Vienna University of Economics and Business

Bachelor Thesis

A .NET Cookbook using ooRexx.NET

Author

Adrian Baginski

supervised by
ao. Univ. Prof. Mag. Dr. Rony G. Flatscher

July 23, 2016

Table of contents

I Introduction

OoRexx.NET is an approach for bridging the .NET framework and Open Object Rexx. It was developed by student Manuel Raffel of the University of Economy Vienna. To accomplish that, BSF4ooRexx is used to get access to Java classes. It then camouflages Rexx classes as if they were .NET classes so it is possible to use the entire framework with all of its advantages. Open Object Rexx is an object-oriented programming language based on Rexx, which was invented in 1979 by IBM.
This bachelor thesis is about exploring the many attributes, methods, classes, namespaces and assemblies from the .NET framework and implement it in the dynamically typed, caseless and easy-to-use ooRexx. It includes both basic, easy to understand examples and more advanced ones which might take a while for the non-experienced reader to understand. Although the author tried to keep the application code well-commentated and therefore well-documented, it is a good approach to first see the outcome of each application before looking at the code. To achieve that the reader could use the "viewAll.rxj" file included in the directory. It launches all applications one-by-one with the "rexxpaws.exe" program, which pauses after each application, so that the reader can choose when to start the next one. After exiting the last application, it will delete all temporary files created in the process to keep the directory clean.

1 Requirements

In order to use this Cookbook examples or create ooRexx.NET applications on your own, the reader will need to have the following programs installed on your computer:

- Open Object Rexx, current version
- BSF4ooRexx, current version
- Java, Version 1.5 or higher
- Microsoft .NET Framework, Version 4 or higher
- Jni4net

Be aware, that installing those components with different bitnesses (32 or 64 bit) might eventually cause problems.

2 Step-by-step ooRexx.NET Installation Guide

This setup guide will help the reader to install all necessary components of ooRexx.NET and its prerequisites. It can be assumed that the reader already has access to the zipped ooRexx.NET files. They will be distributed alongside BSF4ooRexx version 4.52. in fall 2016. If the reader has some parts of this installation guide already installed and ready to go on your computer, the reader can simply skip them. It is recommended though to always use the current versions due to security patches and function extensions - so better have a look at the used version of already installed components before skipping.

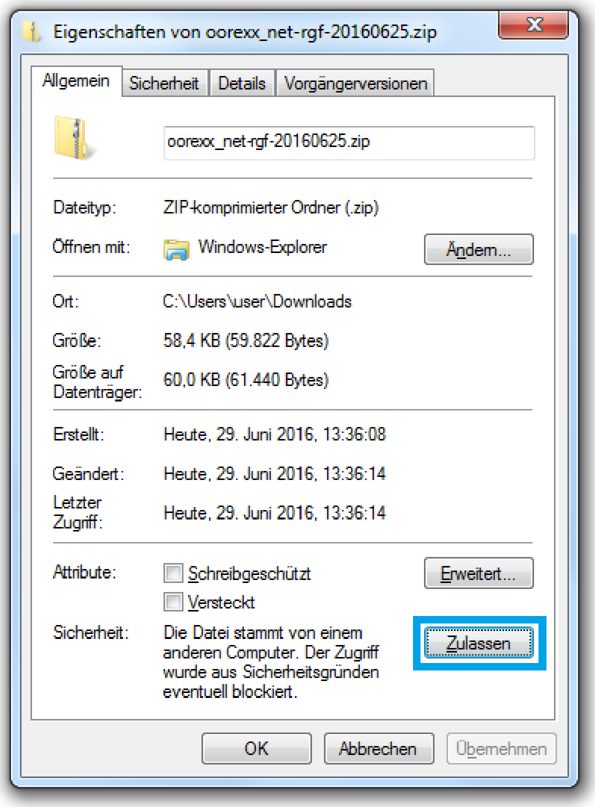
1. Download and install **ooRexx**. A list of all available versions can be found via the following link: `https://sourceforge.net/projects/oorexx/files/oorexx/`.

2. Download and install **Java**. Most systems already have Java installed because it is so popular, but take care that the reader has at least version 1.5 installed. the reader can get it from that the Oracle page directly: `https://www.java.com/de/download/`.

3. Download and install the **.NET framework** from the Microsoft website. Here is a link to the web installer: `https://www.microsoft.com/de-at/download/details.aspx?id=30653`.

4. Download and install **BSF4ooRexx**. This is the link to the latest version:
`https://sourceforge.net/projects/bsf4oorexx/files/latest/`.
Use the installer provided in the following path:
"install/windows/install.cmd".

5. Go into the file settings of the **ooRexx.NET zip archive**. At the bottom of the "general" tab the reader will find a button, which marks this archive and all of its components safe and unblocks all of its contents. See Figure 1 for a screenshot of that form. After doing so the reader can unzip that zip archive and start **install_clr_support.cmd**. This executable Windows Batch file will add jni4net.n-0.8.8.0.dll and oorexx.net .dll to the Global Assembly Cache (GAC) [1]. Both files are needed for ooRexx.NET to bridge the .NET framework using Java. DLL stands for "dynamic-link library" and is a collection of .NET classes.



Eigenschaften von oorexx_net-rgf-20160625.zip

Allgemein | Sicherheit | Details | Vorgängerversionen

oorexx_net-rgf-20160625.zip

Dateityp: ZIP-komprimierter Ordner (.zip)

Öffnen mit: Windows-Explorer

Ort: C:\Users\user\Downloads

Größe: 58,4 KB (59.822 Bytes)

Größe auf Datenträger: 60,0 KB (61.440 Bytes)

Erstellt: Heute, 29. Juni 2016, 13:36:08

Geändert: Heute, 29. Juni 2016, 13:36:14

Letzter Zugriff: Heute, 29. Juni 2016, 13:36:14

Attribute: ☐ Schreibgeschützt ☐ Versteckt

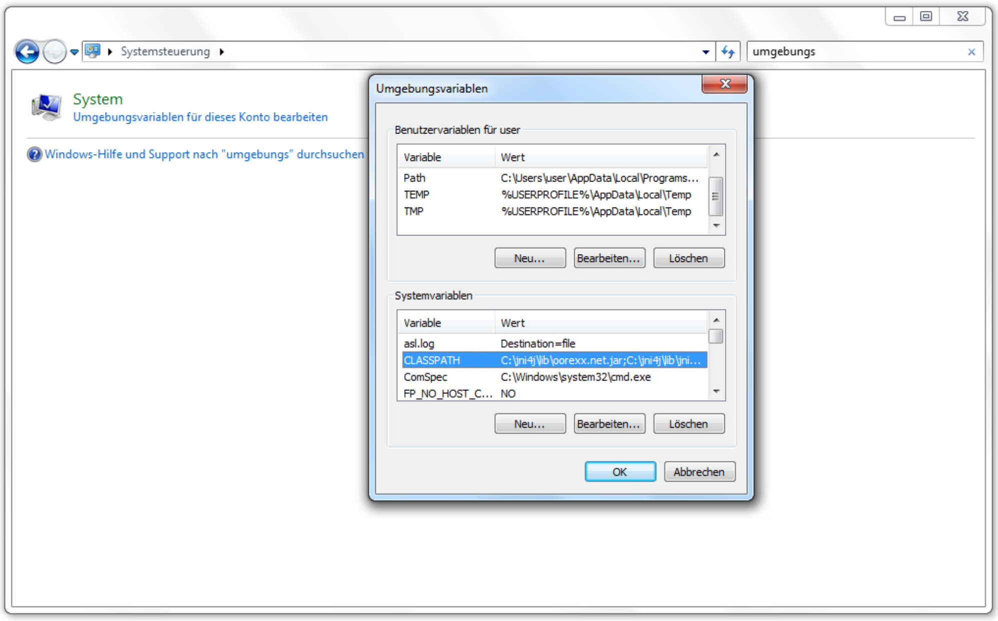
Sicherheit: Die Datei stammt von einem anderen Computer. Der Zugriff wurde aus Sicherheitsgründen eventuell blockiert.

Figure 1: File settings of ooRexx.NET.zip. Activate this button at the bottom of this form to mark the zip archive as safe, which allows the installer to work properly. In an English Version of Windows, the button is labelled *"Unblock"*.

6. **CLR.CLS**: This is an Open Object Rexx file containing all ooRexx .NET classes and its members, as well as several public routines. It also requires BSF.CLS itself, making it possible to use both .NET and Java classes with just one `::REQUIRES` call. For this directive to find this file in your applications, simply copy CLR.CLS into the BSF4ooRexx directory.

7. **jni4net.j-0.8.8.0.jar**: It is important to point the environmental variable "CLASSPATH" to this JAR file directly. To access this variable, use Windows' Command Prompt with the following command: *"SET CLASSPATH=%CLASSPATH%;[absolute path to the jar file];"*. It will maintain the old value of the PATH variable and add a new entry. Overriding some environmental variables can cause system stability issues. Sometimes a system reboot is needed in order to finish the

variable changing process. If the environmental variables do not save after closing the cmd.exe application, there might be a problem with access control. In that case, open the System Control Panel and search for environmental variables. Using that form, the reader can define and change both environmental variables for the current user and for the system.


Systemsteuerung
umgebungs
System
Umgebungsvariablen für dieses Konto bearbeiten
Windows-Hilfe und Support nach "umgebungs" durchsuchen
Umgebungsvariablen
Neu...
Bearbeiten...
Löschen
OK
Abbrechen

Benutzervariablen für user

Variable	Wert
Path	C:\Users\user\AppData\Local\Programs...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Systemvariablen

Variable	Wert
asl.log	Destination=file
CLASSPATH	C:\jni4j\lib\oorexx.net.jar;C:\jni4j\lib\jni...
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO

Figure 2: changing the environmental variables in Windows' System Control Panel

8. **oorexx.net.jar**: Finally, include this JAR file into the classpath variable as described in the previous step. [2, 3]

3 ooRexx.NET in a Nutshell

Here are the 5 most important things to know about ooRexx.NET programming in a nutshell:

1. CLR.CLS is the only needed ooRexx file to access the .NET framework. Include it in all your applications with the `::REQUIRES` directive of ooRexx. Keep in mind, that if problems occur there is always the possibility to have a look at the CLR.CLS code and maybe find a solution there. It is well-structured and also well-documented with a lot of comments to have a better understanding of the code.

2. Use `.clr~new(CLASSNAME, argument1, argument2, ...)` to instantiate a class with constructors. If this class has one or more constructors, it is possible to get an instance of it. The easiest and definitely most popular constructor has no arguments, so that is how it looks like in the .NET documentation: CLASSNAME().

3. Use `clr.import(CLASSNAME)` to import a static class, which has no constructors and therefore does not accept any input variables when instantiating it. Additionally, this routine offers to import normal classes as well and save it into an ooRexx class for future use, as this coding example shows:

```
CALL clr.import "System.IO.StreamWriter", "netStreamWriter" -- import class, save it in
    local as "netStreamWriter"

file = .netStreamWriter~new("tmp.txt") -- create a streamwriter object
file~write("hello, .net world!")
file~close

::REQUIRES CLR.CLS
```

Listing 1: Using the `clr.import` routine to import a .NET class and use it as if it was an ooRexx class

   This is especially useful for classes which are used multiple times in an application to make it more efficient. The `clr.import` routine imports the class stated in the first argument into the second argument. From that point it can be used like a regular ooRexx class.

4. Treat enumerations like static classes. Treat enumeration values as if they were attributes.

5. Use Microsoft's Developer Network (MSDN, `https://msdn.microsoft.com/en-us/library/`) effectively to find what the reader is searching

for. Microsoft's programming languages include C#, F#, Visual Basic, ASP.NET and many more. They have built a great community and are very likely to have already stumbled upon a problem we ooRexx programmers can use to try and solve our own.

For further information on ooRexx.NET have a look at [4].

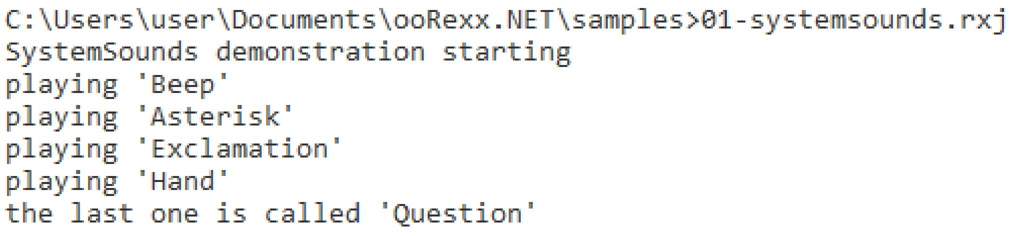
II Coding Examples

1 01-systemsounds.rxj

This application shows how to get access to the static .NET class "System-Sounds" and how to use it properly.

```
/* File:        01-systemsounds.rxj
 * Description: This application demonstrates all available SystemSounds
 * Shows:
 *              - calling the Rexx Method "SysSleep" to do nothing for a specific amount
 *                of seconds
 *              - Access to static .NET classes
 *              - Console Output via .NET
 *              - Using SystemSounds class to play specific system sounds
 *              - Including ooRexx.NET with the REQUIRES directive
 */

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

console~WriteLine("SystemSounds demonstration starting") -- the method "WriteLine" is
    basically identical to ooRexx' SAY command. If no line feed is needed after the
    output, you should use "Write" instead of "WriteLine".
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 support
```

Listing 2: Using SystemSounds and Console in ooRexx.NET



```
C:\Users\user\Documents\ooRexx.NET\samples>01-systemsounds.rxj
SystemSounds demonstration starting
playing 'Beep'
playing 'Asterisk'
playing 'Exclamation'
playing 'Hand'
the last one is called 'Question'
```

Figure 3: Output of 01-systemsounds.rxj

`"System.Media.SystemSounds"` is a static class and cannot be instantiated. Line number 12 shows the public routine `clr.import()` from CLR.CLS to import a static class by giving the fully qualified name of the class as an argument. `Beep`, `Asterisk`, `Exclamation`, `Hand` and `Question` are static properties of the class "SystemSounds". Accessing them will get the sound associated to the corresponding program event in the current Windows sound scheme. Note that some sound schemes do not have any sounds specified for those events, so it might be better to use other sound classes with fixed .mp3 sounds - or to use system sounds as an additional source of confirmation for user behaviour.

[5, 6]

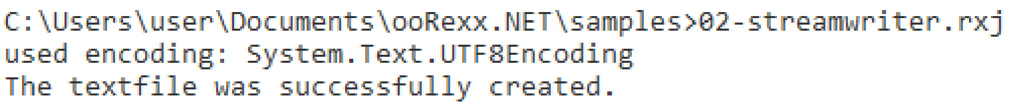
The output of this application is very straight forward and can be seen in Figure 3.

2 02-streamwriter.rxj

This application shows the usage of a file writer in ooRexx.NET.

```
/* File:        02-streamwriter.rxj
 * Description: This application creates a new textfile in the same directory and writes some text into it
 * Shows:
 *              - Usage of static System.IO.Directory class to get the path to the
                  present working directory using .NET support
 *              - Instantiating a .NET class
 *              - Creating a new textfile
 *              - Writing into that textfile
 *              - Getting the current encoding which can be changed if needed
 *              - Closing a filestream
 */


directory = clr.import("System.IO.Directory") -- get access to static class System.IO.Directory
filedirectory = directory~GetCurrentDirectory -- this method returns the path of the current directory of this application. This assures, that this application works on each and every system, regardless of where the files are saved.
-- alternatively, use rexx's built-in function directory() to get the current directory
filename = "02-textfile.txt"
filepath = filedirectory || .file~separator || filename  -- .file~separator: on Windows "\", on Unix "/"


file = .clr~new("System.IO.StreamWriter", filepath) -- get an instance of the public class "System.IO.StreamWriter"

file~WriteLine("~~First Heading~~")
file~WriteLine
file~WriteLine("This ooRexx.NET application will create a new textfile in given file path if it does not exist already.")

encoding = file~Encoding -- get access to enconding of the file, which will be most likely UTF8
SAY "used encoding:" encoding~toString

file~Close               -- close the opened file to free some memory
SAY "The textfile was successfully created."


::REQUIRES CLR.CLS
```

Listing 3: Writing and creating files in ooRexx.NET



```
C:\Users\user\Documents\ooRexx.NET\samples>02-streamwriter.rxj
used encoding: System.Text.UTF8Encoding
The textfile was successfully created.
```

Figure 4: Output of 02-streamwriter.rex

The public class "System.IO.StreamWriter" has many constructors like

an already open stream or the choice of an encoding, although in most cases a filepath will be sufficient. Lines 14 and 15 generate the absolute path of the current directory of this application using the .NET class `"System.IO.Directory"`. Alternatively, ooRexx provides the `directory()` routine. The output of variable "filepath" defined in Line 18 is dynamic and will look differently on every system, depending on the location of the application. In my case, this is how the variable looks like:
"C:\Users\user\Documents\ooRexx.NET\samples\02-textfile.txt"
Note that in Windows absolute paths are always separated with a backslash, while other operating systems such as Linux use normal slashes.
The output of this program can be viewed in Figure 4. There the reader can also see the default encoding "System.Text.UTF8Encoding", which contains a large number of special characters and thus can represent almost anything. There are many other encodings - while utf8 uses at least 1 byte for each letter - utf16 uses twice or utf32 even 4 times the amount of disk space per letter. If file size matters and only english letters are used it is sometimes better to stick to basic encodings like ASCII. [7]

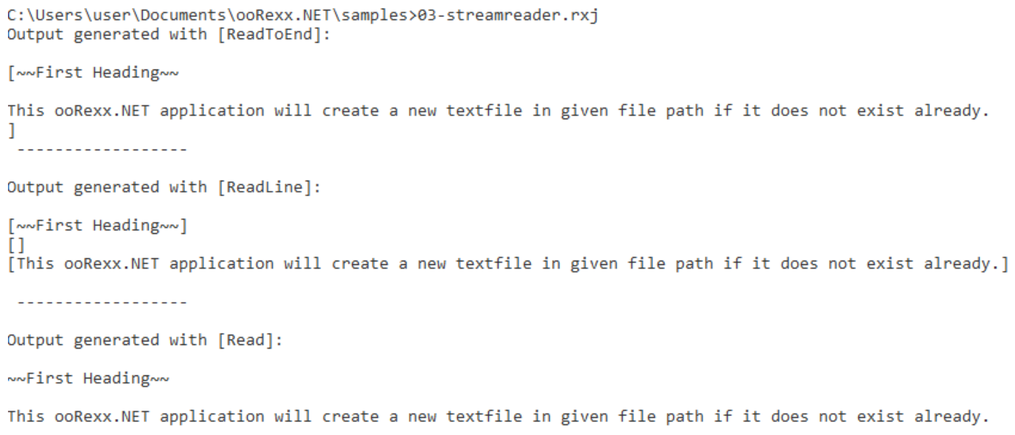
3 03-streamreader.rxj

This application is based on the previous coding example from Listing 3 since it uses the textfile, which was created with the stream writer.

```
 1 /* File:        03-streamreader.rxj
 2  * Description: This application opens a file and outputs it using different techniques
 3  * Shows:
 4  *              - Using a linefeed and pretty print routine
 5  *              - Difference between ReadToEnd, ReadLine and Read
 6  *              - Using the class "System.Convert" to convert a number representing a char
       into a real char
 7  */
 8
 9 SIGNAL ON SYNTAX -- in case an error occurs, go to Syntax jumping point in Line 54
10 returnPointAfterSyntax: -- define a returning point
11
12 lf = "0a"x -- defining a linefeed
13 directory = clr.import("System.IO.Directory")
14 filedirectory = directory~GetCurrentDirectory
15 filename = "02-textfile.txt"
16 filepath = filedirectory || .file~separator  || filename
17
18 /* get access to static classes System.Console and System.Convert for future use */
19 console = clr.import("System.Console")
20 converter = clr.import("System.Convert")
21
22 fileStream = .clr~new("System.IO.StreamReader", filepath) -- open a file stream with the
       file created in 02-streamwriter.rxj
23
24 SAY "Output generated with [ReadToEnd]:" lf
25 prettyOutput = pp(fileStream~ReadToEnd) -- pp stands for pretty print and is a rexx method
       to add square brackets before and after the given argument
26 console~Write(prettyOutput)
27 fileStream~Close -- close the filestream to free resources
28
29 SAY lf "------------------" lf
30
31
32 fileStream = .clr~new("System.IO.StreamReader", filepath) -- open the filestream again
33 SAY "Output generated with [ReadLine]:" lf
34
35 DO UNTIL fileStream~Peek = -1 -- -1 means, that there is no further line to output
36   prettyOutput = pp(fileStream~ReadLine)
37   console~WriteLine(prettyOutput) -- WriteLine is used to insert a line break afterwards,
       which we need here because we read each line
38 END
39 fileStream~Close
40
41 SAY lf "------------------" lf
42
43
44 fileStream = .clr~new("System.IO.StreamReader", filepath) -- open the filestream again
45 SAY "Output generated with [Read]:" lf
46 DO WHILE fileStream~Peek <> -1   -- while there is a character available
47   char = converter~ToChar(fileStream~Read)
48   prettyOutput = char
49   console~Write(prettyOutput)
50 END
51 fileStream~Close
```

```
exit

Syntax:     -- assuming the syntax error was caused because of the missing file "02-textfile.txt"
  "rexx ""02-streamwriter.rex""" -- call the streamwriter application to create 02-textfile.txt
  SIGNAL returnPointAfterSyntax  -- return to Line 10

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

Listing 4: Reading files in ooRexx.NET



```
C:\Users\user\Documents\ooRexx.NET\samples>03-streamreader.rxj
Output generated with [ReadToEnd]:

[~~First Heading~~

This ooRexx.NET application will create a new textfile in given file path if it does not exist already.
]
 -----------------

Output generated with [ReadLine]:

[~~First Heading~~]
[]
[This ooRexx.NET application will create a new textfile in given file path if it does not exist already.]

 -----------------

Output generated with [Read]:

~~First Heading~~

This ooRexx.NET application will create a new textfile in given file path if it does not exist already.
```

Figure 5: Output of 04-streamreader.rxj

A few commands are the same as in the last example, like getting the present working directory with the public method "Directory.GetCurrentDirectory". [8]
It starts by defining a linefeed variable, which can also be called "End Of Line" character or "Line Break", which is exactly what it does - a line break. Since every Rexx SAY commands ends with a line break, it is also possible to use multiple SAY commands instead of linefeeds, but those keep the code clean and easy to read. The "0a" from `lf = "0a"x` uses hexadecimal encoding, which is equivalent to decimal spelling 10 - and "x" implies that we want to access a value from an ASCII table, in that case the 10th value, which is a line break.
`fileStream~peek` returns an integer representing the next character to be read or -1 in case there are no characters left to read, signalising that the document has reached its end. The focus of this application is to show the differences between `"ReadToEnd"`, `"ReadLine"` and `"Read"`. `ReadToEnd` returns a string containing all of the characters in the textfile, including special characters like tabulators or carriage returns. `ReadLine` reads until the next line

break and sets the cursor afterwards, so that the next call to `ReadLine` or `Read` will start at that point. In contrast to `ReadLine`, `Read` will receive only the next character as an integer as opposed to the whole line.

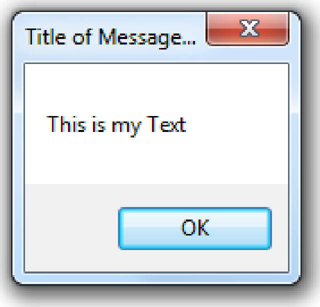
4 04-messagebox.rxj

This is a very easy and basic coding example which shows how to access .NET message boxes.

```
/* File:        04-messagebox.rxj
 * Description: Shows a Message Box in ooRexx.NET with a title and a text
 * Shows:       - MessageBox.Show method with title and a text
 */

text = "This is my Text" -- define text
title = "Title of MessageBox" -- define title
MessageBox = clr.import("System.Windows.Forms.MessageBox") -- get access to static class
    "MessageBox" of namespace "System.Windows.Forms"
MessageBox~Show(text, title) -- start method "show" with two arguments: text and optional
    title

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

Listing 5: Message Boxes in ooRexx.NET

A message box is a nice way to communicate with the user or to display error/success messages. It is used in many programming languages with different names, like Message Dialog in Java or Message Box in Python. How it's displayed is dependent on the operating system and .NET framework version. It is not possible to change either the width or height of a message box - making it an unreliable but still easy-to-implement part of user interaction.



This is my Text

Figure 6: The default MessageBox in .NET

5 05-messagebox.advanced.rxj

This coding example shows just how complex message boxes can get. Sometimes it is useful to give the user a choice when clicking on a message box. This is where additional buttons come in handy, which can be passed as the third argument in the `MessageBox~Show()` method.

```
/* File:        05-messagebox.advanced.rxj
 * Description: Shows multiple Message Boxes with different button setups
 * Shows:       - MessageBox with Yes, No and Cancel options
 *              - MessageBox with OK and Cancel options
 *              - MessageBox with Abort, Retry and Ignore options
 *              - Reacting on user interaction
 */

title = "Title of MessageBox" -- define title
   -- get access to the static class "MessageBox" of namespace "System.Windows.Forms", a
    "Dialog" window
MessageBox = clr.import("System.Windows.Forms.MessageBox")
DialogResult = clr.import("System.Windows.Forms.DialogResult") -- get access to
    DialogResult enumerations"

   -- get access to MessageBoxButtons enumerations
MessageBoxButtons = clr.import("System.Windows.Forms.MessageBoxButtons")

   -- display first message box and save result in Rexx variable "res"
title = 'Message Box # 1'
res = MessageBox~show('This is a YesNoCancel Messagebox. Press the "Yes" button to
    continue', title, MessageBoxButtons~YesNoCancel)

IF res = DialogResult~Yes  THEN DO -- compare message box result with the enumeration
    value defined in DialogResult
  SAY title 'returned "OK"'
  DO LABEL outer FOREVER          -- define a DO block with a name (as a possible target
    for LEAVE in the inner DO block)
     title='Message Box # 2 ("outer")'
     res = MessageBox~show('This is another Messagebox (you pressed the "'res~toString'"
    button). Press the "OK" button to see the next one', title,
    MessageBoxButtons~OKCancel)
     IF res = DialogResult~OK THEN -- compare message box result with the enumeration
    value defined in DialogResult
     DO
        SAY title 'returned "OK"'
        DO LABEL inner FOREVER
          title = 'Message Box # 3 ("inner")'
          res = MessageBox~show('Messagebox # 3 (you pressed the "'res~toString'" button).
    Click whatever you want.' , title, MessageBoxButtons~AbortRetryIgnore)
          IF res = "Abort" THEN    -- compare with a string
          DO
             SAY title 'returned "Abort"'
             LEAVE outer          -- leave do-block named "outer"
          END
          IF res = "Retry" THEN    -- leave do-block named "inner"
          DO
             SAY title 'returned "Retry"'
             LEAVE inner          -- leave do-block named "inner"
          END
          SAY title 'returned "Ignore"'
        END
```

```
44      END
45      ELSE
46      DO
47         SAY title 'returned "Cancel"'
48         LEAVE outer
49      END
50   END
51 END
52 ELSE
53   SAY title 'returned "'res~toString'"'
54
55 ::REQUIRES CLR.CLS   -- get ooRexx.NET support
```

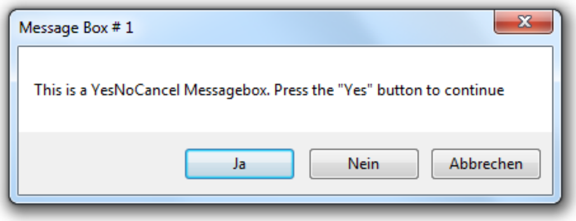
Listing 6: Advanced Message Boxes in ooRexx.NET

Take a second look at Line 15 where we define the Rexx variable MessageBoxButtons as `"System.Windows.Forms.MessageBoxButtons"` class. This is actually a public enumeration - which is a set of predefined constants which can only be used for one specific reason, in this particular case for message boxes. This construct can be seen quite often in the .NET framework, that's why this example shows how to handle them correctly.
Line 14 defines another set of enumerations as the Rexx variable "DialogResult" to indicate the return value of a message box. The members are:

- OK
- Yes
- No
- Abort
- Cancel
- Retry
- Ignore
- and finally **None** if nothing was returned from the message box.

Line 21 compares the result of the shown message box against a DialogResult enumeration, in this example "Yes". The next button presses compare directly to the Rexx strings "Abort", "Retry" and "Ignore", which is also possible. [9, 10]

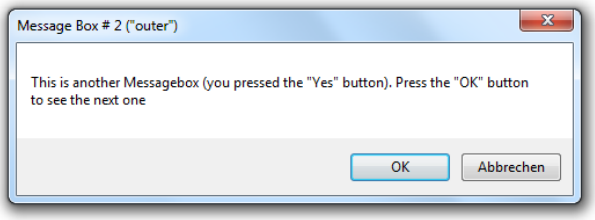
Figure 7 shows the first message box providing the Yes/No/Abort options.


Message Box # 1

This is a YesNoCancel Messagebox. Press the "Yes" button to continue

Figure 7: A Yes/No/Abort MessageBox in ooRexx.NET

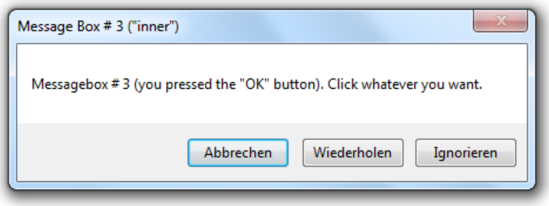
The second message box shows when clicking on "Yes" in the previous one.


Message Box # 2 ("outer")

This is another Messagebox (you pressed the "Yes" button). Press the "OK" button to see the next one

Figure 8: An OK/Abort MessageBox in ooRexx.NET

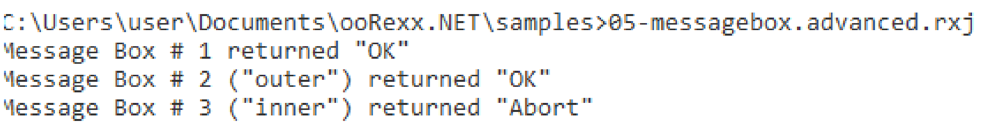
The third message box provides the possibility to show the second message box again when activating the "Retry" option.


Message Box # 3 ("inner")

Messagebox # 3 (you pressed the "OK" button). Click whatever you want.

Figure 9: An Abort/Retry/Ignore MessageBox in ooRexx.NET

Meanwhile, all user input gets documented in the terminal in Figure 9.



```
C:\Users\user\Documents\ooRexx.NET\samples>05-messagebox.advanced.rxj
Message Box # 1 returned "OK"
Message Box # 2 ("outer") returned "OK"
Message Box # 3 ("inner") returned "Abort"
```

Figure 10: The terminal output of 05-messagebox.advanced.rxj

6 06-process.demonstration.rxj

This example demonstrates process handling in ooRexx.NET. It opens a text file in notepad.exe, a website in a web browser and sends the F11 key to toggle fullscreen mode in Internet Explorer.

```
/* File:        04-process.demonstration.rxj
 * Description: This application demonstrates object-oriented process handling in ooRexx.NET
 * Shows:
 *              - Usage of basic message boxes with a specified title
 *              - Starting a process by dispatching "Start"
 *              - Sending keys to manipulate running applications
 *              - Starting a process with maximized window style and additional information
 *              - Retrieving the absolute path to temporary folder
 */



directory = clr.import("System.IO.Directory") -- get access to static class System.IO.Directory
filedirectory = directory~GetCurrentDirectory -- get path to present working directory
filename = "02-textfile.txt"
filepath = filedirectory || .file~separator || filename -- absolute path to the textfile from sample 02-streamwriter.rex
myProcess = .Process~new -- get a reference to class "Process" of this file (Line 67)
messageBox = clr.import("System.Windows.Forms.MessageBox") -- preload messageBox class into a variable for further use

messageBox~show("Let's open our new textfile", "Information") -- open a messageBox with the first argument as text and second argument as title
myProcess~Application = "notepad" -- define class attribute "Application" -> set it to "notepad". No file extensions needed - the .NET framework knows what we want
myProcess~Filepath = filepath -- define class attribute "Filepath" and set it to our textfile
myProcess~Start -- start process "notepad" and open the file given in variable filepath
CALL SysSleep 2 -- wait for 2 seconds

messageBox~show("Now I will open a Website in your favorite Webbrowser", "Information")
myProcess~Reset -- call to class method "Reset" which resets all the values so another Process can be started with the same class instance
myProcess~Filepath = "https://google.com"
myProcess~Start(.true) -- open this website with the standard application for the "https" protocol, e.g. standard web browser
CALL SysSleep 2

messageBox~show("Of course I can force you to use Internet Explorer", "Information")
myProcess~Application = "IExplore"
myProcess~Filepath = "https://google.com" -- start Internet Explorer and open the link specified in Filepath
myProcess~Start
CALL SysSleep 2

messageBox~show("I can even toggle Fullscreen by emulating keyboard presses", "Information")
sendKeys = clr.import("System.Windows.Forms.SendKeys") -- get access to static class System.Windows.Forms.SendKeys
sendKeys~SendWait("{F11}") -- this is a call to the method "SendWait" which presses F11 and waits for a response. F11 toggles fullscreen mode in Internet Explorer.
```

```
CALL SysSleep 1
sendKeys~SendWait("{F11}") -- return to normal mode
CALL SysSleep 2


path = clr.import("System.IO.Path") -- get reference to static class "System.IO.Path"
tempPath = path~GetTempPath -- --Path.GetTempPath returns a string with the absolute path
    to the temporary folder which usually looks like this but can vary for each system:
    C:\Users\[UserName]\AppData\Local\Temp\. Alternatively, it can be accessed using
    %TEMP%
tmpfilepath = tempPath || .file~separator || "temporary.txt"
file = .clr~new("System.IO.StreamWriter", tmpfilepath) -- open a file stream
file~Write("It's also possible to specify additional information, like WindowStyle to
    maximize or minimize the window.")
file~Write("(Program has ended, leaving this window open.)")
file~Close

ProcessStartInfo = .clr~new("System.Diagnostics.ProcessStartInfo", "Notepad") --
    ProcessStartInfo is used to set a few values before starting a process
ProcessStartInfo~Arguments = tmpfilepath -- give notepad the absolute file path to our new
    textfile as an argument
ProcessWindowStyle = clr.import("System.Diagnostics.ProcessWindowStyle")
ProcessStartInfo~WindowStyle = ProcessWindowStyle~Maximized -- now the application
    "Notepad" will start with window style maximized.

myNewProcess = .Process~new
myNewProcess~ProcessStartInfo = ProcessStartInfo
myNewProcess~Start -- start a new process based on the information provided in
    ProcessStartInfo


::REQUIRES PROCESS.CLS -- get access to the Process class which includes CLR.CLS
    (ooRexx.Net support)
```

Listing 7: Process handling in ooRexx.NET

The PROCESS class is defined in PROCESS.CLS. This file requires CLR.CLS and provides multiple methods for easier process handling.

```
/* File:        PROCESS.CLS
 * Class:       Process
 * Description: Starts Windows Processes based on information about application or file
 */


::CLASS Process PUBLIC
  ::ATTRIBUTE Application
  ::ATTRIBUTE Filepath
  ::ATTRIBUTE ProcessStartInfo

  /* Method:      init
   * Description: Constructor which saves the passed arguments in the class and defines
    variable "Process" as .NET class "System.Diagnostics.Process"
   * Argument:    - Application: the application which will be started
   */
  ::METHOD init
    EXPOSE Application Process

    if arg(1,"Exists") then   -- if an argument was supplied, assign it to attribute
    "application"
```

```
       application = arg(1)

    Process = .clr~new("System.Diagnostics.Process") --get access to public class System.Diagnostics.Process

  /* Method:      Start
   * Description: Starts the process based on already submitted information
   * Argument:    - ResetValues: calls class method "Reset" after starting the process
   */
  ::METHOD Start
    EXPOSE ProcessStartInfo Application Filepath Process ResetValues
    USE ARG ResetValues = .false

    IF var("ProcessStartInfo") THEN
      Process~clr.dispatch("Start", ProcessStartInfo) -- start process based on ProcessStartInfo
    ELSE IF var("Application") THEN
      Process~clr.dispatch("Start", Application, Filepath) -- start process based on Application and optionally Filepath
    ELSE IF var("Filepath") Then
      Process~clr.dispatch("Start", Filepath) -- start process based solely on Filepath (uses default application)
    ELSE
      RETURN "ERROR: No values specified"

    IF ResetValues <> .false THEN -- if the call to this method includes an argument, all values will be reset after launching the process
      self~Reset -- call to class method Reset

    RETURN .true

  /* Method:      Reset
   * Description: Resets all class attributes to their default values to launch a different process in the same class instance
   */
    ::METHOD Reset
      EXPOSE Application Filepath ProcessStartInfo
      DROP application filepath processStartInfo   -- drop the attributes

::REQUIRES CLR.CLS
```

Listing 8: PROCESS.CLS

This coding example chains together a lot of commands separated by message boxes, which have to be pressed in order to execute the next commands. It also includes the ooRexx class "Process", which can be used for easy process handling with one or more class instances. After instantiating the public class "Process", one of the three attributes has to be defined or else no process can be launched. Those three attributes are:

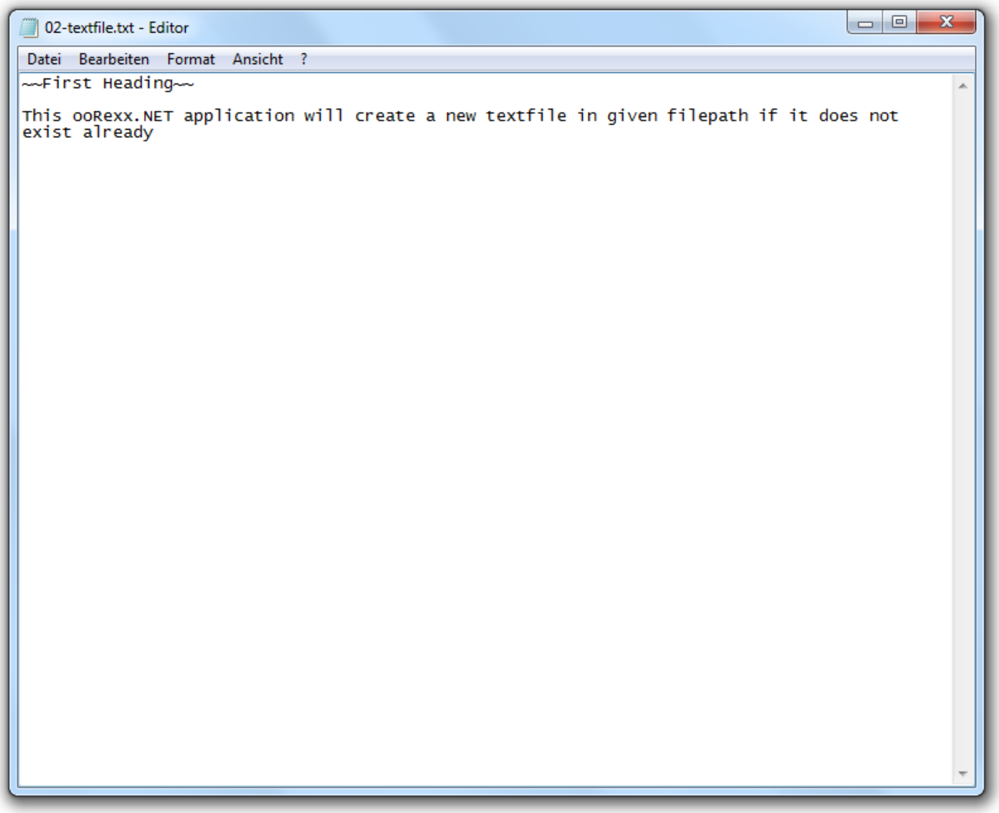
- **Application**: defines the Application to launch
- **Filepath**: defines the file path to a file to launch. If only this information is available, Windows will start the process with the default application based on the file extension, e.g. Microsoft Word for `.docx`,

Acrobat Reader for `.pdf` and Notepad for `.txt` files. These default applications can be changed in the system settings.

- **ProcessStartInfo**: defines a set of values for the process. It has to be an instance of the public class `"System.Diagnostics.ProcessStartInfo"` and has to include at least the application or file path as argument, like in Line 54 of this coding example.

The class `"Process"` is set to public so it can be also used for other examples, it just needs to be included with the `::REQUIRES` directive. Technically, the inclusion of `CLR.CLS` in Line 64 is not needed since `PROCESS.CLS` already requires `CLR.CLS` so therefore 06-process.demonstration.rxj has transitive access to all the files `PROCESS.CLS` has access to.

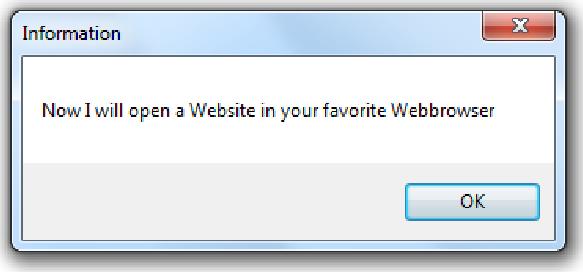
The last process to start in lines 54 to 61 includes a .NET class called `"ProcessStartInfo"`. It can have up to two arguments. The first one is the filepath and the second provides a set of commands to pass to the desired application. Aside from the WindowStyle there are a lot of other Properties which can be set with ProcessStartInfo before starting the Process, for example a (plain text or encrypted) password for applications which need authentication before usage. It is also possible to start the process without creating a new window - that can come in handy sometimes. [11]
When starting the application a text file will be opened as seen in Figure 11.



```
~~First Heading~~

This ooRexx.NET application will create a new textfile in given filepath if it does not
exist already
```

Figure 11: Output of the first Notepad Application

Then, a message box gives the hint that a Website will be opened in the preferred Webbrowser.



Now I will open a Website in your favorite Webbrowser

Figure 12: The first message box

The application opens https://google.at in the preferred browser in Figure 13.

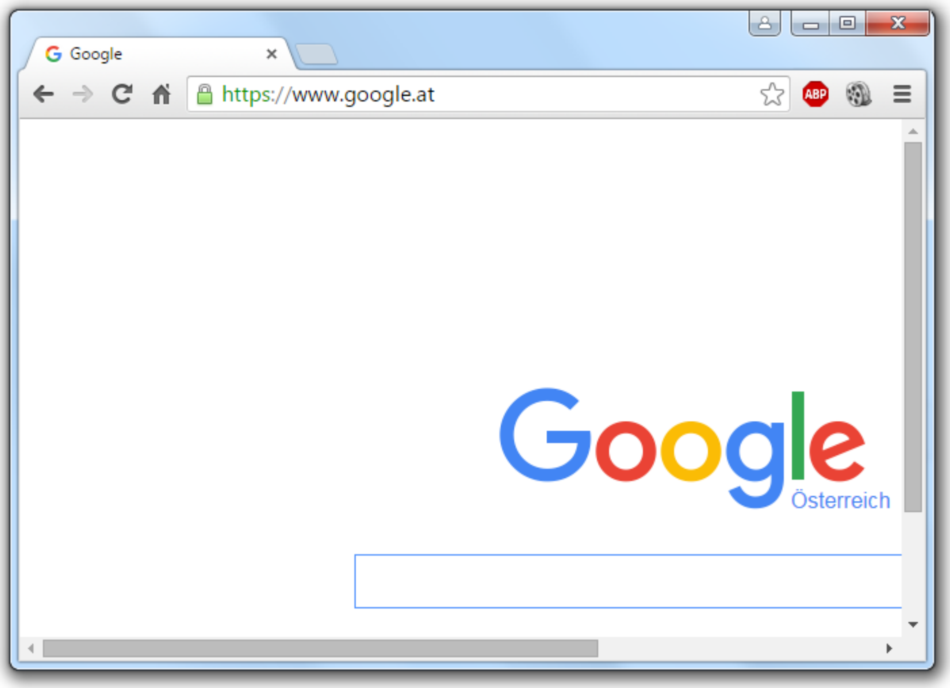
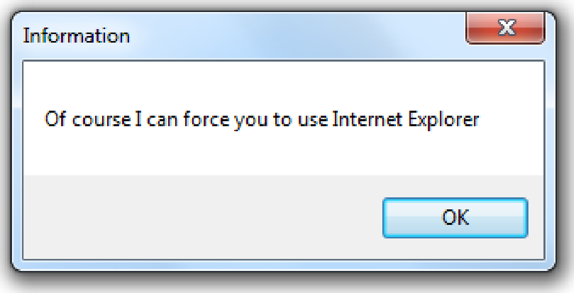


Figure 13: https://google.at in favorite browser

Figure 14 shows another message box stating that the next web page will be opened in Internet Explorer, regardless of the favorite browser setting.



Of course I can force you to use Internet Explorer

Figure 14: The Second Message Box

Figure 15 opens the same web page in the Internet Explorer.

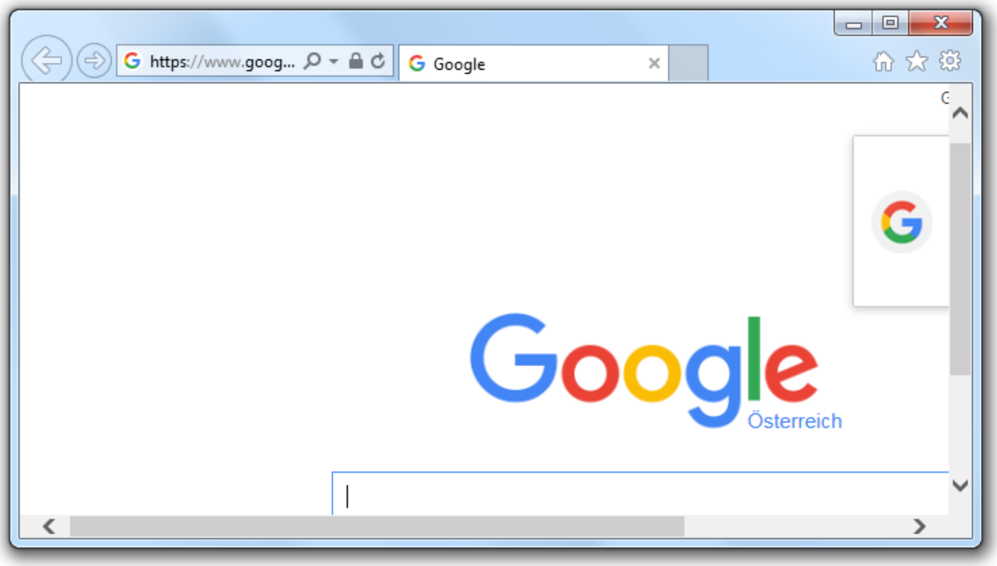
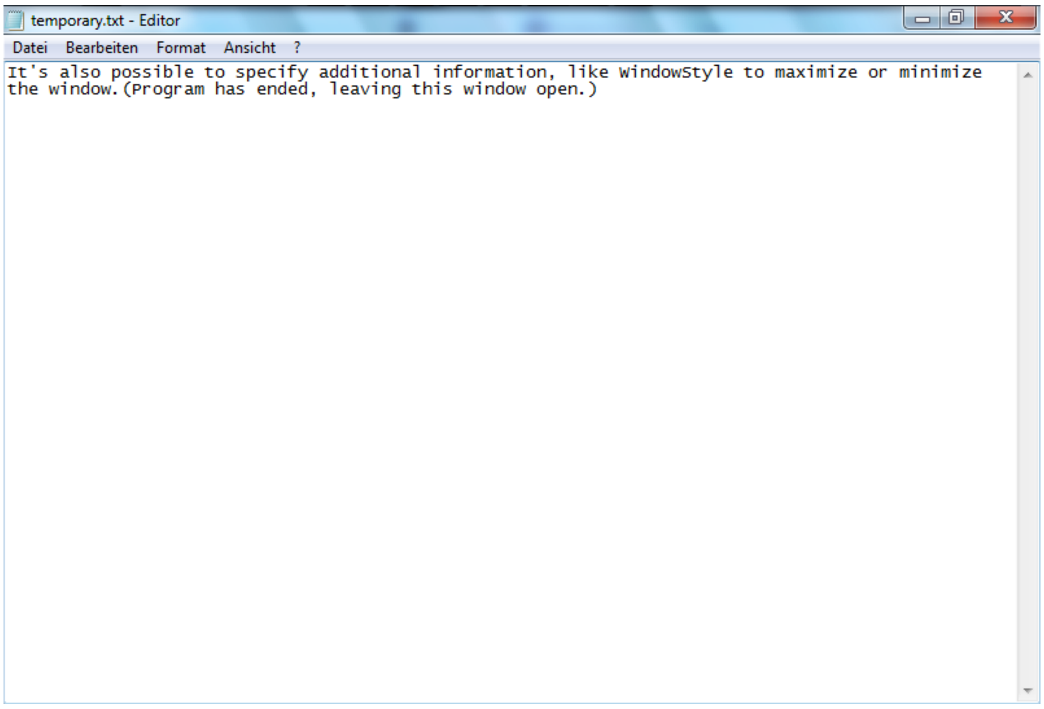


Figure 15: https://google.at in Internet Explorer

Finally, Figure 16 shows the notepad application in full screen mode.



```
It's also possible to specify additional information, like WindowStyle to maximize or minimize
the window.(Program has ended, leaving this window open.)
```

Figure 16: Maximised Notepad Process

7 07-MAC.rxj

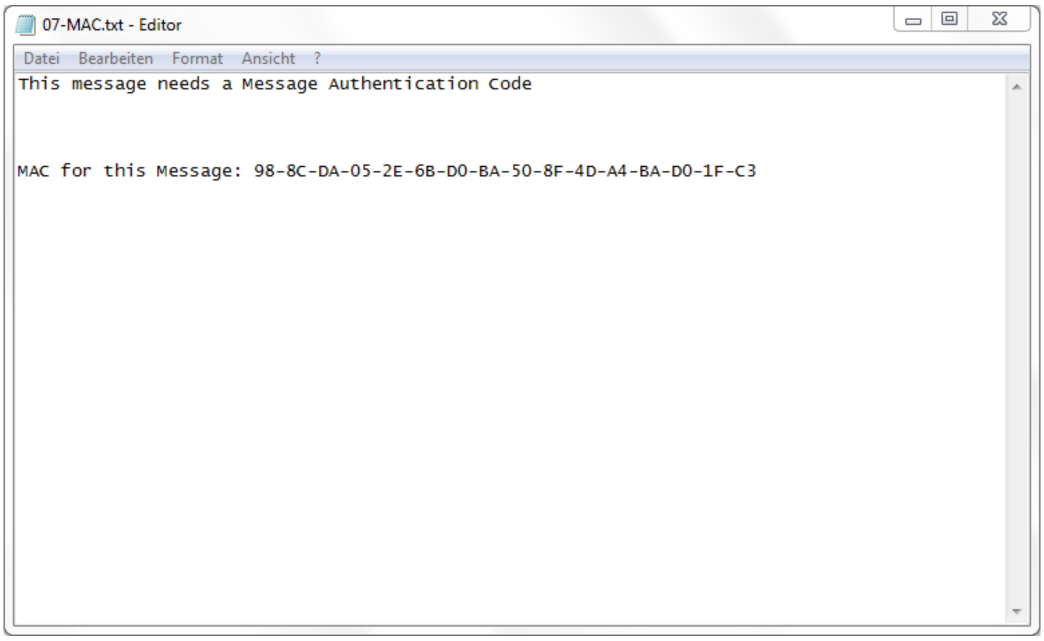
A Message Authentication Code is a tool for secure communication between two subjects. It uses a hash function based on a secret key to generate a hash value for a message. For the Message Authentication Code to work it is really important that the secret key is only available to the sender and recipient of the message. Once a key has been exchanged the communication with a Message Authentication Code provides both integrity and authenticity.

```
/* File:        07-MAC.rxj
 * Description: This application generates a hash string of a message based on a secret
    key. Then, it stores both message and hash into a textfile and opens it afterwards as
    a process with the default text editor program.
 * Shows:
 *              - Encode a string into an array of Bytes
 *              - Generate a hash of a message based on a specific key
 *              - Convert an array of Bytes into a string
 *              - Usage of PROCESS.CLS to start a process
 */

-- rgf: .file~separator statt "\"
-- rgf: aktuelles Verzeichnis kann man direkt in Rexx mit der directory()-Funktion
    erhalten!

message = "This message needs a Message Authentication Code" -- an important message
key = "my123secret321key" -- this is a secret key, which only the sender and recipient
    should have access to

encoding = .clr~new("System.Text.ASCIIEncoding")
keyByte = encoding~GetBytes(key) -- use ASCII encoding to code the key into several bytes
    ( a byte array )

hashkey = .clr~new("System.Security.Cryptography.HMACMD5", keyByte) -- get access to a MAC
    md5 hash function class

messageBytes = encoding~GetBytes(message) -- returns the message as an array of Bytes

hashmessage = hashkey~ComputeHash(messageBytes) -- generates the hash in an array of Bytes

BitConverter = clr.import("System.BitConverter") -- get access to static class
    System.BitConverter
MAC = BitConverter~toString(hashmessage) -- convert System.Byte[] to System.String

filedirectory = directory() -- this method returns the path of the current directory of
    this application. This assures, that this application works on each and every system,
    regardless of where the files are saved.
filename = "07-MAC.txt"
filepath = filedirectory || .file~separator || filename
file = .clr~new("System.IO.StreamWriter", filepath) -- get access to class
    System.IO.StreamWriter
file~WriteLine(message) -- write message
file~~WriteLine~~WriteLine~~WriteLine~~WriteLine -- get 4 additional line breaks
file~Write("MAC for this message:" MAC) -- write MAC
file~Close

openResult = .Process~new -- get an instance of the Process Class
```

```
openResult~Filepath = filename -- set the filepath to our new filename. You could also use
    variable "filepath" there - which is a reference to the absolute path + filename of
    this file. But since it's in the same directory filename is sufficient.
openResult~Start  -- start the process based on the given information (filepath)


::REQUIRES PROCESS.CLS -- get access to Process class
```

Listing 9: A Message Authentication Code in ooRexx.NET

The .NET Framework has quite a few cryptography classes - all under the namespace `"System.Security.Cryptography"`. They include message encoding, decoding, authentication and hash functions in general. Unfortunately, some algorithms are already old and not recommended for using. It is best to read the MSDN documentation on those classes carefully to avoid unsecure applications.[12]
This example uses HMACMD5, a Hashed-Key Message Authentication Code using MD5 Hashing. MD5 is a Message-Digest Algorithm with a 128 Bit/16 Byte hash. Therefore the length of the result of our Message Authentication Code is a Byte array with exactly 16 Elements in it. This array is converted to a string in Line 23 and then written into a file with the message itself. Finally, with the help of PROCESS.CLS the new text file gets opened with the default text editing program, in my case "notepad.exe".



```
This message needs a Message Authentication Code


MAC for this Message: 98-8C-DA-05-2E-6B-D0-BA-50-8F-4D-A4-BA-D0-1F-C3
```

Figure 17: Output of 07-MAC.rxj

8 08-WebClient.rxj

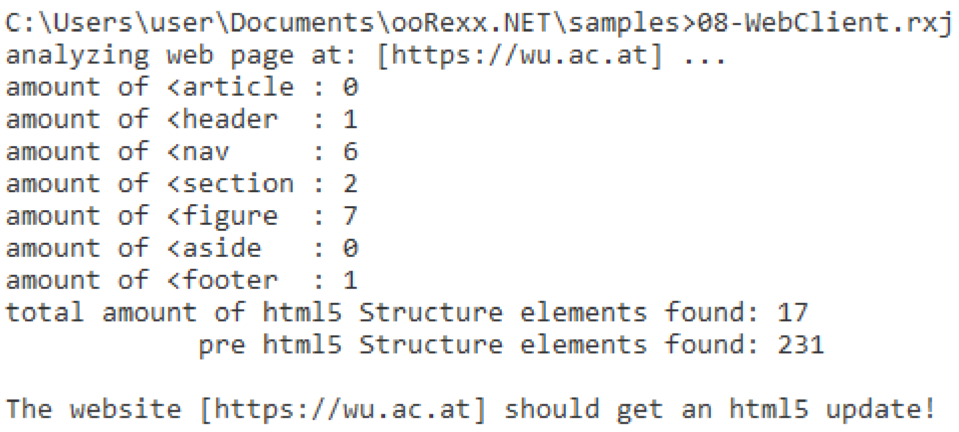
This coding example downloads the text document of an HTTP URL and analyses it for each array element defined in Line 23.

```
 1 /* File:        08-WebClient.rxj
 2  * Description: This application downloads the frontend source code of an html page via
      HTTP and analyses it based on the amount of html5 vs. html4.1 commands.
 3  * Arguments:   - URL including "http://" or "https://"
 4  * Shows:
 5  *              - Setting up a WebClient to send or receive data through a URI
 6  *              - Defining a Rexx array with elements
 7  *              - Using Rexx arguments
 8  *              - Private Routine CountSubStrings to count substrings in a string
 9  *              - Using PROCESS.CLS to start a process
10  *              - Comparing two integer values
11  */
12
13
14 WebClient = .clr~new("System.Net.WebClient") --get Access to the WebClient
15 IF arg(1)~length > 0 & ABBREV(arg(1), "http") -- if an argument was supplied while
      starting this app AND this argument is a valid URL (starts with http at least)
16   THEN URL = arg(1) -- then set URL to that argument
17   ELSE URL = "https://wu.ac.at" -- else set it to the WU Homepage
18
19 SAY "analyzing web page at:" pp(url) "..."
20
21 htmlPage = WebClient~DownloadString(URL) -- download source code from given URL
22 htmlStructureElement = "<div"
23 html5StructureElements = .array~of("<article", "<header", "<nav", "<section", "<figure",
      "<aside", "<footer")
24
25 htmlcount = CountSubStrings(htmlPage, htmlStructureElement) -- use private routine from
      this file to count total amount of substrings
26 html5count = 0
27
28 DO oneElement OVER html5StructureElements
29   oneElementCount = CountSubStrings(htmlPage, oneElement) -- count all substrings for each
      element of the html5StructureElements array
30   html5count += oneElementCount
31   SAY LEFT("amount of" oneElement, 18) ":" oneElementCount
32 END
33
34 s1 = "total amount of html5 Structure elements found:"
35 s2 = "pre html5 Structure elements found:"
36 maxLen = max(length(s1),length(s2))   -- get length of longer string
37 SAY s1~right(maxLen) html5count     -- right adjust
38 SAY s2~right(maxLen) htmlcount      -- right adjust
39 SAY
40
41 IF htmlcount > html5count THEN   -- compare those values
42    SAY "The website" pp(url) "should get an html5 update!"
43 ELSE IF html5count>0 THEN        -- html5 elements available?
44    SAY pp(url)": Great!"
45 ELSE
46    SAY pp(url)": ? Neither old HTML nor any HTML5 tags found, tsk!"
47
48
49
50 ::REQUIRES CLR.CLS
```

```
51
52 ::ROUTINE CountSubStrings PRIVATE
53   USE ARG haystack, needle
54   RegEx = .clr~new("System.Text.RegularExpressions.Regex", needle)
55   /* Although System Class "System.Text.RegularExpressions.Regex" has a constructor with
       no Arguments, you cannot use it because it is PROTECTED.
56   Instead, 3 Constructors can be used: only needle, needle and RegexOptions or needle,
       RegexOptions and Timespan. */
57
58   /* This example will count all non-overlapping occurences of the string defined in
       "needle" in the string defined in "haystack" */
59   Matches = RegEx~Matches(haystack)
60   RETURN Matches~Count
```

Listing 10: Downloading and analysing HTML pages via HTTP in ooRexx.NET

The generated output of this application can be seen in Figure 18:



```
C:\Users\user\Documents\ooRexx.NET\samples>08-WebClient.rxj
analyzing web page at: [https://wu.ac.at] ...
amount of <article : 0
amount of <header  : 1
amount of <nav     : 6
amount of <section : 2
amount of <figure  : 7
amount of <aside   : 0
amount of <footer  : 1
total amount of html5 Structure elements found: 17
            pre html5 Structure elements found: 231

The website [https://wu.ac.at] should get an html5 update!
```

Figure 18: Output of 08-WebClient.rxj while analysing https://wu.ac.at

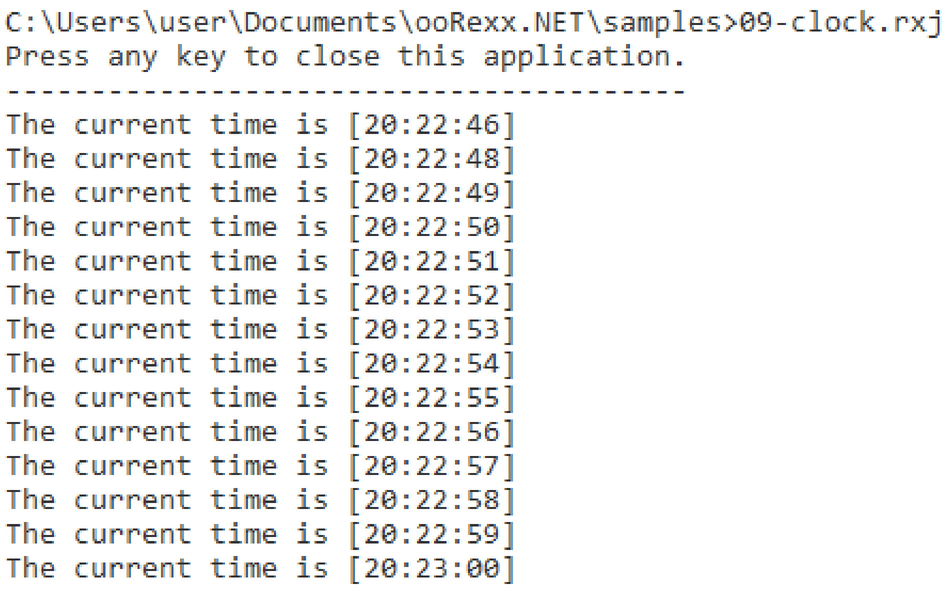
The private routine `"CountSubStrings"` starting in Line 52 creates an regular expression class instance with the array element as argument. A regular expression, also known as RegEx or RegExp, is a string with a specific syntax which is used to search efficiently through sometimes huge texts or to specify how another string should look like. With regex pattern matching, it makes it easy to create a rule that a string should start with an "A" and end with "D" without restricting the letters between "A" and "D" - or that it should or should not have a precise amount of characters. It is often used for "search and replace" functions in many programming languages. Line 59 returns all occurrences of the variable needle in the variable haystack and leaves the routine with the amount of found matches in Line 60. Since html5 introduced quite a few new structural tags it is not sufficient to run the private routine CountSubStrings only once - but for every tag and then store the final results in the variables htmlCount and html5count. The final output depends on whether there are more html5 structure elements than divs on this web page or not. In case the reader wants to try a web page with more html5 structure elements than html4, try `http://html5demos.com/contenteditable`. In reality, no element can replace the good old "div" tag counted in this example in the htmlCount variable. The html5 elements are only an addition to the "div" tag since they allow to give elements of the web page a better-named structure. [13]

9 09-clock.rxj

This coding example displays the current system time using .NET classes.

```
/* File:        09-clock.rxj
 * Description: Continous output of the current time via .NET classes.
 * Shows:       - DateTime class: getting the current time.
 *              - Starting a CLRThread, which is a Thread and runs simultaneously to the main code
 *              - Reading input keys from console
 *               - Exiting an application manually with the "Environment" class
 */



SAY "Press any key to close this application."
SAY "-----------------------------------------"

DateTime = .clr~new("System.DateTime")
.KeyPressThread~new~~start -- create an instance of KeyPressThread class, which is a Thread and start it.

DO FOREVER    -- starts an infinite loop
    CALL SysSleep 1               -- wait 1 second for the time to change
    currentTime = DateTime~Now    -- DateTime~Now usually contains a timeString with that format: "dd.mm.yyyy HH:mm:ss"

       -- convert the timeString to only show the time, ooRexx~counterpart: .dateTime~new~normalTime
    SAY "The current time is" pp(currentTime~toString("HH:mm:ss"))
END


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

/* Class:       KeyPressThread
 * Description: Checks continuously user input in the console. This class will then exit the application using the "Environment" class.
 */
::CLASS KeyPressThread SUBCLASS CLRThread
  ::ATTRIBUTE Console -- define attributes used in the class.
  ::ATTRIBUTE Environment

  ::METHOD init
    EXPOSE Console Environment
    Console = clr.import("System.Console") -- initialise console to use for the whole class
    Environment = clr.import("System.Environment") -- get access to static class "Environment"

  ::METHOD run -- by starting the CLRThread, this method will be run automatically
    EXPOSE Console Environment -- get Access to Console and Environment

    DO FOREVER -- starts an infinite loop
      IF Console~ReadKey <> .nil THEN -- Method "ReadKey" saves user input. If nothing was pressed, it will return the NIL Object
      DO
        Environment~Exit(0)  -- leave application
        EXIT                 -- leave method
      END
    END
```

Listing 11: Displaying the current time as it changes in ooRexx.NET



```
C:\Users\user\Documents\ooRexx.NET\samples>09-clock.rxj
Press any key to close this application.
----------------------------------------
The current time is [20:22:46]
The current time is [20:22:48]
The current time is [20:22:49]
The current time is [20:22:50]
The current time is [20:22:51]
The current time is [20:22:52]
The current time is [20:22:53]
The current time is [20:22:54]
The current time is [20:22:55]
The current time is [20:22:56]
The current time is [20:22:57]
The current time is [20:22:58]
The current time is [20:22:59]
The current time is [20:23:00]
```

Figure 19: Output of 09-clock.rxj

This application runs an infinite loop where it retrieves the current system time, which consists of the date and the time. Since we only need the time, the "toString" method allows for an argument to display only the fields that are needed - in this example the hour, minute and second. The syntax for the hour is either HH or hh. The difference between those is that small caps hh use an 12-hour clock, while large caps HH use an 24-hour clock. This method of retrieving the time may not be the most efficient one, since a lot of calculations have to be made in order to display the current time. On the other hand it is very secure and therefore advisable for server applications. Try the following: change the time in your system settings while running this application. As a result the displayed time will also immediately change! Unfortunately, due to millisecond rounding and lag it is possible that certain seconds will get left out like [20:22:47] in Figure 19. This is quite unfortunate, but it can happen. As a countermeasure the author suggests to implement one of two things:

- Retrieve the system time only every 5 seconds or so. For the other seconds simply count an integer up. This should not create any significant lag whatsoever.

- Reduce SysSleep time of Line 18 by a few milliseconds.

The class `"KeyPressThread"` is a subclass of the class `"CLRThread"`, which allows it to run asynchronously. It waits for the user to press a key and then exits the application. But first it defines two attributes in Lines 32 and 33, which is actually not necessary for the functionality of this application - but still it is easier to read if the class gets expanded.

10 10-gui.introduction.rxj

This application introduces Windows Forms, which can be used to create Graphical User Interfaces (GUIs) using the .NET Framwork.

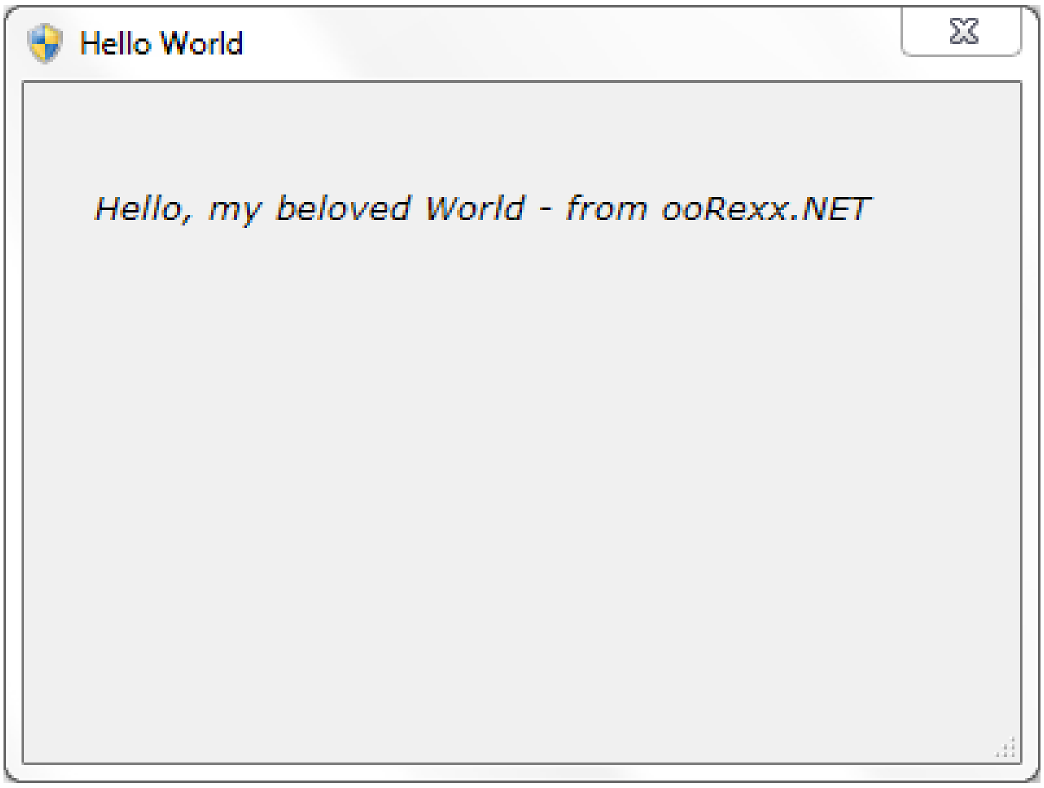
```
/* File:        10-gui.introduction.rxj
 * Description: This application opens a Windows Form including a label with text.
 * Shows:
 *              - A Graphical User Interface (GUI) in ooRexx.NET
 *              - Setting the title of a Form
 *              - Disallowing maximization and minimization of the Form
 *              - Attaching an Icon to a Form
 *              - Setting a Startposition for the Form to the center of the screen
 *              - Creating a simple label with a text
 *              - Setting the Location of the Label
 *              - Changing font family, font size and font style of a Label
 *              - Adding the label to the controls of the Form
 *              - Starting the Form, therefore displaying it.
 */


form = .clr~new("System.Windows.Forms.Form") -- get an instance of the Form class of
    namespace "System.Windows.Forms"
form~Text = "Hello World" -- set the Caption of the Titlebar
form~maximizeBox = .false -- do not allow to maximize the form - therefore, do not even
    display the buttons for maximization/minimization
form~minimizeBox = .false -- do not allow to minimize the form
form~Width = 400 -- set the width of the form to 400px
form~Height = 300 -- set the height of the form to 300px

SystemIcons = clr.import("System.Drawing.SystemIcons")
icon = .clr~new("System.Drawing.Icon", SystemIcons~Shield, 48, 48) -- load the enumation
    "Shield" from the set of system icons in 48x48px size as an icon
form~Icon = icon -- attach that icon to the form

FormStartPosition = clr.import("System.Windows.Forms.FormStartPosition") -- import static
    enumeration "FormStartPosition"
form~startPosition = FormStartPosition~CenterScreen -- set the starting position of the
    form to the center of the screen

label = .clr~new("System.Windows.Forms.Label") -- get an instance of the Label class
label~Text = "Hello, my beloved World - from ooRexx.NET" -- set the text to "Hello World"
label~Location = .clr~new("System.Drawing.Point", 25, 40) -- set the location to 40px from
    top and 25px from left corner of the form
label~AutoSize = .true -- resize this label from default label size to real label size

FontStyle = clr.import("System.Drawing.FontStyle") -- get access to enumation FontStyle
font = .clr~new("System.Drawing.Font", "Verdana", 10, FontStyle~Italic)
label~Font = font -- set the Font of the label to font family "Verdana" with font size
    10px and font style "Italic"


form~Controls~Add(label) -- add the label to the Controls of the form
SAY "close the dialog to end program"
form~showDialog -- show the form

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

Listing 12: Displaying a Windows Form using ooRexx.NET

In ooRexx.NET, one just has to create an instance of the Form Class and add Controls to it. There are many different controls in .NET, for example labels, buttons or a progress bar. Basically anything, which is used for user interaction in a GUI can be added to the form as a control object. Finally, to display the dialog, there are 2 ways to achieve that:

- Use one of the methods `"ShowDialog"` or `"Show"` from the Form Class. There is no difference in functionality between them.
- Use the method `"Run"` from the Application class with `"System.Windows.Forms"` namespace

The opened form can be seen in Figure 20.



Hello, my beloved World - from ooRexx.NET

Figure 20: Output of 10-gui.introduction.rxj

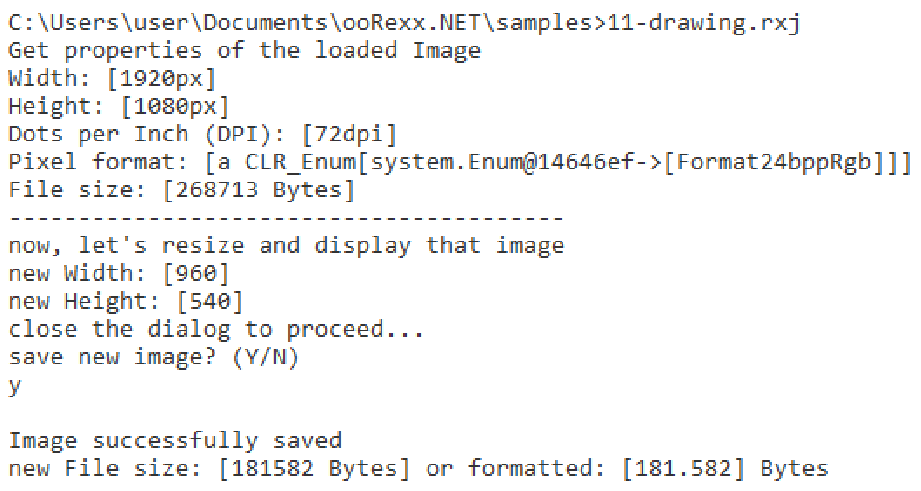
11 11-drawing.rxj

This application demonstrates image and bitmap handling in ooRexx.NET. It first opens the image as a bitmap from the "images" directory and then retrieves the file size and other properties of the image. After displaying it in a "PictureBox" the application crops the image by 50 % .

```
/* File:        11-drawing.rxj
 * Description: This application loads an image from the "images" subfolder, outputs a few
    interesting properties and then crops the image by 50% and saves it under a different
    file name.
 * Shows:
 *              - Loading a Bitmap from the image path
 *              - Accessing various Bitmap properties
 *              - Retrieving image file size with FileInfo Class
 *              - Adding thousand points to an integer for better overview
 *              - Applying a new image size to an image
 *              - Displaying an image in a form
 */

imagePath = "images/html5.jpg"
bitmap = .clr~new("System.Drawing.Bitmap", imagePath) -- load the image as a Bitmap
width = bitmap~width
height = bitmap~height
dpi = bitmap~VerticalResolution
pxFormat = bitmap~PixelFormat

SAY "Get properties of the loaded Image"

SAY "Width:" pp(width"px") -- pp = pretty print, inserts "[" before and "]" after the
    argument
SAY "Height:" pp(height"px")
SAY "Dots per Inch (DPI):" pp(dpi"dpi")
SAY "Pixel format:" pp(pxformat)
/* now let's retrieve the file size of this image */
filesize = .clr~new("System.IO.FileInfo", imagepath)~length -- returns filesize in Bytes
--SAY "File size:" pp(filesize~toString("N0") "Bytes") -- use number 0 default formatting
    to add thousand points while converting Integer64 to String
SAY "File size:" pp(filesize "Bytes")
SAY "----------------------------------------"

SAY "now, let's resize and display that image"
size = .clr~new("System.Drawing.Size", 960, 540) -- half size
bitmap = .clr~new("System.Drawing.Bitmap", bitmap, size) -- load a new bitmap from the old
    bitmap but use the new size
form = .clr~new("System.Windows.Forms.Form") -- a Form represents a window as a user
    interface
form~size = size -- set the width and height of the form to the width and height of the
    (new) bitmap

pictureBox = .clr~new("System.Windows.Forms.PictureBox") -- create a new PictureBox, which
    is used to display images in a form
pictureBox~image = bitmap -- insert our bitmap into the PictureBox
pictureBox~size = size -- also set the the width and height of the PictureBox to the size
    of our bitmap

SAY "new Width:" pp(bitmap~width) -- output width and height of the bitmap to see, if
    resizing was successfull
SAY "new Height:" pp(bitmap~height)
```

```

form~controls~add(pictureBox) -- add the PictureBox to the form
SAY "close the dialog to proceed..."
form~showDialog -- show the form

SAY "save new image? (Y/N)"
PULL save -- retrieve user input in console and save it as variable "save"
IF save = "Y" THEN DO -- since we used command "PULL" instead of "PARSE PULL", small caps user input will be transformed to upper case. Therefore this command will return .true even if the user typed in small caps "y".
  newImagePath = "11-newImage.jpg"
  bitmap~save(newImagePath)  -- save the bitmap as filename supplied as argument
  SAY "0A"x"Image successfully saved" -- insert line feed before actual Output

  newfilesize = .clr~new("System.IO.FileInfo", newImagePath)~length -- returns new filesize in Bytes
   -- turn the number into a CLR/.Net System.Decimal value and use the formatting capabilities of its ToString() method
  d=clr.box("decimal",newfilesize)  -- box as a System.Decimal (range should be large enough for all primitive value types)
  SAY "new File size:" pp(newfilesize "Bytes") "or formatted:" pp(d~toString("N0")) "Bytes"
END
ELSE EXIT 0


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

Listing 13: Image handling with ooRexx.NET



```
C:\Users\user\Documents\ooRexx.NET\samples>11-drawing.rxj
Get properties of the loaded Image
Width: [1920px]
Height: [1080px]
Dots per Inch (DPI): [72dpi]
Pixel format: [a CLR_Enum[system.Enum@14646ef->[Format24bppRgb]]]
File size: [268713 Bytes]
----------------------------------------
now, let's resize and display that image
new Width: [960]
new Height: [540]
close the dialog to proceed...
save new image? (Y/N)
y

Image successfully saved
new File size: [181582 Bytes] or formatted: [181.582] Bytes
```

Figure 21: Console Output of 11-drawing.rxj

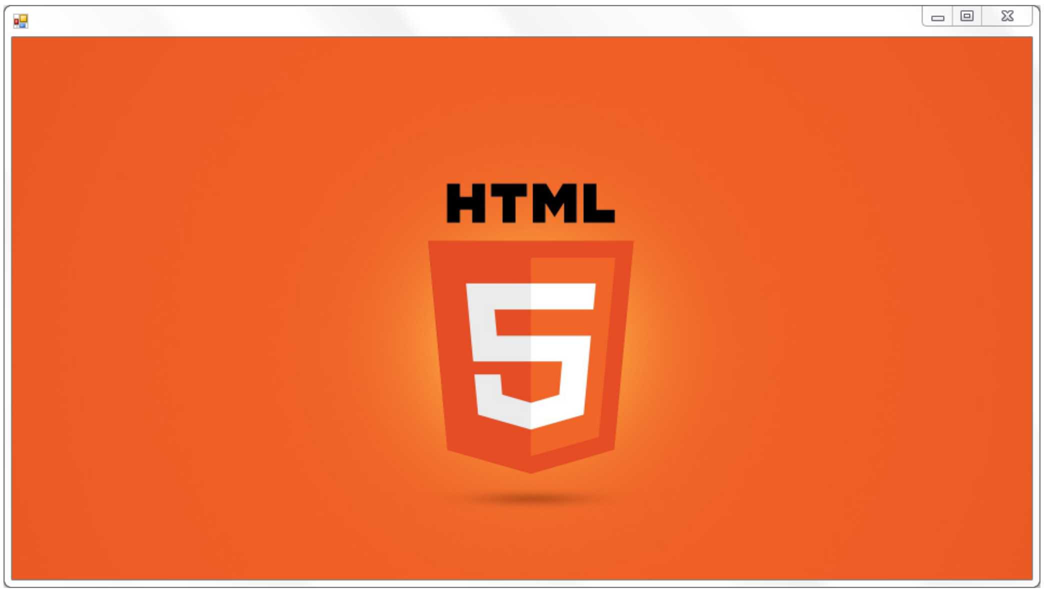


Figure 22: Form Output of 11-drawing.rxj

After the image gets resized, the application outputs the new file size converted to a string with thousand points so it is easier to read for human users. The .NET Framework supports quite a few standard numeric format strings. Here is a list of the most useful ones:

- **"C", en-US** stands for "Currency" and uses the country code of the second argument to represent a value as a currency. The use of this format specifier for the number "132.456" would result in "$123.46"". For the German country code, use de-DE.
- **"D10"** formats the provided integer to the specified length. So this number specifier would result in "0000123456" for number "123456". This is useful, if a database requires a special number format.
- **"N2"** formats to the number of decimal places. This can be extended by the country code in the second argument.
- **"P"** for displaying percentages. 0 = 0 %, 1 = 100 %
- **"H"** formats a decimal number as an hexadecimal value. Open Object Rexx uses `D2X(wholenumber)` Function for that.

[14]

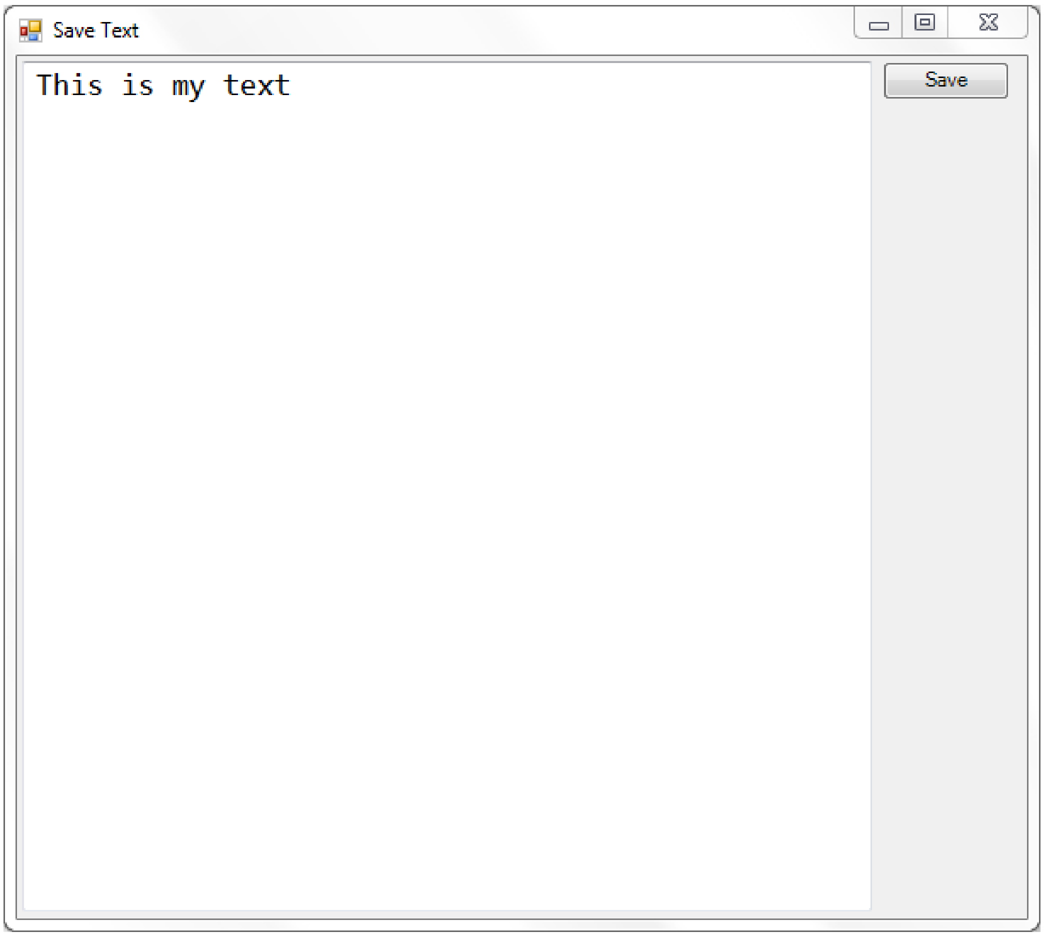
12 12-savefile.rxj

This application shows an easy implementation of a "save file" dialog using forms.

```
/* File:        12-savefile.rxj
 * Description: This application opens a form with a textarea and a save button to save the text as a text file.
 * Shows:
 *              - Customizing a textbox
 *              - Using Threads
 *              - A dialog to save files
 *              - Filtering file names and file types based on regular expressions
 */


form = .clr~new("System.Windows.Forms.Form") -- get an instance of the Form class
form~AutoSizeMode = GrowAndShrink -- set the attribute "AutoSizeMode" to enumeration "GrowAndShrink". This allows the form to automatically adjust to the elements.
form~text = "Save Text"
form~width = 610
form~height = 545

textbox = .clr~new("System.Windows.Forms.TextBox") -- get an instance of the TextBox class
textbox~Multiline = .true -- allows for multiple lines
textbox~AcceptsReturn = .true  -- this attribute allows line breaks
textbox~AcceptsTab = .true -- this attribute allows tabs

textbox~size = .clr~new("System.Drawing.Size", 500, 500) -- set the size of the textbox to 500x500px with

font = .clr~new("System.Drawing.Font", "Consolas", 14) -- get an instance of the Font class and set it to font family "Consolas" with font size 14pt
textbox~font = font -- apply our font instance to the textbox, therefore setting the text of the textbox to Consolas/14

startButton = .clr~new("System.Windows.Forms.Button")
startButton~Text = "Save"  -- set property "Text" to string "Save"

contentPane = .clr~new("System.Windows.Forms.FlowLayoutPanel") -- represent a FlowLayoutPanel to lay out the content horizontally
contentPane~AutoSize = .true  -- set property "AutoSize" to boolean true
contentPane~AutoSizeMode = GrowAndShrink  -- set property "AutoSizeMode" to enumeration value "GrowAndShrink"
form~Controls~Add(contentPane) -- add the contentPane to the form

contentPane~Controls~Add(textbox) -- add textbox to the contentPane (therefore adding it to the form too)
contentPane~Controls~Add(startButton) -- add the button to the contentPane (therefore adding it to the form too)

userData = .directory~new -- create a new Rexx Directory
userData~TextBox = textbox -- add the textbox to the directory
userData~startButton = startButton -- add the button to the directory

mouseEventHandler = clr.createEventHandler(.MouseEventHandler~new(userData))  -- create new event handler from ooRexx class "MouseEventHandler" with the directory "userData" as parameter
startButton~Click += mouseEventHandler  -- register event handler to "Click" event

```

```
application = clr.import("System.Windows.Forms.Application")  -- get reference to static
    class "System.Windows.Forms.Application"
application~Run(form)  -- invoke method "Run" to start the form


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

/* Class:         MouseEventHandler
 * Description:   Instantiates the ooRexx "Save" class with the passed userData as
    parameter, which is a thread, therefore starting it with the "Start" method.
 */
::CLASS MouseEventHandler

  /* Method:      init
   * Description: This constructor saves the passed arguments in the class.
   * Arguments:   - userData: an ooRexx directive, which can reference several form
    elements
   */
  ::METHOD init
    EXPOSE userData
    USE ARG userData

  /* Method:      invoke
   * Description: Called by .NET when the registered event is triggered.
   * Arguments:   - caller: the object the event handler was registered to
   *              - mouseEventArgs: the event arguments passed by the caller
   */

  ::METHOD invoke
    EXPOSE userData
    USE ARG caller, mouseEventArgs
          .Save~new(userData)~start -- start the "Save" class, which is a Thread



::CLASS Save SUBCLASS CLRThread
  ::METHOD init
    EXPOSE userData
    USE ARG userData

  /* Method:      run
   * Description: This method will get executed upon start of the thread by invoking
    "start" method of the CLRThread class, which is a superclass of this class.
   */
  ::METHOD run
    EXPOSE userData        -- expose must be the first command after invocation of a
    method

    textbox = userData~TextBox -- get a (shorter) reference to the textbox instance
    SaveFileDialog = .clr~new("System.Windows.Forms.SaveFileDialog") -- get an instance of
    the SaveFileDialog class
    SaveFileDialog~Filter = "all files|*.*" -- Syntax: [Label]|[Regular Expression]
    SaveFileDialog~FileName = "12-savefile.txt" -- this provides a default value as the
    file name, which can be overwritten in runtime

    DialogResult = clr.import("System.Windows.Forms.DialogResult") -- import static
    DialogResult class in order to check where the user clicked later on
    SaveFileDialog.Result = SaveFileDialog~ShowDialog -- show the dialog and save where
    the user clicked in the variable "SaveFileDialog.Result"

    IF SaveFileDialog.Result~equals(DialogResult~OK) THEN DO
  FileStream = .clr~new("System.IO.StreamWriter", SaveFileDialog~OpenFile) -- open a
filestream based on the filename/filepath+filename from the Save Dialog
  FileStream~Write(textbox~Text) -- write the text from the textbox into that
filestream file
  FileStream~Close
  MessageBox = clr.import("System.Windows.Forms.MessageBox")
  MessageBox~Show("finished saving process", "success") -- show the user that saving
the file was successfull
END
```

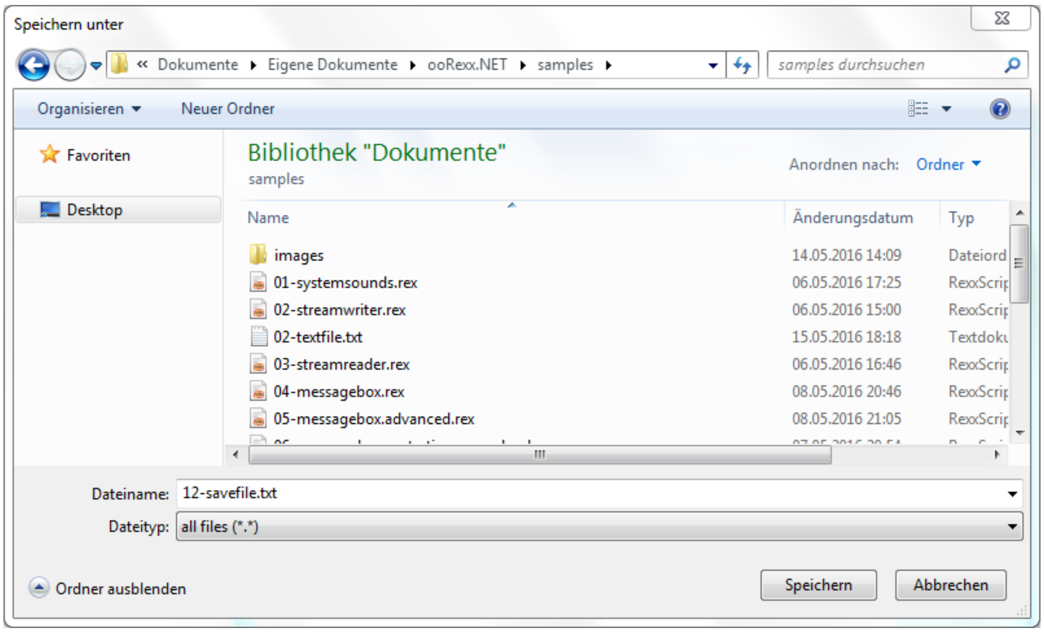
Listing 14: A save file dialog ooRexx.NET



This is my text

Figure 23: The form of 12-savefile.rxj

The "SaveFileDialog" opens when clicking on the "Save" button.



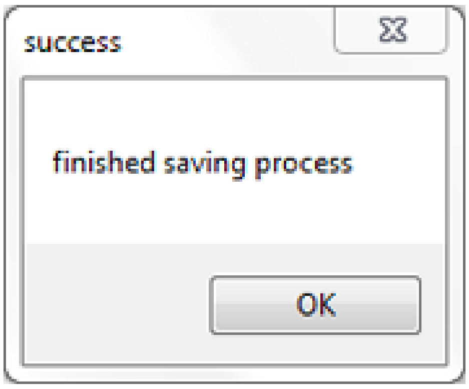
Name	Änderungsdatum	Typ
images	14.05.2016 14:09	Dateiord
01-systemsounds.rex	06.05.2016 17:25	RexxScrip
02-streamwriter.rex	06.05.2016 15:00	RexxScrip
02-textfile.txt	15.05.2016 18:18	Textdoku
03-streamreader.rex	06.05.2016 16:46	RexxScrip
04-messagebox.rex	08.05.2016 20:46	RexxScrip
05-messagebox.advanced.rex	08.05.2016 21:05	RexxScrip

Dateiname: 12-savefile.txt

Dateityp: all files (*.*)

Figure 24: The save file dialog of 12-savefile.rxj

If saving was successful, the user gets notified via a message box in Figure 25.


success

finished saving process

Figure 25: The confirmation dialog which pops up after saving the file in 12-savefile.rxj

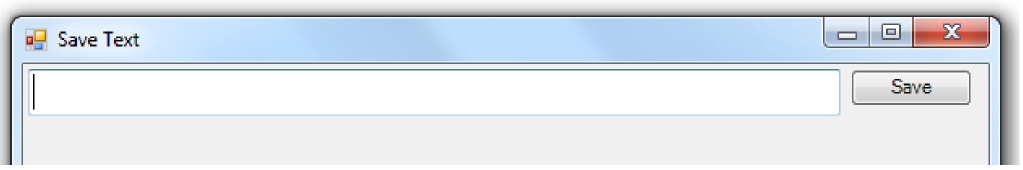


Figure 26: An alternative output when disabling attribute "multiline" of the form in 12-savefile.rxj

The dialog in Figure 24 basically asks the user to select a folder and provide a file name. After clicking on the "save" button of the "save file" dialog it returns the absolute path of the file name without creating that file in Line 94. What happens with that file path is up to the application. 12-savefile.rxj uses the given file path and file name to create a new text file using the `SteamWriter` class of `System.IO` namespace in Line 98.
Interestingly, the `"Multiline"` attribute defined in Line 18 is not bound to the height of the TextBox. Disabling it will result in a single-lined text box absolutely ignoring the height attribute. However, even when setting the size of the TextBox with the `"size"` attribute, not `"width"` and `"height"`, a disabled `"multiline"` attribute will have no impact on the width.
The Rexx constructor used in both ooRexx classes `"MouseEventHandler"` and `"Save"` is a method called "init", which stands for "initialize" and is called directly after instantiating the class. Usually there exist several constructors with different parameters, so that the reader has a choice whether the reader wants to provide information about that class at time of class creation or not.

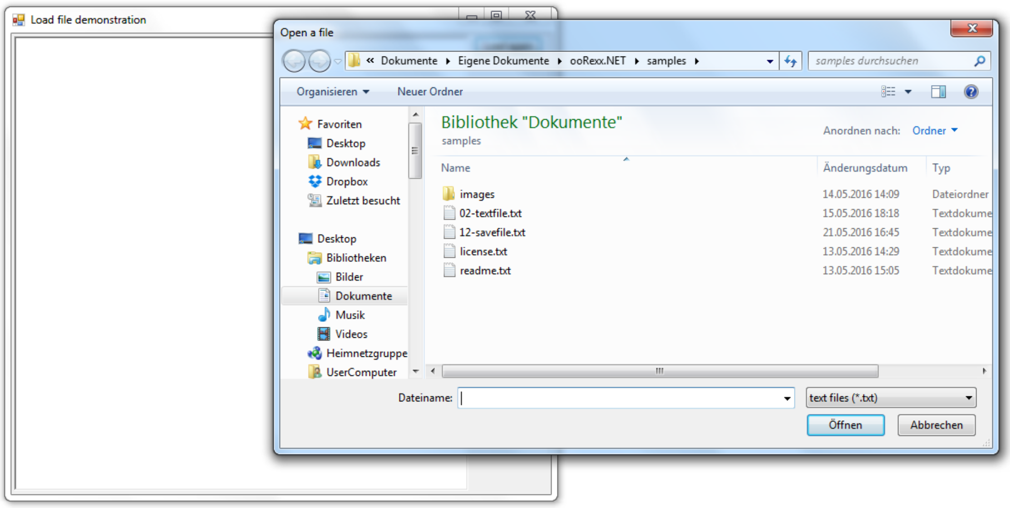
13 13-loadfile.rxj

This application shows the "OpenFileDialog" where the user can select a text file he wants to open.

```
/* File:        13-loadfile.rxj
 * Description: This application opens a form with a textarea and a "load" button to load
    a text file and display it in the textarea.
 * Shows:
 *              - Using a RichTextBox in contrast to the standard TextBox
 *              - Enabling shortcuts in a (rich)textbox
 *              - The Form.Load Event in ooRexx.NET used for executing commands after
    loading the form
 *              - Filtering file names and file types based on regular expressions
 */


form = .clr~new("System.Windows.Forms.Form")
form~AutoSizeMode = GrowAndShrink
defaultFormText = "Load file demonstration"
form~text = defaultFormText
form~width = 610
form~height = 545
textbox = .clr~new("System.Windows.Forms.RichTextBox")
textbox~Multiline = .true
textbox~AcceptsTab = .true
textbox~ShortcutsEnabled = .true -- this attribute allows amongst other things ctrl+a to
    select everything
textbox~size = .clr~new("System.Drawing.Size", 500, 500)
font = .clr~new("System.Drawing.Font", "Consolas", 14)
textbox~font = font

startButton = .clr~new("System.Windows.Forms.Button")
startButton~Text = "Load again"  -- set property "Text" to string "Load again"

contentPane = .clr~new("System.Windows.Forms.FlowLayoutPanel")
contentPane~AutoSize = .true  -- set property "AutoSize" to boolean true
contentPane~AutoSizeMode = GrowAndShrink  -- set property "AutoSizeMode" to enumeration
    value "GrowAndShrink"
form~Controls~Add(contentPane)

contentPane~Controls~Add(textbox)
contentPane~Controls~Add(startButton)

userData = .directory~new
userData~form = form -- add the form to the directory
userData~TextBox = textbox
userData~startButton = startButton
userData~defaultFormText = defaultFormText -- also include the default form text of this
    application to prevent redundancy
mouseEventHandler = clr.createEventHandler(.MouseEventHandler~new(userData))  -- create
    new event handler from ooRexx class "MouseEventHandler"
startButton~Click += mouseEventHandler  -- register event handler to "Click" event
form~Load += mouseEventHandler -- Occurs before a form is displayed for the first time.
    This will open the "load file" dialog when starting the application.

application = clr.import("System.Windows.Forms.Application")  -- get reference to static
    class "System.Windows.Forms.Application"
application~Run(form)  -- invoke method "Run", which starts an application message loop


```

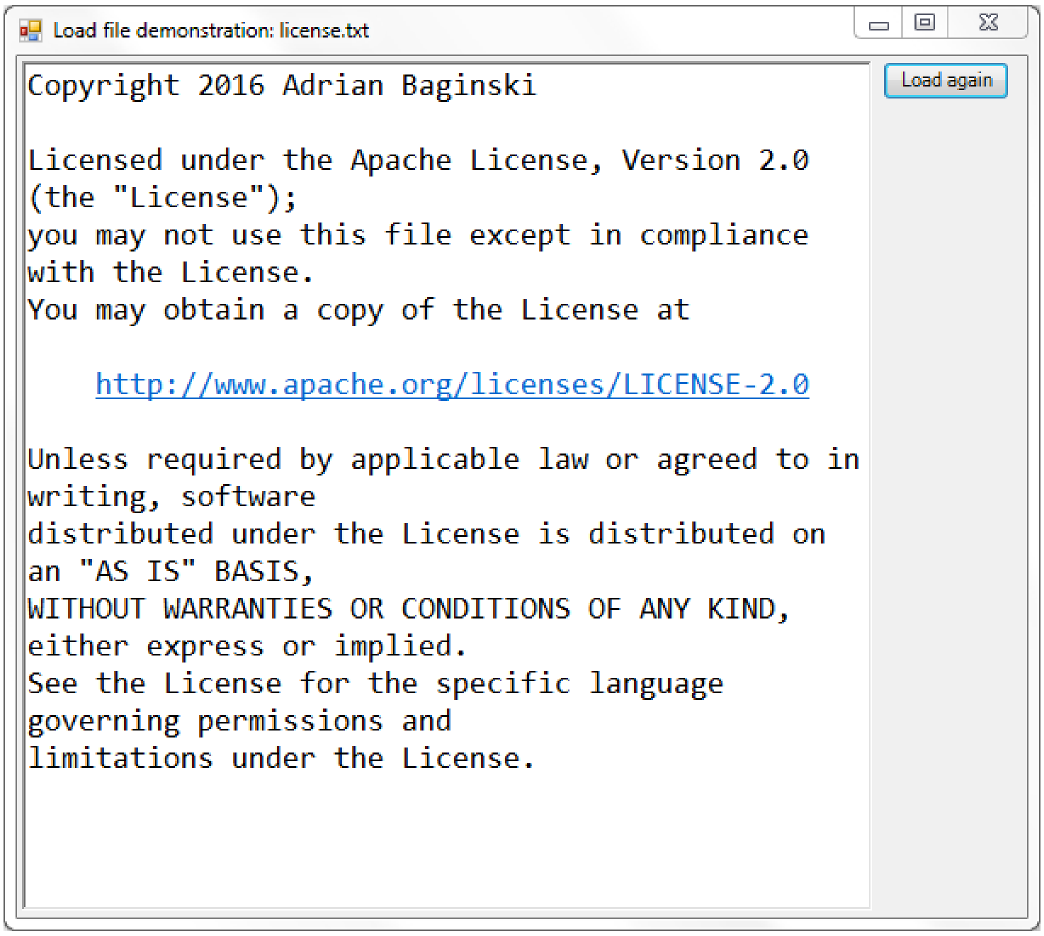
```
::REQUIRES CLR.CLS  -- get ooRexx.NET support


::CLASS MouseEventHandler

  /* Method:      init
   * Description: Constructor which saves the passed arguments in the class.
   * Arguments:   - progressBar: the progress bar which is to be modified
   *              - startButton: the start button which is to be modified
   */
  ::METHOD init
    EXPOSE userData
    USE ARG userData

  /* Method:      invoke
   * Description: Called by .NET when the registered event is triggered.
   * Arguments:   - caller: the object the event handler was registered to
   *              - mouseEventArgs: the event arguments passed by the caller
   */
  ::METHOD invoke
    EXPOSE userData
    USE ARG caller, mouseEventArgs
  .Save~new(userData)~start



::CLASS Save SUBCLASS CLRThread
  ::METHOD init
    EXPOSE userData
    USE ARG userData

  /* Method:      run
   * Description: Executed upon start of the thread (by invoking "start").
   */
  ::METHOD run
    EXPOSE userData
    fileContent = loadFile()
    IF fileContent <> .false THEN DO
      userData~textbox~Text = fileContent[1] -- insert the file content to the textarea
      userData~form~Text = userData~defaultFormText || ":" fileContent[2] -- add the file name (including file extension) to the title of the form
    END

/*
 * ROUTINE:       loadFile
 * Description:   opens a "open file" dialog to select a text or rexx file
 * Returns:       either an array with file content and file name or the ooRexx value ".false" / 0
 */

::ROUTINE loadFile
  OpenFileDialog = .clr~new("System.Windows.Forms.OpenFileDialog")
  DialogResult = clr.import("System.Windows.Forms.DialogResult")
  OpenFileDialog~Filter = "text files|*.txt|rexx files|*.rex" -- Syntax: Label|Regular Expression
  OpenFileDialog~Title = "Open a file" -- set the  title of the OpenFileDialog

  OpenFileDialog.Result = OpenFileDialog~ShowDialog
  IF OpenFileDialog.Result~equals(DialogResult~OK) THEN DO
    FileStream = .clr~new("System.IO.StreamReader", OpenFileDialog~OpenFile)
    FileContent = FileStream~ReadToEnd

    IF FileContent <> .nil THEN DO
      FileStream~Close -- only close the filestream if the content is OK
      FileName = OpenFileDialog~SafeFileName -- retrieve only the file name without file
    path
      returnArray = .array~of(FileContent, FileName)
      RETURN returnArray
    END
    ELSE
      SAY "FileContent is empty :-( Try again!"
  END
  ELSE
    RETURN .false
```

Listing 15: A open file dialog ooRexx.NET



Name	Änderungsdatum	Typ
images	14.05.2016 14:09	Dateiordner
02-textfile.txt	15.05.2016 18:18	Textdokume
12-savefile.txt	21.05.2016 16:45	Textdokume
license.txt	13.05.2016 14:29	Textdokume
readme.txt	13.05.2016 15:05	Textdokume

Figure 27: The form of 13-loadfile.rxj also opens a "open file" dialog



```
Copyright 2016 Adrian Baginski

Licensed under the Apache License, Version 2.0
(the "License");
you may not use this file except in compliance
with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in
writing, software
distributed under the License is distributed on
an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
either express or implied.
See the License for the specific language
governing permissions and
limitations under the License.
```

Figure 28: A huge benefit of RichTextBoxes: They also display http links as real clickable links

This application offers to open a file directly at the start using a `"Form.Load"` event from the form class.
In the previous example there was a huge disadvantage of the "TextBox" class. It would not allow for shortcuts like ctrl+a to select the whole text. That is why we use the `"RichTextBox"` class in this example, because it has always multiple lines - so setting the `"multiline"` attribute will not have any impact on the output of the application. Therefore there is no `"AcceptsReturn"` attribute in the `RichTextBox` class, which allows the user for creating line breaks. Setting it to a boolean (or any other) value like in the previous example will always result in a runtime error when executing the application. A complete list of all available shortcuts can be found on the corresponding Microsoft website - see [15, 16]

14 14-menu.rxj

This application opens a form with a menu bar including multiple menu items.

```
/* File:        14-menu.rxj
 * Description: This application opens a form with a menu bar, a menu inside a menu and a
    button at the bottom of the form.
 * Shows:
 *              - Using a FlowLayoutPanel to group together Control Elements.
 *              - The "Dock" attribute to change the position of Control Elements
    dynamically.
 *              - Using the "caller" attribute when invoking event handlers to determine
    the clicked element
 *              - Creating a border, background color and background image for form
    elements
 *              - Creating a Menu
 *              - Creating a SubMenu
 *              - How to assign a shortcut to a menu item.
 */

form = .clr~new("System.Windows.Forms.Form")
form~text = "Menu Class"
form~AutoSizeMode = GrowOnly
FormStartPosition = clr.import("System.Windows.Forms.FormStartPosition")
form~StartPosition = FormStartPosition~CenterScreen -- start the form in the center of the
    screen

endButton = .clr~new("System.Windows.Forms.Button")
endButton~Text = "Exit Application"  -- set property "Text" to string "Exit Application"
endButton~Width = 275

contentPane = .clr~new("System.Windows.Forms.FlowLayoutPanel") -- represent a
    FlowLayoutPanel to lay out the content horizontally
contentPane~width = form~width -- set the width of the contentPane to the width of the
    form
contentPane~height = 30 -- default height of a button is 24px, so set the contentPane to a
    little more than that: 30px

BorderStyle = clr.import("System.Windows.Forms.BorderStyle")
contentPane~BorderStyle = BorderStyle~FixedSingle -- add a single dimensional border to
    the contentPane

DockStyle = clr.import("System.Windows.Forms.DockStyle")
contentPane~Dock = DockStyle~Bottom -- move the contentPane including all its components
    (the button) to the bottom of the form by docking it


Color = clr.import("System.Drawing.Color") -- get a reference of the static
    System.Drawing.Color class
contentPane~BackColor = Color~LightGray -- set the background color of the contentpane to
    a predefined light gray color
form~BackColor = Color~White
endButton~BackColor = Color~Lavender -- Lavender seems to be an even lighter gray


bgImage = .clr~new("System.Drawing.Bitmap", "images/oorexx_logo.gif") -- import the
    oorexx_logo image as a bitmap
form~backgroundImage = bgImage -- apply the imported background image to the form
bgImageLayout = clr.import("System.Windows.Forms.ImageLayout") -- get a reference to the
    static System.Windows.Forms.ImageLayout class
```

```
form~backgroundImageLayout = bgImageLayout~Center -- set the background image to the center of the form

contentPane~Controls~Add(endButton) -- add the button to the contentPane
form~Controls~Add(contentPane) -- add the contentPane to the form

application = clr.import("System.Windows.Forms.Application")  -- get reference to static class "System.Windows.Forms.Application"

userData = .directory~new
userData~form = form
userData~application = application

mouseEventHandler = clr.createEventHandler(.MouseEventHandler~new(userData))  -- create new event handler from ooRexx class "mouseEventHandler"
endButton~Click += mouseEventHandler  -- register event handler to "Click" event

menu = .clr~new("System.Windows.Forms.MainMenu") -- get an instance of the MainMenu class
menuItemFile = .clr~new("System.Windows.Forms.MenuItem", "File") -- create a new MenuItem with the title "File"
menuItemHelp = .clr~new("System.Windows.Forms.MenuItem", "Help") -- create a new MenuItem with the title "Help"
shortcut = clr.import("System.Windows.Forms.Shortcut") -- get access to static class "System.Windows.Forms.Shortcut"
menuItemHelp~Shortcut = shortcut~F1 -- give menu item "Help" the shortcut "f1"
menuItemHelp~Click += clr.createEventHandler(.showHelp~new) -- if this menu item gets activated, create an event handler and get an instance of the showHelp class

menuItemClose = .clr~new("System.Windows.Forms.MenuItem", "Close")
menuItemClose~Shortcut = shortcut~AltF4
menuItemClose~Click += clr.createEventHandler(.MouseEventHandler~new(userData))  -- register event handler to "Click" event. This menu item does the same as the endButton

openFile = .clr~new("System.Windows.Forms.MenuItem", "Open File") -- this is a menu item to demonstrate menus for menus, e.g. submenus
saveFile = .clr~new("System.Windows.Forms.MenuItem", "Save File")
menuItemFile~MenuItems~~Add(openFile)~~Add(saveFile)

possibilities = .directory~new
possibilities~open = openFile
possibilities~save = saveFile

openFile~Click += clr.createEventHandler(.WhoAmI?~new(possibilities))
saveFile~Click += clr.createEventHandler(.WhoAmI?~new(possibilities))

menu~MenuItems~Add(menuItemFile)
menu~MenuItems~Add(menuItemHelp)
menu~MenuItems~Add(menuItemClose)
form~Menu = menu -- set our menu as the Menu of the form


application~Run(form)  -- invoke method "Run", which starts an application message loop



::REQUIRES CLR.CLS

/* Class:         MouseEventHandler
 * Description:   Creates an instance of the ooRexxx "ExitApplication" class and starts it
 * Argument:      A directory with the form, which has to be closed.
 */

```

```
::CLASS MouseEventHandler
  ::METHOD init
    EXPOSE userData
    USE ARG userData

  ::METHOD invoke
    EXPOSE userData
    USE ARG caller, mouseEventArgs
    .ExitApplication~new(userData)~~start


/* Class:         ExitApplication
 * Description:   This class closes a form.
 * Argument:      A directory with the form, which has to be closed.
 */

::CLASS ExitApplication SUBCLASS CLRThread
  ::METHOD init
    EXPOSE userData
    USE ARG userData

  ::METHOD run
    EXPOSE userData

    MessageBox = clr.import("System.Windows.Forms.MessageBox") -- get access to static class "MessageBox" of namespace "System.Windows.Forms"
    MessageBoxButtons = clr.import("System.Windows.Forms.MessageBoxButtons") -- get access to a set of enumerations

    MessageBox.Result = MessageBox~show("Do you really want to close this applicaton?", "Exit application", MessageBoxButtons~YesNo) -- display MessageBox and save result in Rexx variable "MessageBox.Result"
    DialogResult = clr.import("System.Windows.Forms.DialogResult") -- get a reference to DialogResult enumerations

    IF MessageBox.Result~equals(DialogResult~Yes) THEN DO
      form = userData~form
      form~Close
    END

/* Class:         showHelp
 * Description:   Creates an instance of the ooRexxx "runHelp" class, which is a Thread, and starts it
 */

::CLASS showHelp
  ::METHOD invoke
    USE ARG caller, mouseEventArgs
    .runHelp~new~start


/* Class:         runHelp
 * Description:   This class opens a MessageBox with a hint
 */

::CLASS runHelp SUBCLASS CLRThread
  ::METHOD run
    MessageBox = clr.import("System.Windows.Forms.MessageBox")
    MessageBox~show('hint: press "f1" to access this help box or "alt + f4" to close the application', "Help")


```

```
/* Class:         WhoAmI?
 * Description:   This class validates the clicked element based on the parameter and
    outputs a MessageBox with the name of the clicked element.
 * Argument:      A directory with all available menu items whose click event handler
    target is the "WhoAmI?" class.
 */

::CLASS WhoAmI?
  ::METHOD init
    EXPOSE bothItems MessageBoxTitle
    USE ARG bothItems
    MessageBoxTitle = "WhoAmI: Result" -- set the title to be used later on

  ::METHOD invoke
    EXPOSE bothItems MessageBoxTitle
    USE ARG caller, mouseEventArgs
    saveItem = bothItems~save -- retrieve the reference to the menu item "save"
    openItem = bothItems~open

    MessageBox = clr.import("System.Windows.Forms.MessageBox")
    IF caller~toString = saveItem~toString THEN -- convert both objects into a string
    representation and check for equality
      MessageBox~show("You clicked on the ""save"" menu item!", MessageBoxTitle)
    ELSE IF caller~toString = openItem~toString THEN
      MessageBox~show("You clicked on the ""open"" menu item!", MessageBoxTitle)
```

Listing 16: Using menus and submenus in ooRexx.NET

After executing this application this form will open providing a menu bar, a background image and a button to exit the application as seen in Figure 29.

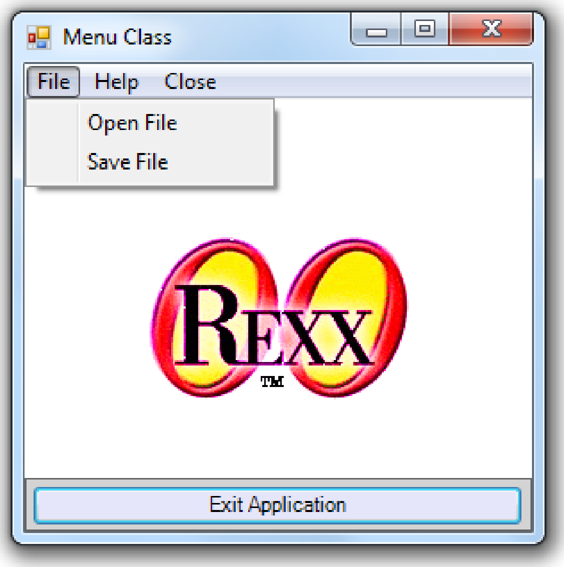
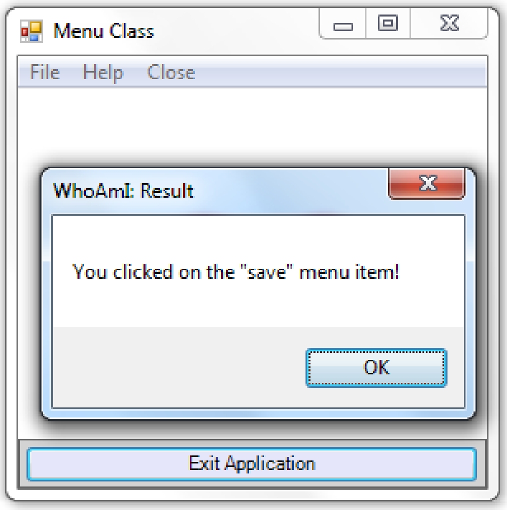


Figure 29: This image shows the form of the example 14-menu.rxj

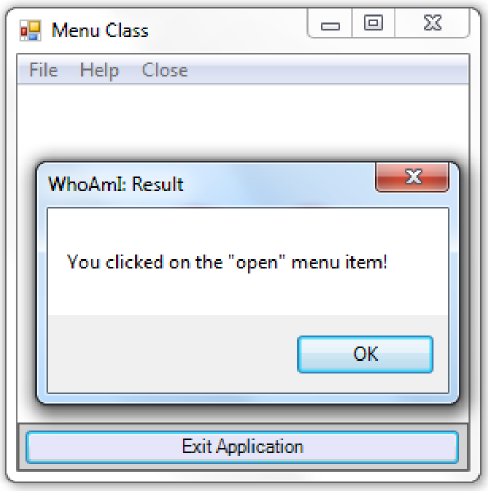
The message box from Figure 30 will pop up if "File > Save" gets clicked. The clicked element gets identified by the "caller" argument in Lines 169-172.



You clicked on the "save" menu item!

Figure 30: This message box invoked in the `WhoAmI?` class.

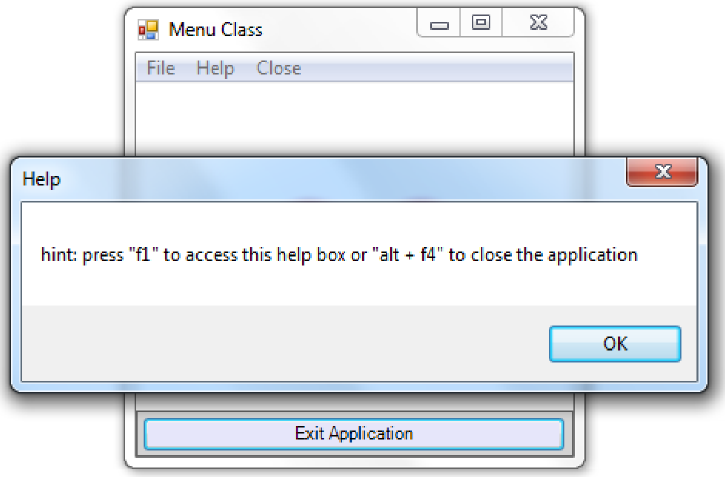
The message box from Figure 31 will pop up if "File > Open" gets clicked.



You clicked on the "open" menu item!

Figure 31: Another message box invoked in the `WhoAmI?` class.

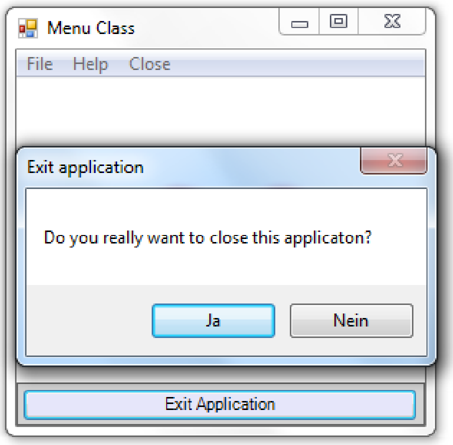
The message box in figure 32 provides some hints for this application. It gets activated when the user clicks on "Help" in the menu bar or presses "f1" on the keyboard.



hint: press "f1" to access this help box or "alt + f4" to close the application

Figure 32: The output of the class `"runHelp"` in Lines 145-148

Finally, this message box confirms the "Exit Application" button at the bottom and the "Exit" menu item in the menu. The application will close once the user confirms by pressing on the "Yes" button



Do you really want to close this applicaton?

Figure 33: The output of the class "ExitApplication" in Lines 112-129.

This application primarily demonstrates menus and shows how much code they actually need to perform well. Including a menu bar in an application are just two lines of code - the first being the import of the main menu bar itself, see Line 57 and the second one is embedding the menu bar in the form. Then, each menu item requires at least two additional lines of code as shown in Line 58 and 79. Of course, menu items have a specific purpose, for example calling a routine or instantiating a class. That requires adding event handlers which add several lines of code. Aside from the many attributes that can be changed in the menu bar and the menu items it can get really confusing when creating applications with one or even more menus. The author therefore advises to use additional files to keep the structure inside the source code of an application. It is possible to include other Rexx files in an application with the `::REQUIRES` directive. This way, menu bars can stay the way they are and yet the programmer has no problem to keep an overview over his code.
Another useful and easy-to-implement thing this application shows are lines 40 to 43, which add a background image to the form. First, the image gets imported as a bitmap from the file path. The "BackgroundImageLayout" property of the form class determines where the image will be displayed. Unfortunately, this property does not allow for much creativity, since it expects a static enumeration value of type "System.Windows.Forms.ImageLayout". The possibilities of this enumeration are quite limited as it for example does not allow for the background image to display in the upper right corner of the

form. Actually, the background image cannot be displayed in **any** corner except the upper left, which is equivalent to a Point with the coordinates 0 and 0. So this is quite a limitation but there are multiple solutions for this problem. A background image can be attached to most of the form elements in .NET. Therefore it is possible to set a content pane with the background image to dock to the upper right corner of the form. Alternatively, one could load the image into a new bitmap with the exact size of the form, just like in example 11-drawing.rxj on page 37.
However, images are a great addition to any application and are worth their troubles, just like menus are. [17]

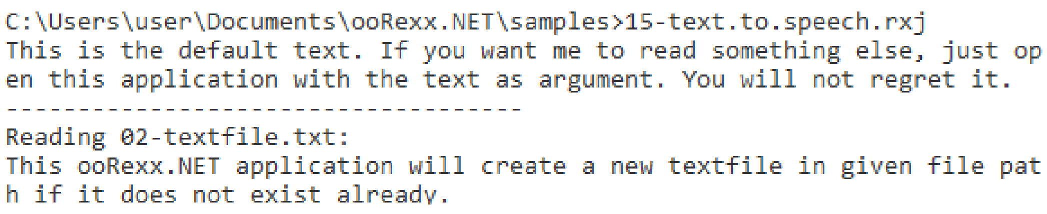
15 15-text.to.speech.rxj

This application shows one of the many great advantages of the .NET framework, a built-in speech synthesis engine. This feature generates audio output based on textual input by analysing linguistic components like phasing, intonation and duration.

```
/* File:        15-text.to.speech.rxj
 * Description: This application first reads a default text, then reads the content of "02-textfile.txt" word by word ignoring the first line.
 * Shows:
 *              - Adding an Assembly
 *              - Regulating volume and rate of the audio output
 *              - Reading a string
 *              - Extracting words of a textfile
 */

CALL clr.addAssembly "System.Speech" -- load System.Speech Assembly to use all System.Speech.* classes

Console = clr.import("System.Console")
SpeechSynthesizer = .clr~new("System.Speech.Synthesis.SpeechSynthesizer") -- get an instance of the SpeechSynthesizer class to use all text-to-speech (TTS) functions
SpeechSynthesizer~SetOutputToDefaultAudioDevice
SpeechSynthesizer~Volume = 100 -- regulate the volume of the voice
SpeechSynthesizer~Rate = -1 -- regulate the speed between -10 and 10.

TextToRead = "This is the default text. If you want me to read something else, just open this application with the text as argument. You will not regret it."
IF arg(1)~length > 0 THEN TextToRead = arg(1)

Console~WriteLine(TextToRead)
SpeechSynthesizer~Speak(TextToRead)

CALL SysSleep 1 -- wait a second

filepath = "02-textfile.txt"
mystream = .stream~new(filepath) -- load a new filestream using ooRexx STREAM Class
mystream~~linein -- ignore first line, which is the heading with a lot of tildes
SpeechSynthesizer~Rate = 1 -- speed the rate of the speaker up a little bit

word = "" -- set the variable "word" to a 0 length string

SAY "-------------------------------------"
Console~WriteLine("Reading" filepath || ":")
SpeechSynthesizer~Speak("Reading")
SpeechSynthesizer~Rate = -6 /*  change the rate to -6 because the filepath consists of a lot of points and slashes.
    Still, sounds weird because she says "zero two to textfile" instead of "zero two textfile".
    therefore, we will now remove the slash:
    */
posOfSlash = filepath~pos("-") -- get the position of the slash inside the filepath
subStrBeforeSlash = filepath~left(posOfSlash - 1) -- retrieve text from position 0 to just before the slash
subStrAfterSlash = filepath~substr(posOfSlash + 1) -- retrieve the text starting at position of slash + 1 (so we do not include the slash in this string) until the end of the variable "filepath"
newSpeakableFilePath = subStrBeforeSlash subStrAfterSlash -- merge both variables

```

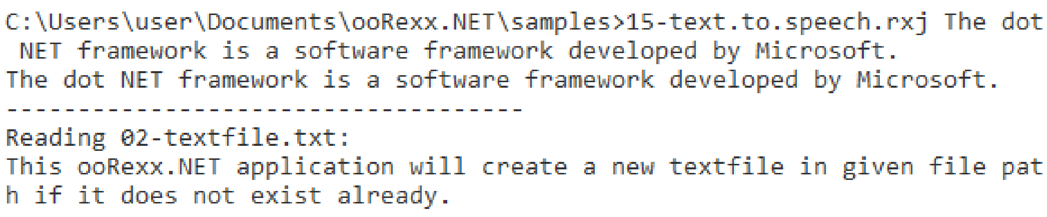
```
SpeechSynthesizer~Speak(newSpeakableFilePath) -- speak the new file path
SpeechSynthesizer~Rate = 1 -- get the rate back up to 1

DO WHILE mystream~chars > 0   -- read the file until it has reached its end
  line = myStream~linein        -- read an entire line from file
  DO i = 1 to words(line)       -- iterate over words
     word = word(line,i)        -- extract word
     Console~Write(word || " ")     -- display the word
     SpeechSynthesizer~Speak(word)  -- speak the word
  END
END

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

Listing 17: Creating a text-to-speech (TTS) application in ooRexx.NET



```
C:\Users\user\Documents\ooRexx.NET\samples>15-text.to.speech.rxj
This is the default text. If you want me to read something else, just op
en this application with the text as argument. You will not regret it.
------------------------------------
Reading 02-textfile.txt:
This ooRexx.NET application will create a new textfile in given file pat
h if it does not exist already.
```

Figure 34: This image shows the default output of 15-text.to.speech.rxj



```
C:\Users\user\Documents\ooRexx.NET\samples>15-text.to.speech.rxj The dot
 NET framework is a software framework developed by Microsoft.
The dot NET framework is a software framework developed by Microsoft.
------------------------------------
Reading 02-textfile.txt:
This ooRexx.NET application will create a new textfile in given file pat
h if it does not exist already.
```

Figure 35: This is another variation of this example. By supplying the application with an argument, it will override the default text with the argument.

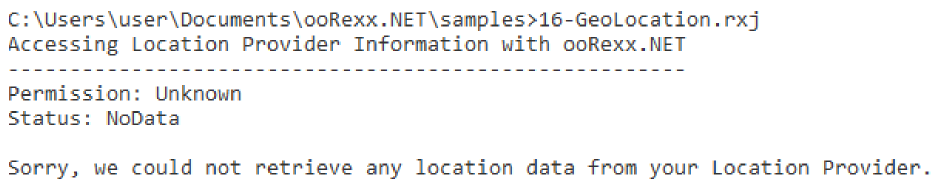
Any programming language with access to the .NET framework has access to this easy-to-use text-to-speech support provided by the framework. The needed class is "SpeechSynthesizer" from the namespace `"System.Speech.Synthesis"`. It does not only provide a lot of customization options like the `"rate"` of the speaker and the `"volume"` of the audio output, but also provides the possibility to select a different speaker (or as Microsoft calls them: "Voice" [18]). For the full list of supported languages, see [19, 20, 21].

16 16-GeoLocation.rxj

This application demonstrates how to extract the location data of the user by using the "System.Device.Location" namespace of the "System.Device" Assembly. It provides programmers a couple of classes using only one application programming interface (API) - independent of the source of the location data, which Microsoft refer to as the "Location Provider".

```
/* File:        16-GeoLocation.rxj
 * Description: This application tries to access location data on the device and then
    computes the location to an adress.
 * Shows:
 *              - Setting up a "GeoCoordinateWatcher" based on a high accuracy level
 *              - Accessing permission and status of the GeoCoordinateWatcher class
 *              - Converting ("Resolving") a GeoCoordinate to the exact civic adress
 */

CALL clr.addAssembly("System.Device") -- add System.Device Assembly to use all classes
    which have a "System.Device.*" namespace

lf = "0A"x
GeoPositionAccuracy = clr.import("System.Device.Location.GeoPositionAccuracy")
GeoCoordinateWatcher = .clr~new("System.Device.Location.GeoCoordinateWatcher",
    GeoPositionAccuracy~High) -- use the most accurate position possible
GeoCoordinateWatcher~clr.dispatch("Start") -- start tracking the location

jumpcounter = 0 -- set jumpcounter to 0
jumpspot:
SAY "Accessing Location Provider Information with ooRexx.NET"
SAY "-----------------------------------------------------"
SAY "Permission:" GeoCoordinateWatcher~Permission~toString
SAY "Status:" GeoCoordinateWatcher~Status~toString lf
GeoPositionStatus = clr.import("System.Device.Location.GeoPositionStatus") -- get access
    to GeoPositionStatus Enumeration

IF GeoCoordinateWatcher~Status~equals(GeoPositionStatus~Ready) THEN DO
  GeoPosition = GeoCoordinateWatcher~Position -- everything is set, retrieve location data
  AdressResolver = .clr~new("System.Device.Location.CivicAddressResolver")
  GeoCoordinate = GeoPosition~Location

  Adress = AdressResolver~ResolveAdress(GeoCoordinate) -- get the adress based on the
    information in GeoCoordinate
  SAY lf"Your adress:" Adress~AdressLine1~toString
  SAY "Your building:" Adress~Building~toString
  SAY "Your postal code:" Adress~PostalCode~toString
  SAY "Your state or province:" Adress~StateProvince~toString
END
ELSE IF GeoCoordinateWatcher~Status~equals(GeoPositionStatus~NoData) THEN DO
  SAY "Sorry, we could not retrieve any location data from your Location Provider."
END
ELSE IF GeoCoordinateWatcher~Status~equals(GeoPositionStatus~Disabled) THEN DO
  SAY "Sorry, the Location Provider is disabled. Unfortunately, your device does not
    support Location Services."
END
ELSE IF GeoCoordinateWatcher~Status~equals(GeoPositionStatus~Initializing) THEN DO
  SAY "Your Location Provider is still initializing."
  SAY "Trying again ..."
  CALL SysSleep 3
  jumpcounter += 1 -- count jumpcounter up
  IF jumpcounter = 5 THEN EXIT -- if jumpcounter has reached 5 (e.g. after 4 jumps) exit the application to prevent unnecessary infinity loops
  CALL jumpspot -- retry after 3 seconds
END

::REQUIRES CLR.CLS
```

Listing 18: Retrieving the geographical location of the user with 16-GeoLocation.rxj



```
C:\Users\user\Documents\ooRexx.NET\samples>16-GeoLocation.rxj
Accessing Location Provider Information with ooRexx.NET
-------------------------------------------------------
Permission: Unknown
Status: NoData

Sorry, we could not retrieve any location data from your Location Provider.
```

Figure 36: This image shows the output of 16-GeoLocation.rxj when access to the location is not granted there is no GPS sensor installed.

The location may come from a GPS sensor, a wifi spot sharing that information or other sources. Programmers using the .NET framework do not have to worry about how they get the location data, they just do. Unfortunately, it is necessary to have some kind of location sensor installed on the computer, preferably a GPS sensor. The output of status "NoData" in line 21 of Listing 17 indicates that there is no such sensor in the device.
Alternatively, the status "Disabled" would mean that there is a location sensor in the device, but it is deactivated and thus needs some action from the user. That is why it is very important to check the status of the "GeoCoordinateWatcher" class instance for all possibilities as shown in this application. Still, this classes with this namespace is of special value for creating applications for mobile devices, because they are more likely to have the needed hardware on-board.
In the case of mobile devices the event "PositionChanged" becomes very interesting. This event gets invoked each time the latitude or longitude of the user changes. [22]

III Summary, Outlook and Future Research

oorexx.NET is an easy way for interacting with .NET objects and classes by using Open Object Rexx and BSF4ooRexx. It can be combined with Java classes because CLR.CLS already requires BSF.CLS - therefore using both .NET and Java classes will generate no additional loading times or lose efficiency in any way. Each and every programming language and framework has its own benefits like Java's platform independence or .NET's cryptography classes and built-in speech synthesiser. Though using those advantages is great, combining multiple languages and engines to maximise the outcome of the application is even better. Once ooRexx.NET gets distributed with BSF4ooRexx in the fall of 2016 after the beta phase, every ooRexx programmer can get his hands on the .NET framework.
As already stated in example "16-GeoLocation.rxj" on page 59 the author sees a great opportunity in developing applications for mobile devices, like smartphones or tablets. As of now mobile programming for Windows Phones offers most of the .NET classes and a few additional ones, which would not make any sense on desktop computers like screen-on time, or access to default applications like text messaging or calling. It would be very interesting to see someone develop mobile applications using an ooRexx.NET-related framework.
All in all the author very much enjoyed using ooRexx.NET for creating the applications for this bachelor thesis. Microsoft's documentation is great, although the loading times of the MSDN website were rather long, especially during business hours.

References

[1] "Wikipedia: Global assembly cache." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Global_Assembly_Cache&oldid=721831523

[2] W. D. Ashley, R. G. Flatscher, M. Hessling, R. McGuire, M. Miesfeld, L. Peedin, R. Tammer, and J. Wolfers, *Open Object Rexx: Programming Guide.* Rexx Language Association, 2009.

[3] R. G. Flatscher, *Introduction to REXX and ooRexx.* Facultas, 2013.

[4] M. Raffel, *ooRexx.NET- Bridging .NET and ooRexx.* WU, 2015.

[5] "Systemsounds class," Apr. 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.media.systemsounds%28v=vs.110%29.aspx

[6] "Console class," Apr. 2016. [Online]. Available: https://msdn.microsoft.com/de-de/library/system.console%28v=vs.110%29.aspx

[7] "Streamwriter class," Mar. 2016. [Online]. Available: https://msdn.microsoft.com/de-de/library/system.io.streamwriter.aspx

[8] "System.io.directory.getcurrentdirectory class," Mar. 2016. [Online]. Available: https://msdn.microsoft.com/de-de/library/system.io.directory.getcurrentdirectory(v=vs.110).aspx

[9] "Messageboxbuttons enumeration," Feb. 2016. [Online]. Available: https://msdn.microsoft.com/de-de/library/system.windows.forms.messageboxbuttons.aspx

[10] "Dialogresult enumeration," Feb. 2016. [Online]. Available: https://msdn.microsoft.com/de-de/library/system.windows.forms.dialogresult.aspx

[11] "Processstartinfo class," Jul. 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.diagnostics.processstartinfo(v=vs.110).aspx

[12] "System.security.cryptography namespace," Jul. 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.110).aspx

[13] “html5 introduction,” Jul. 2016. [Online]. Available: http://www.w3schools.com/html/html5_intro.asp

[14] “Standard numeric format strings,” May 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/dwhawy9k.aspx

[15] “Shortcutsenabled property,” May 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.textboxbase.shortcutsenabled.aspx

[16] “Openfiledialog class,” Apr. 2016. [Online]. Available: https://msdn.microsoft.com/de-de/library/system.windows.forms.openfiledialog%28v=vs.110%29.aspx

[17] “Imagelayout enumeration,” May 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.imagelayout.aspx

[18] “Speechsynthesiser.voice property.” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.speech.synthesis.speechsynthesizer.voice.aspx

[19] “Language support in tts engine of .net,” May 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh378476

[20] “Wikipedia article on speech synthesis,” May 2016. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Speechsynthesis&oldid=722000804

[21] “Download additional voices and add them to the tts engine of .net,” May 2016. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=27224

[22] “Windows 7 desktop without location sensor,” May 2016. [Online]. Available: http://stackoverflow.com/questions/24679099/windows-7-desktop-without-location-sensor