Unicode and REXX

A brief introduction to TUTOR

36th International Rexx Language Symposium

The Wirtschaftsuniversität Vienna, Austria, May 4-7 2025

Josep Maria Blasco

jose.maria.blasco@gmail.com

EPBCN – ESPACIO PSICOANALITICO DE BARCELONA

Balmes, 32, 2° 1ª — 08007 Barcelona, Spain

May the 4th, 2025

Notice

This whole document[1] is an experiment in CSS printing. It comfortably mixes normal text, emojis, and programs beautified by the heavy prettyprinting produced by the Rexx Highlighter.

If you are viewing this file as a PDF, it will look as a presentation. If you are viewing it as a web page, the suggested print settings to produce a presentation are: no headers or footers, and background images active. This is true for the Chrome browser at the time of this writing (Jan-May 2025).

The default style for Rexx fenced code blocks is `dark`. You can choose the light style by adding a `style=light` query string to the url of this document.

---

1. HTML version: https://rexx.epbcn.com/publications/2025-05-06-The-Rexx-Highlighter/.
   PDF version (slides): http://www.epbcn.com/pdf/josep-maria-blasco/2025-05-04-Unicode-and-Rexx.pdf.↩

What is TUTOR?

- TUTOR is an acronym for *T*he *U*nicode *T*ools *O*f *R*exx.
- A collection of software utilities showing how a possible implementation of Unicode for Rexx could be.
- In this sense, TUTOR is simply a set of ideas, to encourage discussion about Rexx and Unicode, and an aid to visualize a possible way in which Rexx and Unicode could match.
- It is a prototype, not tuned for efficiency. It is neither a standard, nor a finished product.
- Written in 2024 by Josep Maria Blasco, it incorporates many useful suggestions discussed in the Rexx Architecture Review Board.
- Originally *toys*, not *tools*. Changed by popular suggestion, although "toys" seems to best describe what the software collection has ended up being.
- Originally "toys *for* Rexx", not "toys *of* Rexx". Chip Davis suggested the change which produced the current, really nice, acronym.

This presentation will be devoted to highlighing the main features of the TUTOR package.

Unicode and Rexx, Today

With the right editor and an appropiately configured terminal program, one can *use* Unicode strings today.

```
-- Accented characters and non-latin characters
Say "José María Blasco नमस्ते"

-- Emojis
Say "🍆🍐🦞🍐"
```

These Unicode strings can be stored in files, retrieved, used as file names,[2] etc.

BUT... Although we can **use** Unicode strings, we cannot properly **manipulate** them. Assuming a UTF-8 encoding,

```
Say Length("🍆🍐🦞🍐")    -- 16, instead of the expected 4
Say Right("José",1)       -- "A9"X (unprintable), instead of the expected "é"
```

2. With some limitations, depending on many different factors. For example, Dropbox cannot synchronize files which contain emojis in their filenames.↩

What is a character?

To properly manipulate Unicode strings and still be compatible with existing programs, TUTOR needs to be able to recognize "old", Classic Rexx, strings, composed of characters that are **bytes**, and "new", Unicode, strings, where the notion of "character" becomes much more complex.

```
/* As a UTF-8 character string, "🦢🍐🦞🍐" consists of 16 bytes   */
/* As a Unicode string, "🦢🍐🦞🍐" consists of 4 emoji codepoints */
```

Indeed, the Unicode standard provides *two* different definitions of what a Unicode character is: **codepoints**, which roughly correspond to letters, numbers, other symbols (including emojis) and diacritics, and **extended grapheme clusters**, which are defined to be "user-perceived characters".

For example, the emoji sequence "👬🎅" consists of **two** extended grapheme clusters, namely "👬" and "🎅"; **eight** codepoints, that is, the emojis for "👱" (man), "👱‍♀️" (woman), "👧" (girl) and "👱" (boy), "glued" together with three *Zero-width-joiners*, a special Unicode codepoint used to build compound characters and emojis, and "🎅", the emoji for "Father Christmas"; and, finally, **twenty-nine** bytes, corresponding to the UTF-8 representation of the string.

String types

The same string, then, can be viewed (1) as an array of **bytes**, (2) as an array of **codepoints** and (3) as an array of **extended grapheme clusters**. We say that a string "comes with" its own way of being viewed, which we will call its **type**.

A `BYTES` string is an array of bytes. A `CODEPOINTS` string is an array of codepoints. A `GRAPHEMES` string is an array of graphemes. A `TEXT` string is a `GRAPHEMES` string that has been NFC-normalized at string creation time.

Classic Rexx strings are `BYTES` strings.

`BYTES`, `CODEPOINTS`, `GRAPHEMES` and `TEXT` are also BIF names. You convert between types by using these BIFs. Left to right conversions are called **promotions**, and right to left conversions are called **demotions**. Converting from `BYTES` to `GRAPHEMES` is a promotion, and converting from `TEXT` to `CODEPOINTS` is a demotion. Demotion will always work. Promotion will fail if the argument string is not valid UTF-8. Converting to `TEXT` may alter the original string, because of the extra NFC normalization step, and, in this case, we won't be dealing with *the same* string, but with an *equivalent* one.

New string suffixes

A string wih a `Y` suffix is a `BYTES` string. `BYTES("string")` and `"string"Y` have the same value. `BYTES("string")` is a BIF call (it has a run-time cost), but `"string"Y` is checked at translation time.

- `"string"Y` is equivalent to `BYTES("string")`.
- `"string"P` is equivalent to `CODEPOINTS("string")`.
- `"string"G` is equivalent to `GRAPHEMES("string")`.
- `"string"T` is equivalent to `TEXT("string")`.

```
Say Length("नमस्ते"Y)              -- 18 (bytes)
Say Length("नमस्ते"P)              -- 6  (codepoints)
Say Length("नमस्ते"G)              -- 3  (graphemes)
Say Length("नमस्ते"T)              -- 3  (same as "G")
```

The type of a string can be obtained by using the `STRINGTYPE` BIF.

```
Say StringType("नमस्ते"Y)          -- BYTES
Say StringType("🍆🍐🐙🍐"G)      -- GRAPHEMES
```

Unsuffixed strings: The default string type

Determining what should be the semantics of a unsuffixed string in Unicode-enabled Rexx is quite involved. In Classic Rexx, a unsuffixed string is what we call a `BYTES` string, but a new user would most probably expect the default to be `TEXT`, and have immediate access to all the Unicode features.

```
Options DefaultString Codepoints  -- All strings are CODEPOINTS by default
Say Length("🦢🍐🦞🍐")             -- 4
Options DefaultString Bytes       -- All strings are BYTES by default
Say Length("🦢🍐🦞🍐")             -- 16
```

TUTOR defines the default to be `TEXT`, but it also allows to configure a different default, by providing an `OPTIONS DEFAULTSTRING option` instruction. *Option* has to be one of `BYTES`, `CODEPOINTS`, `GRAPHEMES`, `TEXT` or `NONE`. When *option* is not `NONE`, every number, variable, or constant symbol are first checked to see if they have a stringtype assigned and, if not, they are automatically converted to the *option* type. When *option* is `NONE`, no conversion is attempted.

Mixing string types

What happens when we mix different string types in the same expression? For example, when a `TEXT` string and a `BYTES` string are concatenated, what is the type of the result?

```
Options Coercions Promote        -- Convert to the strongest type
Options Coercions Demote         -- Convert to the weakest type
Options Coercions Left           -- Convert to the type of the left operand
Options Coercions Right          -- Convert to the type of the right operand
Options Coercions None           -- Do not allow conversions (produce a syntax error)
```

TUTOR is agnostic in this respect. There are only three sensible things to do when mixing operands of different types in the same expression:

- Forbid the operation and produce a `SYNTAX` error.
- Promote the weakest operand (towards "more Unicode").
- Demote the strongest operand (towards "less Unicode").

For completeness, TUTOR also allows to convert to the type of the left or the right operand.

U strings

In addition to Y, P, G and T strings, TUTOR defines a new, specialized type of string, the U string. U strings allow the designation of individual code points by hexadecimal value, by name, by alias or by label:

```
Say "(Man)"U                    -- "👨", an emoji
Say "(Man)(Woman)"U             -- "👨👩", two emojis
Say "(Man)(ZWJ)(Woman)"U        -- "👨‍👩", an emoji grapheme cluster
Say "61"U                       -- "a"
Say "0061"U                     -- "a"
Say "U+0061"U                   -- Also "a". The "U+" prefix is optional
Say "(Latin Small Letter A)"U   -- "a"
Say "(LatinSmallLetterA)"U      -- "a": blanks are ignored
Say "(LATINSMALL LETTERA)"U     -- "a": case is also ignored
```

U strings are low-level constructs, as are binary and hexadecimal strings, and therefore they are always BYTES strings. If you need them in another role, you will have to promote them first.

New low-level character manipulation functions

`C2U` and `U2C`, the U versions of `C2X` and `X2C`:

```
Say C2U("🍐🦞")                    -- "1F350 1F99E", two hexadecimal codepoint
Say C2U("🍐🦞","Names")            -- "(PEAR) (LOBSTER)"
Say U2C("(Pear) 1f99e")            -- "🍐🦞"
```

`N2P` and `P2N`, *N*ame to code*P*oint and code*P*oint to *N*ame conversions:

```
Say N2P("LATIN CAPITAL LETTER F")  -- "0046" (padded to four digits)
Say N2P("BELL")                    -- "1F514" (codepoint for "🔔")
Say P2N("1F342")                   -- "FALLEN LEAF" (The "🍂" emoji)
Say P2N("0012")                    -- "<control-0012>" (a label, not a name)
```

A single API for all string types

To manipulate `CODEPOINTS`, `GRAPHEMES` or `TEXT` strings, we will use the usual set of built-in functions (`LENGTH`, `SUBSTR`, `UPPER`, ...) and operators (`||`, `+`, ...): they will operate on the corresponding definition of what a *character* is.

```
  Call Print "👩‍👩‍👦"T                    -- "👩‍👩‍👦" -- 1F468 200D 1F469
  Call Print "👩‍👩‍👦"P                    -- "👩" -- 1F468
                                        -- "" -- 200D
                                        -- "👦" -- 1F469

  Say Length("Jose"P || "301"U)         -- 5 codepoints   ["301"U is the acute accent]
  Say Length("Jose"G || "301"U)         -- 4 graphemes
  Say C2X("Jose"G || "301"U)            -- "4A6F7365CC81"X
  Say C2X("Jose"T || "301"U)            -- "4A6F73C3A9"X (normalized to NFC)
Exit

Print: string = Arg(1)
  Do i = 1 To Length(string)
    c = SubStr(string, i, 1)
    Say '"'c'" --' C2U(c)
  End
Return
```

Unicode-enabled streams

By default, streams continue to be byte-oriented, but we can also specify an encoding when opening a stream:

```
Call Stream filename, "Command", "Open read ENCODING IBM-1047"
```

Rexx will automatically convert to and from Unicode when using encoded streams. When an ill-formed character is encountered, the default action is to substitute it by the Unicode Replacement Character, `"�"` (`"FFDD"U`).

```
Call Stream filename, "Command", "Open read ENCODING IBM-1047 SYNTAX"
```

If you specify `SYNTAX`, a syntax error is produced instead.

You can also specify the target string type as `TEXT` (the default), `GRAPHEMES`, or `CODEPOINTS`. Remember that `TEXT` strings are automatically normalized to NFC format.

A procedural-first definition of Unicode for Rexx

As you will probably have realized, all the TUTOR Rexx extensions we have described are **purely procedural**, that is, they can be implemented by extending a non-object oriented implementation of Rexx, like Regina.

The **current implementation** of TUTOR, though, uses ooRexx 5.0 and four new basic classes, which correspond to the four basic string types (`BYTES` becomes a conceptually synonym for `STRING`, with some small Unicode extensions). Although these classes show a possible way to extend ooRexx to accomodate Unicode functionality, they are a contingency of the actual implementation.

One should be able to implement the procedural funcionality described by TUTOR in a non-object oriented version of Rexx. We describe this fact by saying that our definition of Unicode for Rexx is **procedural-first**.

The Rexx Preprocessor for Unicode (1/2)

If you want to experiment with TUTOR, you can use the **Rexx Preprocessor for Unicode**. The Preprocessor, called RXU (*R*e*X*x and *U*nicode) is implemented by a Rexx program, `rxu.rex`.

The simplest way to use the preprocessor is the following: you write a RXU program, that is, a Rexx program with Unicode TUTOR features, and give it a `.rxu` extension.

```
/* isnamevalid.rxu */
Signal On Syntax
Parse Arg name
c = U2C( "("name")" )
Say "The name '"name"' is a valid Unicode name."
Say "It corresponds to '"c"' ('"C2U(c)"'U)."
Exit 0

Syntax: Say "'"name"' is not a valid Unicode name."
Exit 1
```

The Rexx Preprocessor for Unicode (2/2)

You can then readily call your program using `rxu`:

```
C:\test>rxu isnamevalid guitar
The name 'guitar' is a valid Unicode name.
It corresponds to '🎸' ('1F3B8'U).
C:\test>rxu isnamevalid papoola
'papoola' is not a valid Unicode name.
```

The preprocessor invokes the Rexx Parser to obtain a parsed version of your program, which is then translated into a standard ooRexx program, in this case `isnamevalid.rex`.

The translated program is called with the supplied arguments, and the return code is stored by `rxu`; the translated program is then deleted, and the return code is passed to the caller.

The net effect is that you can write Unicode-extended programs, give them a `.rxu` extension, and call them with the `rxu` command, in the same way that you can write a `.rex` program and call it using the `rexx` command.

The rxutry.rex utility

The `rxutry` utility is a version of `rexxtry` modified to support Unicode.

```
C:\test>rxutry
REXX-ooRexx_5.1.0(MT)_64-bit 6.05 21 Nov 2024
👉rxutry.rex lets you interactively try Unicode-REXX statements.
    Each string is executed when you hit Enter.
    Enter 'call tell' for a description of the features.
👉   Options DefaultString is Text
👉   Options Coercions     is Promote
  Go on - try a few...            Enter 'exit' to end.
Say "Jose" || "301"U
José
  ..... rxutry.rex on WindowsNT
couple = Text("(Man)(Zero Width Joiner)(Woman)"U)
  ..... rxutry.rex on WindowsNT
Say couple Length(couple) Length(Bytes(couple))
👫 1 11
  ..... rxutry.rex on WindowsNT
exit
C:\test>
```

The net-oo-rexx software bundle and ooRexxShell

Rony Flatscher is curating a nice software bundle, called *net-oo-rexx*, which puts together a number of packages, including TUTOR and the Rexx Parser. After installing net-oo-rexx, you will enjoy immediate access to all the packages, without having to individually install every one of them separately.

```
D:\Dropbox\net-oo-rexx-packages>oorexxshell
loadPackage OK for extension/stringChunk.cls
(...)
Unicode-REXX (TUTOR) loaded
    Options DefaultString is Text
    Options Coercions     is Promote
REXX-ooRexx_5.1.0(MT)_64-bit 6.05 20 Feb 2025
Input queue name: S00000000000015ECQ000001809C06EC70
D:\Dropbox\net-oo-rexx-packages
ooRexx[CMD]>
```

In particular, net-oo-rexx includes a copy of Jean Louis Faucher's ooRexxShell, a RexxTry-like environment with many useful extensions and enhancements, distributed with his Executor ooRexx variant. The version of ooRexxShell distributed with net-oo-rexx automatically loads TUTOR at initialization time, and therefore you can choose to use ooRexxShell instead of rxutry if you want to play with TUTOR.

Acknowledgements

Jean Louis Faucher has integrated TUTOR into ooRexxShell, and Rony Flatscher has included TUTOR and the Rexx Parser in the net-oo-rexx distribution. TUTOR could not have been developed without the extensive conversations held in the Rexx Architecture Review Board (ARB) in 2024, in particular in meetings with René Jansen and Jean Louis Faucher.

I also want to thank my colleagues at EPBCN, Laura Blanco, Silvina Fernández, Mar Martín, David Palau, Olga Palomino and Amalia Prat, who have read several drafts of this presentation and helped to improve it with their comments and suggestions.

Questions?

References

TUTOR can be downloaded at:

- https://rexx.epbcn.com/TUTOR/ (preferred: better Rexx highlighting)
- https://github.com/JosepMariaBlasco/TUTOR

The Rexx Parser can be downloaded at:

- https://rexx.epbcn.com/rexx-parser/ (preferred: better Rexx highlighting)
- https://github.com/JosepMariaBlasco/rexx-parser

Executor can be downloaded at:

- https://github.com/jlfaucher/executor

The net-oo-rexx bundle can be downloaded at:

- https://wi.wu.ac.at/rgf/rexx/tmp/net-oo-rexx-packages/