Creating Powerful and Portable GUIs with JavaFX

The 2024 International Rexx Symposium
Brisbane, Queensland, Australia
March 3rd – March 6th 2024
Agenda

• Brief historic overview
  – Java GUI packages for creating portable GUIs

• Overview of JavaFX
  – Concepts
  – ooRexx nutshell examples

• Roundup
Brief Historic Overview, 1

- Java package "java.awt"
  - "awt": "abstract window toolkit"
    - Java GUI classes for creating a GUI consisting of GUI components
      - Abstracts from concrete operating systems
    - Uses JNI (Java Native Interface) to interact with the platform's GUI
      - "heavy" interaction with peer native GUI controls
    - Insulates the Java programmer from the platform
    - GUI control and event management carried out in a separate "awt"/"GUI"-thread
      - Released with Java 1.0 (1996)
Brief Historic Overview, 2

- Java package "javax.swing"
  - "javax": Java extension
  - Java GUI classes for the most important GUI controls
    - "light-weight"
      - Uses Java2d to draw the controls
        - Text fields can be formatted with HTML style-attributes of that time
      - Contained in awt container
    - Swing class names may start with "J", if an awt class of the same name exists already
      - e.g. javax.swing.JButton vs. java.awt.Button
  - Adds PLAF
    - Pluggable Look and Feel
  - Released with Java 1.2 (1998)
Brief Historic Overview, 3

- **Java package "javafx."**
  - 2008 a standalone Java package
  - Also included a proper script engine named "JavaFX Script"
    - Reason why the Java scripting framework gets fully supported
    - Removed with JavaFX 2.0 (2011)
  - Replaces `java.awt` and `javax.swing`
    - Introduces "Properties"
    - Totally new class hierarchy
    - Many new multiplatform classes for
      - e.g. charts, sound, video
  - Released with Java 1.8/8 (2014) as part of the JRE/JDK as "JavaFX8"
  - Already included in Java 1.7/7 updates as part of the JRE/JDK (7u15)
Concepts of JavaFX, 1

- "Property"
  - Contains a value, has setter and getter methods
  - Can be bound to other properties
    - Auto-update values!
  - GUI classes use properties to display and interact with
import the Java class, allow it to be used like an ooRexx class thereafter

```rexx
sipClz=bsf.import("javafx.beans.property.SimpleIntegerProperty")
num1 = sipClz~new(1)
num2 = sipClz~new(2)
sum=num1~add(num2)
say "'num1=1' (an IntegerProperty) and 'num2=2' (an IntegerProperty), 'sum' (a NumberBinding):" sum~getValue
num1~set(2)
say "after setting 'num1' to '2', sum:" sum~getValue
num2~set(3)
say "after setting 'num2' to '3', sum:" sum~getValue
def requires "BSF.CLS" -- get Java support
```

Output:

'num1=1' (an IntegerProperty) and 'num2=2' (an IntegerProperty), 'sum' (a NumberBinding): 3
after setting 'num1' to '2', sum: 4
after setting 'num2' to '3', sum: 5
FX Markup Language (FXML), 1

- Allows to define the GUI as an XML file
  - Tool SceneBuilder to create GUIs interactively!

- Allows to set up an available javax.script engine
  - Run script code, e.g. for events!

- A Java loader class will read the FXML and create the GUI
  - GUI controls with 'fx:id' attribute directly addressable!
 Invoking script code occurs with the help of `javax.script`

Creates a separate **Engine** for each **FXML** document

Each invocation gets its own **ScriptContext** with a **GLOBAL_SCOPE** and **ENGINE_SCOPE** Binding

**GLOBAL_SCOPE** Binding contains
- The created **JavaFX** GUI controls that have the attribute 'fx:id' set!
- A Rexx script can access all of these GUI controls
Concepts of JavaFX, 5

**Model-View-Controller (MVC)**

- Introduced with Smalltalk-76
- **Model** – the data to maintain
  - Our program
- **View** – the program to display the data
  - Our program, JavaFX or a combination of both
  - View and model can be bound with Properties!
- **Controller** – to control (co-ordinate) interaction with the model and the view
  - Our program serving as the bridge between the model and the view(s)
Concepts of JavaFX, 6

Example 1

Output (fxml_01.rex):

REXXout>2023-01-24T15:17:15.177000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[]
REXXout>...     new value of label=[Clicked at: 2023-01-24T15:17:15.177000]
REXXout>
REXXout>2023-01-24T15:17:35.416000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[Clicked at: 2023-01-24T15:17:15.177000]
REXXout>...     new value of label=[Clicked at: 2023-01-24T15:17:35.416000]
REXXout>
Three Files

1. File: “FXML_01_Document.fxml”
   - The FXML file defines the GUI
     - Defines "rexx" to be used as the script language
     - Defines an AnchorPane GUI container which contains
       - Button with fx:id="idButton1" (with Rexx code) and a
       - Label with fx:id="idLabel1"
       - Text (textFill property) of both controls is GREEN

2. File: “fxml_01_controller.rex”
   - Defines a public Rexx routine "buttonClicked"

3. File: “fxml_01.rex”
   - Runs the program using the javafx package
Concepts of JavaFX, 8

Using "SceneBuilder" for the Dialog

- [https://gluonhq.com/products/scene-builder/] (2022-12-11)
<?xml version="1.0" encoding="UTF-8"?>
<AnchorPane id="idAnchorPane" prefHeight="200" prefWidth="400"
xmlns:fx="http://javafx.com/fxml/1">
    <!-- call Rexx program, its public routine "buttonClicked" is known afterwards -->
    <fx:script source="fxml_01_controller.rex" />
    <children>
        <!-- the Rexx code in the 'onAction' attribute will be invoked by JavaFX; note: last argument is the slotDir argument from BSF4ooRexx -->
        <Button fx:id="idButton1" layoutX="170.0" layoutY="89.0"
             onAction="use arg event, slotDir; call buttonClicked slotDir;"
             text="Click Me!" textFill="GREEN" />
        <Label fx:id="idLabel1" alignment="CENTER" contentDisplay="CENTER"
             layoutX="76.0" layoutY="138.0"
             minHeight="16" minWidth="49"
             prefHeight="16.0" prefWidth="248.0"
             textFill="GREEN" />
    </children>
</AnchorPane>
Concept of "Rexx Script Annotation"

- A "boon" implemented into the ooRexx `javax.script RexxEngine`
  - A Rexx block comment, which may be one of
    ```
    //@get(idx1 idx2 ...)*/
    - Fetches entries named "idx1", "idx2" from the `ScriptContext's Bindings` and makes them available as Rexx variables by the same name ("idx1", "idx2")
    ```
    ```
    //@set(idx1 idx2 ...)*/
    - Sets the entries named "idx1", "idx2" in the `ScriptContext Bindings`, using the values of the Rexx variables "idx1", "idx2"
    ```
    ```
    //@showsourse*/
    - Displays the Rexx code that gets executed by the `RexxEngine`
    ```
2. File: “fxml_01_controller.rex”

- Defines the public Rexx routine "buttonClicked"
  - Usually there is one controller for each FXML file
  - Fetches the supplied slotDir argument
    - Can be used to access the ScriptContext and its Bindings
    - This example uses "Rexx script annotations"
  - Fetches and updates the Label with fx:id="label"
    - Taking advantage of "Rexx script annotations"
      /*@get(label)*/ instead of coding:
      label=slotDir~scriptContext~getAttribute("label")

- Outputs information to stdout
Responsible for updating the Label object using the (fx:)id value (case-sensitive!) "idLabel1"

```rexx
/* This routine will be called from the Rexx code defined with the "onAction" attribute in Button's definition */
::routine buttonClicked public
use arg slotDir -- using Rexx script annotation instead
now=.dateTime~new -- time of invocation
say now": arrived in routine 'buttonClicked' ...
/* RexxScript annotation fetches the Label object with the id "idLabel1" from
the ScriptContext and makes it available as the Rexx variable "IDLABEL1" */
/* @get(idLabel1) */
say ... current value of label=pp(idLabel1~getText)
idLabel1~text="Clicked at:" now -- set text property
say ... new value of label=pp(idLabel1~getText)
say
```

Responsible for the output:

```
REXXout>2023-01-24T15:17:15.177000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[]
REXXout>... new value of label=[Clicked at: 2023-01-24T15:17:15.177000]
REXXout>
REXXout>2023-01-24T15:17:35.416000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[Clicked at: 2023-01-24T15:17:35.416000]
REXXout>... new value of label=[Clicked at: 2023-01-24T15:17:35.416000]
REXXout>
```
A JavaFX application uses

- **Stages** to display **Scenes**
  - A **Stage** is usually some kind of a window
  - A **Scene** is a GUI container placed on a **Stage** for interaction
- There may be multiple **Stages** and **Scenes**

**Abstract class** `javafx.application.Application`

- Initializes **JavaFX**, creates a ("primary") **Stage** and invokes the abstract method `start(Stage primaryStage)` in its `launch` method
- A Rexx program defines a Rexx class that implements the abstract method `start`
- Uses `BsfCreateRexxProxy()` to create a proxied Application, and sends it the `launch` message (which in turn will invoke the `start` method implemented in Rexx)
1. File: “fxml_01.rex”

- Defines the Rexx class `RexxApplication`
  - Implements the abstract method `start`
  - A Rexx instance will be used in `BsfCreateRexxProxy()`
    - The resulting Java object (of type `javafx.application.Application`) gets the `launch` message sent to it, which eventually will invoke the method `start`, causing a Rexx message of that name to be sent to the embedded Rexx instance
Concepts of JavaFX, 15

1. File: “fxml_01.rex”

```rexx
rxApp=.RexxApplication-new -- create Rexx object that will control the FXML set up
jrxApp=BSFCreateRexxProxy(rxApp, "javafx.application.Application")
jrxApp-launch(jrxApp~getClass, .nil) -- launch the application, invokes "start"

::requires "BSF.CLS" -- get Java support

-- Rexx class defines "javafx.application.Application" abstract method "start"
::class RexxApplication -- implements the abstract class "javafx.application.Application"

::method start -- Rexx method "start" implements the abstract method
use arg primaryStage -- fetch the primary stage (window)
primaryStage~setTitle("Hello JavaFX from ooRexx! (Green Version)")
-- create an URL for the FMXMLDocument.fxml file (hence the protocol "file:")
fxmlUrl=.bsf~new("java.net.URL", "file:fxml_01.fxml")
-- use FXMLLoader to load the FXML and create the GUI graph from its definitions:
rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)

scene=.bsf~new("javafx.scene.Scene", rootNode) -- create a scene for our document
primaryStage~setScene(scene) -- set the stage to our scene
primaryStage~show -- show the stage (and thereby our scene)
```
Roundup Example 1 - Output of Running File: “fxml_01.rex”

Output (fxml_01.rex):

REXXout>2023-01-24T15:17:15.177000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[]
REXXout>... new value of label=[Clicked at: 2023-01-24T15:17:15.177000]
REXXout>2023-01-24T15:17:35.416000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[Clicked at: 2023-01-24T15:17:15.177000]
REXXout>... new value of label=[Clicked at: 2023-01-24T15:17:35.416000]
JavaFX without Employing FXML

- **FXML** contains all GUI declarations
  - Which **javafx** controls
  - Position of **javafx** controls
  - Attributes of **javafx** controls, e.g.
    - Color information
    - Position and size information
    - Unique and case-sensitive **fx:id** values for **javafx** controls

- **Without taking advantage of FXML**
  - The code needs to do all this setting up
  - Needs to take over event handling
JavaFX Nutshell Example (without FXML), 1 – 1
"javafx_01.rex"

```plaintext
rexxHandler=.RexxAppHandler-new
  -- instantiate the abstract JavaFX class, the abstract "start" method will be served by rexxHandler
rxApp=BSFCreateRexxProxy(rexxHandler, ,"javafx.application.Application")
  -- launch the application, invoke "start" and then stay up until the application closes
rxApp-launch(rxApp-getClass, .nil)
::requires "BSF.CLS"  -- get Java support
::class RexxAppHandler -- the Rexx handler for javafx.application.Application
::method start  -- will be called by JavaFX, allows to setup everything
  use arg primaryStage
primaryStage~setTitle("Hello JavaFX from ooRexx!")  -- we could use primaryStage~title="..." instead!

colorClz=bsf.loadClass("javafx.scene.paint.Color")  -- get access to the JavaFX colors
cdClz=bsf.loadClass("javafx.scene.control.ContentDisplay")  -- get access to ContentDisplay constants
alClz=bsf.loadClass("javafx.geometry.Pos")  -- get access to alignment constants (an Enum class)

root=.bsf~new("javafx.scene.layout.AnchorPane")  -- create the root node
root~prefHeight=200
root~prefWidth=400

  -- define the Label
lbl=.bsf~new("javafx.scene.control.Label")
lbl~textFill=colorClz~BLUE
lbl~setLayoutX(76)
lbl~setLayoutY(138)
lbl~prefHeight="16.0"
lbl~prefWidth="248.0"
lbl~contentDisplay=cdClz~CENTER  -- center ContentDisplay
lbl~alignment=alClz~valueOf("CENTER")  -- center align

... continued on next slide ...
```
... continued on next slide ...

-- define and add the Button
btn= .bsf-new("javafx.scene.control.Button")
btn-textFill=colorClz-BLUE
btn-layoutX=170  -- assign as if it was a Rexx attribute
btn-layoutY=89   -- assign as if it was a Rexx attribute
btn-text="Click Me!"  -- assign as if it was a Rexx attribute

-- create a RexxButtonHandler, wrap it up as a Java RexxProxy implementing all methods of "javafx.event.EventHandler":
bh=BSFCreateRexxProxy(RexxButtonHandler-new(lbl), "javafx.event.EventHandler")
btn-setOnAction(bh)  -- let this instance's Java RexxProxy handle the event

-- add the button to
root-getChildren--add(btn)--add(lbl)

-- put the scene on the stage (using AnchorPane's preferred height and width)
primaryStage-setScene(.bsf-new("javafx.scene.Scene", root))
primaryStage-show

::class RexxButtonHandler
::method init
  expose label  -- define an attribute
  use arg label  -- save reference to javafx.scene.control.Label

::method handle  -- will be invoked by the Java side
  expose label
  say .dateTime-new": arrived in code defined for Button’s setOnAction method, i.e. the 'handle' method"

  say "... current value of 'pp(label)': label-getText='pp(label-text)" ;
  label-text="Clicked at:" .dateTime-new -- set the label

  say .dateTime-new": returning from the event handler" ;
  say
All `javafx` controls are organized in a **DOM** tree
- **DOM**: Document Object Model
- **W3C** standard

All `javafx` controls can be formatted using **CSS**
- **CSS**: Cascading Style Sheets
- Defining styles for all nodes of the DOM tree

**JavaFX** employs **webkit** for rendering
- Open source rendering engine
- e.g. Apple uses it for Safari, Google forked it for Chrome
JavaFX Nutshell Example (Using FXML), 2 – 1

Six Files

• Image files
  - bsf4oorexx_032.png (application icon), oorexx_032.png (background)

• Dialog files
  1. File: fxml_02.css
  2. File: FXML_02_Document.fxml
  3. File: fxml_02_controller.rex
  - Automatic substitution of values if the value of the property named text starts with % or $
    • %year, %clickMe: substitute string with ResourceBundle translation files
    4. File (German), 5. File (English): FXML_02_de.properties, FXML_02_en.properties
    • $name: fetch value for name from ScriptContext Bindings at startup
    • ${name}: $-prefix and curly braces, value gets continuously fetched from ScriptContext
  - Starting the application (main program)
  6. File: fxml_02.rex
* define the background of the scene, will be applied to AnchorPane: */
.root {
    -fx-background-image: url("bsf4oorexx_032.png");
    -fx-background-color: LightGoldenRodYellow;
}

/* this is the basic formatting for all Label:s */
.label {
    -fx-font-size: 11px;
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
    -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
    -fx-border-color: red;
    -fx-border-radius: 3px;
    -fx-border-style: dashed;
    -fx-border-width: 1px;
}

/* this will change the appearance of Button a little bit: */
.button {
    -fx-text-fill: royalblue;
    -fx-font-weight: 900;
}

/* this will apply alpha (fourth value) to get the background to shine thru the
label with the class "rexxInfo"; to be able to apply the alpha, one
needs to turn the hexadecimal values into their decimal representations like:
hence: oldlace = #fdf5e6 -> fd~x2d f5~x2d e6~x2d -> rgb(253, 245, 230)
*/
.rexxStarted {
    -fx-background-color: rgb(253, 245, 230, 0.75);
    -fx-text-fill: royalblue;
}
<xml version="1.0" encoding="UTF-8"?>
<import javafx.scene.control.Button/>
<import javafx.scene.control.Label/>
<import javafx.scene.layout.AnchorPane/>
<!-- comment: this file can be displayed and edited with the JavaFX SceneBuilder, e.g. from
<http://gluonhq.com/labs/scene-builder/>; however note that as of fall 2016
(SceneBuilder 2.0) there are bugs present that may remove text-field values
that start with a $ character, so you need to fill them back in with a plain
text editor! -->
<!-- comment: the following process instruction (PI) defines the Java script engine named 'rexx'
to be used for the code in event attributes like 'onAction' -->
<?language rexx?>
<AnchorPane fx:id="idAnchorPane" prefHeight="240.0" prefWidth="480.0" styleClass="root"
/>  
<!-- comment: defines the attribute in GLOBAL_SCOPE named 'rexxStarted' to be used for labelStart -->
<fx:script source="fxml_02_controller.rex" />

<!-- comment: define the JavaFx controls that make up the GUI, all controls that possess a fx:id
attribute are stored by their id value in the ScriptContext's GLOBAL_SCOPE -->
<children>
  <!-- comment: "$rexxStarted" will cause fetching the value of the attribute "rexxStarted"
from the ScriptContext's Bindings, when initially setting up the Label;
note: SceneBuilder as of 2016-11-22 cannot handle (and deletes) the text attribute's
value: "$rexxStarted" -->
  <Label fx:id="labelRexxStarted" alignment="CENTER" layoutX="50.0" layoutY="26.0" minHeight="16" minWidth="69"
prefHeight="16.0" prefWidth="380.0" styleClass="rexxStarted" stylesheets="@fxml_02.css" text="$rexxStarted" /></children>

... continued on next slide ...
... continued from previous slide ...

<!-- comment: the Rexx code in the 'onAction' attribute will be invoked by JavaFX via a
Rexx call; note that JavaFX will remove any newline characters between the
double-quotes ("), hence each Rexx statement is explicitly ended with the
semi-colon character; note the text attribute which gets localized -->

<Button fx:id="idButton" layoutX="210.0" layoutY="137.0" onAction=""
    say ' ==> ---
      arrived in button''s onAction-code ...
'>
  /*@showsource*/ /* show this onAction-Rexx code in original and edited state */
  use arg event, slotDir /* this argument is always sent by BSF4ooRexx */
  /*@get('label')*/ /* a Rexx script annotation: incorporates the attribute 'label' as a local Rexx variable */
  say ' ... label-getText=pp(label-getText)  ...
    changing text in label to current date and time ...
    label-text=dateTime~new~string
    say ' ... label-getText=pp(label-getText)  ... now invoking the public Rexx routine ''klickButtonAction''
    call klickButtonAction slotDir /* do additional work */;        
</Button>

<!-- comment: "$year" will be replaced by the value in the ResourceBundle's properties files -->

<Label fx:id="year" layoutX="50.0" layoutY="175.0" minHeight="16" minWidth="20"
    style="-fx-background-color: palegoldenrod;" text="$year"/>

<Label fx:id="label" layoutX="95.0" layoutY="175.0" minHeight="16" minWidth="49"
    prefHeight="16.0" prefWidth="335.0" style="-fx-background-color: honeydew;"
/>  
<!-- comment: "${rexxInfo}" will cause continuous fetching of the value of the attribute
"rexxInfo" from the ScriptContext's Bindings;
note: SceneBuilder as of 2016-11-22 cannot handle the text attribute's
value: "${rexxInfo}", displays a warning icon and does not display this entry! -->

<Label fx:id="labelRexxInfo" alignment="CENTER" layoutX="50.0" layoutY="200.0" minHeight="16.0" minWidth="49.0"
    prefHeight="16.0" prefWidth="380.0" style="-fx-background-color: skyblue; -fx-cursor: wait; -fx-font-family: serif; -fx-font-weight: lighter;"
    text="${rexxInfo}" />

</children>
</AnchorPane>
Using "SceneBuilder" for the Dialog
JavaFX Nutshell Example (Using FXML), 2 – 6

Output of Running "rexx fxml_02.rex" (English GUI)
Prof. Rony G. Flatscher

/*@showsource*/   /* this Rexx script annotation will cause this script's source code to be shown */

started = dateTime ~ new
-- take the date and time

slotDir = arg(arg())
-- fetch the slotDir argument (BSF4ooRexx adds this as the last argument at the Java side)

scriptContext = slotDir.scriptContext
-- get the ScriptContext from the slotDir (last) argument

parse source s
say "just arrived at" pp(started): parse source ->" pp(s)

engine_scope = 100
-- define numeric value for engine scope Bindings

global_scope = 200
-- define numeric value for global scope Bindings

-- add an attribute to the ScriptContext's GLOBAL_SCOPE Bindings, used for "labelStartTime" in the fxml-document

scriptContext.setAttribute("rexxStarted", "Rexx started at: " starts-tring, global_scope)

parse version v
-- get Rexx version, display it in the "rexxInfo" label

scriptContext.setAttribute("rexxInfo", "Rexx version:" v, global_scope)

-- set attribute in ENGINE_SCOPE Bindings (visible for this script engine only):

scriptContext.setAttribute("title", "---> " engine_scope)

-- set attribute in GLOBAL_SCOPE Bindings (visible for all script engines):

scriptContext.setAttribute("count", 1, global_scope)

/* *************************************************************************************************** */
/* --------------------------------------------------------------------------------------------------- */
/* This routine will be called from the Rexx code defined with the "onAction" event attribute; cf. */
/* the JavaFX control with the id "label" in the fxml document */

::routine klickButtonAction public

slotDir = arg(arg())
-- fetch the slotDir argument (BSF4ooRexx adds this as the last argument at the Java side)

scriptContext = slotDir.scriptContext
-- get the slotDir (the last) argument, get the entry "SCRIPTCONTEXT"

say " --> arrived in public Rexx routine 'klickButtonAction' "

/* the following Rexx script annotation will incorporate the denoted attributes as local */

Rexx variables which can be used immediately thereafter by Rexx */

/*@get(rexxInfo label count event title)*/

... continued on next page ...
... continued from previous page ...

/* changing the attribute that gets constantly updated (once we return from
this event handler) thanks to the FXMLLoader: */

rexxInfo="Updated from public Rexx routine 'klickButtonAction'."

if count//2=0 then rexxInfo=rexxInfo-reverse  -- if even, reverse the current text
/* the following Rexx script annotation will update the value of the attribute
named 'rexxInfo' setting it to the current value of the Rexx variable named REXXINFO */

/*@set(rexxInfo)*/   -- update the attribute with the Rexx variable's current (new) value

/* show the currently defined attributes in all ScriptContext's scopes */

say "getting all attributes from all ScriptContext's scopes..."

st=.stringTable~new  -- contains the scope numbers of the Bindings

st[100]="ENGINE_SCOPE"
st[200]="GLOBAL_SCOPE"

do sc over 100, 200

  say "ScriptContext scope:" pp(sc) ": ("st-entry(sc))", available attributes:

  bin=scriptContext-getBindings(sc)

  if bin=nil then iterate  -- inexistent scope

  keys=bin-keySet  -- get key values

  it=keys-makearray  -- get the keys as a Rexx array

  do key over it-sortWith(.CaselessComparator~new)  -- sort keys (attributes) caselessly

    val=bin-get(key)  -- fetch the key's value

    str=""

    if val-isa(.bsf) then str="~toString:" pp(val~toString)

    say "  ", pp(key)~left(35, ")." pp(val) str

  end

  say ":"~copies(79)

end

-- access the "label" JavaFX Label, change its text

label~setText(title .dateTime~new-string "(count #: count")"

/* now explicitly update the count attribute in global scope bindings; if the
attribute does not exist, it would be created */

scriptContext~setAttribute("count", count+1, 200)

say " ==<--- returning from public Rexx routine 'klickButtonAction'."
3. File: “fxml_02_controller.rex” – Klick # 1 Output

... cut (output of /*@showsource*/ Rexx script annotation not shown) ...

REXXout>just arrived at [2023-01-24T19:03:46.208000]: parse source -> [WindowsNT SUBROUTINE fxml_02_controller.rex]
REXXout> ... arrived in button's onAction-code ...
REXXout> ... label-getText=[]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label-getText=[2023-01-24T19:04:30.388000]
REXXout> ... now invoking the public Rexx routine 'klickButtonAction' ...
REXXout> ... getting all attributes from all ScriptContext's scopes...
REXXout>ScriptContext scope: [180] (ENGINE_SCOPE), available attributes:
REXXout>   [event]............................ [javafx.event.ActionEvent@52fa7f17] ~toString: [javafx.event.ActionEvent[source=Button[id=idButton, styleClass=button]'Click Me!']
REXXout>   [javax.script.argv]................ [Ljava.lang.Object;@6a543dfc] ~toString: [Ljava.lang.Object;@6a543dfc]
REXXout>   [javax.script.engine].............. [Open Object Rexx (ooRexx)]
REXXout>   [javax.script.engine_version]...... [101.20220806]
REXXout>   [javax.script.filename]............ [FXML_02_Document.fxml-onAction_attribute_in_element_ending_at_line_43]
REXXout>   [javax.script.language]............ [ooRexx]
REXXout>   [javax.script.language_version].... [REXX-ooRexx_5.1.0(MT)_64-bit 6.05 6 Jan 2023]
REXXout>   [javax.script.name]................ [rexx]
REXXout>   [title]............................ [--> -> >]
REXXout>-------------------------------------------------------------------------------
REXXout>ScriptContext scope: [200] (GLOBAL_SCOPE), available attributes:
REXXout>   [count]............................ [1]
REXXout>   [idAnchorPane]..................... [javafx.scene.layout.AnchorPane@49153b46] ~toString: [AnchorPane[id=idAnchorPane, styleClass=root root]]
REXXout>   [idButton]......................... [javafx.scene.control.Button@48ce0f4e] ~toString: [Button[id=idButton, styleClass=button]'Click Me!']
REXXout>   [label]............................ [javafx.scene.control.Label@14f9c17c] ~toString: [Label[id=label, styleClass=label]'2023-01-24T19:04:30.388000']
REXXout>   [labelRexxInfo].................... [javafx.scene.control.Label@52979ac4] ~toString: [Label[id=labelRexxInfo, styleClass=label]"Updated from public Rexx routine 'klickButtonAction'."]
REXXout>   [labelRexxStarted]................. [javafx.scene.control.Label@312aef3] ~toString: [Label[id=LabelRexxStarted, styleClass=label]rexxStarted started at: 2023-01-24T19:03:46.208000']
REXXout>   [location]......................... [java.net.URL@49a8db7a] ~toString: [file:FXML_02_Document.fxml]
REXXout>   [resources]........................ [java.util.PropertyResourceBundle@76469188] ~toString: [java.util.PropertyResourceBundle@76469188]
REXXout>   [rexxFile]....................... [updated from public Rexx routine 'klickButtonAction']
REXXout>   [year]............................. [javafx.scene.control.Label@1e2c5f16] ~toString: [Label[id=year, styleClass=label]"Year->"]
REXXout>-------------------------------------------------------------------------------
REXXout> <== <--- returning from public Rexx routine 'klickButtonAction'.
REXXout>

... continued on next page ...
... continued from previous page ...

REXXout> ==> ... arrived in button's onAction-code ...
REXXout> ... label.getText=[--> -> > 2023-01-24T19:04:30.408000 (count # 1)]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label.getText=[2023-01-24T19:04:50.505000]
REXXout> ... now invoking the public Rexx routine 'klickButtonAction' ...
REXXout> ==> ... arrived in public Rexx routine 'klickButtonAction' ...
REXXout> getting all attributes from all ScriptContext's scopes...
REXXout>ScriptContext scope: [100] (ENGINE_SCOPE), available attributes:
REXXout>   [event]............................ [javafx.event.ActionEvent@667188e4] ~toString: [javafx.event.ActionEvent[source=Button[id=idButton, styleClass=button]'Click Me!']]
REXXout>   [javax.script.argv]................ [Ljava.lang.Object;@35074c73] ~toString: [Ljava.lang.Object;@35074c73]
REXXout>   [javax.script.engine].............. [Open Object Rexx (ooRexx)]
REXXout>   [javax.script.engine_version]...... [101.20220806]
REXXout>   [javax.script.filename]............ [FXML_02_Document.fxml-onAction_attribute_in_element_ending_at_line_43]
REXXout>   [javax.script.language]............ [ooRexx]
REXXout>   [javax.script.language_version].... [REXX-ooRexx_5.1.0(MT)_64-bit 6.05 6 Jan 2023]
REXXout>   [javax.script.name]................ [rexx]
REXXout> [title]............................ [--> -> >]
REXXout>-------------------------------------------------------------------------------
REXXout>ScriptContext scope: [200] (GLOBAL_SCOPE), available attributes:
REXXout>   [count]............................ [2]
REXXout>   [idAnchorPane].................... [javafx.scene.layout.AnchorPane@49153b46] ~toString: [AnchorPane[id=idAnchorPane, styleClass=root root]]
REXXout>   [idButton]......................... [javafx.scene.control.Button@48ce0f4e] ~toString: [Button[id=idButton, styleClass=button]'Click Me!']
REXXout>   [label]............................ [javafx.scene.control.Label@14f9c17c] ~toString: [Label[id=label, styleClass=label]'2023-01-24T19:04:50.505000']
REXXout>   [labelRexxInfo].................... [javafx.scene.control.Label@52979ac4] ~toString: [Label[id=labelRexxInfo, styleClass=label]'.'noitcAnottuBkcilk' enituor xxeR cilbup morf detadpU']
REXXout>   [labelRexxStarted]................. [javafx.scene.control.Label@312aef3] ~toString: [Label[id=labelRexxStarted, styleClass=label rexxStarted]'Rexx started at: 2023-01-24T19:04:50.505000']
REXXout>   [location]......................... [java.net.URL@49a8db74] ~toString: [file:FXML_02_Document.fxml]
REXXout>   [resources]........................ [java.util.PropertyResourceBundle@76469108] ~toString: [java.util.PropertyResourceBundle@76469108]
REXXout> [rexxInfo]......................... ['noitcAnottuBkcilk' emituor xxeR cilbup morf detadpu']
REXXout> [rexxStarted]...................... [Rexx started at: 2023-01-24T19:04:50.505000]
REXXout> [year]............................. [javafx.scene.control.Label@1e2c5f16] ~toString: [Label[id=year, styleClass=label]'Year->']
REXXout>-------------------------------------------------------------------------------
REXXout> <== <--- returning from public Rexx routine 'klickButtonAction'.
REXXout>... continued on next page ...
3. File: “fxml_02_controller.rex” – Klick # 3 Output

... continued from previous page...

REXXout> >>> ... arrived in button's onAction-code ...
REXXout> ... label.getText=[-> ->] > 2023-01-24T19:05:58.518000 (count # 2)]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label.getText=[2023-01-24T19:05:58.769000]
REXXout> ... now invoking the public Rexx routine 'klickButtonAction'
REXXout> >>> ... arrived in public Rexx routine 'klickButtonAction' ...
REXXout> getting all attributes from all ScriptContext's scopes...

REXXout> ScriptContext scope: [180] (ENGINE_SCOPE), available attributes:

REXXout> [event]............................ [javafx.event.ActionEvent@78f29099] ~toString: [javafx.event.ActionEvent[source=Button[id=idButton, styleClass=button]'

REXXout> [javax.script.engine].............. [Open Object Rexx (ooRexx)]

REXXout> [javax.script.engine_version]...... [REXX-ooRexx_5.1.0(MT)_64-bit 6.05 6 Jan 2023]

REXXout> [javax.script.filename]............ [FXML_02_Document.fxml-onAction_attribute_in_element_ending_at_line_43]

REXXout> [javax.script.language]............ [ooRexx]

REXXout> [javax.script.language_version].... [REXX-ooRexx_5.1.0(MT)_64-bit 6 05 6 Jan 2023]

REXXout> [javax.script.name]................ [rexx]

REXXout> [title]............................ [--> -> >]

REXXout>-------------------------------------------------------------------------------

REXXout> ScriptContext scope: [200] (GLOBAL_SCOPE), available attributes:

REXXout> [count]............................ [3]

REXXout> [idAnchorPane].................... [javafx.scene.layout.AnchorPane@49153b46] ~toString: [AnchorPane[id=idAnchorPane, styleClass=root root]

REXXout> [idButton]......................... [javafx.scene.control.Button@48ce8fe4e] ~toString: [Button[id=idButton, styleClass=button]'

REXXout> [label]............................ [javafx.scene.control.Label@18149c17c] ~toString: [Label[id=label, styleClass=label]'

REXXout> [labelRexxInfo].................... [javafx.scene.control.Label@8f297ac4] ~toString: [Label[id=labelRexxInfo, styleClass=label]'

REXXout> [labelRexxStarted]................ [javafx.scene.control.Label@312aef3] ~toString: [Label[id=labelRexxStarted, styleClass=label rexxStarted]'

REXXout> [year]............................. [javafx.scene.control.Label@e2c5f16] ~toString: [Label[id=year, styleClass=label]'

REXXout>-------------------------------------------------------------------------------

REXXout> <== <--- returning from public Rexx routine 'klickButtonAction'.

REXXout>

... ...
/* only "de" has an effect and will use the German translation for the */
parse arg locale . -- get locale from user ("en", default, or "de" for German)
   -- create Rexx object that will control the FXML set up
if locale<>"" then rexApp=.RexxApplication-new(locale)
else rexApp=.RexxApplication-new
   -- instantiate the abstract JavaFX class, the abstract "start" method will be served by rexApp
jRexxApp=BSfCreateRexxProxy(rexApp,"javax.application.Application")
   -- launch the application, invoke "start" and then stay up until the application closes
jRexxApp-launch(jRexxApp->getClass,.nil)
   -- need to use this version of launch in order to work
::requires "BSF.CLS" -- get Java support
/* implements the abstract method "start" for the Java class javafx.application.Application
   (BSF4ooRexx also supplies another (trailing) slotDir (a Rexx Directory) argument, as "start" is
   invoked from Java) */
::class RexxApplication
::method init -- constructor to fetch a locale string ("de" for German, file "fxml_01_de.properties"), if any
expose locale
use strict arg locale="en" -- default to English
use arg stage -- we get the stage to use for our UI
   -- create an URL for the FXMLDocument.fxml file (hence the protocol "file:")
rootDocUrl=bsf~new("java.net.URL", "file:FXML_02_Document.fxml")
   -- use Java translation services
  jLocale=bsf~new("java.util.Locale", locale)
  jRB=bsf.loadClass("java.util.ResourceBundle")-getBundle("FXML_02", jLocale)
root=bsf.loadClass("javafx.fxml.FXMLLoader")-load(rootDocUrl, jRB) -- load the fxml document
scene=bsf~new("javafx.scene.Scene", root) -- create a scene for our document
stage-setScene(scene) -- set the stage to our scene
img=bsf~new("javafx.scene.image.Image", "oorexx_032.png")
stage-getIcons-add(img) -- set application icon
stage-show -- show the stage (and thereby our scene)
4./5. File: “FXML_02_{de|en}.properties”

FXML_02_en.properties

! This is the English (en) translation for two terms.
!
! the following key is used in the Label with the fx:id="text", where its text attribute states (note the percentage char): text="%year"
year = Year->
!
! the following key is used in the Button with the fx:id="button", where its text attribute states (note the percentage char): text="%clickMe"
clickMe = Click Me!

FXML_02_de.properties

! This is the German (de) translation for two terms.
!
! the following key is used in the Label with the fx:id="text", where its text attribute states (note the percentage char): text="%year"
year = Jahr->
!
! the following key is used in the Button with the fx:id="button", where its text attribute states (note the percentage char): text="%clickMe"
clickMe = Drück mich!
JavaFX Nutshell Example (Using FXML), 2 – 13

Output of Running "rexx fxml_02.rex de" (German GUI)
An Address Book Application

- Simple address book example
  - Data loaded from JSON file, if available
  - Data stored in JSON file
  - List persons
  - Allow for
    - Adding, deleting, changing persons
    - Create and show statistics about the months of birth
    - Print persons according to the current list order
Files

- Rendering, graphics: address_book_128.png, DarkTheme.css, DarkThemePrint.css
- Rexx-Utilities: json-rgf.cls, put FXID_objects_into.my.app.rex
- Controlling the application
  - MainApp.rex
    - For each FXML file a Rexx class is defined to control it
- FXML-files defined with SceneBuilder
  - RootLayout.fxml, PersonOverview.fxml, BirthdayStatistics.fxml, PersonEditDialog.fxml, PersonPrinterDialog.fxml
JavaFX Nutshell Example (Using FXML), 3 – 3

Overview

- Needs ooRexx 5.0.0 or higher
- **MainApp.rex**
  - In addition creates an entry "MY.APP" in global .environment
  - The controller classes will be able to fetch the **JavaFX** objects to interact with from .MY.APP stored in a directory named after the FXML file
- **put_FXID_objects_into.my.app.rex**
  - Will be called at the end of each FXML file, after all JavaFX objects got defined
  - If there is no entry named MY.APP in the global Rexx .environment, then one will get created by that name referring to a newly created Rexx directory, such that it can be referred to by its environment symbol .MY.APP
  - Will store all JavaFX objects with an fx:id attribute in .MY.APP under the name of the FXML file name (location entry in global ScriptContext) for later retrieval
Sample JSON Content

```json
[
  {
    "birthday": "1979-03-11",
    "city": "Some City",
    "firstName": "Hans",
    "lastName": "Muster",
    "postalCode": 8985,
    "street": "some unknown street"
  },
  {
    "birthday": "2014-04-08",
    "city": "Some City",
    "firstName": "Ruth",
    "lastName": "Mueller",
    "postalCode": 9940,
    "street": "some unknown street"
  },
  ...
  ...
  ..., cut ...
  ..., cut ...
  {
    "birthday": "1978-05-20",
    "city": "Some City",
    "firstName": "Martin",
    "lastName": "Mueller",
    "postalCode": 4979,
    "street": "some unknown street"
  }
]
```
GUI Output, 1

JavaFX Nutshell Example (Using FXML), 3 – 3

Prof. Rony G. Flatscher
JavaFX Nutshell Example (Using FXML), 3 – 4

GUI Output, 2
Additional Information

• Nutshell examples
  – Cf. BSF4ooRexx850 installation in "bsf4oorexx850/samples/JavaFX"
    • Menu "BSF4ooRexx850 → Samples → JavaFX"

• Information ad JavaFX
  – Menu "BSF4ooRexx850 → Samples → JavaFX → index.html"
    • Link list to many, interesting information around JavaFX
  – Fee open source JavaFX controls
    • http://jfxtras.org/
    • http://fxexperience.com/controlsfx/features/
Roundup

• **JavaFX**
  - A great and extremely powerful GUI programming infrastructure
  - Allows meeting the most challenging GUI demands
  - **SceneBuilder** makes it easy to take full advantage of **JavaFX**
  - **DOM** and **CSS (webkit)**

• **BSF4ooRexx' javax.script** support makes it very easy to use **JavaFX** from **ooRexx**!
  - Allows for powerful and portable (!) **ooRexx** applications
  - No excuse not to create great GUIs with ooRexx! :}

Prof. Rony G. Flatscher