

Anatomy of a GUI (Graphical User Interface)

Rony G. Flatscher, WU
2018 International Rexx Symposium

Overview

- Interaction with users
- Windows dependent interactions
- Platform independent interactions
 - [BSF4ooRexx](#)
 - Platform independent class [BSF.Dialog](#)
- Anatomy of a GUI
 - Using [awt](#) and [swing](#)
 - Using [JavaFX](#)
- Roundup and outlook

Interaction with Users, 1

- From time to time input from users is needed
 - Fetch input data to be processed
 - Display and allow input for desired choices
- Sometimes the user needs to be informed
 - About conditions that have occurred
 - Progress a longer running function takes
- Command-line Rexx
 - **SAY** statements for outputs
 - **PARSE PULL** statements for inputs

Interaction with Users, 2

- Rexx programs that use **SAY** and **PARSE PULL** statements execute in a single thread
- Getting input halts execution of the Rexx program until the user pressed enter
 - No need to synchronize the Rexx program with the user input

Windows Dependent Interactions

- **ooDialog**
 - Windows only GUI solution
 - Originally with a development environment
 - Included a user interface builder
- **Open Object Rexx version**
 - Without development environment
 - For legal reasons IBM could not open source it
 - Instead manually creating Windows resource files
 - Possible to use some resource editor programs
 - Outdated, no active development, yet still feasible

Platform Independent Interactions

- [BSF4ooRexx](#)
 - ooRexx Java bridge
 - Java: *"compile once, run everywhere"*
 - Truly enables platform independence
 - BSF4ooRexx exists for Windows, Linux, MacOSX, AIX, s390x
 - All BSF4ooRexx samples run unchanged on all those platforms
 - Opens access to Java GUI classes and infrastructure!
 - [java.awt](#) (abstract windows toolkit) package
 - [javax.swing](#) package
 - [javafx](#) package plus [SceneBuilder](#) to create GUIs interactively

BSF.CLS – Class **BSF.DIALOG**, 1

- Class **BSF.DIALOG** defines class methods to create *blocking* popup windows on all operating systems
 - **messageBox**
 - Informs the user, can be also a warning or error
 - **dialogBox**
 - Allows the user to chose which button to click
 - **inputBox**
 - Allows the user to supply input to the program
 - Waits until user pressed a button or picked a choice
 - Comparable to using **PARSE PULL**, just much more versatile!
 - Cf. [samples/1-020_demo.BSF.dialog.rxd](#)

BSF.CLS – Class **BSF.DIALOG**, 2

```
say "Demonstrating .bsf.dialog~messageBox(...):"
/* arguments: message, title, messageType */
.bsf.dialog~messageBox("This is an informal message")
.bsf.dialog~messageBox("This is an informal message", "A title text")
.bsf.dialog~messageBox("This is an informal message", "A title text", "info")
.bsf.dialog~messageBox("This is an error message", "A title text", "error")
say "---"

say "Demonstrating .bsf.dialog~dialogBox(...):"
/* arguments: message, title, messageType, optionType, icon, textOfButtons, defaultButton */
res=.bsf.dialog~dialogBox("Shall we delete?", , "question", "YesNoCancel")
say "dialogBox: you picked button #" res

txtButtons=.list~of("Tickle Alice", "Tickle Berta", "Tickle Cindy")
defButton ="Tickle Berta"
res=.bsf.dialog~dialogBox("Please pick a button", , "question", , , txtButtons, defButton)
say "dialogBox: you picked button #" res
say "---"

say "Demonstrating .bsf.dialog~inputBox(...):"
/* arguments: message, title, messageType, icon, textOfOptions, defaultValue */
res=.bsf.dialog~inputBox("Enter something!")
say "inputBox: you entered" pp(res)

txtOptions=.list~of("Tickle Alice", "Tickle Berta", "Tickle Cindy")
defaultTxtOption="Tickle Berta"
res=.bsf.dialog~inputBox("Pick something!", "Choice Dialog", "plain", , txtOptions,
defaultTxtOption)
say "inputBox: you picked" pp(res)

::requires BSF.CLS
```


Anatomy of a GUI, 1

- If a GUI element gets created, then
 - The GUI subsystem creates an own "GUI Thread" !
 - Interaction with GUI elements/objects is only allowed on the "GUI Thread"
 - Otherwise the GUI hangs, the program blocks!
 - The user cannot interact with the program anymore!
- Usually
 - One supplies a callback method that will be invoked on the "GUI Thread"
 - Then it is safe to interact with all GUI elements

Anatomy of a GUI, 2

- Interacting with GUI elements/objects from another Rexx thread
 - Usually a service function/method from the GUI management is needed to be used instead
 - One needs to register the need for a callback on the GUI thread
 - The next time the GUI thread is used by the GUI management the registered callbacks get carried out on the "GUI Thread"

Anatomy of a GUI, 3

- Graphical subsystems in operating systems
 - Windows
 - Linux
 - MacOSX
- Programming environment with "GUI Thread"
 - Windows GUIs including ooDialog
- Java packages available on all operating systems
 - `java.awt`, `javax.swing` and `javafx` GUIs

GUI With Synchronisation Needs, 1

- If "**GUI Thread**" totally independent of others
 - Need to synchronize with "**GUI Thread**"!
 - Otherwise the Rexx program ends, tearing down the GUI
 - Java packages [java.awt](#), [javax.swing](#)
- ooRexx multi-threading to the rescue! ;)
 - Setup the GUI
 - User will become able to interact immediately
 - Block the main Rexx program by calling a blocking method after setup, waiting for the GUI to close
 - Define a callback for the GUI event that indicates that Window closes, that releases the blocked method
 - Blocked main Rexx program will be able to continue its work

GUI With Synchronisation Needs, 2

- A simple "helloWorld.rxj" example
 - Creates a window with a title (a "Frame")
 - Closing it should end the Rexx program via a callback
 - Creates a button with a text
 - Clicking it should end the Rexx program via a callback
 - After creating the GUI and displaying the frame
 - The Rexx program waits/blocks until the frame gets closed or the button clicked
 - There is an ooRexx class defined that will
 - Allow blocking
 - Defines the necessary callback methods

GUI With Synchronisation Needs, 3a

*-- The Rexx class implements blocking and the methods for the Java callbacks
-- "actionPerformed" (ActionListener) and "windowClosing" (WindowListener)*

```
::class RexxCloseAppEventHandler
```

```
::method init                -- Rexx constructor method  
  expose lock  
  lock=.true                -- if set to .false, then release block
```

```
::method waitForExit        -- method blocks until attribute is set to .true  
  expose lock  
  guard on when lock=.false -- clever ooRexx way to block! :)
```

```
::method actionPerformed    -- event method (from ActionListener)  
  expose lock  
  lock=.false              -- indicate that the app should close
```

```
::method unknown            -- intercept unhandled events, do nothing
```

```
::method windowClosing      -- event method (from WindowListener)  
  expose lock  
  lock=.false              -- indicate that the app should close
```

GUI With Synchronisation Needs, 3b

```
-- create instance/value of our Rexx class  
rexCloseEH =.RexCloseAppEventHandler~new -- Rexx event handler
```

```
-- Create Java RexxProxy for the Rexx event handler  
javaCloseEH=BsfCreateRexxProxy(rexCloseEH, , - /* Rexx object to box */  
    "java.awt.event.ActionListener", - /* actionPerformed */  
    "java.awt.event.WindowListener" ) /* windowClosing */
```

```
-- create a Java awt window with a title  
window=.bsf~new("java.awt.Frame", 'Hello World!')  
window~addWindowListener(javaCloseEH) -- register event handler
```

```
-- create a Java awt window with a title  
button=.bsf~new("java.awt.Button", 'Press Me !')  
button~addActionListener(javaCloseEH) -- register event handler
```

```
-- prepare window and show it, using cascading messages (two twiddles '~')  
window ~~add(button) ~~pack ~~setSize(200,60) ~~setVisible(.true) ~~toFront
```

```
rexCloseEH~waitForExit -- blocks until user closes the Window (Frame)
```

```
::REQUIRES BSF.CLS -- get the Java support
```

GUI With Synchronisation Needs, 3c

```
-- create instance/value of our Rexx class
rexxCloseEH = .RexxCloseAppEventHandler~new -- Rexx event handler

-- Create Java RexxProxy for the Rexx event handler
javaCloseEH=BSfCreateRexxProxy(rexxCloseEH, , - /* Rexx object to box */
    "java.awt.event.ActionListener", - /* actionPerformed */
    "java.awt.event.WindowListener" ) /* windowClosing */

-- create a Java awt window with a title
window=.bsf~new("java.awt.Frame", 'Hello World!')
window~addWindowListener(javaCloseEH) -- register event handler

-- create a Java awt window with a title
button=.bsf~new("java.awt.Button", 'Press Me !')
button~addActionListener(javaCloseEH) -- register event handler

-- prepare window and show it, using cascading messages (two twiddles '~')
window ~~add(button) ~~pack ~~setSize(200,60) ~~setVisible(.true) ~~toFront

rexxCloseEH~waitForExit -- blocks until user closes the Window (Frame)

::REQUIRES BSF.CLS -- get the Java support

/* ----- */
-- The Rexx class implements blocking and the methods for the Java callbacks
-- "actionPerformed" (ActionListener) and "windowClosing" (WindowListener)
::class RexxCloseAppEventHandler

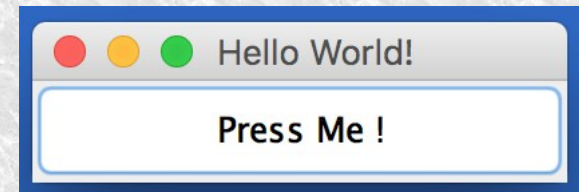
::method init -- Rexx constructor method
expose lock
lock=.true -- if set to .false, then release block

::method waitForExit -- method blocks until attribute is set to .true
expose lock
guard on when lock=.false -- clever ooRexx way to block! :)

::method actionPerformed -- event method (from ActionListener)
expose lock
lock=.false -- indicate that the app should close

::method unknown -- intercept unhandled events, do nothing

::method windowClosing -- event method (from WindowListener)
expose lock
lock=.false -- indicate that the app should close
```



GUI Without Synchronisation Needs, 1

- JavaFX
 - Creating GUI with *SceneBuilder*
 - GUI stored in FXML file
 - Creating a Rexx program
 - Need to extend/subclass [javafx.application.Application](#)
 - Implement its method [start](#)
 - Run the [launch](#) method
 - JavaFX will block that application object until the user closes the GUI!

GUI Without Synchronisation Needs, 2a

```
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 implements "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:")
  fxmUrl=.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(fxmUrl)

  scene=.bsf~new("javafx.scene.Scene", rootNode) -- create a scene
  primaryStage~setScene(scene) -- set the stage to our scene
  primaryStage~show -- show the stage (and thereby our scene)
```

GUI Without Synchronisation Needs, 2b

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

```
<?import javafx.scene.control.Button?>
```

```
<?import javafx.scene.control.Label?>
```

```
<?import javafx.scene.layout.AnchorPane?>
```

```
<!-- use the Java scripting engine named 'rexx' in this file -->
```

```
<?language rexx?>
```

```
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="400"
```

```
xmlns:fx="http://javafx.com/fxml/1">
```

```
<!-- Rexx buttonClicked callback -->
```

```
<fx:script source="FXML_01_controller.rex" />
```

```
<children>
```

```
<Button fx:id="idButton1" layoutX="170.0" layoutY="89.0"
```

```
onAction="slotDir=arg(arg()); 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>
```

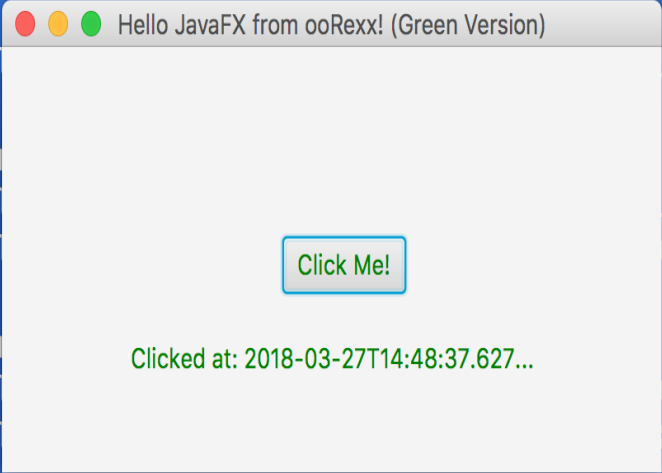
```
</AnchorPane>
```

GUI Without Synchronisation Needs, 2c

```
::routine buttonClicked public
  slotDir=arg(arg()) -- note: last argument is the slotDir argument from BSF4ooRexx
  now=.dateTime~new -- time of invocation
  say now": arrived in routine 'buttonClicked' ..."

/* @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
```

```
ronymac2014:code rony$ rexxj.sh fxml_01.rex
REXXout>2018-03-27T14:48:24.646971: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[]
REXXout>... new value of label=[C
REXXout>
REXXout>2018-03-27T14:48:28.443930: a
REXXout>... current value of label=[C
REXXout>... new value of label=[C
REXXout>
REXXout>2018-03-27T14:48:30.251728: a
REXXout>... current value of label=[C
REXXout>... new value of label=[C
REXXout>
REXXout>2018-03-27T14:48:37.627544: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[Clicked at: 2018-03-27T14:48:30.251728]
REXXout>... new value of label=[Clicked at: 2018-03-27T14:48:37.627544]
REXXout>
```



Updates to GUI From Another Thread, 1

- Possible to have Rexx threads in parallel
 - Long running operations
 - Need to give user feedback about progress
 - Desire to use the GUI to inform the user
 - Updating a GUI element from a Rexx thread
 - Hangs the GUI and as a result
 - Hangs the application for the user
 - Proper way
 - Inform the GUI to call back on the "GUI Thread" later
 - Depends on the GUI system one uses

Updates to GUI From Another Thread, 2

- JavaFX
 - Own GUI management
 - "GUI Thread" dubbed "JavaFX Application Thread"
 - Too long of a term
 - "javafx.application.Platform"
 - Class method `runLater(Runnable)`
 - Allows to have the Runnable code executed on the "GUI Thread" later
- Students, even skilled and informed were not able to properly use this class
 - Need to find a more "human centric" solution

Updates to GUI From Another Thread, 3

- BSF4ooRexx
 - Class `FXGuiThread` methods
 - `IsGuiThread`
 - Returns `.tru/.false`
 - `runLater(GUI_object, message, ...)`
 - Returns `GUIMessage` object
 - `runLaterLatest(GUI_object, message, ...)`
 - Returns `GUIMessage` object
 - Class `GUIMessage`
 - Modelled after ooRexx class `Message`
 - Can directly use its documentation
 - Possible to wait on message to have executed

Updates to GUI From Another Thread, 4

- [samples/JavaFX/fxml_06](#)
 - GUI progress bar sample
 - GUI progress indicator gets updated from a worker REXX thread
 - As the user may interrupt the REXX thread at any time via the GUI the worker thread needs to learn about it
 - Need to create a communication protocol!
 - Cf. class [Action](#) in [fxml_pb_controller.rex](#)
 - Communication via REXX can be done without problems between the "[GUI Thread](#)" and the REXX worker thread

Updates From REXX Worker Thread, 1

```
::requires "BSF.CLS"
```

```
::class Worker public
```

```
::method go
```

```
  use arg clzAction  -- get class object
```

```
  reply           -- return to caller, keep working on a separate thread
```

```
  fxml=.my.app~fxml_pb.fxml  -- get the corresponding FXML REXX directory
```

```
  pb                =fxml~idProgressBar
```

```
  lblCurrent=fxml~idLabelCurrent
```

```
  do i=1 to 100 while clzAction~state="running"
```

```
    -- update GUI controls on the "JavaFX Application Thread"
```

```
    d=box("Double",i/100)
```

```
    .FXGuiThread~runLaterLatest(pb, "setProgress", "individual", d)
```

```
    .FXGuiThread~runLaterLatest(lblCurrent, "setText", "indiv" , i "%")
```

```
    -- instead of sleeping, do the real work here!  <-- <-- <--
```

```
    call SysSleep 0.01  -- sleep 1/100 of a second
```

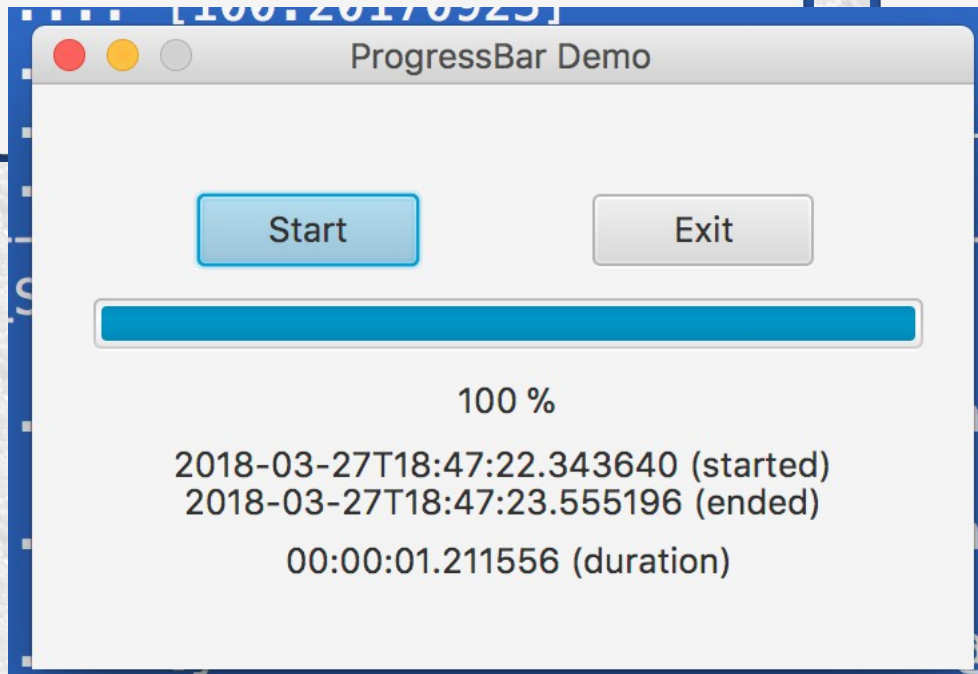
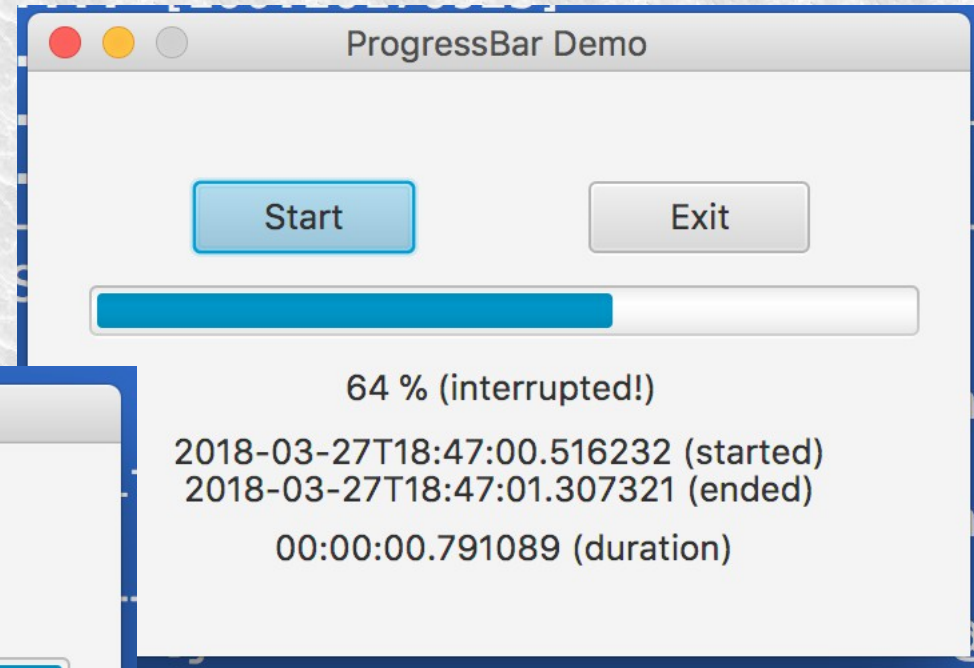
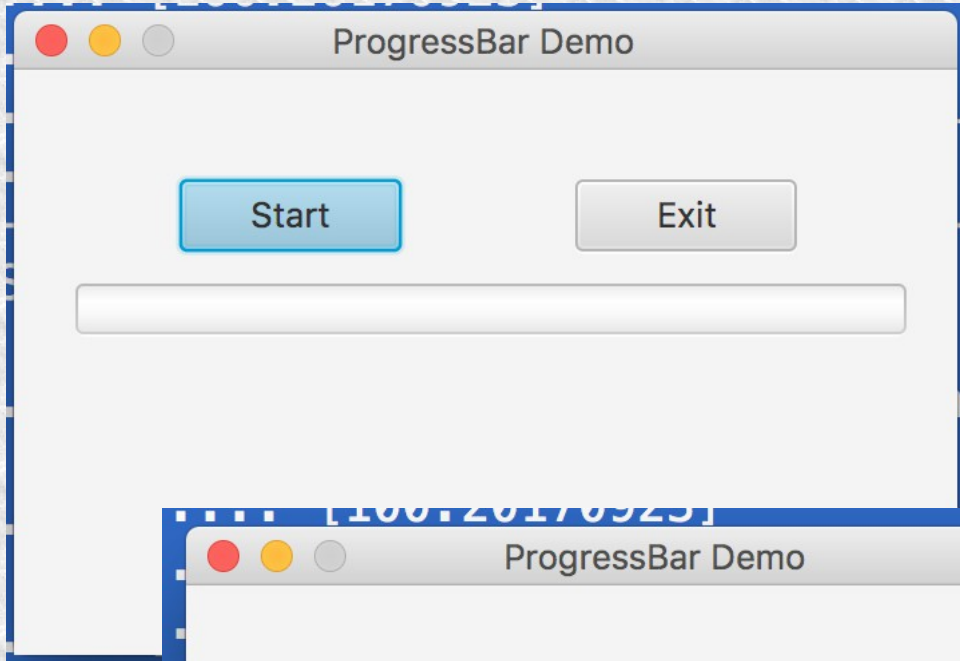
```
  end
```

```
    -- we need to send the message on the "JavaFX Application Thread"
```

```
  msg=.FXGuiThread~runLater(clzAction, "setIdle")
```

```
  res=msg~result  -- this blocks until message was executed
```

Updates From Rexx Worker Thread, 2



Roundup and Outlook

- Creating cross-operating system GUIs easy
- Possible problem
 - Interacting with GUI element from a non "GUI Thread"
 - Hangs the GUI, hangs the user interface!
 - Solution for JavaFX applications
 - REXX class `FXGuiThread`
 - Easy to use
 - Makes it easy to create bullet-proof REXX-GUI applications!