

# ooRexx 5.0 Beta

---

New Features and Nutshell Examples

Thomas Dabernig, h01451744

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

4167 – Seminar aus BIS

SS 2020

## ***Abstract***

The scripting language ooRexx is an easy to learn human-oriented programming language. Finally, in 2018 the most recent ooRexx 5.0 Beta version released. The new version offers several new features that provide the user with a number of benefits. The new features regard notations, directives, keywords, classes and built-in-functions. The benefits include improved functionality and performance.

# Table of contents

1	Introduction.....	1
2	Overview.....	2
2.1	The Language “ooRexx” .....	2
2.2	History .....	3
2.3	Changes “ooRexx 5.0beta” .....	3
2.4	Source Control.....	4
3	Notations.....	5
3.1	Array Notation Term .....	5
3.2	Namespaces .....	6
3.3	Variable Reference Notation Term.....	7
4	Directives .....	8
4.1	::ANNOTATION.....	8
4.2	::ATTRIBUTE.....	9
4.3	::CLASS.....	9
4.4	::CONSTANT.....	10
4.5	::OPTIONS .....	10
4.6	::RESOURCE.....	10
5	Keyword Instructions.....	12
5.1	ADDRESS .....	12
5.2	“DO” Keyword Instruction.....	12
5.3	SELECT CASE .....	14
5.4	USE LOCAL .....	15
6	Classes.....	17
6.1	AlarmNotification.....	17
6.2	EventSemaphore .....	18
6.3	Json .....	19
6.4	MessageNotification.....	19
6.5	MutexSemaphore.....	20
6.6	RexxInfo .....	21
6.7	StringTable .....	22
6.8	Ticker.....	23
6.9	Variable Reference .....	24
6.10	Validate.....	25
7	Other Changes.....	26

7.1	Built-in-functions.....	26
7.1.1	CONDITION() .....	26
7.1.2	DATATYPE() .....	27
7.1.3	DATE().....	27
7.1.4	XRANGE().....	28
7.2	Rexxutil .....	28
7.2.1	SysFormatMessage.....	29
7.2.2	SysGetLongPathName .....	29
8	Conclusion .....	30
	References .....	31

# 1 Introduction

The scripting language ooRexx is an easy to learn human-oriented programming language. Although its first version goes back to the early days of programming and the language is past its most popular days, it is still constantly reworked and improved.

The objectives of this paper are to present the most important features of the most recent ooRexx 5.0 beta version. The central focus lies on illustrating the changes and additions, which are briefly described. The features will be highlighted to give the reader a basic understanding. Additionally, a nutshell example is given to ease the understanding.

The paper consists of eight main chapters

Following the introduction, the second chapter provides general background information, including a brief overview and history of the language. Furthermore, the research method will be described, and it will be explained how changes and new features are documented. Each of the following chapters outlines one of the major concepts in the language and illustrates the changes implemented in the new version. The third chapter aims to highlight changes concerning the notation, the general writing rules and syntax of the language. In the fourth chapter directive instructions will be outlined. The following chapter is about keywords that are reserved words instructing the interpreter to execute certain commands. The language ooRexx is object-based, one of the key principles to implement the object-based system is classes and methods. These topics will be reviewed in the fifth chapter. The last chapter investigates several other changes. This includes built-in-functions and Rexx utility.

My research is mainly based on internet research, research publications and the ooRexx Documentation. Internet research includes Source Forge to get a first impression and general ideas and it also informs about the reason why the feature was implemented. Further internet research comprises various other articles and source code files. Research publications and the ooRexx Documentation are a good anchor point for general explanation of new features and they present nutshell examples, which makes it easy to get familiar with the topic.

## 2 Overview

### 2.1 *The Language “ooRexx”*

Open Object Rexx is a high-level message-based programming language. High level refers to the language being human-oriented with a very simple syntax. In consequence, the language is very easy to learn. It is one of the main design points of the Rexx language. Most instructions are in common English, in contrast to other languages like Java. There are several more issues that built upon the concept of human orientation.<sup>1</sup>

Firstly, it is case-insensitive, which makes it less prone to human error. Furthermore, variables need not to be declared and the language is kept intentionally small. As a result, the documentation is shorter, and the language is faster to learn. It is an interpreted language. In other words, it does not need to be compiled to be executed. The smaller overhead results in a faster run speed, reducing development time significantly.<sup>2</sup>

The message-based concept was taken over from Smalltalk. To send a message, the tilde character (~) is used and once received by the object, a method which has the same name as the message will be executed. In addition, arguments for the message can be given in parenthesis.<sup>3</sup>

As already indicated, ooRexx is object-oriented. General concepts of object-oriented programming include objects, classes, methods, modularity, abstraction and inheritance. There are several advantages associated with object-oriented programming scalability, adaptability and reuse of code blocks, which results in faster development times.<sup>4</sup>

ooRexx 5 offers a wide variety of in-built-functions and methods. One of the languages' strong points is its very good handling with strings. It is free and open source with a public source code. The Rexx Language Association manages its development and spread. ooRexx nearly runs on every system including Linux, Solaris and Windows. All these characteristics make it the perfect scripting language and macro language.<sup>5</sup>

The interface to system utilities and make the language very versatile. As a result, ooRexx is the perfect command programming language. In addition,

---

<sup>1</sup> (Cowlshaw, 1987, p. 331)

<sup>2</sup> (Cowlshaw, 1987, pp. 331-333) (Flatscher & Müller, 2019)

<sup>3</sup> (Flatscher & Müller, 2019, p. 2)

<sup>4</sup> (About RexxLA, 2020) (Object-oriented Programming Wikipedia, 2020)

<sup>5</sup> (About RexxLA, 2020) (Rexx Blog, 2020)

the quality of its character-manipulation designs it to be a good macro language.<sup>6</sup> (mike) Furthermore, new functions and features can be added over modules. One example for this is BSF4ooRexx.<sup>7</sup>

*“BSF4ooRexx, the "Bean Scripting Framework for ooRexx", is a Java bridge for ooRexx, allowing ooRexx to interact with Java classes and Java objects, as if they were ooRexx classes and ooRexx objects by requiring the supplied ooRexx package BSF.CLS.”*<sup>8</sup>

## **2.2 History**

Mike Cowlshaw developed the Rexx Restructured Extended Executor language to replace exec and exec-2 for IBM Mainframes. Rexx was later adopted for the desktop operating system by IBM. Due to object-oriented influence foremost by Smalltalk, an object-oriented Rexx was created and released at the end of the 1990s. After the Rexx Language Association had received the source code, ooRexx 3.0 was released in 2004. It became open source. In 2009 ooRexx 4.0 was made public. The kernel was rewritten, and a new native interface was implemented. In 2010 BSF4ooRexx was released. It bridges the gap between Javan and ooRexx, offering all the utility Java classes offer. Finally, in 2018 the most recent ooRexx 5.0 Beta version released.<sup>9</sup>

## **2.3 Changes “ooRexx 5.0beta”**

The 5.0 Beta version presents a lot of new improvements. It is a stable beta release, which offers better performance and stability. The new version is backwardly compatible with older ones. This means that older programs are still able to run on the new version. There are new functions, classes, methods and keywords. These offer new features and utilities that make life easier for the programmer. You do not need administrative rights anymore to install ooRexx. It is now possible to run ooRexx from a USB-stick or to run multiple interpreters on the same system at the same time. It is now possible to access the runtime environment with the local environment symbol.<sup>10</sup>

---

<sup>6</sup> (Cowlshaw, 1987)

<sup>7</sup> (About RexxLA, 2020)

<sup>8</sup> (Flatscher R. G., The New BSF4ooRexx 6.00, 2018, p. 1)

<sup>9</sup> (Flatscher & Müller, 2019, p. 1) (Rexx Wikipedia, 2020) (Flatscher R. G., Open Object Rexx Tutorial, 2017)

<sup>10</sup> (Flatscher & Müller, 2019)

## **2.4 Source Control**

The source files are distributed on Source Forge. It is a service to manage free open source software projects on the web. Its features include a source code repository, bug tracking and a wiki. For ooRexx the version control works over a ticket system. Bugs and requests for features can be submitted there and if approved, they will be implemented into ooRexx. There is a tracker to trace request history for new features in the forum. This makes it easy to look up the request content and the person who requested the feature.<sup>11</sup>

List of full changes:

<https://sourceforge.net/p/ooorexx/code-0/HEAD/tree/main/trunk/CHANGES>

Furthermore, the language ooRexx possesses a very detailed programming reference in which changes and additions are marked as new, \*NEW\*, or changed, \*CHG\*.

ooRexx 5.0.0 Reference:

<https://sourceforge.net/projects/ooorexx/files/ooorexx-docs/5.0.0beta/>

The examples were created with IntelliJ with the ooRexx plugin and executed in cmd with the rexxpaws command or with the pause command. Although, it is recommended to execute them directly in IntelliJ.

JetBrains IntelliJ IDEA Community Edition for windows:

<https://www.jetbrains.com/de-de/idea/download/#section=windows>

ooRexx plugin 2.0.0 for IntelliJ IDEA download:

<https://sourceforge.net/projects/bsf4ooorexx/files/Sandbox/aseik/ooRexxIDEA/GA/2.0.0/>

---

<sup>11</sup> (Source Forge Wikipedia, 2020)



## 3 Notations

Notations decide the syntax and the basic meaning of the symbols of a language. The following main changes have been made concerning notations.<sup>12</sup>

### 3.1 Array Notation Term

The new update introduces a new way to note down arrays. It is now possible to create arrays by using the delimiter comma on a list of items. If the values are already used by ooRexx or have a different meaning, the values should be put between brackets. It is now easier to create arrays with less writing needed. The items are indexed from the left to the right, as shown in the example. The generated array cannot be smaller than the size of 2, but it can have 0 items.<sup>13</sup>

Example:

```
-- new array notation
animals = "cat", "dog", "cow"

SAY "There are " animals~items "animals"

animals[2] = "new dog"
animals[5] = "old cat"

Supp = animals~SUPPLIER -- create Supplier Object

SAY
SAY "list of animals: "

DO with index x item y over Supp
SAY "index:" x " item:" y
END
```

14

Output:



---

<sup>12</sup> (Notation Wikipedia, 2020)

<sup>13</sup> (Flatscher & Müller, 2019, p. 2) (Ashley, et al., 2020, pp. 24-25)

<sup>14</sup> (Flatscher & Müller, 2019, p. 2) (Ashley, et al., 2020, p. 25) (Flatscher R. G., ooRexx 5.00 New Features, 2017, p. 9)

## 3.2 Namespaces

Namespaces were introduced to differentiate between classes and routines of the same name. To do so, the `::REQUIRES` directive needs to be tagged with a namespace name. The `ooRexx` namespace can be used to call `ooRexx` classes if another class is already using the same name. The addressed class needs a public statement, or it will not be visible outside of its containing `ooRexx` program.<sup>15</sup>

Example:

filename: namespace.rex

```
say .dog~bark

-- new namespace notation
say animal:dog~bark

::class dog
::method bark class
return "woof"

::requires 'namespace2.rex' namespace animal
```

filename: namespace2.rex

```
::class dog public
::method bark class
return "WOOF from the animal namespace"
```

16

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
woof
WOOF from the animal namespace
Drücken Sie eine beliebige Taste . . .
```

---

<sup>15</sup> (Flatscher & Müller, 2019, p. 3) (Ashley, et al., 2020, pp. 35-36, 94)

<sup>16</sup> (Ashley, et al., 2020, pp. 35-36)

### 3.3 *Variable Reference Notation Term*

The Variable Reference term is a reference to a variable, for example in a routine, allowing the original variable to be modified. This makes it possible to change the value of arguments to new objects, new strings, new arrays or to nil. The operators, > or <, are used to indicate a reference.<sup>17</sup>

Example:

```
a = "hello"
call work >a
say a

PAUSE

::routine work
  use arg >tmp

  tmp="from the work routine"
```

<sup>18</sup>

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
from the work routine
Drücken Sie eine beliebige Taste . . .
```

---

<sup>17</sup> (Ashley, et al., 2020, p. 25) (Flatscher & Müller, 2019, p. 2)

<sup>18</sup> (Flatscher & Müller, 2019, p. 2) (Ashley, et al., 2020, p. 26)

## 4 Directives

Directives are executable code units that stand at the end of a program. They are indicated with two consecutive colons (::) and the first directive separates the main code block from the other directive instructions. When a program is executed, it will first run a syntax check and then start invoking the first directive. This ensures that classes, routines and the environment are set up and available before the program is executed.<sup>19</sup>

### 4.1 ::ANNOTATION

The ANNOTATION directive is used to create annotations for packages, classes, methods, attributes, routines and constants. This means that metadata will be saved in a pair which is comprised of a name and a string value. The data is saved in a string table. A new method, “annotations”, is added to query the name and value pair.<sup>20</sup>

Example:

```
tmpSupp = .test~annotations~SUPPLIER
DO WHILE tmpSupp~AVAILABLE
SAY "index ["tmpSupp~INDEX"] item ["tmpSupp~ITEM"]"
tmpSupp~NEXT
END

::annotate package author "M. Mustermann"

::class test
::annotate class test version "1.0.0"
```

21

Output:



---

<sup>19</sup> (Flatscher & Müller, 2019, pp. 2-3) (Ashley, et al., 2020, p. 90)

<sup>20</sup> (Ashley, et al., 2020, pp. 90-91. 159) (Flatscher & Müller, 2019, p. 3)

<sup>21</sup> (Flatscher R. G., ooRexx 5.00 New Features, 2017, pp. 10-11)

## 4.2 **::ATTRIBUTE**

The DELEGATE sub-keyword is added to the ATTRIBUTE directive. The directive defines attributes and properties of methods. Following common design patterns, it delegates the execution of the method to an object.<sup>22</sup>

## 4.3 **::CLASS**

The CLASS directive defines and creates classes in ooRexx. The newly addition to this class is the added ABSTRACT sub-keyword. Marked with the abstract keyword, it will not be possible to create instances of this abstract class, otherwise it will generate an error. In this way, it is possible to provide a base for the subclasses, which is one of the fundamentals of object-based programming.<sup>23</sup>

Example:

```
d1 = .smalldog~New
d1~bark

::CLASS dog ABSTRACT
::METHOD bark
SAY "woof woof"
return

::CLASS smalldog SUBCLASS dog
```

<sup>24</sup>

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
woof woof
Drücken Sie eine beliebige Taste . . .
```

---

<sup>22</sup> (Ashley, et al., 2020, pp. 92-93, 97-98) (::ATTRIBUTE Delegate sub-keyword, 2020)

<sup>23</sup> (Ashley, et al., 2020, pp. 94-95) (Abstract Wikipedia, 2020) (::CLASS Abstract keyword, 2020)

<sup>24</sup> (Ashley, et al., 2020, p. 127)

## 4.4 **::CONSTANT**

A constant directive is a method that returns constant values for a class. In the new version the value is optional, so you will not get an error if you leave the value empty.<sup>25</sup>

Example:

```
SAY .test~pi
SAY .test~empty

::Class test
::Constant pi 3.14
::Constant empty
```

26

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
3.14
EMPTY
Drücken Sie eine beliebige Taste . . .
```

## 4.5 **::OPTIONS**

The **::OPTIONS** directive is used to set option settings and default values during runtime. Two additional options are added to this directive **NOVALUE** and **PROLOG/NOPROLOG**. The option **NOVALUE** decides if an uninitialized variable will raise a **NOVALUE** condition or a **SYNTAX** condition. The **NOVALUE CONDITION** is the default setting, but using **NOVALUE ERROR** raises a syntax error, which is useful for debugging. The setting **PROLOG** or **NOPROLOG** defines if the prolog code should be run or not. This is set by default to **PROLOG**.<sup>27</sup>

## 4.6 **::RESOURCE**

The **RESOURCE** directive defines multiline string data, which can be useful for multiline SQL statements. It is possible to signal the ending of the string with “**::END**” or define a

---

<sup>25</sup> (Ashley, et al., 2020, pp. 96-97)

<sup>26</sup> (Ashley, et al., 2020, p. 96)

<sup>27</sup> (Ashley, et al., 2020, pp. 100-101)

delimiter with “end”. Both are shown in the following example. The string ends automatically in the string “Zeile4”. The values are saved in a .RESOURCES string table and indexed with the directive name.<sup>28</sup>

Example:

```
SAY "pw: " .resources~pw~makeString~decodeBase64
SAY "text:"
SAY .resources~text

PAUSE

::RESOURCE pw
dG9wc2VjcmV0cGFzc3dvcmQ=
::END

::RESOURCE text end "Zeile4"

Zeile1

    Zeile2
Zeile3
Zeile4
```

29

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
pw: topsecretpassword
text:
Zeile1
    Zeile2
Zeile3
Drücken Sie eine beliebige Taste . . .
```

---

<sup>28</sup> (Flatscher & Müller, 2019, p. 3) (Ashley, et al., 2020, pp. 103-104, 439-440)

<sup>29</sup> (Ashley, et al., 2020, pp. 103-104) (Flatscher & Müller, 2019, p. 3) (Flatscher R. G., ooRexx 5.00 New Features, 2017, pp. 12-13)

## 5 Keyword Instructions

Keywords are short phrases or words that offer flow control or a service to the programmer. They are not case-dependent. This means that the compiler will read “*DO*” and “*do*” in the same way. The programming language ooRexx version 5 has around 30 different keyword instructions. The following main changes and additions have been made.<sup>30</sup>

### 5.1 ADDRESS

The ADDRESS keyword is used to access the system environment of ooRexx. The new addition to this feature allows the redirection of standard input, standard output and standard error into stream and collection ooRexx objects. The following example uses “*ADDRESS system*”, which addresses *cmd* on Windows and *sh* on Unix. It inputs the array “in” over *stdin*, sorts it and prints the output from *stdout*.<sup>31</sup>

Example:

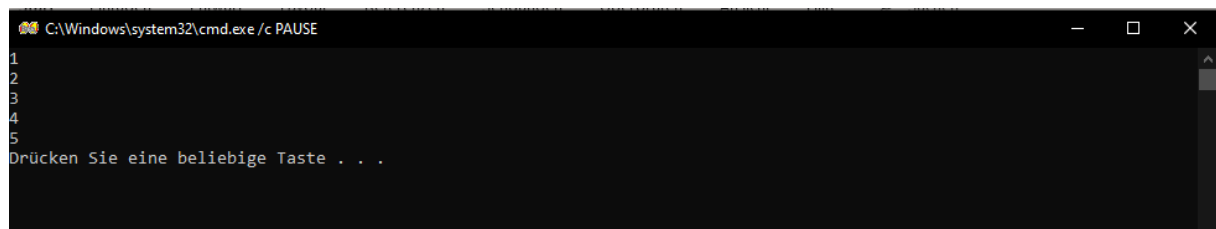
```
in = "2", "1", "3", "5", "4"
out = .array~new

ADDRESS system "sort" with input using (in) output using (out)

DO item OVER out
  SAY item
END
```

32

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
1
2
3
4
5
Drücken Sie eine beliebige Taste . . .
```

### 5.2 “DO” Keyword Instruction

The DO keyword is used to create code blocks and loops. The new DO instruction “*do with index x item y over supplierObject*” can be used to iterate over supplier objects, which is used to match index and item pairs. This makes it easier than using the Supplier class with a next

---

<sup>30</sup> (Ashley, et al., 2020, p. 42) (Flatscher & Müller, 2019, pp. 3-5)

<sup>31</sup> (Flatscher & Müller, 2019, pp. 3-4) (Ashley, et al., 2020, pp. 44-45)

<sup>32</sup> (Flatscher & Müller, 2019, p. 4)



message. The loop has two control variables, x that refers to current index value and y, which is associated to the item value. The FOR keyword limits the iterations of the loop.<sup>33</sup>

Example:

```
-- create a new array "animals"
animals = .array ~new
animals[1] = "cat"
animals[2] = "dog"
animals[3] = "cow"

Supp = animals~SUPPLIER -- create Supplier Object

DO with index x item y over Supp FOR 3
SAY "index:" x " item:" y
END

SAY

-- example of FOR added, out prints items of animals
DO item over animals FOR 2
SAY item
END
```

34

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
index: 1 item: cat
index: 2 item: dog
index: 3 item: cow

cat
dog
Drücken Sie eine beliebige Taste . . .
```

Furthermore, the COUNTER phrase was added to repetitive DO loops. It provides a count value for the loop. The value starts with 1 and increases by 1 with each repetition of the loop. The value for zero repetitions has a 0 value. Once the loop is finished, the variable has the value of the last repetition, which is illustrated in the example.<sup>35</sup>

Example:

---

<sup>33</sup> (Ashley, et al., 2020, pp. 50-52. 645) (Flatscher & Müller, 2019, p. 4) (DO supplier iteration Ticket, 2020) (FOR modifier Ticket, 2020)

<sup>34</sup> (Flatscher R. G., ooRexx 5.00 New Features, 2017, p. 8) (Ashley, et al., 2020, p. 645)

<sup>35</sup> (Ashley, et al., 2020, pp. 646-647)

```

DO COUNTER ct x = 2 TO 8 BY 2
  SAY "counter: " ct " value: " x
END

SAY "finished loop counter:" ct

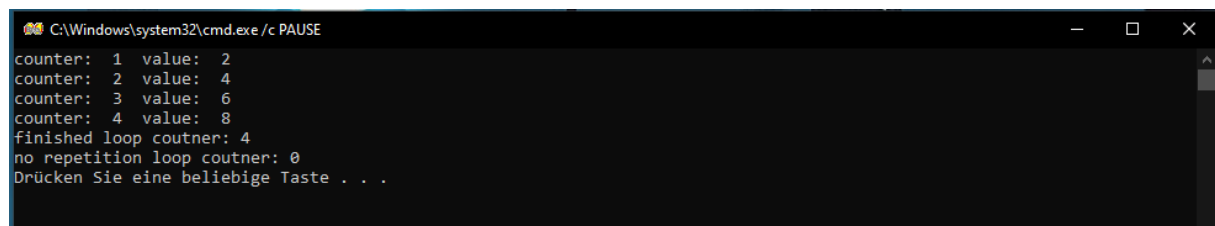
DO COUNTER ct2 y = 2 TO 1 BY 2
  SAY "counter: " ct2 " value: " y
END

SAY "no repetition loop counter:" ct2

```

36

Output:



```

C:\Windows\system32\cmd.exe /c PAUSE
counter: 1 value: 2
counter: 2 value: 4
counter: 3 value: 6
counter: 4 value: 8
finished loop counter: 4
no repetition loop counter: 0
Drücken Sie eine beliebige Taste . . .

```

### 5.3 *SELECT CASE*

The SELECT CASE expression is added to SELECT. It is used to make the language more human like and easier to learn for novices. At first, the case is evaluated and then the result of the case expression is compared to each of the options. As the example shows, the options get handled from top to bottom. if the highest option is false, the next one is checked until the case expression matches with the options. The option in which the case holds true is executed and the other options are ignored. To prevent an error, there needs to be an “otherwise” statement at the end in case none of the options hold true.<sup>37</sup>

Example:

```

SELECT CASE x
WHEN y THEN say "first y"
WHEN x THEN say "first x"
WHEN x THEN say "second x"
otherwise say "otherwise"
END

```

38

---

<sup>36</sup> (Ashley, et al., 2020, pp. 646-647)

<sup>37</sup> (Ashley, et al., 2020, pp. 77-79)

<sup>38</sup> (Flatscher R. G., ooRexx 5.00 New Features, 2017, p. 9) (Ashley, et al., 2020, pp. 78-79)

Output:



## 5.4 *USE LOCAL*

USE LOCAL defines local variables in a method. The keyword instruction must be the first instruction in the method. After defining the local variables, it changes all the other variables into object variables, which makes them public and exposes them. In a sense, it does the opposite of the keyword EXPOSE. This can be especially useful when there is a need to expose multiple variables like in the init method. The follow example shows its general function and how it is exchangeable with EXPOSE.<sup>39</sup>

Example:

```
p1 = .Player~New("Player1","1","1000")
p2 = .Player~New("Player2","2","1000")

SAY "Name: "p1~name
SAY "Authentication Number: "p1~authentication
SAY "Elo: "p1~elo
SAY
SAY "Player1 Elo: " p1~~increaseElo(5)~elo
SAY "Player2 Elo: " p2~~reduceElo(5)~elo

::CLASS Player
::METHOD INIT
USE LOCAL authentication
USE ARG name, authentication, elo

::METHOD name ATTRIBUTE
::METHOD authentication ATTRIBUTE
::METHOD elo ATTRIBUTE

::METHOD increaseElo
USE LOCAL
USE ARG increase
elo = elo + increase

::METHOD reduceElo
EXPOSE elo
USE ARG reduction
elo = elo - reduction
```

---

<sup>39</sup> (Ashley, et al., 2020, pp. 88-89) (USE LOCAL Ticket, 2020)

## Output:

A screenshot of a Windows command prompt window. The title bar shows the icon of a yellow notepad and the text 'C:\Windows\system32\cmd.exe /c PAUSE'. The window has standard Windows window controls (minimize, maximize, close) on the right. The command prompt displays the following text:

```
Name: Player1
Authentication Number: AUTHENTICATION
Elo: 1000

Player1 Elo: 1005
Player2 Elo: 995
Drücken Sie eine beliebige Taste . . .
```

---

<sup>40</sup> (Ashley, et al., 2020, p. ()) (Flatscher R. G., ooRexx 5.00 New Features, 2017, p. 14) (Flatscher R. G., An Introduction to Procedural and Object-oriented Programming (ooRexx) 4, 2016, p. 22)

## 6 Classes

Classes are code templates in order to create objects. It is possible to send messages to created objects and execute methods with the given message name. The objects can have variables, which are called attributes. There are four different types of classes object, mixin, abstract and metaclass.<sup>41</sup>

### 6.1 AlarmNotification

The new class AlarmNotification is added to ooRexx; it is a notification interface for alarm and ticker class. It allows messages to be sent later at a different time and date. The abstract method “triggered” gets notified when the message has been sent. The method “cancel” cancels the alarm or the ticker. This class is especially useful for multi-threading.<sup>42</sup>

Example:

```
a1 = .Alarm~new(1,.Target~new,"alarm 1 triggered")
a2 = .Alarm~new(3,.Target~new,"alarm 2 triggered")
call SysSleep 2
a2~cancel

::class Target inherit AlarmNotification
::method triggered
use arg alarm
say alarm~attachment

::method cancel
say "alarm cancelled"
```

43

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
alarm 1 triggered
alarm cancelled
Drücken Sie eine beliebige Taste . . .
```

---

<sup>41</sup> (Classes Wikipedia, 2020) (Ashley, et al., 2020, pp. 107-110)

<sup>42</sup> (Ashley, et al., 2020, p. 332) (Flatscher & Müller, 2019, p. 5)

<sup>43</sup> (Ashley, et al., 2020, pp. 419-420)

## 6.2 EventSemaphore

The new class EventSemaphore allows multiple threads and objects to be synchronized. Other activities get signalized once an event or a condition is triggered.<sup>44</sup> This class is especially useful for concurrency which is the process of running multiple objects and methods at the same time.<sup>45</sup> The execution of the program halts when the method “wait” is used and continues once the event semaphore is posted with the method “post”. The method “isposted” checks if the event is posted or not and returns 1 or 0, true or false.<sup>46</sup>

Example:

```
sem = .EventSemaphore~new
sem~reset
say "main starts tasks"

.task~new~run(sem, "task" 1)
.task~new~run(sem, "task" 2)

Call SysSleep 2
say "main posts"
sem~post
say "main ends"

SAY sem~isposted -- checks state
sem~reset
SAY sem~isposted -- checks state

::class Task
::method run
reply
use strict arg sem, name
say name "waits"
sem~wait
say name "runs"
```

47

Output:

---

<sup>44</sup> (Ashley, et al., 2020, pp. 357-358) (Flatscher & Müller, 2019, p. 5)

<sup>45</sup> (Ashley, et al., 2020, pp. 601-612)

<sup>46</sup> (Ashley, et al., 2020, pp. 359-360)

<sup>47</sup> (Ashley, et al., 2020, pp. 358-360)

```
C:\Windows\system32\cmd.exe /c PAUSE
main starts tasks
task 2 waits
task 1 waits
main posts
main ends
task 2 runs
task 1 runs
1
0
Drücken Sie eine beliebige Taste . . .
```

## 6.3 *Json*

The newly added Json class makes it possible to convert ooRexx objects into Json objects and vice versa. Json is mainly used for data exchange and is widely used in other languages with implemented json parsers that can interpret this format.<sup>48</sup> The REQUIRES directive is needed to load the class if a class is not preloaded.<sup>49</sup>

Example:

```
rexarray = "cat", "dog", "cow"

rexjson = .json~new()~toJSON(rexarray)
Say rexjson

::REQUIRES 'json.cls'
```

Output:

```
C:\Windows\system32\cmd.exe /c PAUSE
["cat","dog","cow"]
Drücken Sie eine beliebige Taste . . .
```

## 6.4 *MessageNotification*

The new class MessageNotification is a notification interface for the message class. The message class allows to send asynchronous messages, the notification will be sent with the notify method from the message class. By executing the notify method, the method

---

<sup>48</sup> (Json Wikipedia, 2020) (Json.cls, 2020) (Ashley, et al., 2020, p. 625)

<sup>49</sup> (Ashley, et al., 2020, pp. 102, 392)

“messageCompleted” of the MessageNotification class will be called after the target has processed the message.<sup>50</sup>

Example:

```
-- arguments = message target, message content
msg = .Message~new( .task~new, "perform")

msg~notify(.information~new)
msg~start

::class task
::method perform
call SysSleep 2
SAY "task performed"

::class information inherit MessageNotification
::method messageComplete
use strict arg message
call SysSleep 1
SAY "message sent"
```

51

Output:



## 6.5 MutexSemaphore

The new class MutexSemaphore allows multiple ooRexx threads to use one resource. It locks one resource while one thread uses the resource. In the meantime, the other thread is in queue. After the first thread has been finished, the semaphore block releases, and the next waiting thread can use the resource.<sup>52</sup>

Example:

```
mutex = .MutexSemaphore~new
.local~x = 1
.Task~new~start(mutex, "write")
```

---

<sup>50</sup> (Flatscher & Müller, 2019, p. 5) (Ashley, et al., 2020, pp. 132-137. 370-371)

<sup>51</sup> (Ashley, et al., 2020, pp. 136-137)

<sup>52</sup> (Flatscher & Müller, 2019, p. 5) (Ashley, et al., 2020, pp. 387-388)



```

.Task~new~start(mutex, "read")
say "started ..."

::class Task
::method start unguarded
use local
use arg mutex, name
reply
self~execute

::method execute unguarded
use local
IF .x = 5 THEN DO
    .local~x = 1
    return
END

mutex~acquire
say name .x
call syssleep 1

.local~x = .x + 1
self~execute
53

```

Output:

```

started . . .
write 1
write 2
write 3
write 4
read 1
read 2
read 3
read 4
Press ENTER key to exit...

```

## 6.6 *RexxInfo*

The new class `RexxInfo` provides ooRexx settings, “Rexx language information and other platform-specific information”.<sup>54</sup> The class can be accessed over the environment symbol. The use of the class is limited to one instance at a time.<sup>55</sup> In the first part of the examples, it checks on which system the program is running. In the second part, a few examples of `RexxInfo` methods are given, a full list can be found in the Rexx Reference.

Example:

```

IF .rexinfo~platform = "WindowsNT" THEN
    DO

```

---

<sup>53</sup> (Ashley, et al., 2020, p. 388)

<sup>54</sup> (Ashley, et al., 2020, p. 400)

<sup>55</sup> (Ashley, et al., 2020, pp. 400-401)

```

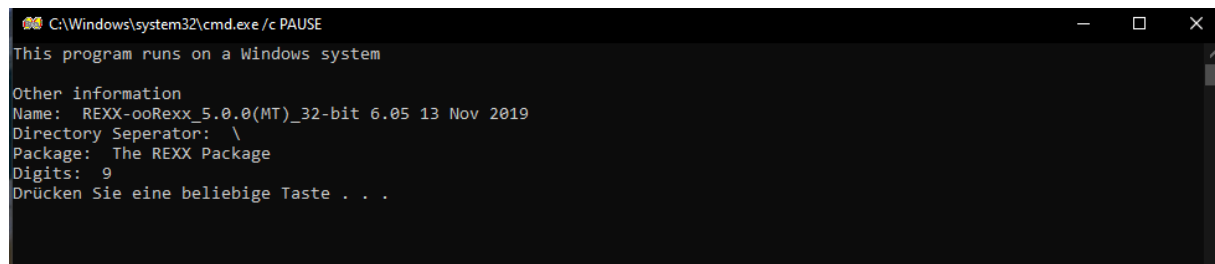
    SAY "This program runs on a Windows system"
END
ELSE IF .rexxinfo~platform = "Linux" THEN
DO
    SAY "This program runs on a Linux system"
END
ELSE SAY "System unknown"

-- other information
SAY
SAY "Other information"
SAY "Name: " .rexxinfo~name
SAY "Directory Separator: " .rexxinfo~directorySeparator
SAY "Package: " .rexxinfo~package
SAY "Digits: " .rexxinfo~digits

```

56

Output:



## 6.7 *StringTable*

The new class `StringTable`, a MIXIN class, is a `MapCollection`, which is similar in functionality to a `Directory` class and uses unique strings to index items.<sup>57</sup> Since the ooRexx interpreter uses this class internally, this results in better runtime performance in ooRexx 5, due to the reduced overhead when looking up and executing method objects.<sup>58</sup>

Example:

```

--creates table
coll = .StringTable~new

-- checks if the StringTable is empty returns true or false
SAY "is empty: " Coll~isEmpty

--insert values: "item", "name"
coll~put("0123334", "Mike")
coll~put("0123567", "Bob")

```

<sup>56</sup> (Ashley, et al., 2020, pp. 400-409) (Flatscher R. G., ooRexx 5.00 New Features, 2017, pp. 15-16)

<sup>57</sup> (Ashley, et al., 2020, p. 314)

<sup>58</sup> (Flatscher & Müller, 2019, p. 5)

```

coll~put("1234010", "Alice")


-- to look up a specific entry
SAY "number of the person: " coll~at("Alice")
SAY "person with the number: " coll~index("1234010")

--number of items
SAY "number of items: " coll~items

--outprint the item name pair
SAY
SAY "full table:"
supp = coll~SUPPLIER -- create Supplier Object
DO with index idx item itm over supp
  SAY "index:" idx " item:" itm
END
59

```

Output:



```

C:\Windows\system32\cmd.exe /c PAUSE
is empty: 1
number of alice: 1234010
person with the number "1234010": Alice
number of items: 3

full table:
index: Bob item: 0123567
index: Alice item: 1234010
index: Mike item: 0123334
Drücken Sie eine beliebige Taste . . .

```

## 6.8 Ticker

The new class Ticker sends a notification message to a target which must be a subclass of the AlarmNotification class, in certain intervals, this can be a fixed time span or a string object. It is possible to cancel the Ticker with the cancel method. This class can be useful for monitoring task progress information.<sup>60</sup> The following example showcases a simple countdown.

Example:

```

startCount = 5
.local~x = 1
SAY startCount

timer = .Ticker~new(1, .Target~new, startCount)
call SysSleep startCount
timer~cancel

::class Target inherit AlarmNotification
::method triggered

```

<sup>59</sup> (Ashley, et al., 2020, pp. 314-318)

<sup>60</sup> (Ashley, et al., 2020, pp. 419-421)

```

use arg ticker
say ticker~attachment - .x
.local~x = .x + 1

::method cancel
SAY "Timer finished"
61

```

Output:



```

C:\Windows\system32\cmd.exe /c PAUSE
5
4
3
2
1
0
Timer finished
Drücken Sie eine beliebige Taste . . .

```

## 6.9 Variable Reference

The new class Variable Reference is responsible to maintain the connection between the object and the variable and link the name and value of the referenced variable. It enables to create new instances of this class with the new keyword, just with the variable reference term.<sup>62</sup>

Example:

```

a = "123"
-- displays name of the variable reference
SAY >a~name
-- value method of variable reference class
SAY >a~class~id
SAY >a~value~class~id
63

```

Output:



```

C:\Windows\system32\cmd.exe /c PAUSE
A
VariableReference
String
Drücken Sie eine beliebige Taste . . .

```

<sup>61</sup> (Ashley, et al., 2020, pp. 419-420)

<sup>62</sup> (Flatscher & Müller, 2019, p. 5) (Ashley, et al., 2020, p. 432)

<sup>63</sup> (Ashley, et al., 2020, p. 433)

## 6.10 Validate

The new class `Validate` checks if arguments are of a certain type, class type or length. Concerning numbers, it is possible to check whether they are positive or negative, or if they are in a certain range. If the argument value is invalid, a syntax error will be raised.<sup>64</sup>

Example:

```
Call test(1)
EXIT

test :
use arg time
.validate~number(time, time)
call sysleep time
say "validation completed"
```

<sup>65</sup>

Output:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe /c PAUSE'. The window content displays the output 'validation completed' on the first line, followed by the German text 'Drücken Sie eine beliebige Taste . . .' on the second line. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

---

<sup>64</sup> (Ashley, et al., 2020, pp. 428-432)

<sup>65</sup> (Ashley, et al., 2020, pp. 428-432)

## 7 Other Changes

### 7.1 Built-in-functions

A built-in function is a routine that returns a single object as a result, compared to a subroutine it is called with a function call expression. It consists of a function name and function arguments in round brackets.<sup>66</sup>

#### 7.1.1 *CONDITION()*

The condition function gives information regarding the current trapped condition. There are several kinds of information that can be queried, for example the name, status or condition-specific information.<sup>67</sup> “A condition is an event or state that CALL ON or SIGNAL ON can trap.”<sup>68</sup> A condition trap, also called exception, changes the execution flow of the Rexx program, whereby a trapname specifies the kind of the trap.<sup>69</sup> For the built-in-function condition the option reset is added, current trapped conditions will be reset back to their default values and the null string is returned.<sup>70</sup>

Example:

```
SIGNAL ON SYNTAX
RAISE SYNTAX 001
EXIT

SYNTAX:
SAY "SYNTAX IS RAISED."
SAY "before reset, error name: " CONDITION(C)
SAY "resetting . . . " CONDITION(RESET)
SAY "after reset, error name: " CONDITION(C)

EXIT
```

71

---

<sup>66</sup> (Ashley, et al., 2020, pp. 441, 447-448)

<sup>67</sup> (Ashley, et al., 2020, pp. 459-460, 591)

<sup>68</sup> (Ashley, et al., 2020, p. 591)

<sup>69</sup> (Condition Trap IBM, 2020) (Ashley, et al., 2020, pp. 459-460)

<sup>70</sup> (Ashley, et al., 2020, p. 459)

<sup>71</sup> (Ashley, et al., 2020, p. 460) (Flatscher R. G., An Introduction to Procedural and Object-oriented Programming (ooRexx) 3, 2017, p. 11)

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
SYNTAX IS RAISED.
before reset, error name: SYNTAX
resetting . . .
after reset, error name:
Drücken Sie eine beliebige Taste . . .
```

### 7.1.2 DATATYPE()

The datatype function checks a string and compares the string with an option. There are different type options, for example alphanumeric, mixed case or binary. If the string is equal to the option, it returns a 1, if they are not equal, the function will return a 0. A new option type is added to this function Internal, which checks the string for whole numbers.<sup>72</sup>

Example:



```
x = "9123213"
y = "123A231"

-- DATATYPE(string, type)
SAY "x: " DATATYPE(x, Internal)
SAY "y: " DATATYPE(y, Internal)
```

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
x: 1
y: 0
Drücken Sie eine beliebige Taste . . .
```

### 7.1.3 DATE()

The date function returns current date as object result. The ISO format is added as a new function and it returns the date in yyyy-mm-dd format, which is specified by ISO 8601.<sup>73</sup> The ISO standard is established to create a standard representing dates and times to ease interchange and to minimize misinterpretation.<sup>74</sup>

---

<sup>72</sup> (Ashley, et al., 2020, pp. 462-463)

<sup>73</sup> (DATE() ISO Ticket, 2020) (Ashley, et al., 2020, pp. 463-467)

<sup>74</sup> (ISO Wikipedia, 2020)

Example:

```
SAY DATE("ISO")
```

Output:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe /c PAUSE'. The window content displays the date '2020-05-22' and the prompt 'Drücken Sie eine beliebige Taste . . .'. The output of the command is visible on the first line.

### 7.1.4 *XRANGE()*

The Xrange function returns a string, that is specified in the round brackets. Usually, a start and an end should be specified, but it is possible to query the string with a keyword as well. It is now possible to chain multiple queries in a single operation.<sup>75</sup>

Example:

```
-- XRANGE(start, end)  
SAY XRANGE(xdigit, "0", "9", "A", "C")
```

Output:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe /c PAUSE'. The window content displays a long alphanumeric string '0123456789ABCDEFabcdef0123456789ABC' and the prompt 'Drücken Sie eine beliebige Taste . . .'. The output of the command is visible on the first line.

## 7.2 *Rexxutil*

The REXXUTIL is a package of functions to communicate with the embedded system, issue commands or get information. It includes function for Windows and Unix platforms. The rexxutil packages are preloaded by the interpreter on start up.<sup>76</sup>

---

<sup>75</sup> (XRANGE Ticket, 2020) (Ashley, et al., 2020, pp. 508-509)

<sup>76</sup> (Ashley, et al., 2020, p. 510)



### 7.2.1 *SysFormatMessage*

`SysFormatMessage` is a new window function that formats and replaces “&n” in a text, n stands for a number, for example &1. The maximum number of replacements is nine.<sup>77</sup>

Example:

```
say SysFormatMessage("STRING111222 &1 &2", ("Replacement A", "Replacement B"))
```

78

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
STRING111222 Replacement A Replacement B
Drücken Sie eine beliebige Taste . . .
```

### 7.2.2 *SysGetLongPathName*

The new function returns the long version or path to a file or directory for windows systems. If the path does not exist or the search process fails, it returns the null string.<sup>79</sup>

Example:

```
say SysGetLongPathName("C:\PROGRA~1")
```

80

Output:



```
C:\Windows\system32\cmd.exe /c PAUSE
C:\Program Files
Drücken Sie eine beliebige Taste . . .
```

---

<sup>77</sup> (Ashley, et al., 2020, pp. 529-530)

<sup>78</sup> (Ashley, et al., 2020, pp. 529-530)

<sup>79</sup> (Ashley, et al., 2020, p. 533)

<sup>80</sup> (Ashley, et al., 2020, p. 533)

## 8 Conclusion

The new ooRexx 5.0.0 beta version introduced a significant number of new features. As a matter of fact, it resulted in performance gains and improved functionality. Furthermore, the numerous bugfixes led to an increase in stability. All these changes evolved the language, while retaining its initial design and being backwardly compatible to older versions. The paper has set out to describe and provide nutshell examples of the new features. Based on analyzing them, following benefits are associated with the new ooRexx version.

Firstly, ooRexx is more tailored towards object-oriented programming. This includes the DELEGATE sub-keyword for the CLASS directive and namespaces. In addition, there are improvements regarding user friendliness, for example the USE LOCAL keyword.

Secondly, ooRexx is kept human-oriented, lightweight and easy to learn for novices. An intuitive array notation was added, and the keyword SELECT CASE was added. In fact, the total amount of keywords is kept low.

The new version is easy and simple, however, powerful in performance. In fact, the multithreading has been significantly improved by adding message and alarm notification interfaces and by adding the two semaphore classes.

Furthermore, the versatility as macro and command language is improved. The RESOURCE directive enhances ooRexx string dealing and macro language capabilities. This, for example, makes it easier to issue multiline sql commands. The expanded usage of the ADDRESS keyword improves the capabilities as a command language and the way ooRexx interacts with the system. In addition, new built-in-functions for the system were added.

Definitely, the version update has brought new features and functionality that prepare ooRexx to be competitive in the future.

# References

- ::*ATTRIBUTE Delegate sub-keyword*. (2020, May 16). Retrieved from Source Forge:  
<https://sourceforge.net/p/oorexx/feature-requests/649/>
- ::*CLASS Abstract keyword*. (2020, June 1). Retrieved from Source Forge:  
<https://sourceforge.net/p/oorexx/feature-requests/631/>
- Abstract Wikipedia*. (2020, May 16). Retrieved from Wikipedia:  
[https://en.wikipedia.org/wiki/Abstract\\_type](https://en.wikipedia.org/wiki/Abstract_type)
- Ashley, W. D., Flatscher, R. G., Hessling, M., McGuire, R., Peedin, L., Sims, O., & Wolfers, J. (2020). *ooRexx Documentation 5.0.0.r12043 Open Object Rexx Reference* (0.2020.04.04 ed.). (R. L. Association, Ed.)
- Classes Wikipedia*. (2020, May 21). Retrieved from Wikipedia:  
[https://en.wikipedia.org/wiki/Class\\_\(computer\\_programming\)](https://en.wikipedia.org/wiki/Class_(computer_programming))
- Condition Trap IBM*. (2020, May 22). Retrieved from IBM:  
[https://www.ibm.com/support/knowledgecenter/en/SSLTBW\\_2.2.0/com.ibm.zos.v2r2.ikja300/ikja30073.htm](https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.ikja300/ikja30073.htm)
- Cowlshaw, M. (1987, February). The design of the Rexx language. *IBM Systems Journal*, pp. 326-335. doi:10.1147/sj.234.0326
- DATE() ISO Ticket*. (2020, May 17). Retrieved from Source Forge:  
<https://sourceforge.net/p/oorexx/feature-requests/407/>
- DO supplier iteration Ticket*. (2020, April 29). Retrieved from Source Forge:  
<https://sourceforge.net/p/oorexx/feature-requests/158/>
- Flatscher, R. G. (2016, October 13). An Introduction to Procedural and Object-oriented Programming (ooRexx) 4. Augasse 2-6, Wien.
- Flatscher, R. G. (2017, October 17). An Introduction to Procedural and Object-oriented Programming (ooRexx) 3. Augasse 2-6, Wien.
- Flatscher, R. G. (2017, April 9-12). ooRexx 5.00 New Features. Amsterdam, Netherlands: 28th Annual Rexx Symposium.
- Flatscher, R. G. (2017, April 9-12). Open Object Rexx Tutorial. Amsterdam, Netherlands: Open Object Rexx Tutorial. 28th Annual Rexx Symposium.
- Flatscher, R. G. (2018). *The New BSF4ooRexx 6.00*. The 2018 International Rexx Symposium. Aruba, Dutch West Indies: René Jansen, Chip Davis. Retrieved from <https://www.rexxla.org/events/2018/presentations/201803-BSF4ooRexx-6.0-Article.pdf>
- Flatscher, R. G., & Müller, G. (2019). *ooRexx 5 Yielding Swiss Army Knife Usability*. Hursley, Great Britain, United Kingdom: The 2019 International RexxLA Symposium.
- FOR modifier Ticket*. (2020, April 29). Retrieved from Source Forge:  
<https://sourceforge.net/p/oorexx/feature-requests/605/>

*ISO Wikipedia*. (2020, May 22). Retrieved from Wikipedia:  
[https://en.wikipedia.org/wiki/ISO\\_8601](https://en.wikipedia.org/wiki/ISO_8601)

*Json Wikipedia*. (2020, May 27). Retrieved from Wikipedia:  
<https://en.wikipedia.org/wiki/JSON>

*Json.cls*. (2020, May 27). Retrieved from Idenburg:  
<http://www.idenburg.net/ooRexx/wip/showSource.php?sfn=../code/json.cls>

*Notation Wikipedia*. (2020, June 1). Retrieved from Wikipedia:  
<https://en.wikipedia.org/wiki/Notation>

*Source Forge Wikipedia*. (2020, April 1). Retrieved from Wikipedia:  
<https://en.wikipedia.org/wiki/SourceForge>

*USE LOCAL Ticket*. (2020, April 29). Retrieved from Source Forge:  
<https://sourceforge.net/p/ooRexx/feature-requests/654/>

*XRANGE Ticket*. (2020, May 17). Retrieved from Source Forge:  
<https://sourceforge.net/p/ooRexx/feature-requests/639/>