



# Creating Powerful and Portable GUIs with JavaFX

The 2024 International Rexx Symposium  
Brisbane, Queensland, Australia  
March 3<sup>rd</sup> – March 6<sup>th</sup> 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





## Example "property\_binding.rex"

```
-- import the Java class, allow it to be used like an ooRexx class thereafter
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

::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!
    - Cf. <http://gluonhq.com/labs/scene-builder/>
- Allows to set up an available `javafx.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!







## FX Markup Language (FXML), 2

- Invoking script code occurs with the help of `javafx.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





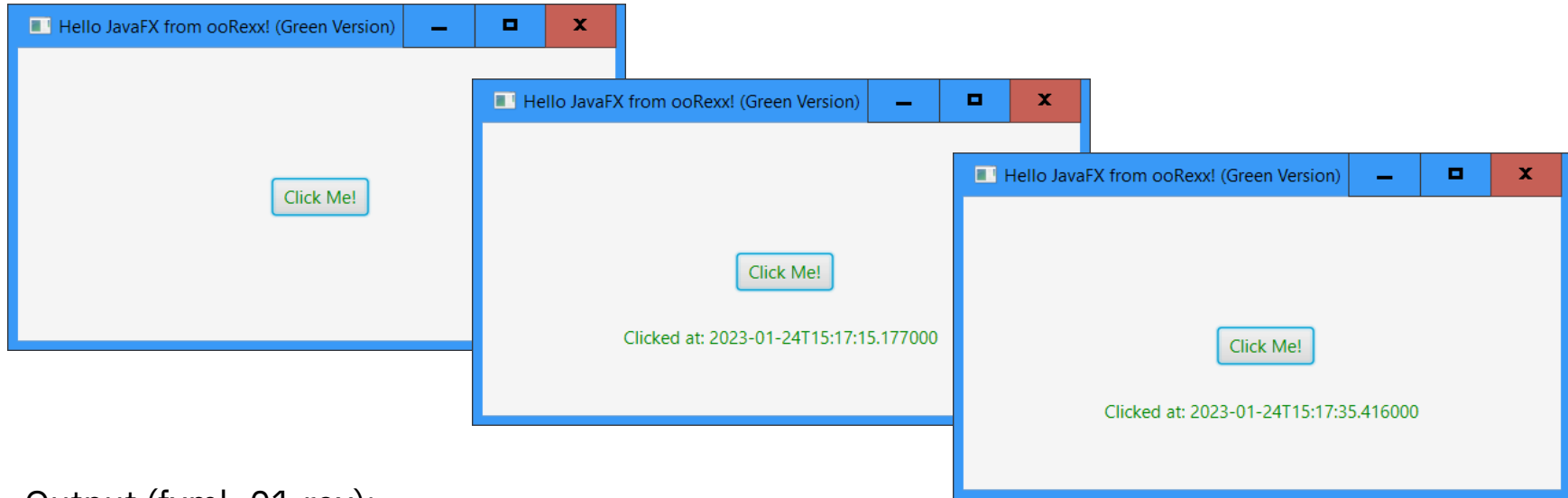
## Model-View-Controller (MVC)

- Introduced with Smalltalk-76
- **M**odel – the data to maintain
  - Our program
- **V**iew – the program to display the data
  - Our program, JavaFX or a combination of both
  - View and model can be bound with Properties!
- **C**ontroller – 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>
```



## 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”

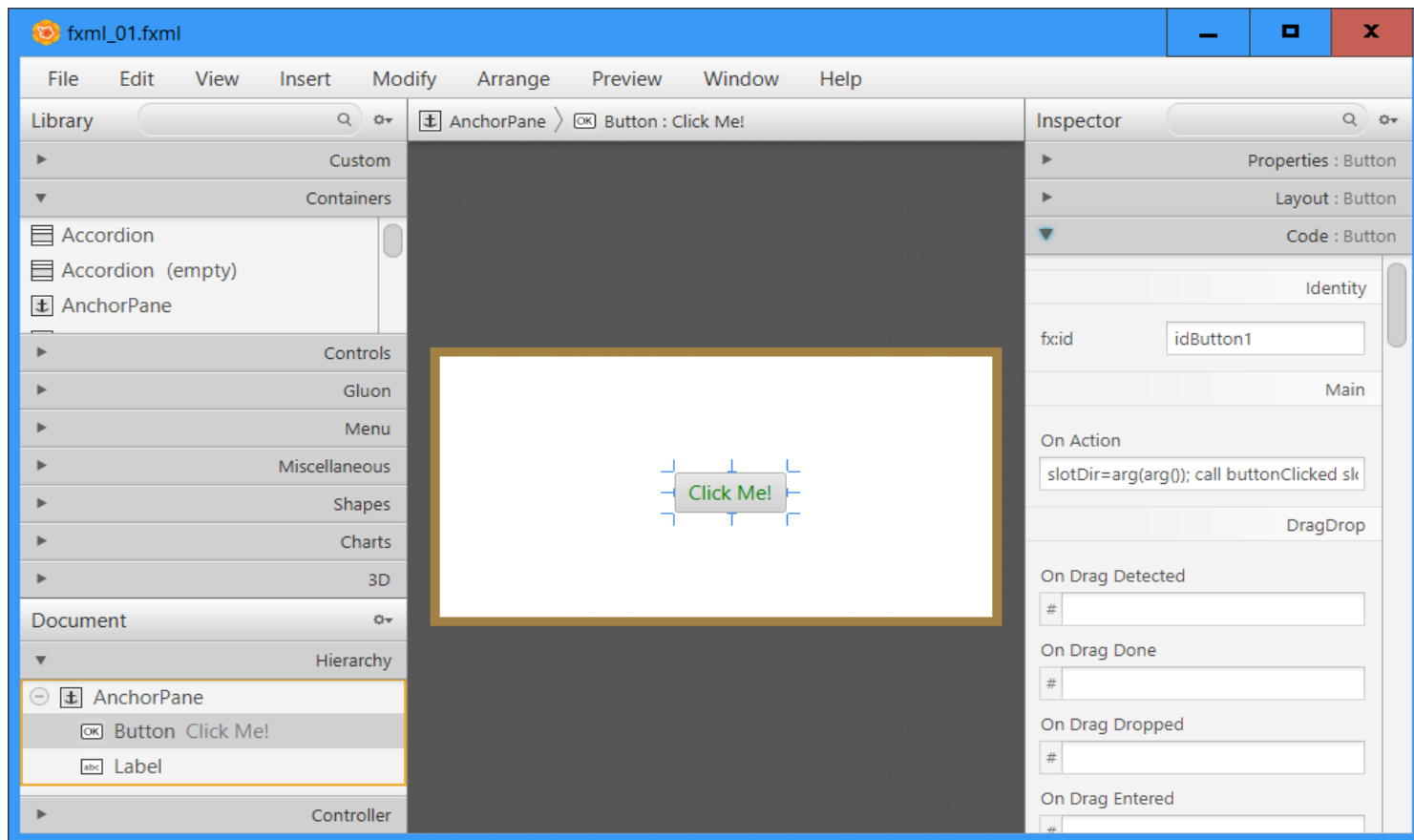
- Defines a public REXX routine "buttonClicked"

## 3. File: “FXML\_01.rex”

- Runs the program using the `javafx` package

# Using "SceneBuilder" for the Dialog

- <https://gluonhq.com/products/scene-builder/> (2022-12-11)



## 2. File: “FXML\_01\_Document.fxml”

```

<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<!-- comment: the following process instruction (PI) defines the Java script engine named 'rexx'
      to run "FXML_01_controller.rex" and the code in the event attribute 'onAction' -->
<?language rexx?>
<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"

```
/*@showsource*/
```

- 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`





## 2. File: “fxml\_01\_controller.rex”



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

```

/* 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: 2023-01-24T15:17:15.177000]
REXXout>... new value of label=[Clicked at: 2023-01-24T15:17:35.416000]
REXXout>
    
```



# Running a JavaFX Application

- 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.rEXX`”

- 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



## 1. File: "FXML\_01.rer"



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

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

```
-- Rerx class defines "javafx.application.Application" abstract method "start"
::class RerxApplication -- implements the abstract class "javafx.application.Application"
```

```
::method start -- Rerx method "start" implements the abstract method
```

```
use arg primaryStage -- fetch the primary stage (window)
```

```
primaryStage~setTitle("Hello JavaFX from ooRerx! (Green Version)")
```

```
-- create an URL for the FXMLDocument.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:
```

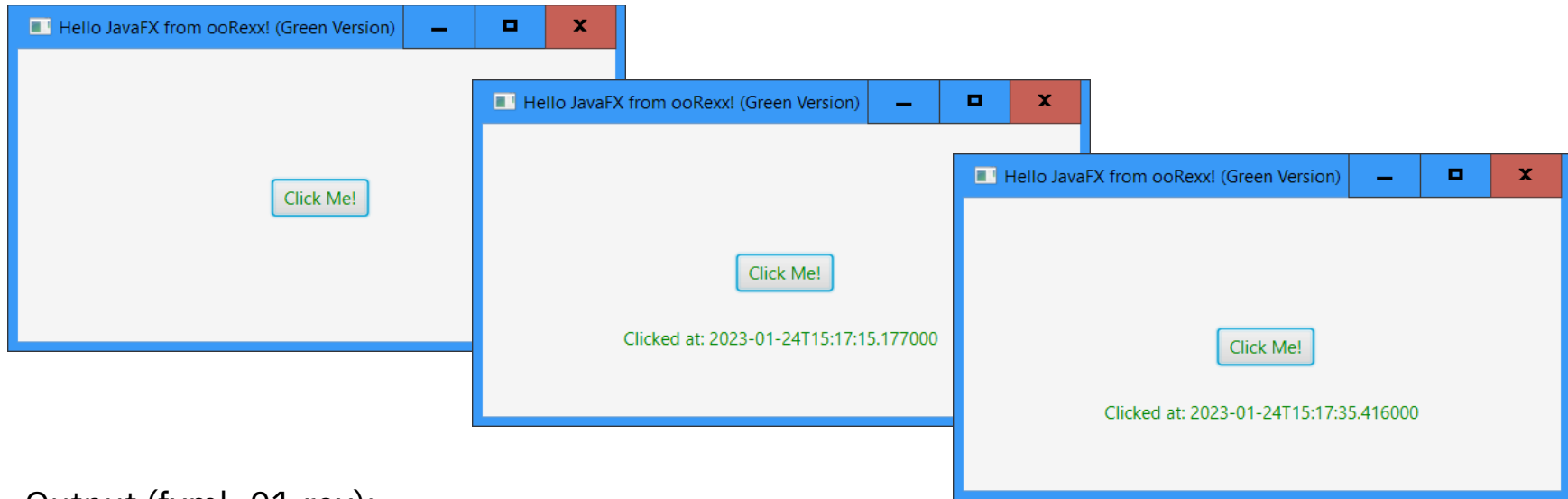
```
rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(FXMLUrl)
```

```
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>
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>
```



# 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





```
rexHandler=.RexxAppHandler~new
  -- instantiate the abstract JavaFX class, the abstract "start" method will be served by rexHandler
rxApp=BSFCreateRexxProxy(rexHandler, ,"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

-- REXX class which handles the button presses
::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
```







## DOM and CSS

- 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





## 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)



## 1. File: “fxml\_02.css”

```
* 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 e5~x2d -> rgb(253, 245, 230)
*/
.rexxStarted {
    -fx-background-color: rgb(253, 245, 230, 0.75) ;
    -fx-text-fill: royalblue;
}
```





## 2. File: “FXML\_02\_Document.fxml”, 1/2

```

<?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"
stylesheets="@FXML_02.css" xmlns="http://javafx.com/javafx/8.0.65" xmlns:fx="http://javafx.com/fxml/1">
  <!-- 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>
</AnchorPane>

```

... continued on next slide ...



## 2. File: “FXML\_02\_Document.fxml”, 2/2

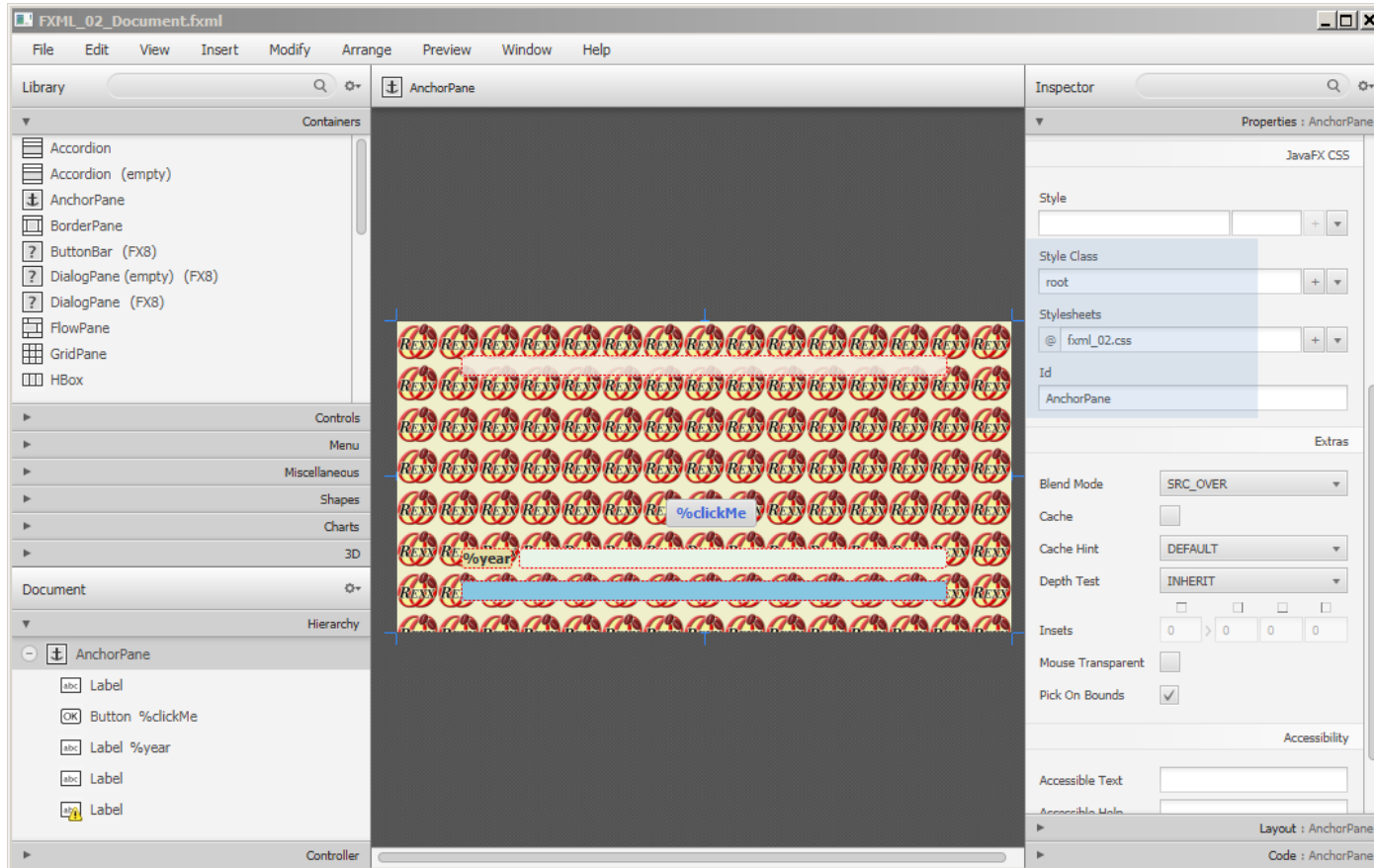
```

... continued from previous slide ...
<!-- comment: the Rextx code in the 'onAction' attribute will be invoked by JavaFX via a
Rextx call; note that JavaFX will remove any newline characters between the
double-quotes ("), hence each Rextx 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 ' ==> ---&gt; arrived in button's onAction-code ... ' ;
  /*@showsourcex*/ /* show this onAction-Rextx code in original and edited state */ ;
  use arg event, slotDir /* this argument is always sent by BSF4ooRextx */ ;
  /*@get('label')*/ /* a Rextx script annotation: incorporates the attribute 'label' as a local Rextx variable */ ;
  say ' ... label~getText='pp(label~getText) ' ;
  say ' ... changing text in label to current date and time ... ' ;
  label~text=.dateTime~new~string
  ;
  say ' ... label~getText='pp(label~getText) ' ;
  say ' ... now invoking the public Rextx routine 'klickButtonAction'' ' ;
  call klickButtonAction slotDir /* do additional work */;
  "
  text="%clickMe"
/>
<!-- 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="labelRextxInfo" 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>

```

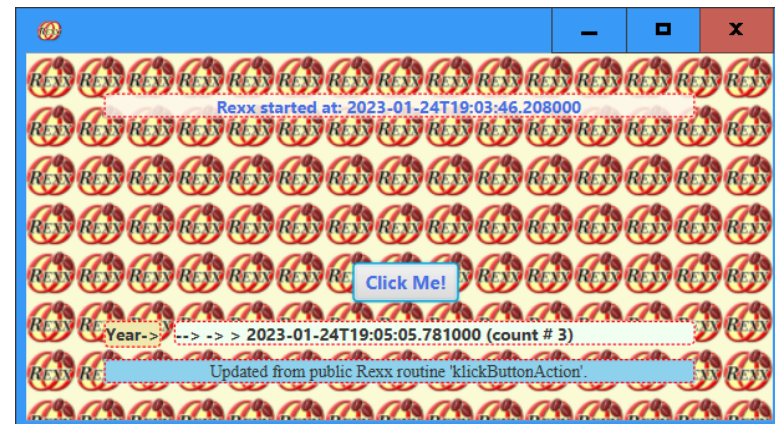
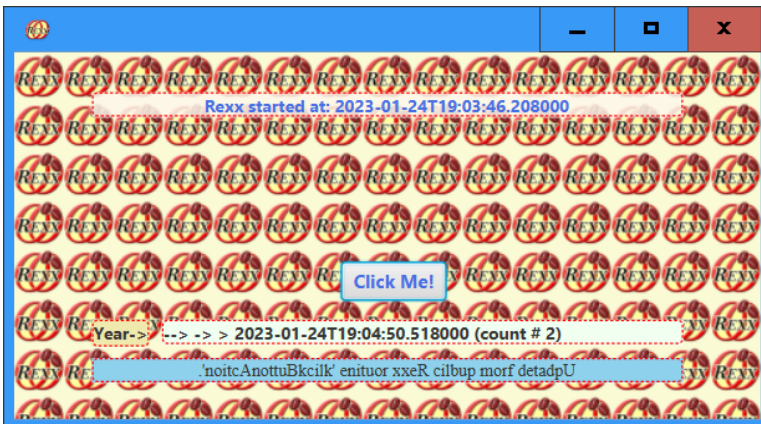
# JavaFX Nutshell Example (Using FXML), 2 – 5

## Using "SceneBuilder" for the Dialog





# Output of Running "rexx fxml\_02.rex" (English GUI)



## 3. File: “fxml\_02\_controller.rex”, 1/2

```

/*@showsourcex/* /* 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("rexStarted", "Rexx started at:" started~string, global_scope)
parse version v -- get REXX version, display it in the "rexInfo" label
scriptContext~setAttribute("rexInfo", "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(rexInfo label count event title)*/
... continued on next page ...

```





## 3. File: “fxml\_02\_controller.rex”, 2/2

```

... continued from previous page ...
/* changing the attribute that gets constantly updated (once we return from
this event handler) thanks to the FXMLLoader: */
rexInfo="Updated from public REXX routine 'klickButtonAction'."
if count//2=0 then rexxInfo=rexInfo~reverse -- if even, reverse the current text
/* the following REXX script annotation will update the value of the attribute
named 'rexInfo' 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'."
say

```



## 3. File: “fxml\_02\_controller.rex” – Klick # 1 Output

... cut (output of /\*@showsourc\*/ 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> ==> ---- 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@52fa7f17] ~toString: [javafx.event.ActionEvent[source=Button[id=idButton, styleClass=button]~> >]]
REXXout> [javafx.script.argv]..... [[Ljava.lang.Object;@6a543dfc] ~toString: [[Ljava.lang.Object;@6a543dfc]
REXXout> [javafx.script.engine]..... [Open Object REXX (ooRexx)]
REXXout> [javafx.script.engine_version]..... [101.20220806]
REXXout> [javafx.script.filename]..... [FXML_02_Document.fxml-onAction_attribute_in_element_ending_at_line_43]
REXXout> [javafx.script.language]..... [ooRexx]
REXXout> [javafx.script.language_version].... [REXX-ooRexx_5.1.0(MT)_64-bit 6.05 6 Jan 2023]
REXXout> [javafx.script.name]..... [rex]
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]'REXX started at: 2023-01-24T19:03:46.208000']
REXXout> [location]..... [java.net.URL@49a8db74] ~toString: [file:FXML_02_Document.fxml]
REXXout> [resources]..... [java.util.PropertyResourceBundle@76469108] ~toString: [java.util.PropertyResourceBundle@76469108]
REXXout> [rexInfo]..... [Updated from public REXX routine 'klickButtonAction'.']
REXXout> [rexStarted]..... [REXX started at: 2023-01-24T19:03:46.208000]
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 ...



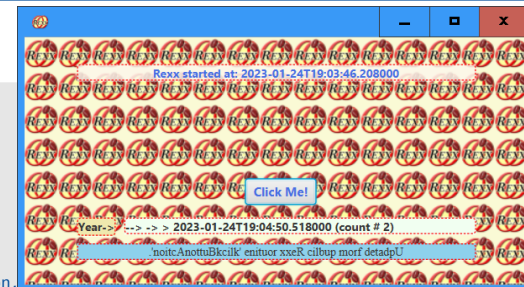


## 3. File: “fxml\_02\_controller.rex” – Klick # 2 Output

... 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,
REXXout> [javafx.script.argv]..... [[Ljava.lang.Object;@35074c73] ~toString: [[Ljava.lang.Object;@35074c73]
REXXout> [javafx.script.engine]..... [Open Object REXX (ooRexx)]
REXXout> [javafx.script.engine_version]..... [101.20220806]
REXXout> [javafx.script.filename]..... [FXML_02_Document.fxml-onAction_attribute_in_element_ending_at_line_43]
REXXout> [javafx.script.language]..... [ooRexx]
REXXout> [javafx.script.language_version].... [REXX-ooRexx_5.1.0(MT)_64-bit 6.05 6 Jan 2023]
REXXout> [javafx.script.name]..... [rex]
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: 2023-01-24T19:03:46.208000']
REXXout> [location]..... [java.net.URL@49a8db74] ~toString: [file:FXML_02_Document.fxml]
REXXout> [resources]..... [java.util.PropertyResourceBundle@76469108] ~toString: [java.util.PropertyResourceBundle@76469108]
REXXout> [rexInfo]..... ['.noitcAnottuBkcilk' enituor xxeR cilbup morf detadpU]
REXXout> [rexStarted]..... [Rexx started at: 2023-01-24T19:03:46.208000]
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=[--> -> > 2023-01-24T19:04:50.518000 (count # 2)]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label-getText=[2023-01-24T19:05:05.769000]
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: [100] (ENGINE_SCOPE), available attributes:
REXXout> [event]..... [javafx.event.ActionEvent@78f29099] ~toString: [javafx.event.ActionEvent[source=Button[id=idButton
REXXout> [javax.script.argv]..... [[Ljava.lang.Object;@91be3e6] ~toString: [[Ljava.lang.Object;@91be3e6]
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]..... [3]
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]'2023-01-24T19:05:05.769000']
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]'REXX started at: 2023-01-
24T19:03:46.208000']
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]..... [Updated from public REXX routine 'klickButtonAction'.]
REXXout> [rexxStarted]..... [REXX started at: 2023-01-24T19:03:46.208000]
REXXout> [year]..... [javafx.scene.control.Label@1e2c5f16] ~toString: [Label[id=year, styleClass=label]'Year->']
REXXout>-----
REXXout> <== <--- returning from public REXX routine 'klickButtonAction'.
REXXout>
... ..

```





## 6. File: “fxml\_02.rex” (Main Program)

```

/* 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 rexxApp=.RexxApplication-new(locale)
    else rexxApp=.RexxApplication-new
-- instantiate the abstract JavaFX class, the abstract "start" method will be served by rexxApp
jRexxApp=BsfCreateRexxProxy(rexxApp, "javafx.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
/* loads the fxml document defining the GUI elements, sets up a scene for it and shows it */
::method start  -- will be invoked by the "launch" method
expose locale
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", "oorex_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

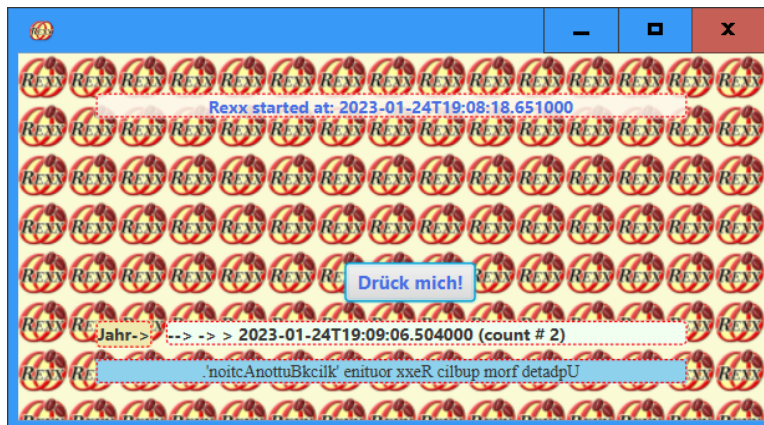
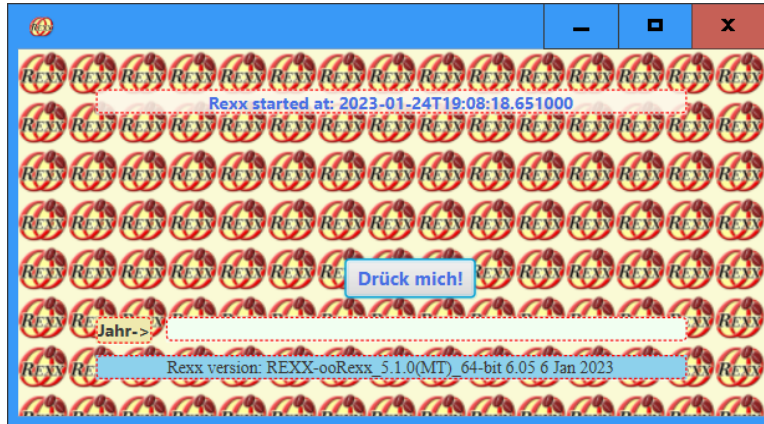
```
! This is the German (de) translation for two terms.  
!  
! the following key is used in the Label with the fx:id="text", where its text attribute states (note the percentage char): text="%year"  
year = Jahr->  
  
! the following key is used in the Button with the fx:id="button", where its text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







# Output of Running "rexx fxml\_02.rex de" (German GUI)





# An Address Book Application

- Cf. <http://code.makery.ch/library/javafx-8-tutorial/>
- 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](#)





## 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



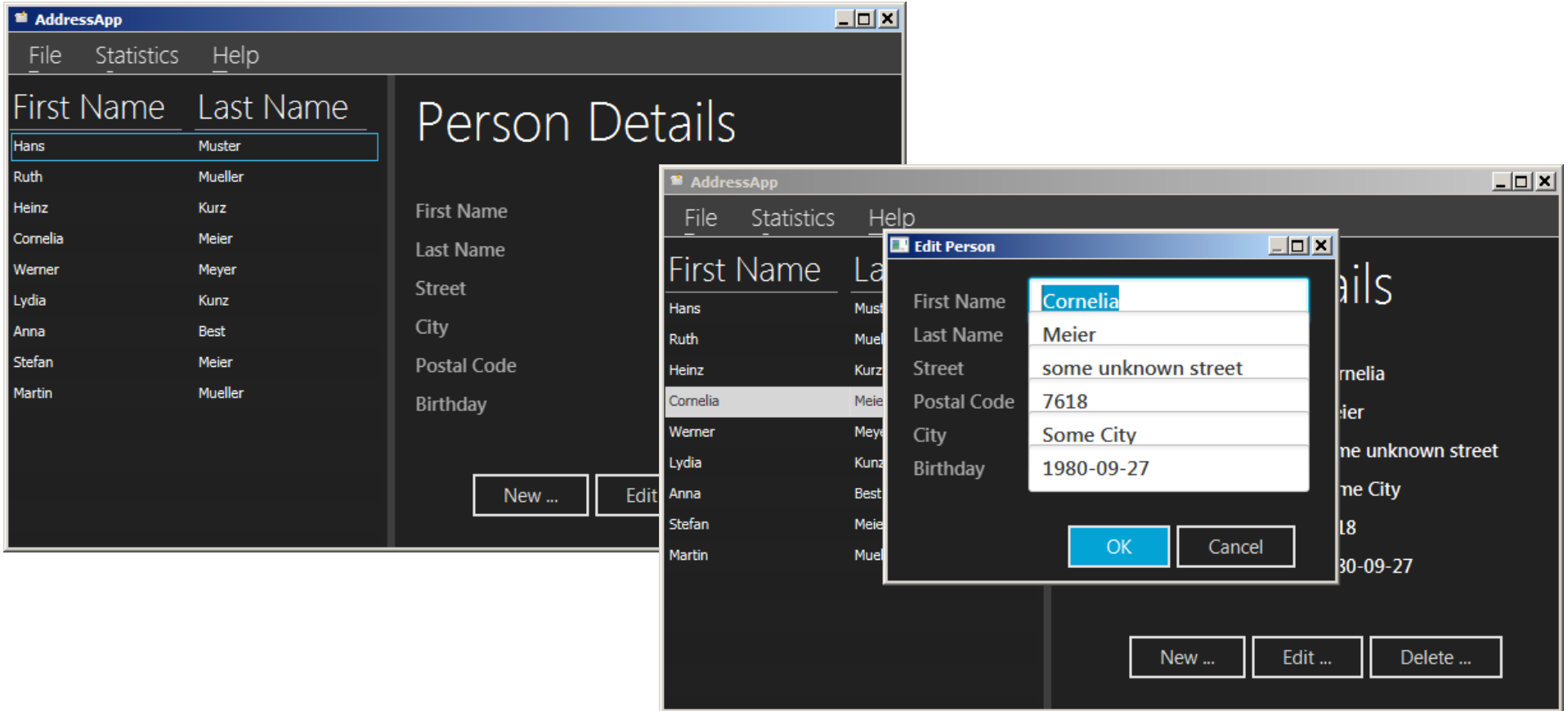
## Sample JSON Content



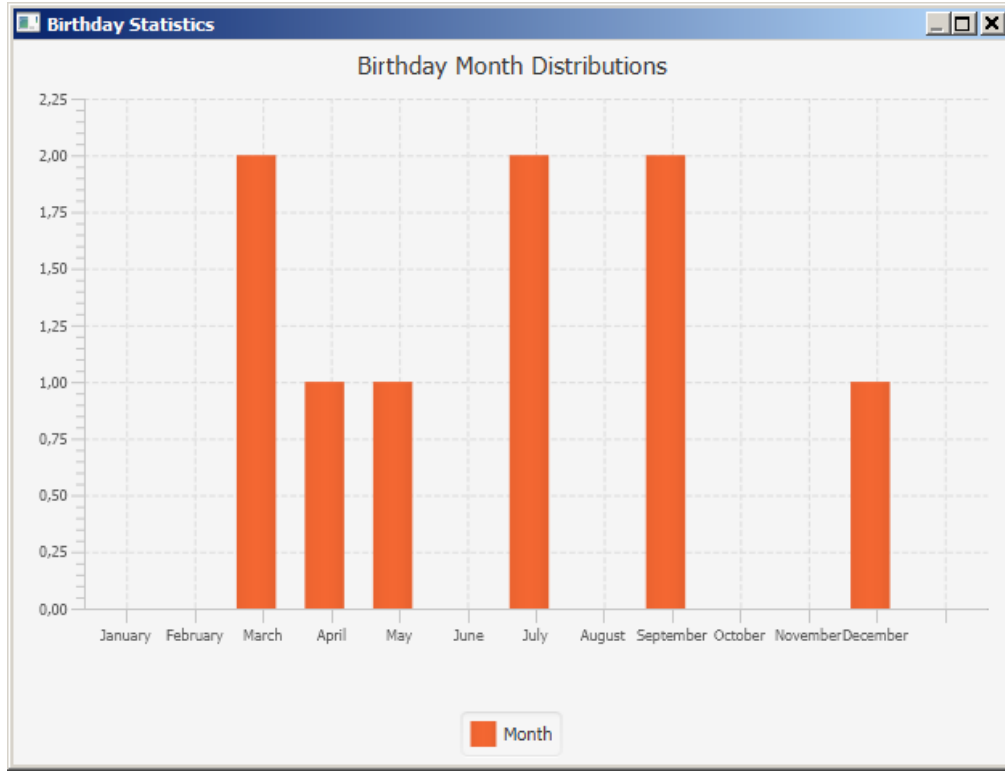
```
[
  {
    "birthday": "1979-03-11",
    "city": "Some City",
    "firstName": "Hans",
    "lastName": "Muster",
    "postalCode": 8985,
    "street": "some unknown street"
  },
  {
    "birthday": "2014-04-08",
    "city": "Some City",
    "firstName": "Ruth",
    "lastName": "Mueller",
    "postalCode":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



# GUI Output, 1



## GUI Output, 2



**Address Book Printing**

#1 **Muster, Hans**  
1979-03-11  
some unknown street  
8985 Some City

#2 **Mueller, Ruth**  
2014-04-08  
some unknown street  
9940 Some City

#3 **Kurz, Heinz**  
1950-12-06  
some unknown street  
2659 Some City

#4 **Meier, Cornelia**  
1998-08-27

2017-01-19T23:15:02.624000 -> loadWorker status=[SUCCEEDED]

Print ... Cancel

- 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/>

- [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](#)' [javafx.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! :)