple dialog with a push button and a label.

- A JavaFX node can be optionally styled using cascading style sheet (CSS) rules where the CSS properties get prefixed with the string "*-fx-*".
- The JavaFX application will be closed by invoking the static *exit()* method of the *javafx.application.Platform* class or if its static field *implicitExit* is set to true and the last JavaFX window (*Stage*) gets closed in the application.

3.2 Java Scripting Framework

The Java scripting framework (package *javax.script*) a.k.a. JSR-223 defines Java interface classes that need to be implemented for supporting a scripting programming language for Java. This implementation need is eased considerably as the *javax.script* package already offers implementations of most of these interface classes, notably the Java classes named *AbstractScriptEngine*, *SimpleBindings*, and *SimpleScriptContext*.

The interface *ScriptContext* defines methods that maintain *Bindings* which are directories that map names to values and allow interacting with them. The *SimpleScriptContext* defines two *SimpleBindings* (implementing in addition the interface *java.util.Map<String,Object>*), one is named the "global scope" with the *int* value *200* and the other is named "engine scope" with the *int* value *100*. The *ScriptContext* interface defines methods for fetching the different managed *Bindings* and adding, querying or deleting entries in them.

JavaFX will use the *SimpleScriptContext* to store all JavaFX nodes that possess an *fx:id* attribute in the global scope *SimpleBindings* (*int* value *200*) and the *event* object in callback situations to the engine scope *SimpleBindings* (*int* value *100*) when invoking scripts via the Java scripting framework.

There are many programming languages that qualify as "Java scripting languages", i.e. that have an implementation of *ScriptEngine* for Java. The scripting languages may be implemented in Java, respectively Java byte code as is the case for the family of "JVM Languages" (JVM Languages 2021) among them Groovy, JRuby and Nashorn. It is also possible to use JNI (Java native interface) to bridge any programming language with Java and implement a *ScriptEngine* for it which is the case for ooRexx which is implemented in C++ and the Java bridge BSF4ooRexx which implements a JSR-223 compliant *RexxScriptEngine*.

3.3 FXML (FX Markup Language)

JavaFX nodes can be defined in a file with text marked up using FXML. The *javafx.fxml.FXMLLoader* class gets used to process such FXML files and instantiate and set up all of the defined JavaFX elements by creating corresponding JavaFX objects that represent nodes at runtime. Fig. 2 depicts the FXML definitions of the GUI in Fig. 1 above.

The XML process instruction (PI) target import denotes three fully qualified JavaFX classes that get used in the FXML file and which will be instantiate at runtime by *FXMLLoader*: the JavaFX container *AnchorPane* maintains a *Button* and a *Label* defined with layout information including the color *GREEN* to use for text. Note that the *fx:id* attribute uniquely identifies the button ("*idButton*") and the label ("*idLabel*"). To demonstrate the ability to have script code in external files executed and thereafter use functions defined in them from event attribute code, a controller file ("*hello_controller.rex*") gets defined. The button's "*onAction*" attribute defines code that invokes the controller's public function "*buttonClicked*" to get the text to be displayed in the label named "*idLabel*".

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?language rexx?>
<AnchorPane id="AnchorPane" prefHeight="104.0" prefWidth="270.0"
            xmlns:fx="http://javafx.com/fxml/1">
  <children>
    <!-- JavaFX runs the ooRexx code in the 'onAction' attribute -->
    <Button fx:id="idButton" layoutX="100.0" layoutY="23.0"
            onAction="/* @get(idLabel) */; idLabel~text=buttonClicked()"
            text="Click Me!" textFill="GREEN" />
    <Label fx:id="idLabel" alignment="CENTER" contentDisplay="CENTER"
           layoutX="21.0" layoutY="74.0" minHeight="16" minWidth="49"
           prefHeight="16.0" prefWidth="229.0" textFill="GREEN" />
  </children>
  <!-- call Rexx program, makes the routine "buttonClicked" visible -->
  <fx:script source="hello_controller.rex" />
</AnchorPane>
```

Figure 2. FXML GUI definition file "*hello.fxml*"

```
#!/usr/bin/env rexx
rxApp=.RexxApplication~new     -- Rexx class implements "start" method
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil) -- launch JavaFX Application
call sysSleep 0.1              -- sleep a bit

::requires "BSF.CLS"           -- get the Java bridge, camouflage Java as ooRexx

   /* Rexx class: implements abstract method "start" of JavaFX "Application" */
::class RexxApplication        -- defines a Rexx class

::method start                 -- implements abstract method "start"
  use arg primaryStage         -- fetch the primary stage (window)
  primaryStage~title="CECIIS 2021"  -- set stage (window) title
      -- create a URL object for the file "hello.fxml"
  fxmlUrl=.bsf~new("java.net.URL", "file:hello.fxml")
        -- load the FXMLLoader class, load the FXML file, returns DOM's root
  rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)
  scene=.bsf~new("javafx.scene.Scene", rootNode)     -- create the scene
  primaryStage~setScene(scene)       -- set the stage to our scene
  primaryStage~show                  -- show the stage
```

Figure 3. ooRexx program "*main.rex*" that launches the JavaFX application

```
-- Controller routine buttonClicked() in ooRexx
parse version v; say v

::routine buttonClicked public
  return "Clicked at:" .dateTime~new
```

Figure 4. ooRexx controller ("*hello_controller.rex*")

The PI with the target language defines the programming language rexx to be used for executing code which is defined in the button's *onAction* attribute. The code that is stored in the file denoted by the source attribute in the *fx:script* element after the *children* end tag will get executed by a *ScriptEngine* that serves the file type ".*rex*" which is the *rexx* engine as well. The Java script engine corresponding to the defined programming language gets resolved using the Java scripting framework (method *getEngineByName(String shortName)* and *getEngineByExtension(String extension)* in the class *javax.script.ScriptEngineManager*).

FXML files can be easily created and maintained with the WYSIWYG ("what you see is what you get") editor named *SceneBuilder* (SceneBuilder 2021a) which allows for defining all aspects of a GUI declaratively. Furthermore it supports creating event handler code in Java right in the editor, a support which is not really available for scripting languages. For this reason it becomes necessary to manually edit an FXML file to inject the *language* PI to define the scripting language that must be used to execute the event handler code and all script code stored inline in the FXML file. One is able to define different FXML files to use different programming languages than Java that get used to implement the event handler and inline script (*fx:script*) code. The JavaFX *FXMLLoader* class in this case employs the Java scripting framework and therefore one can use code written in any of the many available Java scripting languages.

The static *load(...)* method of the *javafx.fxml.FXMLLoader* class that processes the FXML file *"hello.fxml"* will add all known JavaFX nodes with an *fx:id* attribute to the global scope *SimpleBindings*. Therefore the script denoted in the source attribute (value: *"hello_controller.rex"*) in Fig. 2 (*fx:script* element) will be able to access the *idButton* and the *idLabel* JavaFX objects using the *SimpleScriptContext* if this is needed. Whenever the button event fires the event handler (*onAction* attribute) will get executed after the event object got placed into the local scope *SimpleBindings* using the name "*event*".[2]

3.4 Making a Portable GUI from ooRexx

ooRexx is a dynamically-typed, message based, object-oriented programming language, which is easy to learn yet quite powerful. The message operator is the tilde (~) that separates the receiver object on the left from the message name and its arguments to the right of it. Conceptually, the receiver object looks for a method by the name of the received message, invokes it with the received arguments and returns any result that may be returned. As ooRexx programs use English like keyword instructions and English method names they can sometimes be read almost like pseudo code.

As designed and implemented, the BSF4ooRexx bidirectional Java bridge allows ooRexx programmers to interact with the Java runtime environment and all its classes. If a Java application uses the Java scripting framework, then the names "rexx", "Rexx", "oorexx" or "ooRexx" will use the ooRexx interpreter to execute

[2] Starting with JavaFX 15 the *event* object will be made directly available as the single argument additonally.

script code. For this feature a *RexxScriptEngine* (Flatscher 2017a) got implemented which among other things offers these special services:

- Rexx script annotations: these allow the programmer to denote the names of the *SimpleBinding* entries that should be fetched and made available as local Rexx variables in the script in the form of a specifically formed block comment like: */* @get(names) */*
- A *RexxScriptEngine* instance, unlike standard ooRexx, will by default accumulate all public routines and all public classes encountered in all of the Rexx programs that have been already executed, and will make them available to any Rexx scripts that get invoked at a later time, irrespective whether that program explicitly requires them.
- ooRexx will use the *SimpleScriptContext* fields *input* (a *java.io.Reader*), *output* (a *java.io.Writer*) and *errorOutput* (a *java.io.Writer*) as a replacement for *stdin*, *stdout* and *stderr*. Accessing one of the standard files *stdin*, *stdout* and *stderr* from Rexx will cause a prefix to be injected ("*REXXin>*", "*REXXout>*", "*REXXerr>*") that makes it possible to distinguish Rexx generated input or output from Java generated input or output.

The external Rexx function *BsfCreateRexxProxy(rexxObject, optionalData, javaAbstractClass)* allows to extend any abstract Java class and to redirect invocations of the abstract Java methods to the supplied *rexxObject* by sending it appropriate Rexx messages, supplying any Java arguments if necessary. The result of that Rexx function is a Java object (an instance of the dynamically created extended abstract Java class).

The Rexx program *main.rex* in Fig. 3 defines the Rexx class *RexxApplication* that implements the abstract Java method *start* as an ooRexx method: it fetches the *primaryStage* (a window), sets its title, uses the *javafx.fxml.FXMLLoader* class to load the FXML "*hello.fxml*" file in Fig. 2 above. When processing the *Button* child element the *onAction* attribute will cause the appropriate JavaFX *onAction* property to be set up such that the Java scripting framework will be employed to execute the Rexx code defined in the *onAction* attribute each time the button gets pressed: the code will cause the routine *buttonClicked* in *"hello_controller.rex"* to be called which returns the string "Clicked at:" concatenated with the current date and time.

FXMLLoader will use the Java scripting framework and employ the *RexxScriptEngine* when processing the *fx:script* element with the *source* attribute set to *"hello_controller.rex"* in order to execute that Rexx program via the Java scripting framework. The initialisation part of *"hello_controller.rex"* will output the current version of ooRexx on the terminal and upon return its public routine *buttonClicked* will be visible to any Rexx code in event attributes (in this case "*onAction*") that gets executed thereafter in the context of "*hello.fxml*".

The JavaFX node that the load method returns is then used to create a *javafx.scene.Scene* (the GUI) which then gets placed on the *primaryStage* (a window) which in turn will be made visible and shown to the user. The method *show* of the *primaryStage* will block the calling Rexx program until the JavaFX GUI gets closed by the user. Upon return the *start* method of the *RexxApplication* class concludes and the application ends.

At the top of the Rexx program in Fig. 3 the *RexxApplication* Rexx class gets instantiated and the Rexx object is used for extending the Java class *javafx.application.Application* thereby supplying its *start* method. Sending the *launch* message to the returned Java object will cause the JavaFX event dispatcher thread ("JavaFX Application Thread") to be created and its *start* method to be invoked on that newly created GUI thread which in turn will cause the Rexx object to receive the Rexx message *start* with the *primaryStage* as its argument which then executes the statements as described above.

While the JavaFX GUI is up and running, the user can interact with it by pressing the push button or

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?language groovy?>
<AnchorPane id="AnchorPane" prefHeight="104.0" prefWidth="270.0"
            xmlns:fx="http://javafx.com/fxml/1">
  <children>
    <Button fx:id="idButton" layoutX="100.0" layoutY="23.0"
            onAction="idLabel.setText(buttonClicked())"
            text="Click Me!" textFill="MAROON" />
    <Label fx:id="idLabel" alignment="CENTER" contentDisplay="CENTER"
           layoutX="21.0" layoutY="74.0" minHeight="16" minWidth="49"
           prefHeight="16.0" prefWidth="229.0" textFill="MAROON" />
  </children>
  <fx:script source="hello_controller.groovy" />
</AnchorPane>
```

Figure 5. FXML Using Groovy (*"hello.fxml"*)

```
// Controller routine buttonClicked() in Groovy
println  "Groovy version: " + GroovySystem.version

def buttonClicked () {
    def now = new java.util.Date()
    def df  = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
    return "Clicked at: " + df.format(now);
}
```

Figure 6. Groovy controller (*"hello_controller.groovy"*)

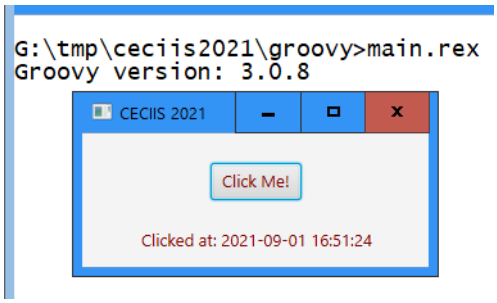

Figure 7. JavaFX GUI controlled by Groovy

closing the GUI by pressing the close button on the frame.

The JavaFX GUI will react upon button press events which will cause the *onAction* Rexx code in the *idButton* element in the FXML file in Fig. 2 to be executed using the Java scripting framework. The Rexx script annotation (*/* @get(idLabel) */*) will fetch the reference to the "*idLabel*" object and make it locally available under the same variable name to Rexx. The Rexx routine *buttonClicked* in Fig. 4 gets called and returns a string "Clicked at:" concatenated with the date and time of its invocation which then is assigned to the label's *text* field.

For Java scripting languages that do not offer a comparable functionality it would be possible to create a Java class that extends *javafx.application. Application* and takes command line arguments that denote the scripting language and the name of a file that contains the code to be run as the *start* method. With this information the extended Application of such a Java class could launch the application and in its own implementation of the *start* method and run the script code from the denoted file using the Java scripting framework, supplying it the received *javafx.stage.Stage* object. Upon return from executing the script the implementation would then invoke the show method of the Stage object to allow the user to interact with the resulting JavaFX GUI.

Alternatively, one could forgo Java altogether by creating an ooRexx script that takes its command line arguments and implements the startup functionality as described above.

3.5 Groovy, JRuby and Nashorn for Controlling the GUI

For demonstration purposes the ooRexx program "*main.rex*" gets used to launch the JavaFX application where the single GUI gets controlled by other scripting languages than ooRexx.

In essence the FXML file "*hello.fxml*" gets reused but needs to be adjusted for each language by changing the *language* PI, the *onAction* code and the name of the controller file to be used accordingly. Note that also the color of the button and label text gets changed in the FXML files (attribute "*textFill*") such that one can see that such layout related changes can be done declaratively, but also to get a visual feedback which programming language is used to control the GUI. As the same name, "*hello.fxml*", gets used for all three different scripting languages, it is necessary to keep the files "*main.rex*", "*hello.fxml*" and the respective controller files in separate directories.

Groovy: Fig. 5 shows "*hello.fxml*" in which the language PI defines "*groovy*" to be the programming language used for all codings in event attributes like "*onAction*", such that JavaFX will use the Groovy script engine to execute the code which got changed to Groovy. The controller code for this GUI is stored in "*hello_controller.groovy*" which the *fx:script* element denotes as its *source* and which gets displayed in Fig. 6. JavaFX will execute that file, which will display the Groovy version that gets used in the terminal. Upon return the *buttonClicked* routine becomes accessible from the context of the "*hello.fxml*" defined GUI, such that the Groovy code in the "*onAction*" attribute can invoke it and use the return value to set the text (a string) of the "*idLabel*" object which is located underneath the button as can be seen in Fig. 7. Note: the file extension "*.groovy*" will be used to determine with the help of the *javax.script.ScriptEngineManager* the script engine that gets used to execute it. For this sample at least *groovy-3.0.8.jar* and *groovy-jsr223-3.0.8.jar* (Groovy 2021) need to be supplied on the Java *CLASSPATH*. Fig. 7 was taken from a Windows 10 installation.

JRuby: Fig. 8 shows "*hello.fxml*" in which the language PI defines "*jruby*" to be the programming language used for all codings in event attributes like "*onAction*", such that JavaFX will use the JRuby script engine to execute the code which got changed to

JRuby. The controller code for this GUI is stored in "*hello_controller.rb*" which the *fx:script* element denotes in its *source* attribute and which gets displayed in Fig. 9. JavaFX will execute that file, which will display the JRuby and Ruby version that gets used in the terminal. Upon return the *buttonClicked* routine becomes accessible from the context of the "*hello.fxml*" defined GUI, such that the JRuby code in the "*onAction*" attribute can invoke it and use the return value to set the text (a string) of the "*idLabel*" object which is located underneath the button as can be seen in Fig. 10. Note: the file extension "*.rb*" will be used to determine with the help of the *javax.script.ScriptEngineManager* the script engine that gets used to execute it. For this sample *jruby-complete-9.2.19.0.jar* (JRuby 2021) was supplied on the Java *CLASSPATH*. Fig. 10 was taken from an Ubuntu Linux 5.3.01-generic installation.

```
<?language jruby?>
...
   onAction="idLabel.setText buttonClicked()"
   text="Click Me!" textFill="BLUE" />

   <fx:script source="hello_controller.rb" />
```

Figure 8. FXML Changes for JRuby ("*hello.fxml*")

```
# Controller routine buttonClicked() in JRuby
puts "JRUBY_VERSION: "+ JRUBY_VERSION + " RUBY_VERSION: " + RUBY_VERSION

def buttonClicked ()
    return "Clicked at: " + Time.new.strftime("%Y-%m-%d %H:%M:%S")
end
```

Figure 9. JRuby controller ("*hello_controller.rb*")

```
rony@rony-linux:~/Dropbox/xfer/temp/ceciis2021/jruby$ rexxj.sh main.rex
Gtk-Message: 18:05:43.531: Failed to load module "topmenu-gtk-module"
JRUBY_VERSION: 9.2.19.0 RUBY_VERSION: 2.5.8
```

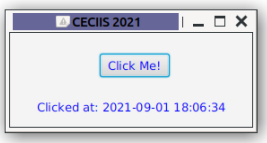

Figure 10. JavaFX GUI controlled by JRuby

```
<?language nashorn?>
...
   onAction="idLabel.setText( buttonClicked() );"
   text="Click Me!" textFill="MAGENTA" />
...
   <fx:script source="hello_controller.js" />
```

Figure 11. FXML Changes for Nashorn ("*hello.fxml*")

```
// Controller routine buttonClicked() in Nashorn
var factory = (new
(Java.type("javax.script.ScriptEngineManager"))).
        getEngineByName("nashorn").getFactory() ;
print("engine name: "       +factory.getEngineName()+
      ", language version: "+factory.getLanguageVersion());

function buttonClicked() {
    return "Clicked at: " + new Date() ;
}
```

Figure 12. Nashorn controller ("*hello_controller.js*")

```
G:\tmp\ceciis2021\nashorn>main.rex
engine name: Oracle Nashorn, language version: ECMA - 262 Edition 5.1
```

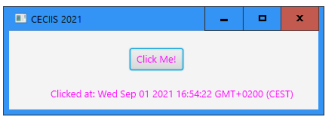

Figure 13. JavaFX GUI controlled by Nashorn

Nashorn (JavaScript): Fig. 11 shows "*hello.fxml*" in which the language PI defines "*nashorn*" to be the programming language used for all codings in event attributes like "*onAction*", such that JavaFX will use the Nashorn script engine to execute the code which got changed to Nashorn JavaScript. The controller code for this GUI is stored in "*hello_controller.js*" which the *fx:script* element denotes in its *source* attribute and which gets displayed in Fig. 12. JavaFX will execute that file, which will display the Nashorn version that gets used in the terminal. Upon return the *buttonClicked* routine becomes accessible from the context of the "*hello.fxml*" defined GUI, such that the JavaScript code in the "*onAction*" attribute can invoke it and use the return value to set the text (a string) of the "*idLabel*" object which is located underneath the button as can be seen in Fig. 13. Note: the file extension "*.js*" will be used to determine with the help of the *javax.script.ScriptEngineManager* the script engine that gets used to execute it. For this sample OpenJDK 11 was used, which includes Nashorn which claims to be able to process files with the extension "js". However, starting with OpenJDK 15 Nashorn is not part of the Java distribution anymore. Instead one needs to download the Nashorn module (Nashorn 2021) and the module needs to be either added to the Java startup command with the "*--modulepath*" option, if feasible, or otherwise the Nashorn module needs to be used for creating an OpenJDK distribution from a collection of modules using the OpenJDK tool *jlink*. Fig. 10 was taken from a Windows 10 installation. Also note that the version information was retrieved using the *javax.script* infrastructure as interestingly there exists no standardized JavaScript API to retrieve it otherwise.

3.6 The JavaFX "Address Book" Example

To promote using JavaFX and FXML for creating Java applications a somewhat popular tutorial, „Creating an Address Book with FXML", was originally created by Sun (Fedortsova 2021, Oracle 2021) and serves as the basic application specification for another "Address Book" tutorial that in addition also demonstrates applying cascading stylesheets (CSS) (Jacob 2021) and which needs at least JavaFX 8. All of these tutorials use textual FXML definitions for the individual GUIs and Java to implement all logic.

The full installation of BSF4ooRexx (BSF4ooRexx 2021) comes with an ooRexx only implementation of the address book tutorial in (Jacob 2021) (cf.

bsf4oorexx/samples/JavaFX/fxml_99), includes FXML files for defining the GUIs and uses the *DarkTheme.css* cascading stylesheet for formatting the GUIs accordingly. Unlike the Java tutorials, the ooRexx implementation includes in addition the ability to print out all addresses using another CSS file for formatting, and a little "about" popup window. From the GUI appearances it cannot be inferred, which programming language, Java or ooRexx, was used to implement the "Address Book" application (Flatscher 2017b).

In this way it serves as a proof-of-concept that it is indeed possible to create quite complex JavaFX applications with scripting languages. ooRexx just serves as an initial example, the address book application sample could be implemented in Groovy, JRuby or Nashorn, or any other Java scripting language.

4 Roundup and Conclusion

This article briefly introduced JavaFX, FXML and the Java scripting framework from a bird-eyes view to allow the reader to understand and relate to the possibilities of employing this powerful and platform independent GUI environment for scripting languages.

The scripting language used to introduce and demonstrate the JavaFX infrastructure is the open-source programming language ooRexx which is easy to learn, to read and is available for all major platforms. It gets successfully used to teach Business administration students programming from scratch in a four hour course in a single semester. The open-source Java bridge BSF4ooRexx has been developed for almost 20 years and has achieved a wealth of functionality that gets exploited in this endeavor, taking advantage of the possibility to run ooRexx code using its *RexxScriptEngine* via the Java scripting framework, making ooRexx also into a "Java scripting language".

As the Java scripting framework gets applied it is also possible to use any other "Java scripting language", i.e. any programming language or any JVM language that implements the *javax.script.ScriptEngine* interface for controlling FXML based GUIs. This article therefore used the JVM languages Groovy, JRuby and Nashorn to demonstrate how easy it is to control an FXML defined GUI in these languages as well, even allowing controllers to be used that get implemented in separate files. As can be seen the JavaFX concepts introduced with ooRexx can be directly applied and exploited in any other Java scripting language.

If it is desired to create standalone JavaFX applications with the means of a Java scripting language (to forgo any need for programming in Java) then such a scripting language needs to be able to extend the JavaFX class *javafx.application.Application* and implement its abstract method *start*. BSF4ooRexx allows for implementing abstract Java methods in ooRexx and even offers the ability to extend abstract Java classes at runtime as demonstrated in Fig. 3 above. Because of this it is not necessary for ooRexx programmers to learn the Java programming language at all in order to become able to launch ooRexx programs that take advantage of the portable JavaFX GUI environment for their GUI needs. It has been demonstrated that it is even possible to combine different scripting languages that control JavaFX FXML defined GUIs by using ooRexx for launching Groovy, JRuby and Nashorn controlled GUIs.

It is possible to define a set of FXML GUIs, each controlled by programs written in different scripting languages and hosted by any program that is able to subclass the *javafx.application.Application* class and implement its abstract *start* method. The Java scripting framework in combination with FXML would even allow a single FXML GUI to host and run script programs written in different scripting languages from different files that would be able to exchange data with each other using the FXML related global scope *javax.script.ScriptContext*. FXML GUIs can be constructed by incorporating FXML GUIs, each controlled by a different scripting language. It would be interesting to explore the possibilities that exist with this use case in mind.

JavaFX can also be used for creating mobile applications specifically for Android and iOS (JavaFX 2021a). Further research and experiments would be interesting that aim at using scripting languages with JavaFX and that run on these specific mobile platforms.

References

BSF4ooRexx (2021): ooRexx-Java Bridge, Download Site for the Latest Beta Versions. https://sourceforge.net/projects/bsf4oorexx/files/beta/20200928/

Cowlishaw, M. F. (1990): *The REXX Language* (Second Edi.). New Jersey, Englewood Cliffs.

Epple, A. (2015): *JavaFX 8*. Heidelberg, dpunkt.verlag.

Fedortsova, I. (2021): Creating an Address Book with XML (JavaFX 2), https://docs.oracle.com/javafx/2/fxml_get_started/fxml_tutorial_intermediate.htm, last accessed

Eden-Rump, E. (2021): JavaFX Button Events and How to Use Them. https://edencoding.com/javafx-button-events-and-how-to-use-them/

Flatscher, R.G. (2010). The 2010 Edition of BSF4ooRexx. In C. Davis & R. V. Jansen (Eds.), *Proceedings of the 2010 International Rexx Symposium* (pp. 1-35). Rexx Language Association, Amsterdam.

Flatscher, R.G. (2013): *Introduction to Rexx and ooRexx*. Vienna: Facultas.

Flatscher, R.G. (2017a): RexxScript – Rexx Scripts Hosted and Evaluated by Java (Package javax.script). In C. Davis & R. V. Jansen (Eds.), *Proceedings of the 2017 International Rexx Symposium* (pp. 1-19). Rexx Language Association, Amsterdam. https://www.rexxla.info/events/2017/presentations/201704-RexxScript-Article.pdf

Flatscher, R.G. (2017b): JavaFX for ooRexx. In C. Davis & R. V. Jansen (Eds.), *Proceedings of the 2017 International Rexx Symposium* (pp. 1-43). Rexx Language Association, Amsterdam. https://www.rexxla.info/events/2017/presentations/201711-ooRexx-JavaFX-Article.pdf

Flatscher, R.G. (2018): Anatomy of a GUI (Graphical User Interface). In C. Davis & R. V. Jansen (Eds.), *Proceedings of the 2018 International Rexx Symposium* (pp. 1-40). Rexx Language Association, Aruba.

Flatscher, R.G., Müller G. (2021): „Business Programming" – Critical Factors from Zero to Portable GUI Programming in Four Hours. In M. Kolaković & T. Horvatinović. & I. Turčić (Eds.), *Proceedings of the 6th Business & Entrepreneurial Economics 2021 (BEE 2021)* (pp. 76-82). University of Zagreb, Faculty of Economics and Business, Croatia.

Groovy (2021): Groovy Scripting Language, Download Site. https://groovy.apache.org/download.html

Jacob, M. (2021): JavaFX Tutorial. https://code.makery.ch/library/javafx-tutorial/

JavaFX (2021a): JavaFX Resources. https://openjfx.io/

JavaFX (2021b): Wikipedia JavaFX. https://en.wikipedia.org/wiki/JavaFX

JavaFX Script (2021): Wikipedia JavaFX Script. https://en.wikipedia.org/wiki/JavaFX_Script

Jenkov, J. (2021): JavaFX FXML, http://tutorials.jenkov.com/javafx/fxml.html

JRuby (2021): JRuby Scripting Language, Download Site. https://www.jruby.org/download

JSR-223 (2021): JSR-000223 Scripting for the JavaTM Platform. https://jcp.org/aboutJava/communityprocess/final/jsr223/index.html

JVM Languages (2021): Wikipedia List of JVM Languages. https://en.wikipedia.org/wiki/List_of_JVM_languages

Nashorn (2021): OpenJDK JavaScript Language. https://github.com/openjdk/nashorn

ooRexx (2021): ooRexx Scripting Language, Download Site for the Latest 5.0 Beta Installation Packages. https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0beta/

OpenFX (2021): Homepage. https://wiki.openjdk.java.net/display/OpenJFX/Main

OpenJDK (2021): Homepage. http://openjdk.java.net

Oracle (2021a): Creating an Address Book with XML (JavaFX 8), https://docs.oracle.com/javase/8/javafx/fxml-tutorial/fxml_tutorial_intermediate.htm

Oracle (2021b): The Java Scripting API. https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/prog_guide/api.html#CDEGJDJF

Pomarolli, A. (2021): JavaFX FXML Tutorial, https://examples.javacodegeeks.com/desktop-java/javafx/fxml/javafx-fxml-tutorial/

Ruzicka, V. (2019): JavaFX Tutorial: FXML and SceneBuilder. https://www.vojtechruzicka.com/javafx-fxml-scene-builder/

SceneBuilder (2021a). SceneBuilder Download Site. https://gluonhq.com/products/scene-builder/

SceneBuilder (2021b): Basic JavaFX project with Scene Builder. https://github.com/gluonhq/scenebuilder/wiki/Basic-JavaFX-project-with-Scene-Builder