.NET for ooRexx
(ooreexx.net)

The 2016 International Rexx Symposium

Rony G. Flatscher
Agenda

• **.NET/CLR**
  – Brief overview

• **Overview of "oorexx.net"**
  – Based on BSF4ooRexx
  – jni4net

• **Examples**
  – ooRexx scripts using CLR classes as if they were ooRexx ones
  – .NET/CLR callbacks to ooRexx scripts

• **Roundup**
.NET/CLR

- "Microsoft's Java"

- CLR: Common Language Runtime
  - Bytecode format and runtime environment
  - Programming languages like C# ("Anti-Java"), VB.NET

- Alternative implementations
  - MONO
    - Opensource version (stripped down) of MS .NET

- .NET evolves to DNX
.NET/CLR, 2

• .NET/CLR
  – Assembly
    • A standalone .NET/CLR program
    • A .NET/CLR library
      – All exported .NET/CLR classes reside usually in such an assembly
  – GAC
    • Global assembly cache
      – Any signed (= strong named) .NET/CLR library
    • Globally available
Overview of "oorexx.net", 1

• "oorexx.net"
  – Bachelor thesis at WU Vienna
    • Created by a student, Manuel Raffel, fall 2015
    • Nutshell examples by a student, Adrian Baginski, summer 2016
      – Caused the presenter to rework part of the support (to ease and simplify it for the programmer)
  – Uses the opensource Java to .NET/CLR bridge named "jni4net"
    • As a result employs BSF4ooRexx, the ooRexx to Java bridge transparently
  – Supplies an ooRexx package named "CLR.CLS" that camouflages and supports .NET/CLR as ooRexx
Overview of "oorexx.net", 2

- **ooRexx package "CLR.CLS"**
  - Modelled after "BSF.CLS"
  - Requires "BSF.CLS"
    - All of BSF4ooRexx features become immediately available
  - Public class "CLR" serves as the ooRexx proxy class
    - Camouflages any .NET/CLR class as an ooRexx class
    - **NEW** method
      - First argument must be the fully qualified name of a .NET/CLR class
      - Optionally followed by all the arguments that the desired constructor defines
      - Returns an ooRexx proxy for the newly created .NET/CLR object
• ooRexx package "CLR.CLS"
  – Public class "CLR_Proxy"
    • Represents a .NET/CLR value (object) returned by .NET/CLR
    • Camouflages any .NET/CLR class as an ooRexx class
  – Public class "CLR_Enum"
    • Represents a .NET/CLR value of type System.Enum
    • Implements comparison methods that allow to compare System.Enum values with other System.Enum values and with Rexx strings carrying the name of the System.Enum value
Overview of "oorexx.net", 4

- **ooRexx package "CLR.CLS"**
  - Public class "CLR_Event"
    - Represents a .NET/CLR event object
      - Defines the method "+" for adding event handlers
      - Defines the method "-" for removing event handlers
  - Public class "CLR_Thread"
    - Meant to be subclassed by a Rexx class needing to execute concurrently .NET/CLR related code
    - Method "run" is abstract and must be implemented by the subclass
    - Method "start" creates and starts the thread and sends the "run" message to invoke the "run" method in the subclass
Overview of "oorexx.net", 5

• ooRexx package "CLR.CLS"
  
  – Public routine clr.createEventHandler(rexxObject,rexxData)
    
    • *rexxObject* is the Rexx object handling the event
    
    • The optional *rexxData* value, if present, will be added to the *slotDir* argument (see below) with an index named "USERDATA"
    
    • When the {.NET/CLR} event "invoke" is fired all arguments will be passed in the same order to the message sent to *rexxObject*
      
      – In addition a *slotDir* argument (a Rexx *Directory* object) will be always appended as the last argument before the message is sent to the ooRexx object
• ooRexx package "CLR.CLS"

  – Public routine `clr.import(className)`
    • Returns an ooRexx proxy class for the fully qualified .NET/CLR `className`
      – Allows access to static members (like fields, properties, methods)
      – Gets a NEW method defined that allows the immediate creation of proxy objects of that imported class

  – Public routine `clr.addAssembly(assemblyName)`
    • Queries and remembers all exported public classes from `assemblyName`
    • Needed for assemblies not available to the runtime by default
Overview of "oorexx.net", 7

- **ooRexx package "CLR.CLS"**
  - Public routine `clr.box(type, value)`
    - Uses *value* to create an instance of the .NET/CLR wrapper class representing it
    - *type* can be one of
      - `SString (System.String)`, `BBoolean (System.Boolean)`, `BByte (System.Byte)`, `SByte (System.SByte)`, `Char (System.Char)`, `DDec (System.Decimal)`, `DDouble (System.Double)`, `INT16 (System.Int16)`, `UINT16 (System.UInt16)`, `INT32 (System.Int32)`, `UINT32 (System.UInt32)`, `INT64 (System.Int64)`, `UINT64 (System.UInt64)`, `SSingle (System.Single)`
    - *value* is any Rexx string representable as the given *type*
overview of "oo rexx.net", 8

• ooRexx package "CLR.CLS"
  – Public routine clr.wrap(value)
    • Uses value to create an instance of the .NET/CLR wrapper class representing it
      – If a whole number (under NUMERIC DIGITS 29) the routine returns
        - A 32-Bit (System.Int32), a 64-Bit (System.Int64) or a Decimal (System.Decimal) .NET/CLR object (using clr.box())
      – If a Rexx string it returns a boxed .NET/CLR value (a System.String)
      – If a CLR proxy object
        - If a System.Enum value, but not wrapped as a CLR_Enum proxy, then creates and returns a CLR_Enum proxy object
        - Returns the CLR proxy object unchanged
Overview of "oorexx.net", 9

- **ooRexx package "CLR.CLS"**
  - Public routine `clr.unbox(value)`
    - Converts primitive `.NET/CLR` values and `.NET/CLR` strings into Rexx strings and returns them
    - All other values are returned unchanged
  - Public routine `clr.createArray(typeName,capacity)`
    - Uses the class `System.Array` to create a `.NET/CLR` array object of `typeName` and with the given `capacity`
Example "HelloWorld" Using Java, 1

- Loads the Java class "java.lang.System"
- Fetches its "out" field and uses its "println" method to output the text to stdout
Example "HelloWorld" Using Java, 2

```java
system = bsf.import("java.lang.System")
system~out~println("Hello World from Java (via BSF4ooRexx)"

::requires BSF.CLS
```

**Output:**

Hello World from Java (via BSF4ooRexx)
Example "HelloWorld" Using .NET, 1

- Loads the .NET class "System.Console"
- Uses its static method "WriteLine" to output the text to stdout
Example "HelloWorld" Using .NET, 2

```exx
console = clr.import("System.Console")
console~WriteLine("Hello World from ooRexx.NET (via BSF4ooRexx and jni4net)")
```

::REQUIRES CLR.CLS -- get ooRexx.NET support

Output:

```
Hello World from ooRexx.NET (via BSF4ooRexx and jni4net)
```
Example "SystemSounds" Using .NET, 1

- Loads the .NET class "System.Media.SystemSounds"
- Uses its static sound properties and plays them
Example "SystemSounds" Using .NET, 2

```rexx
sounds = clr.import("System.Media.SystemSounds")
console = clr.import("System.Console")

console~ WriteLine("SystemSounds demonstration starting")
CALL SysSleep .5 -- wait for 500 ms

console~ WriteLine("playing 'Beep'")
sounds~Beep~Play
CALL SysSleep 1

SAY "playing 'Asterisk'")
sounds~Asterisk~Play
CALL SysSleep 1

SAY "playing 'Exclamation'"
sounds~Exclamation~Play
CALL SysSleep 1

SAY "playing 'Hand'"
sounds~Hand~Play
CALL SysSleep 1

SAY "the last one is called 'Question'"
sounds~Question~Play
CALL SysSleep 1

::REQUIRES CLR.CLS -- get ooRexx.NET (CLR, common language runtime) support
```

Outputs sound together with that sound's text on the console:

```
SystemSounds demonstration starting
playing 'Beep'
playing 'Asterisk'
playing 'Exclamation'
playing 'Hand'
the last one is called 'Question'
```
Example "MessageBox" Using .NET, 1

- Loads the .NET class "System.Windows.Forms.MessageBox"
- Uses its static method "Show" to display a messagebox with a supplied title and text value
Example "**MessageBox**" Using **.NET**, 2

```plaintext
text = "This is my Text"       -- define some text
title = "Title of MessageBox"  -- define some title
MessageBox = clr.import("System.Windows.Forms.MessageBox")
MessageBox~Show(text, title)   -- start method "show" with two arguments: text and title

::REQUIRES CLR.CLS             -- get ooRexx.NET (CLR, common language runtime) support
```

![MessageBox dialog window](image)
Example "Client/Server", "Server", 1

- Creates a server socket that listens to port 2015 on localhost (127.0.0.1) for client connections
- Reads the bytes sent from client and creates an UTF-8 encoded string off them
Example "Client/Server", "Server", 2

-- create "System.Net.IPAddress" object representing the localhost IP
ipAddress = clr.import("System.Net.IPAddress") ~ Parse("127.0.0.1")

-- create instance of class "System.Net.Sockets.TcpListener"

SAY "Starting server..."
tcpListener~clr.dispatch("Start") -- start the TCP listener
SAY "Waiting for connections..."
tcpSocket = tcpListener~AcceptSocket -- wait for connections
SAY "Client connected."

buffer = clr.createArray("System.Byte", 1024) -- create array of "System.Byte"
count = tcpSocket~Receive(buffer) -- write received data into "buffer"

SAY "Message received:"
   -- convert UTF-8 encoded message from byte array to string
decodedMessage = clr.import("System.Text.Encoding") ~ UTF8~GetString(buffer, 0, count)
SAY pp(decodedMessage)

tcpListener~Stop -- stop TCP listener

::REQUIRES CLR.CLS -- get ooRexx.NET support
Example "Client/Server", "Client", 1

- Connects to server on port 2015 on localhost (127.0.0.1)
- Gets the message to send from the user
- Encodes the message as UTF-8 and sends it to the server
Example "Client/Server", "Client", 2

tcpClient = .clr~new("System.Net.Sockets.TcpClient")    -- create instance of class

SAY "Connecting to 127.0.0.1:2015..."
tcpClient~Connect("127.0.0.1", 2015)
SAY "Connected."

SAY "Input message to server:"
PARSE PULL message    -- fetch message from user
    -- encode message as UTF8 and return a byte array representing it
encodedMessage = clr.import("System.Text.Encoding")~UTF8~GetBytes(message)

SAY "Sending message..."
tcpClient~GetStream~Write(encodedMessage, 0, encodedMessage~Length)    -- send to server
SAY "Message was sent to server."

tcpClient~Close

::REQUIRES CLR.CLS    -- get ooRexx.NET support
Example "Client/Server", Output

Output:

<server> Starting server...
<server> Waiting for connections...

    <client> Connecting to 127.0.0.1:2015...

<server> Client connected.

    <client> Connected.
    <client> Input message to server:
    <client> Über den Wölkchen ... (äöüÄÖÜß)
    <client> Sending message...
    <client> Message was sent to server.

<server> Message received:
<server> [Über den Wölkchen ... (äöüÄÖÜß)]
Example "ProgressBar", 1

- Creates a `System.Windows.Forms.Form` consisting of
  - A `System.Windows.Forms.FlowLayoutPanel`,
  - A `System.Windows.Forms.ProgressBar` and

- Defines an `ooRexx` class `MouseEventHandler` for processing events
  - Reacts upon pressing of the button which will cause the message `invoke` to be sent ("fired off") to it
  - Method `invoke` creates an instance of `Processor` (a subclass of `CLRThread`), sends it `start` which will send the `run` message
Example "ProgressBar", 2

```
winForm = .clr~new("System.Windows.Forms.Form")  -- create instance
winForm~Text = "Processor"  -- set property "Text" to string "Processor"
winForm~AutoSize = .true  -- set property "AutoSize" to boolean true
winForm~AutoSizeMode = GrowAndShrink  -- set property to enum value "GrowAndShrink"

contentPane = .clr~new("System.Windows.Forms.FlowLayoutPanel")  -- create instance
contentPane~AutoSize = .true  -- set property "AutoSize" to boolean true
contentPane~AutoSizeMode = GrowAndShrink  -- set property to enum value "GrowAndShrink"
winForm~Controls~Add(contentPane)  -- add "FlowLayoutPanel" to "Form"

progressBar = .clr~new("System.Windows.Forms.ProgressBar")
progressBar~Minimum = 0  -- set property "Minimum" to integer 0
progressBar~Maximum = 100  -- set property "Maximum" to integer 100
progressBar~Value = 0  -- set property "Value" to integer 0
contentPane~Controls~Add(progressBar)  -- add "ProgressBar" to "FlowLayoutPanel"

startButton~Text = "Start"  -- set property "Text" to string "Start"
contentPane~Controls~Add(startButton)  -- add "Button" to "FlowLayoutPanel"

-- create new event handler from the ooRexx class "MouseEventHandler" below
mouseEventHandler = clr.createEventHandler(.MouseEventHandler~new(progressBar, startButton))
startButton~Click += mouseEventHandler  -- register event handler to "Click" event

-- import "System.Windows.Forms.Application" class/type and use its static method "Run"
application = clr.import("System.Windows.Forms.Application")
application~Run(winForm)  -- invoke method "Run", which starts an application message loop

::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

... continued on next page ...
/* mouse event handler, will be invoked by the "Click" event */
::CLASS MouseEventHandler
  ::METHOD init -- constructor which saves the received objects in attributes
    EXPOSE progressBar startButton
    USE ARG progressBar, startButton

  ::METHOD invoke -- will get invoked by .NET when event gets triggered
    EXPOSE progressBar startButton
    USE ARG caller, mouseEventArgs

    -- creates a new instance of ooRexx class "Processor", which inherits the "start"
    -- method from its superclass "CLRThread" (defined in CLR.CLS), which creates a thread
    -- in which the "run" method gets executed
    .Processor~new(progressBar, startButton)~start

    ... continued on next page ...
Example "ProgressBar", 4

/* class that inherits from CLRThread (defined in CLR.CLS), its "run" method will be called
from a new thread that CLRThread creates when it receives the "start" message */
::CLASS Processor SUBCLASS CLRThread

::METHOD init -- constructor which saves the received objects in attributes
    EXPOSE progressBar startButton
    USE ARG progressBar, startButton

::METHOD run -- will be invoked from superclass' "start" method
    EXPOSE progressBar startButton

    startButton~Enabled = .false -- disable start button to prevent multiple clicks

    DO i = 1 TO 100
        progressBar~Value = i -- set value of the progress bar (from 1 to 100)
        startButton~Text = i"%"
        CALL syssleep .1 -- sleep 100 milliseconds to prevent reaching 100 immediately
    END

    startButton~Text = "Finished" -- set text on button to "Finished"
Example "**ProgressBar**", Output

![ProgressBar](image1)

![ProgressBar](image2)

![ProgressBar](image3)
Roundup and Outlook

**Roundup**

- "oorexx.net" (ie. CLR.CLS) camouflages .NET/CLR as ooRexx
- Straight-forward usage of .NET/CLR classes on Windows
  - Adds a missing link to ooRexx on Windows!
- All of BSF4ooRexx/Java is available
  - Java and .NET/CLR can be mixed, if necessary
- BSF4ooRexx comes with these (and more) .NET/CLR samples
- Once "jni4net" supports MONO and/or Microsoft's opensource .NET/CLR, BSF4ooRexx will become able to support both on all operating systems