Institut für Betriebswirtschaftslehre und Wirtschaftsinformatik

Abteilung für Wirtschaftsinformatik

JavaFX for ooRexx

(Creating Powerful Portable GUIs)

Business Programming 2

Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Welthandelsplatz 1 ■ A-1020 Wien

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": "**a**bstract **w**indow **t**oolkit"
    - Java GUI classes for the most important GUI controls
      - Implement what
    - 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
    - Event management and handling carried out in an "awt"-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 exists
      - e.g. javax.swingJButton vs. java.awt.Button
    - Adds PLAF
      - **P**luggable **L**ook **a**nd **F**eel
  - 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"
      - 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)

Overview of JavaFX, 1
Concepts

- "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

Overview of JavaFX, 2
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
```

Overview of JavaFX, 3
Concepts

- FXML
  - **FX M**arkup **L**anguage
  - 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 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!

Overview of JavaFX, 4
Concepts

- FXML (continued)
  - 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

Overview of JavaFX, 5
Concepts

- MVC
  - **M**odel-**V**iew-**C**ontroller (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 interaction with the model and view
      - Our program serving as the bridge between the model and the view(s)

Overview of JavaFX, 6
Example 1

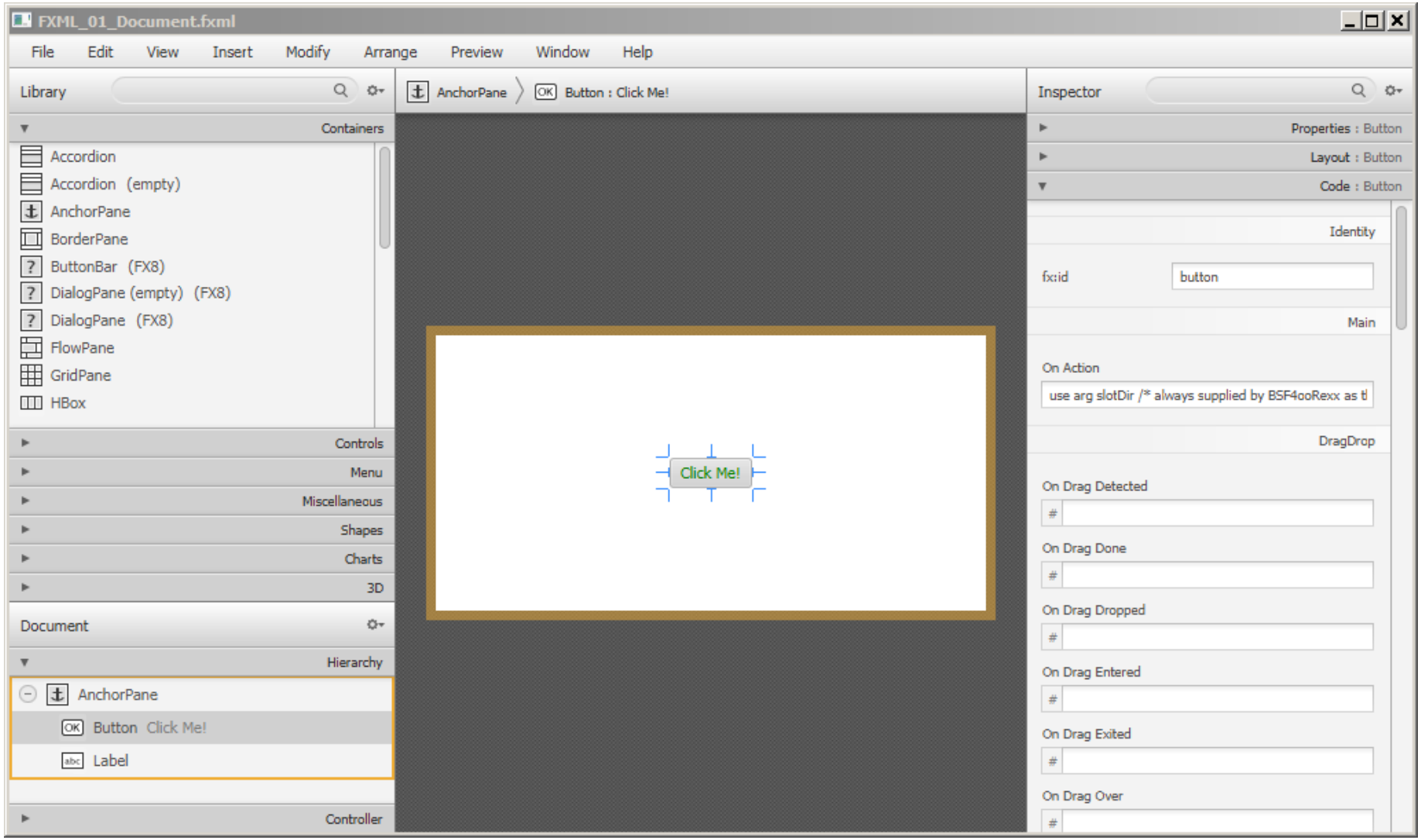
Output:

```
E:\fxml_01>fxml_01.rex

REXXout>2017-01-19T16:51:51.399000: arrived in code defined for Button's setOnAction method, i.e. the public routine 'klickButtonAction'
REXXout>... current value of [javafx.scene.control.Label@1d18228]: label~getText=[]
REXXout>2017-01-19T16:51:51.432000: returning from the event handler
REXXout>

REXXout>2017-01-19T16:52:16.525000: arrived in code defined for Button's setOnAction method, i.e. the public routine 'klickButtonAction'
REXXout>... current value of [javafx.scene.control.Label@1d18228]: label~getText=[Clicked at: 2017-01-19T16:51:51.431000]
REXXout>2017-01-19T16:52:16.555000: returning from the event handler
REXXout>
```

Overview of JavaFX, 7
Example 1, Three Files

- FXML_01_Document.fxml
  - The FXML file defining our GUI
    - Defines "rexx" to be used as the script language
    - Defines an AnchorPane containing a
      - Button with fx:id="button" (with Rexx code) and a
      - Label with fx:id="label"
      - Text (textFill) of both controls is GREEN
- fxml_01_controller.rex
  - Defines a public Rexx routine "klickButtonAction"
- fxml_01.rex
  - Runs the program using the javafx package

Overview of JavaFX, 8
Example 1, Using "SceneBuilder" for the Dialog



Overview of JavaFX, 9
Example 1, "FXML_01_Document.fxml"

```
<?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 be used for the code in event attributes like 'onAction' -->
<?language rexx?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="400" 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_01_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 in the ScriptContext's GLOBAL_SCOPE -->

    <children>
        <!-- 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 -->
        <Button fx:id="button" layoutX="170.0" layoutY="89.0" onAction=
                      "slotDir=arg(arg()) /* always supplied by BSF4ooRexx as the last argument */ ;
                       call klickButtonAction slotDir /* now process the event, pass on slotDir */ ;"
                      text="Click Me!" textFill="GREEN"
        />

        <Label fx:id="label" alignment="CENTER" contentDisplay="CENTER"
                             layoutX="76.0" layoutY="138.0"
                             minHeight="16" minWidth="49"
                             prefHeight="16.0" prefWidth="248.0"
                             textFill="GREEN" />
    </children>
</AnchorPane>
```

Overview of JavaFX, 10
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

Overview of JavaFX, 11
Example 1, "fxml_01_controller.rex"

- Defines the *public* Rexx routine "klickButtonAction"
  - 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

Overview of JavaFX, 12
Example 1, "fxml_01_controller.rex"

```
/* This routine will be called from the Rexx code defined with
   the "onAction" attribute in Button's definition */
::routine klickButtonAction public
  use arg slotDir    -- allows access to ScriptContext
  say .dateTime~new": arrived in code defined for Button's setOnAction method, i.e. the" -
                  "public routine 'klickButtonAction'"

      /* the following Rexx script annotation will incorporate the attribute (the Label
         object named "label" as this is its fx:id attribute in the FXML file and make
         it immediately available as a Rexx variable named "label" */
    /*@get("label")*/
    /* now the local Rexx variable named 'label' is available and can be immediately used: */
  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
```

Responsible for updating the Label object using the (fx:)id value (case-sensitive!) "label" and for the following output to stdout:

```
E:\fxml_01>fxml_01.rex

REXXout>2017-01-19T16:51:51.399000: arrived in code defined for Button's setOnAction method, i.e. the public routine 'klickButtonAction'
REXXout>... current value of [javafx.scene.control.Label@1d18228]: label~getText=[]
REXXout>2017-01-19T16:51:51.432000: returning from the event handler
REXXout>

REXXout>2017-01-19T16:52:16.525000: arrived in code defined for Button's setOnAction method, i.e. the public routine 'klickButtonAction'
REXXout>... current value of [javafx.scene.control.Label@1d18228]: label~getText=[Clicked at: 2017-01-19T16:51:51.431000]
REXXout>2017-01-19T16:52:16.555000: returning from the event handler
REXXout>
```

Overview of JavaFX, 13
Concepts, 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
  - Defines a Rexx class implementing the method start
  - Uses BsfCreateRexxProxy() to create a proxied Application, send it the launch message

Overview of JavaFX, 14
Example 1, "fxml_01.rex"

- Defines the Rexx class RxDocHandler
  - 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

Overview of JavaFX, 15
Example 1, "fxml_01.rex"

```
rexxHandler=.rxDocHandler~new  -- create Rexx object that will control the FXML set up
   -- 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)     -- need to use this version of launch in order to work

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

::class RxDocHandler    -- Rexx class that implements the start method

   /* 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
  use arg stage   -- we get the stage to use for our UI

  stage~setTitle("Hello JavaFX from ooRexx!")   -- we could use stage~title="..." instead!

   -- create an URL for the FMXLDocument.fxml file (hence the protocol "file:")
  rootDocUrl=.bsf~new("java.net.URL", "file:FXML_01_Document.fxml")
  root=bsf.loadClass("javafx.fxml.FXMLLoader")~load( rootDocUrl  )   -- 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
  stage~show                                    -- show the stage (and thereby our scene)
```


Overview of JavaFX, 15
Concept, 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

Overview of JavaFX, 16
Example 1, "javafx_01.rex"

```
rexxHandler=.RexxAppHandler~new -- create Rexx object that will control the FXML set up
   -- 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 ...
```

Overview of JavaFX, 17
Example 1, "javafx_01.rex"

```
... continued from previous 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
```

Overview of JavaFX, 18
Concepts

- 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

Overview of JavaFX, 19
Example 2, Six Files

- Image files  
  - bsf4oorexx_032.png (application icon), oorexx_032.png (background)
- Dialog files
  - fxml_02.css, FXML_02_Document.fxml, fxml_02_controller.rex
  - Automatic substitution of values (problems with SceneBuilder 2.0!)
    - %year, %clickMe: FXML_02_de.properties, FXML_02_en.properties
    - $-prefix: fetch value from ScriptContext at startup
    - ${name}: fetch value continuously from ScriptContext
- Starting the application
  - fxml_02.rex

Overview of JavaFX, 20
Example 2, "xml_02.css"

```
/* Some Java-FX CSS definitions, cf. <http://docs.oracle.com/javafx/2/get_started/css.htm>,
   especially: <https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>.
   The following style definitions only have the purpose to demonstrate the power available.
   These definitions are only meant for fun and starting point for experiments, not for
   professional use! :)    2016-11-22, rgf  */

/* 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 appearence 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;
}
```

Overview of JavaFX, 21
Example 2, "FXML_02_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: 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 id="AnchorPane" fx:id="root" 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 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" />
```

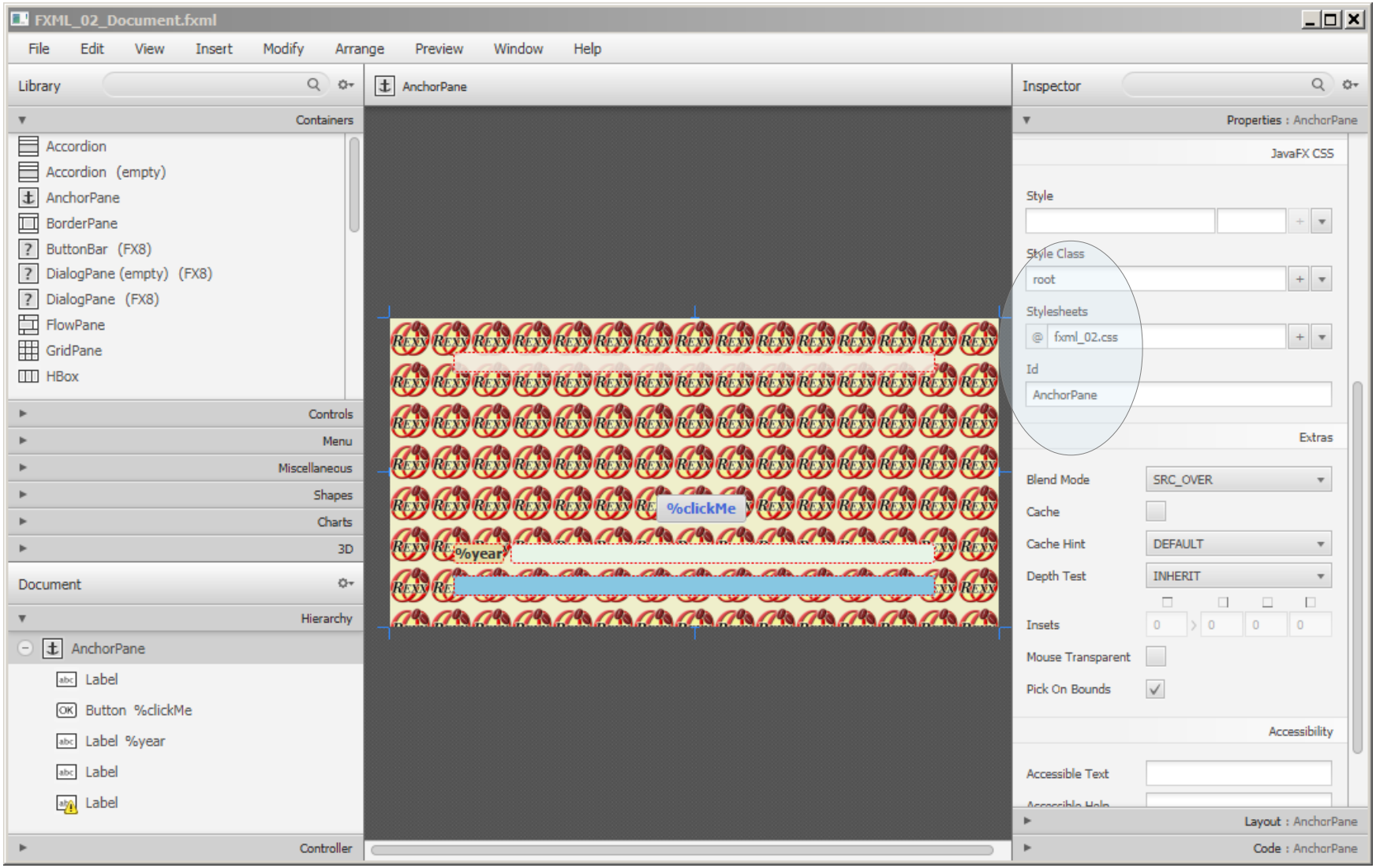
… continued on next page …

Overview of JavaFX, 22
Example 2, "FXML_02_Document.fxml"

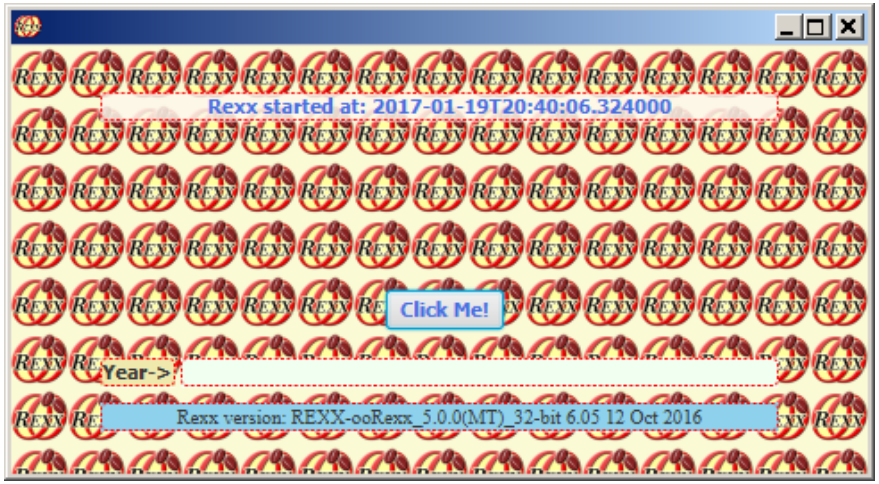
… continued from previous page …

```
        <!-- 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="button" layoutX="210.0" layoutY="137.0" onAction=
           "
                  say ' ==&gt; ---&gt; arrived in button''s onAction-code ...'  ;
                  /*@showsource*/ /* the previous Rexx script annotation will cause this code to be shown in original and edited state */;
                  use arg slotDir /* this argument is always sent by BSF4ooRexx */ ;
                  /*@get('label')*/ /* the previous Rexx script annotation will incorporate the attribute 'label' as a Rexx 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 Rexx routine ''klickButtonAction''' ;
                  call klickButtonAction slotDir /* now process the event */;
           "
           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="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>
```

Overview of JavaFX, 23
Example 2, Using "SceneBuilder" for the Dialog



Overview of JavaFX, 24
Example 2



Overview of JavaFX, 25
Example 2, "fxml_02_controller.rex"

```
started=.dateTime~new    -- take the date and time
/*@showsource*/   /* the previous Rexx script annotation will cause this script's source code to be shown */
use arg slotDir          -- 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)

   -- add an attribute to the ScriptContext's GLOBAL_SCOPE Bindings, used for "labelStartTime" in the fxml-document
.JSR223~setAttribute(scriptContext, "rexxStarted", "Rexx started at:" started~string, .jsr223~global_scope)

parse version v          -- get Rexx version, display it in the "rexxInfo" label
.JSR223~setAttribute(scriptContext, "rexxInfo", "Rexx version:" v, .jsr223~global_scope)

   -- set attribute at ENGINE_SCOPE (visible for this script engine only):
.JSR223~setAttribute(scriptContext, "title", "--> -> >")

   -- set attribute at global scope (visible for all script engines):
.JSR223~setAttribute(scriptContext, "count", "1", .jsr223~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
    use arg slotDir  -- 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 attributes as local
          Rexx variables which can be used immediately thereafter from Rexx:  */
    /*@get("rexxInfo label count event title")*/

... continued on next page ...
```

Overview of JavaFX, 26
Example 2, "fxml_02_controller.rex"

```
... 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 an existing
          attribute in one of the ScriptContext's scopes to the current value of the
          Rexx variable:   */
    /*@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..."
    dir=.directory~new
    dir[.jsr223~engine_scope]="ENGINE_SCOPE"
    dir[.jsr223~global_scope]="GLOBAL_SCOPE"
    arr=.array~of(.jsr223~engine_scope, .jsr223~global_scope)
    do sc over arr
        say "ScriptContext scope:" pp(sc) "("dir~entry(sc)"), available attributes:"
        bin=.jsr223~getBindings(scriptContext,sc)
        if bin=.nil then iterate -- inexistent scope
        keys=bin~keySet          -- get kay 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        */
    .JSR223~setAttribute(scriptContext, "count", count+1,  .jsr223~global_scope)
    say " <== <--- returning from public Rexx routine 'klickButtonAction'."
    say
```

Overview of JavaFX, 27
Example 2, "fxml_02_controller.rex", Some Output

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

REXXout> ==> ---> arrived in button's onAction-code ...
REXXout> ... label~getText=[]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label~getText=[2017-01-19T20:40:56.722000]
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@1af2deb] ~toString: [javafx.event.ActionEvent[source=Button[id=button,
styleClass=button]'Click Me!']]
REXXout>   [javax.script.engine]............. [Open Object Rexx (ooRexx)]
REXXout>   [javax.script.engine_version]..... [100.20161221]
REXXout>   [javax.script.language]........... [ooRexx]
REXXout>   [javax.script.language_version].... [REXX-ooRexx_5.0.0(MT)_32-bit 6.05 12 Oct 2016]
REXXout>   [javax.script.name]............... [Rexx]
REXXout>   [title]........................... [--> -> >]
REXXout>-------------------------------------------------------------------------------
REXXout>ScriptContext scope: [200] (GLOBAL_SCOPE), available attributes:
REXXout>   [button].......................... [javafx.scene.control.Button@1d18228] ~toString: [Button[id=button, styleClass=button]'Click Me!']
REXXout>   [count]........................... [1]
REXXout>   [label]........................... [javafx.scene.control.Label@1b892a4] ~toString: [Label[id=label, styleClass=label]'2017-01-
19T20:40:56.722000']
REXXout>   [labelRexxInfo]................... [javafx.scene.control.Label@1f604d6] ~toString: [Label[id=labelRexxInfo, styleClass=label]'Updated
frompublic Rexx routine 'klickButtonAction'.']
REXXout>   [labelRexxStarted]................ [javafx.scene.control.Label@ebb052] ~toString: [Label[id=labelRexxStarted, styleClass=label
rexxStarted]'Rexx started at: 2017-01-19T20:40:06.324000']
REXXout>   [location]........................ [java.net.URL@dbbb3] ~toString: [file:FXML_02_Document.fxml]
REXXout>   [resources]....................... [java.util.PropertyResourceBundle@507c1] ~toString: [java.util.PropertyResourceBundle@507c1]
REXXout>   [rexxInfo]........................ [Updated from public Rexx routine 'klickButtonAction'.]
REXXout>   [rexxStarted]..................... [Rexx started at: 2017-01-19T20:40:06.324000]
REXXout>   [root]............................ [javafx.scene.layout.AnchorPane@138d8b7] ~toString: [AnchorPane[id=AnchorPane, styleClass=root
root]]
REXXout>   [year]............................ [javafx.scene.control.Label@11659ae] ~toString: [Label[id=year, styleClass=label]'Year->']
REXXout>-------------------------------------------------------------------------------
REXXout> <== <--- returning from public Rexx routine 'klickButtonAction'.
REXXout>

... continued on next page ...
```

Overview of JavaFX, 28
Example 2, "fxml_02_controller.rex", Some Output

```
... continued from previous page ...

REXXout> ==> ---> arrived in button's onAction-code ...
REXXout> ... label~getText=[--> -> > 2017-01-19T20:40:56.792000 (count # 1)]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label~getText=[2017-01-19T20:41:47.371000]
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@cc2042] ~toString: [javafx.event.ActionEvent[source=Button[id=button,
styleClass=button]'Click Me!']]
REXXout>   [javax.script.engine]............. [Open Object Rexx (ooRexx)]
REXXout>   [javax.script.engine_version]..... [100.20161221]
REXXout>   [javax.script.language]........... [ooRexx]
REXXout>   [javax.script.language_version].... [REXX-ooRexx_5.0.0(MT)_32-bit 6.05 12 Oct 2016]
REXXout>   [javax.script.name]............... [Rexx]
REXXout>   [title]........................... [--> -> >]
REXXout>-------------------------------------------------------------------------------
REXXout>ScriptContext scope: [200] (GLOBAL_SCOPE), available attributes:
REXXout>   [button].......................... [javafx.scene.control.Button@1d18228] ~toString: [Button[id=button, styleClass=button]'Click Me!']
REXXout>   [count]........................... [2]
REXXout>   [label]........................... [javafx.scene.control.Label@1b892a4] ~toString: [Label[id=label, styleClass=label]'2017-01-
19T20:41:47.371000']
REXXout>   [labelRexxInfo]................... [javafx.scene.control.Label@1f604d6] ~toString: [Label[id=labelRexxInfo,
styleClass=label]'.'noitcAnottuBkcilk' enituor xxeR cilbup morf detadpU']
REXXout>   [labelRexxStarted]................ [javafx.scene.control.Label@ebb052] ~toString: [Label[id=labelRexxStarted, styleClass=label
rexxStarted]'Rexx started at: 2017-01-19T20:40:06.324000']
REXXout>   [location]........................ [java.net.URL@dbbb3] ~toString: [file:FXML_02_Document.fxml]
REXXout>   [resources]....................... [java.util.PropertyResourceBundle@507c1] ~toString: [java.util.PropertyResourceBundle@507c1]
REXXout>   [rexxInfo]........................ [.'noitcAnottuBkcilk' enituor xxeR cilbup morf detadpU]
REXXout>   [rexxStarted]..................... [Rexx started at: 2017-01-19T20:40:06.324000]
REXXout>   [root]............................ [javafx.scene.layout.AnchorPane@138d8b7] ~toString: [AnchorPane[id=AnchorPane, styleClass=root
root]]
REXXout>   [year]............................ [javafx.scene.control.Label@11659ae] ~toString: [Label[id=year, styleClass=label]'Year->']
REXXout>-------------------------------------------------------------------------------
REXXout> <== <--- returning from public Rexx routine 'klickButtonAction'.
REXXout>

... continued on next page ...
```

Overview of JavaFX, 29
Example 2, "fxml_02_controller.rex", Some Output

```
... continued from previous page ...

REXXout> ==> ---> arrived in button's onAction-code ...
REXXout> ... label~getText=[--> -> > 2017-01-19T20:41:47.481000 (count # 2)]
REXXout> ... changing text in label to current date and time ...
REXXout> ... label~getText=[2017-01-19T20:42:10.265000]
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@45d3b2] ~toString: [javafx.event.ActionEvent[source=Button[id=button,
styleClass=button]'Click Me!']]
REXXout>   [javax.script.engine]............. [Open Object Rexx (ooRexx)]
REXXout>   [javax.script.engine_version]..... [100.20161221]
REXXout>   [javax.script.language]........... [ooRexx]
REXXout>   [javax.script.language_version].... [REXX-ooRexx_5.0.0(MT)_32-bit 6.05 12 Oct 2016]
REXXout>   [javax.script.name]............... [Rexx]
REXXout>   [title]........................... [--> -> >]
REXXout>-------------------------------------------------------------------------------
REXXout>ScriptContext scope: [200] (GLOBAL_SCOPE), available attributes:
REXXout>   [button].......................... [javafx.scene.control.Button@1d18228] ~toString: [Button[id=button, styleClass=button]'Click Me!']
REXXout>   [count]........................... [3]
REXXout>   [label]........................... [javafx.scene.control.Label@1b892a4] ~toString: [Label[id=label, styleClass=label]'2017-01-
19T20:42:10.265000']
REXXout>   [labelRexxInfo]................... [javafx.scene.control.Label@1f604d6] ~toString: [Label[id=labelRexxInfo, styleClass=label]'Updated
frompublic Rexx routine 'klickButtonAction'.']
REXXout>   [labelRexxStarted]................ [javafx.scene.control.Label@ebb052] ~toString: [Label[id=labelRexxStarted, styleClass=label
rexxStarted]'Rexx started at: 2017-01-19T20:40:06.324000']
REXXout>   [location]........................ [java.net.URL@dbbb3] ~toString: [file:FXML_02_Document.fxml]
REXXout>   [resources]....................... [java.util.PropertyResourceBundle@507c1] ~toString: [java.util.PropertyResourceBundle@507c1]
REXXout>   [rexxInfo]........................ [Updated from public Rexx routine 'klickButtonAction'.]
REXXout>   [rexxStarted]..................... [Rexx started at: 2017-01-19T20:40:06.324000]
REXXout>   [root]............................ [javafx.scene.layout.AnchorPane@138d8b7] ~toString: [AnchorPane[id=AnchorPane, styleClass=root
root]]
REXXout>   [year]............................ [javafx.scene.control.Label@11659ae] ~toString: [Label[id=year, styleClass=label]'Year->']
REXXout>-------------------------------------------------------------------------------
REXXout> <== <--- returning from public Rexx routine 'klickButtonAction'.
REXXout>

... ...
```

Overview of JavaFX, 30
Example 2, "fxml_02.rex"

```
   /* only "de" has an effect and will use the German translation for the */
parse arg locale .

   -- create Rexx object that will control the FXML set up with or without local
if locale<>"" then rexxHandler=.rxDocHandler~new(locale)
              else rexxHandler=.rxDocHandler~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)     -- need to use this version of launch in order to work
say center(" after rxApp~launch ", 70, "-")

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

/* implements the abstract method "start" for the Java class javafx.application.Application */
::class RxDocHandler

::method init  -- constructor to fetch a locale string ("de" for German: "fxml_01_de.properties")
  expose locale
  use strict arg locale="en"

   /* 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 FMXLDocument.fxml file (hence the protocol "file:")
  rootDocUrl=.bsf~new("java.net.URL", "file:FXML_02_Document.fxml")
  jLocale=.bsf~new("java.util.Locale", locale)
  jRB=bsf.importClass("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
  scene~getStyleSheets~add("fxml_02.css")        -- use this CSS to style the UI

  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)
```

Overview of JavaFX, 31
Example 2, "FXML_02_{de|en}.properties"

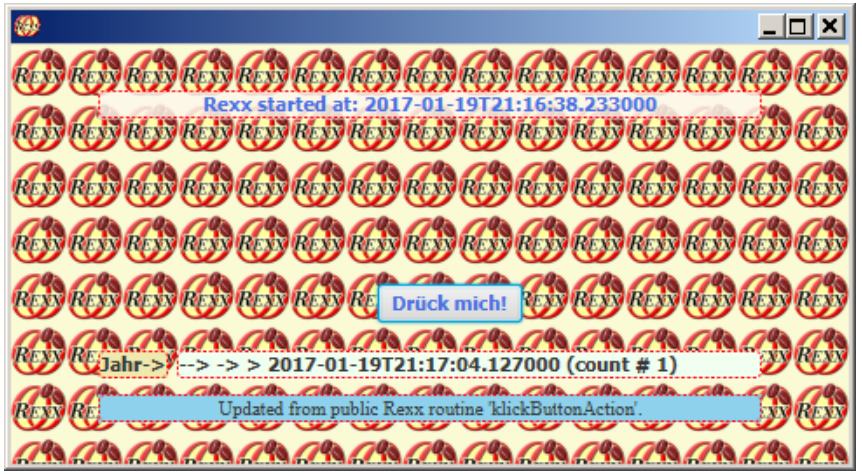
FXML_02_en.properties

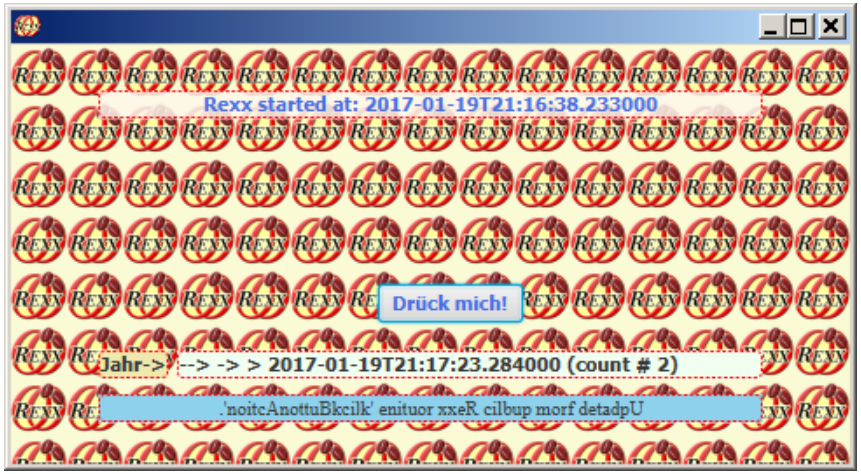
```
! This is the English (en) translation for two terms.
!
! the following key is used in the Label with the fx:id="text", where
! its text attribute states (note the percentage char): text="%year"
year = Year->

! 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!
```

Overview of JavaFX, 32
Example 2





An Address Book Application with JavaFX, 1
Example 3, Overview

- 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

An Address Book Application with JavaFX, 2 Example 3, 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

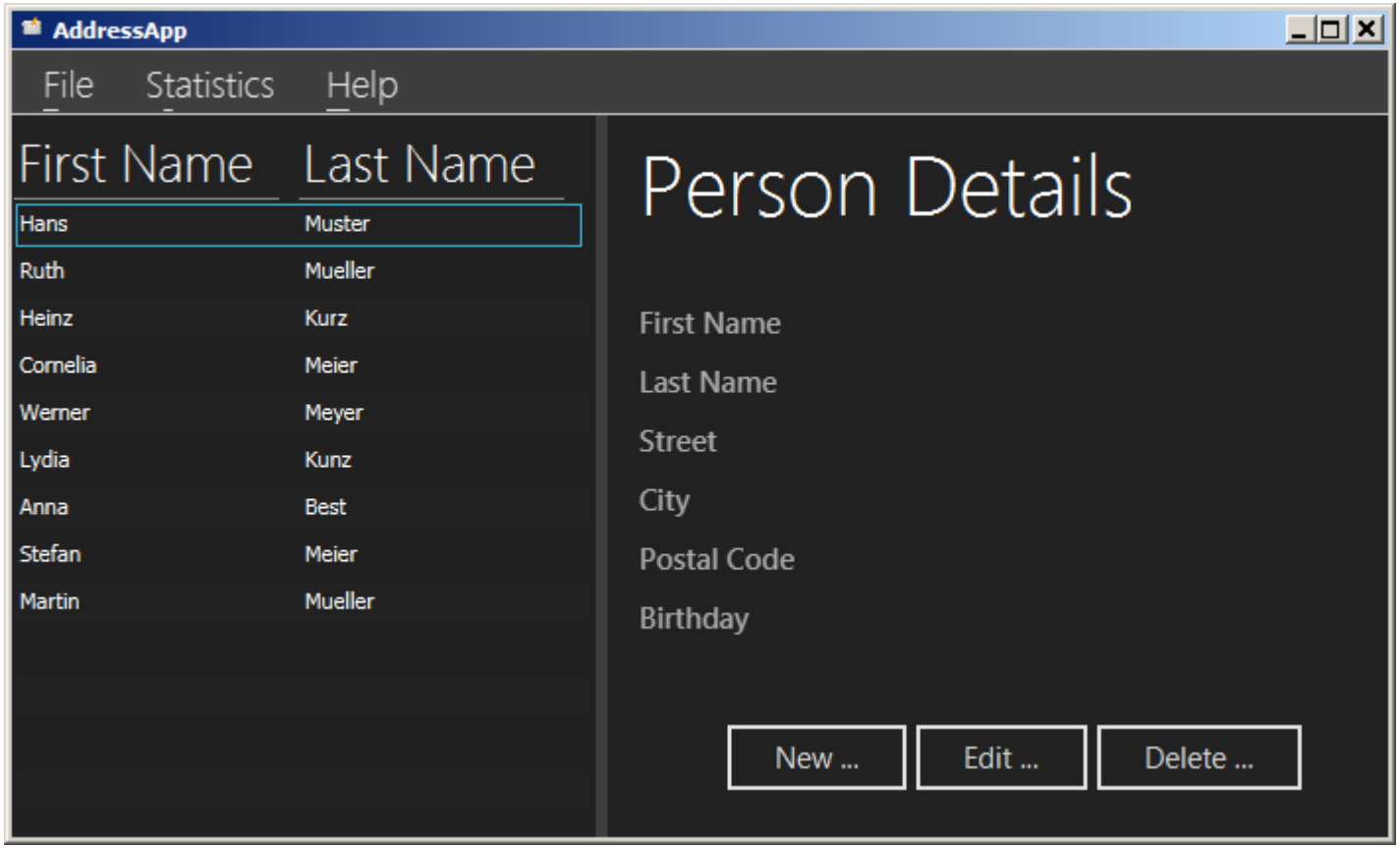
An Address Book Application with JavaFX, 3 Example 3, Overview

- Needs ooRexx 5.0 or higher (beta as of 2017-01-19)
- 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

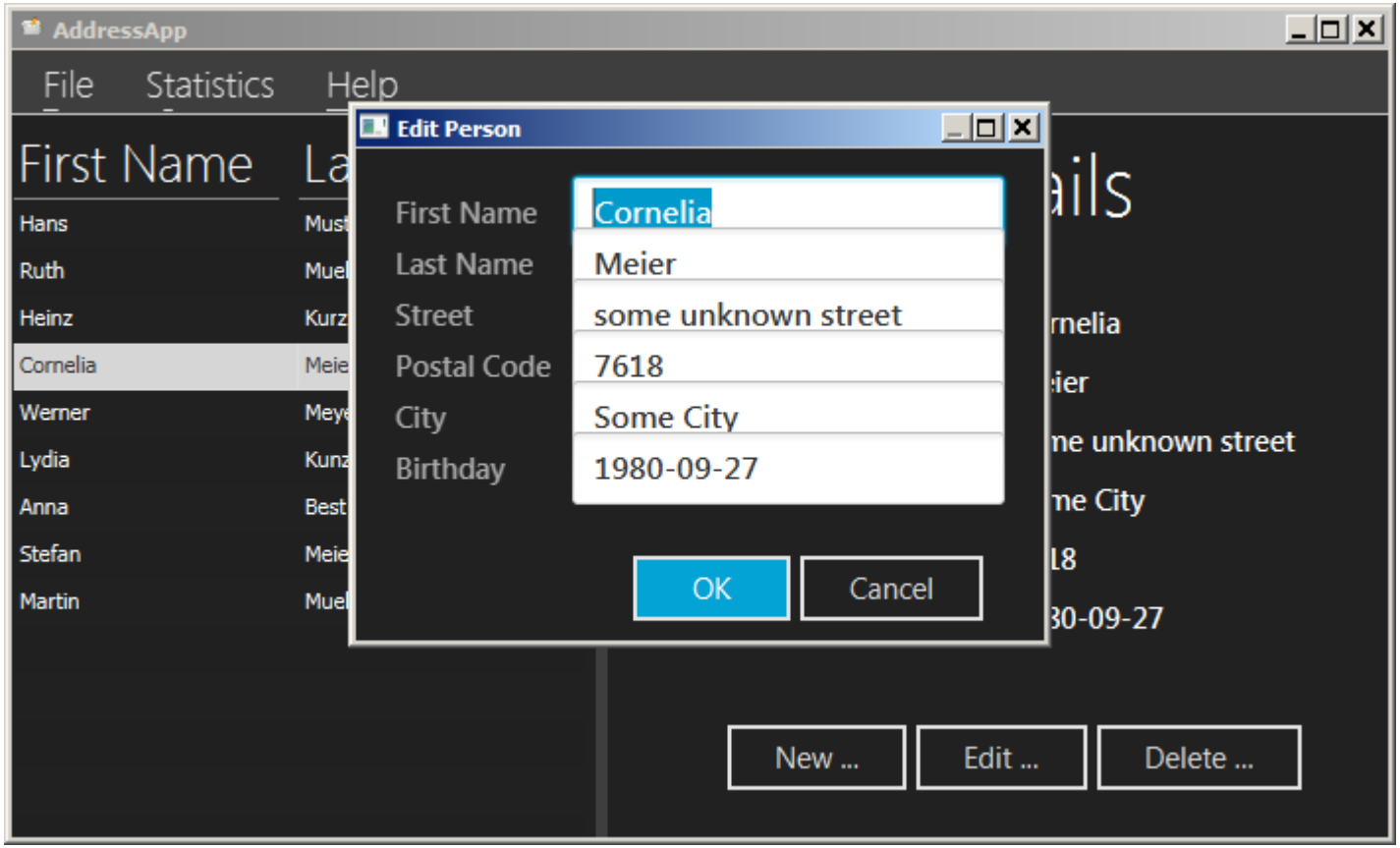
An Address Book Application with JavaFX, 4
Example 3, Sample JSON Content

```
[
  {
    "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"
  }
]
```

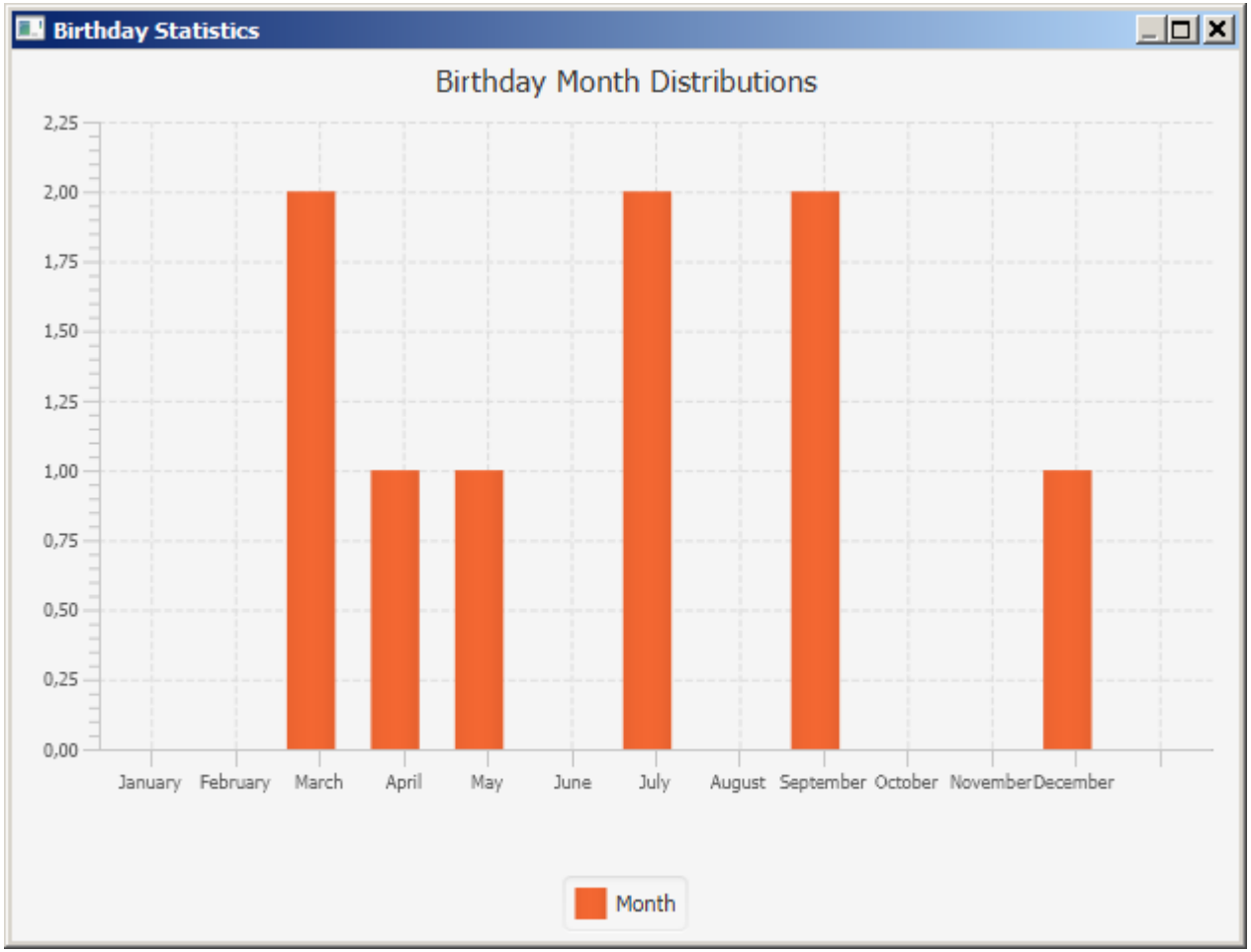
An Address Book Application with JavaFX, 5 Example 3, Screenshots 1/4



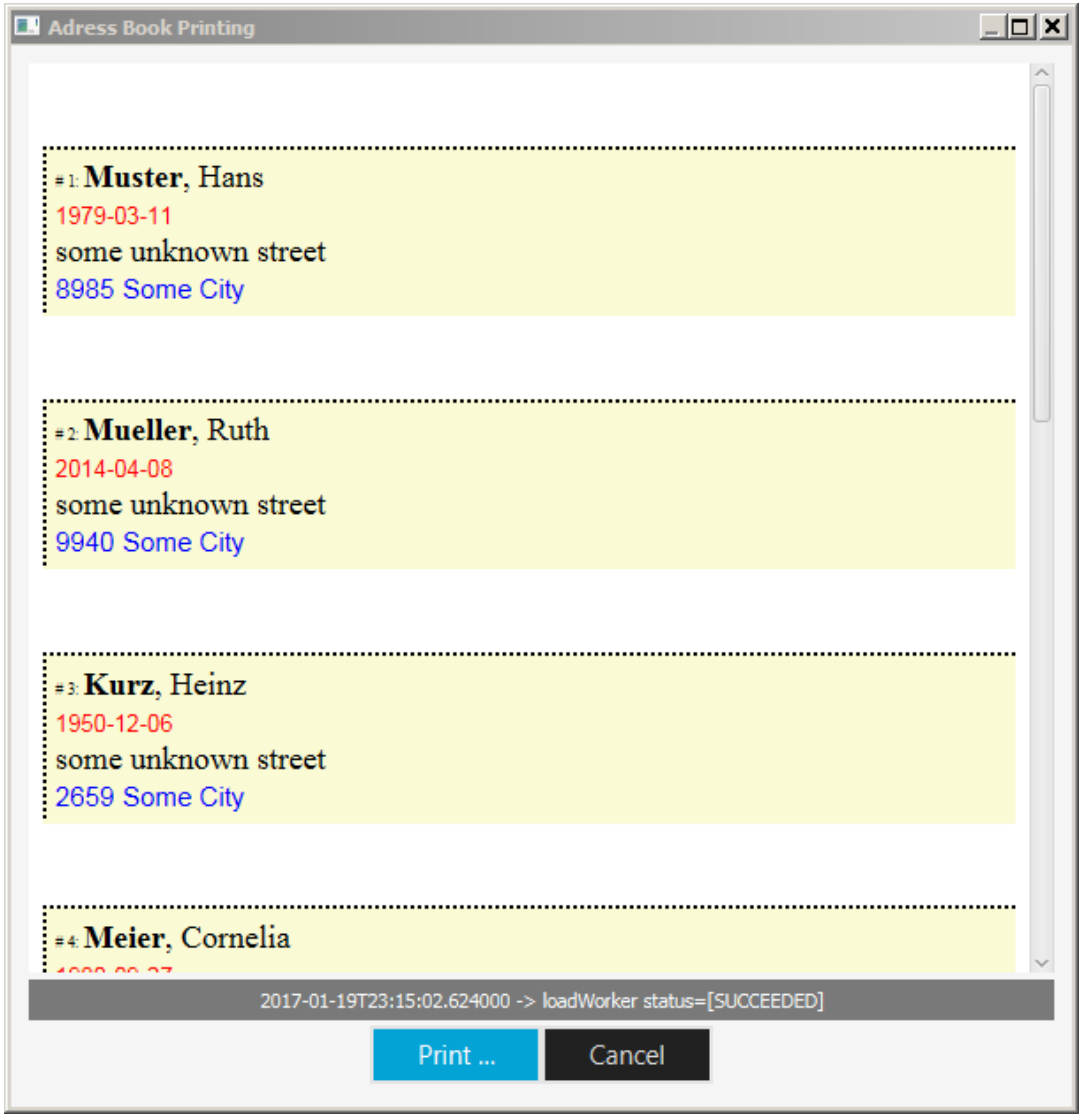
An Address Book Application with JavaFX, 5 Example 3, Screenshots 2/4



An Address Book Application with JavaFX, 5 Example 3, Screenshots 3/4



An Address Book Application with JavaFX, 5 Example 3, Screenshots 4/4



Roundup and Outlook

- 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! :)