# The Rexx Highlighter

## 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 6th, 2025

# Notice

*This whole document[1] is an experiment in CSS printing. It comfortably mixes normal text and programs beautified by the heavy prettyprinting produced by the Rexx Highlighter, and it can be viewed both as a standard web page and as a slide show.*

*If you are viewing this file as a PDF, chances are good that you are looking at it as a slide show. If you are viewing it as a web page, the suggested print settings to produce a PDF containing a slide show 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): https://www.epbcn.com/pdf/josep-maria-blasco/2025-05-06-The-Rexx-Highlighter.pdf.↩

# The Rexx Highlighter

Introduction

Architecture

Features

The three highlighters

Tools

Further work

Acknowledgements

# Introduction

# Introduction

**The Rexx Highlighter** is a child project of the Rexx Parser, and was patient and lovingly written by Josep Maria Blasco in the last half of 2024 and the first half of 2025. A working knowledge of the functionality of the Rexx Parser is required to be able to follow this presentation.[2]

Developed around a common code base, the Highlighter currently includes output drivers for **three modes**: HTML, ANSI terminals emulators, and (Lua)LaTeX.

````
~~~rexx
Say "Done!" -- Inform the user
~~~
````

```rexx
Say "Done!" -- Inform the user
```

The figure above shows the HTML highlighter in action, or, to be more precise, the effect of a Rexx fenced code block in a Markdown file.

---

2. Please refer to the extensive documentation at `https://rexx.epbcn.com/rexx-parser/doc/`, or to the accompanying presentation, `https://www.epbcn.com/pdf/josep-maria-blasco/2025-05-05-The-Rexx-Parser.pdf`.↩

# Architecture

The HTML class assignment system

Coarse- vs. fine-grained class assignments

Specifying a style with CSS

The three highlighters

Tools

Future work

Acknowledgements

# The HTML class assignment system

Irrespective of whether we are going to produce output for ANSI terminals, HTML, or LaTeX, the Highlighter assigns one or more **HTML class names** to every element category and subcategory. This is done in the `HTMLClasses` routine.

In many cases, an element category is mapped directly to **a single HTML class**. For example, `.EL.SHEBANG`, the category that identifies shebang lines, is assigned the class `"shb"`.

In many other cases, an element category is mapped to **two HTML classes**: the first is **a generic one**, which is the same for a whole set of elements, and the second one is **a more specialized one**, which uniquely identifies the category. For example, `.EL.OP.MULTIPLICATION`, which identifies the `"*"` operator, is assigned a generic class of `"op"` (for "operator"), and also a specialized class of `"mul"`.

These class names are then prefixed with a customizable prefix (by default, `"rx-"` is used) to avoid conflicts with classes defined by other programs or frameworks. In our example, the `"*"` operator would be assigned classes `"rx-op"` and `"rx-mul"`.

# Coarse- vs. fine-grained class assignments

The element categories that may be assigned two classes instead of only one class belong to one the following sets: *assignments* (`"asg"` + a specific class), *operators* (`"op"` + a specific class), *special characters and special character sequences* (`"spe"` + a specific class) and *taken constants* (`"const"` + a specific class).

Depending on the supplied program options, each run of the Highlighter may assign one or both of the classes to the elements that have a category belonging to each of these sets.

For example, when using `rexx` fenced code blocks, the `operator` attribute may have a value of `"group"` (assign the first class only), `"detail"` (assign the second class only), or `"full"` (assign both classes).

This mechanism allows to choose, for every of the above sets of elements, between very fine-grained and more coarse-grained class assignments on a run-by-run basis.

```
~~~rexx {assignment=detail special=group operator=full constant=detail classprefix="P"}
```

# Specifying a style with CSS

Once the HTMLClasses routine has mapped the element categories and subcategories to their respective HTML classes, a CSS style file is used, to transform these classes into effective highlighing attributes.

The Highlighter is distributed with two sample style files, `rexx-light.css` and `rexx-dark.css`, although you can of course create your own.

```rexx
Say "Done!" -- Inform the user
```

```rexx
Say "Done!" -- Inform the user
```

The choice of style file is determined by the `style=` attribute in Rexx fenced code blocks, by the `--style=` option of the *highlight* utility, and by similar mechanisms when using other tools. Specifying `style=`*`name`* will select the `rexx-`*`name`*`.css` style file.

CSS files are fetched by the browser when using the HTML highlighter, and are parsed by the Highlighter when other output formats are requested. In the latter case, only a very limited subset of the CSS specification is recognized.

# Style patches

The **style patch system** allows one-time, simple and easy patching of a CSS style.

```
Say "Done!" -- Inform the user
```

*Standard highlighting*

```
Say "Done!" -- Inform the user
```

*With* `patch="all comments yellow"`

*Inline style patches* can be specified by using the `patch=` attribute of a Rexx fenced code block, or the `--patch` option of the Highlight utility. This mechanism is suitable for very small patches, which fit on a single line. For example, the patched code block above has been created with

```
~~~rexx {patch="all comments yellow"}
```

*Larger patches* can be stored in a separate file (see the `patchfile` and `--patchfile=` options). The *parse* method of the Highlighter class also accepts an optional style patch argument.

# Features

# Variable symbol highlighting (1/2)

Depending on the context, an element with the morphology of a variable symbol may play the role of a *local variable*, of an *instance variable*, of an instruction or directive *keyword*, or be taken as a *constant*, like in the case of labels or method names.

The Rexx Parser assigns different element categories to each symbol, depending on its role (and, in the case of taken constants, it also assigns different subcategories), and these differences are passed to the various highlighter versions, so that they can highlight each case appropriately.

# Variable symbol highlighting (2/2)

```
::Method then                           -- A method name
  Expose then expose                    -- Instance variables
If:                                     -- "If" is a label here
  If (then = 2)                         -- "then" is an instance variable
    Then else = 3                       -- "Then" keyword, "else" variable
    Else Do Label 4                     -- Three keywords
      end = 4                           -- A variable called "end"
      Signal Then                       -- Signaling the "then" label
    End 4                               -- A keyword
  self~then                             -- "then" as a method name
Then:                                   -- "Then" is a label
  value = Else()                        -- A ::ROUTINE call

::Routine Else                          -- A ::ROUTINE name
  Use Arg while                         -- Assign to the "while" variable
  Loop Label Forever forever = 1 By 1 - -- "forever", label and control variable
       While (while > forever)          -- Two uses of "while"
    If forever > 16 Then Return while   -- "forever" and "while" are variables
  End Forever                           -- "Forever" as a label
  Return forever                        -- "forever" as a variable
```

# Compound variable highlighting (1/2)

**Compound variables** are special, in the sense that they have two simultaneous aspects: they are, at the same time, *variables*, and *indexed stem references*. The Rexx Highlighter honours this duality by returning compound variables as single elements that include a number of *sub-parts*; you can decide which of the two aspects of a compound variable will determine the highlighting mode.

```rexx
-- As an indexed reference
Say Matrix.1.2A.j..
```

```rexx
-- As a whole
Say Matrix.1.2A.j..
```

You can select the compound variable highlighting mode using the `compound=true|false` attribute on the `rexx` fenced code block marker. The default is to highlight all the components individually.

````
```rexx {compound=true}
(rexx code goes here)
```
````

````
```rexx {compound=false}
(rexx code goes here)
```
````

# Compound variable highlighting (2/2)

When highlighted as a single element, a compound variable will have a class of `.EL.COMPOUND_VARIABLE` or, if the variable is exposed (i.e., it is an instance variable), of `.EL.EXPOSED_COMPOUND_VARIABLE`.

```
::Method myMethod
  Expose var stem.
  local = var + 1
  Say stem.12..2E.var.local
```

When taking sub-parts into account (which is the default), different highlighting attributes will be used for the *stem name* (a `.EL.STEM_VARIABLE` or a `.EL.EXPOSED_STEM_VARIABLE`) and for all the components of its *tail*. The first dot in a compound variable is part of the *stem name*. The rest of the symbol, the *tail*, is an arbitrary sequence of: *variables* (either *local*, `.EL.SIMPLE_VARIABLE`, or *exposed*, `.EL.EXPOSED_SIMPLE_VARIABLE`); signless *integers* (`.EL.INTEGER_NUMBER`); pure dotless *constant symbols* (`.EL.SYMBOL_LITERAL`), and *tail separators* dots (`.EL.TAIL_SEPARATOR`).

# Function and subroutine calls

The Rexx Highlighter is able to differentiate between *internal*, *built-in*, *local* `::ROUTINE`, *namespaced* `::ROUTINE` and *external* function and subroutine calls.

```
  len =  Length(  var )                -- Call a BIF as a function
  Call   Length   var                  -- Call a BIF as a subroutine
  b   =  Verify(  var )                -- Internal routine, as a function
  Call   Verify   var                  -- Internal routine, as a subroutine
  Call  "VERIFY"  var                  -- BIF subroutine call
  Call  "Verify"  var                  -- External subroutine
  Call   meaningOfLife                 -- Package-local ::ROUTINE, function
  Call   Name:myRoutine                -- External ::ROUTINE as a subroutine
  Call   External                      -- External subroutine
  Signal On Syntax Name  Verify        -- A label
  Call   On Error  Name  Length        -- A BIF
  Call   On Error  Name "Length"       -- External

"VERIFY": Return .True                 -- An internal routine

::Routine meaningOfLife; Return 42     -- A locally defined ::ROUTINE
```

# Taken constant highlighting

In many places of the Rexx syntax, a "taken constant" is required. Generally speaking, a "taken constant" is a string or a symbol which is taken as a constant, although some few contexts may impose additional limitations to the acceptable elements.

```
::Class     myClass  Public              -- A class name
::Method    myMethod Class               -- A method name
  Return    myRoutine(12)                -- Calling a ::Routine
::Routine   myRoutine                    -- A routine name
  Return  .Resources[myResource][2]|| -
   .myClass~myMethod                     -- A message term
::Resource myResource End "The end"      -- A resource
A resource line
Another resource line
This is line number 3
The end is near (additionally, "is near" and what follows it are ignored)
```

The Highlighter assigns different highlighting classes to every subcategory of taken constants. This allows to specify different highlighting choices for labels, method, routine and resource names, etc.

# Documentation comments

**Documentation comments** or **doc-comments** are a special form of comment, similar to JavaDoc comments.

Doc-comments can be placed before a directive, or before a callable label.

**Standard doc-comments** start with `"/**` and end with `"*/"`.

```
/*****************************************************************************/
/* This is a set of three classic comments, forming a box.                   */
/*****************************************************************************/

/**
 *  This is a doc-comment. It Starts with "/**" and it ends with "*/", and it
 *  is placed immediately before a directive. A style patch has been applied
 *  to highlight the doc-comment in reverse fuchsia, and a pad of 80 has been
 *  specified as a fenced code block attribute, to embellish the display.
 */
::Routine R
```

# Markdown doc-comments

**Markdown doc-comments** are contiguous sequences of line comments starting with exactly three dashes (that is, with `"---"` but not with `"----"`).

```
--------------------------------------------------------------------------------
-- This is a set of three line comments, forming a box.                       --
--------------------------------------------------------------------------------

---
--- This is a markdown doc-comment. It Starts with "---", and it is placed
--- immediately before a callable label. A style patch has been applied
--- to highlight the doc-comment in yellow over dark blue, and a pad
--- of 80 has been specified as a fenced code block attribute,
--- to embelish the display.
---
Proc: Procedure Expose a b c
```

# Returned doc-comment elements

Documentation comments are always returned as *a single element*.

- In the case of standard doc-comments, the Rexx Parser adds to the element all the whitespace which can be found, if any, in the first comment line, before the first `"/"` character, and in the last line, after the last `"/"` character. The returned element has an element category of `.EL.DOC_COMMENT`.
- In the case of Markdown doc-comments, the Rexx Parser combines all the line comments, and preceding whitespace, if present, into a single element. The returned element has an element category of `.EL.DOC_COMMENT_MARKDOWN`.

# The three highlighters

# The HTML highlighter (1/2)

**The HTML highlighter** wraps every non-inserted element in a `<span>` tag that will contain its corresponding HTML class or classes. For example, when `operator=full` is in effect, a multiplication operator `"*"` might be transformed into:

```
<span class="rx-op rx-mul">*</span>
```

These `<span>` tags are in turn collected in lines, which are enclosed in `<code>` tags. If the user has requested line numbering, the value of the `lineno` attribute will be added to the printed line by using a `::before` pseudo-element. Indentation has been added to the following figure to improve legibility.

```
<code lineno="n">
  <span tag 1>
  ...
  <span tag n>
</code>
```

# The HTML highlighter (2/2)

Lines are then wrapped inside a `<pre>` tag, and the `<pre>` block is itself enclosed in a `<div>`:

```
<div class="highlight-rexx-dark"><pre>
  (code lines)
</pre></code>
```

The `<div>` tag has a class of `highlight-rexx-`*`style`*, where *style* is the style specified in the corresponding program options. Style patches are handled by adding a random `id` attribute to the `<div>` tag, which will be met by a corresponding inline `<style>`:

```
<div id="rx63e218035994" class="highlight-rexx-dark">
  <style>
    #rx63e218035994 .rx-var {font-weight:bold; color:#000000; background-color:#cccc00; }
  </style>
  <pre>
    ...
```

# The ANSI highlighter (1/2)

**The ANSI Highlighter** uses ANSI SGR (*Select Graphic Rendition*) codes to highlight a Rexx program. Only a subset of all the ANSI SGR codes are used, namely (ESC denotes character "1B"X):

| Code | Meaning |
|---|---|
| `ESC[0m` | All attributes off |
| `ESC[1m` | Bold |
| `ESC[3m` | Italic |
| `ESC[4m` | Underline |
| `ESC[38;2;r;g;bm` | Set foreground colour to RGB(*r*,*g*,*b*) |
| `ESC[48;2;r;g;bm` | Set background colour to RGB(*r*,*g*,*b*) |

Different SGR codes can be combined in a single sequence by separating them using semicolons. For example, `ESC[1;3;38;2;255;0;0m` means "choose a bold font, choose an italic font, and set the foreground colour to pure red".

# The ANSI highlighter (2/2)

Terminal applications and emulators differ wildly in their support of ANSI SGR codes. The default configuration of the Windows terminal, for example, implements boldface by substituting colours with a lighter version of themselves, and that, only for basic colours (i.e., codes 30-37, 40-47, 90-97 and 100-107); the same is true of the default Ubuntu terminal under WSL.

Additionally, support for basic colours is abundantly erratic: see, as an example, the colour chart that appears in the Wikipedia. In an attempt to minimize this problem, all the colours generated by the ANSI highlighter are emitted using the 38 and 48 codes, i.e., they are pure RGB colours. CSS named colours[3] are also translated to their RGB definitions: you can be sure that foreground `blue` will be generated as `#0000ff`, that is, as `ESC[38;2;0;0;255m`. How this displays in your emulator may vary.

---

3. https://www.w3.org/TR/css-color-4/#named-colors↩

# The LaTeX highlighter (1/2)

The current version of the LuaLaTeX highlighter is based on the following **packages**:

- `xcolor`, for basic colour support.
- `lua-ul`, for proper background highlighting.
- `listings`, to hold the listings themselves.
- `tcolorbox`, to fix an annoying problem with extra space appearing between listing lines in some circumstances.

The default **mono font** does not support **boldface**. If you need boldface in your highlighting schemas (the default ones supplied with the Rexx Parser use boldface), you should use a different mono font. The Hack font[4], for example, seems to produce decent results.

The LuaLaTeX highlighting framework defines `\textexclamup` as `!`: the exclamation mark is used as an escape character, and the command is needed when the exclamation appears in a Rexx program.

---

4. https://sourcefoundry.org/hack/.↩

# The LaTeX highlighter (2/2)

You will most probably need to adjust **the size of your font**, depending on the page size, and **the number of characters** you want displayed on the largest line. The following style definition, for example, allows for exactly 80 characters per line, when using the Hack font and an A4 paper size:

```
\lstdefinestyle{rexx}
{
    basicstyle=\fontsize{8.0pt}{11.0pt}\selectfont\color{white}\ttfamily
}
```

## Notice

The LaTeX highlighter should be considered **experimental**. It would need some love from a LaTeX expert. Any volunteers?

# Tools

- Style patch syntax
- The command line: `highlight.rex`
- Fenced code blocks: `FencedCode.cls`
- Apache httpd CGI integration

# Style patch syntax (1/3)

**Style patches** follow a very simple syntax:

- A style patch is an ordered set of lines, separated by semicolons, or line-end separators, or both.
- Leading and trailing blanks are removed.
- Null lines and comments (i.e., lines starting with `"--"`) are ignored.
- Once comments are discarded, dashes `"-"` are replaced by blanks `" "` (may come useful when specifying patches in a Linux shell).

Highlighting patches for **element categories**, **category sets**, and **taken constant names**:

```
Element category     highlighting
All     set          highlighting
Name    constantName highlighting
```

# Style patch syntax (2/3)

Element *categories* may omit the `"EL."` prefix; *sets* may omit the `"ALL."` prefix; and *constant names* may omit the `".NAME"` suffix.

*Highlighting* is a blank-separated sequence of case-insensitive elements, selected between

- **Foreground colours**, in the format `#rgb`, `#rrggbb`, or one of the 147 standard CSS named colours.
- **Foreground/background colour combinations**, in the format `fg/bg` (with no blanks), where `fg` and `bg` are either `#rgb`, `#rrggbb`, or one of the 147 CSS named colours.
- One of the single words **bold**, **italic** or **underline**, optionally preceeded by the single word **no**.

# Style patch syntax (3/3)

A simple patch:

```
-- Patch simple variable elements to display as bold black over 75% yellow
element EL.SIMPLE_VARIABLE #000/#cc0 bold
-- Patch method names to display as black over 75% magenta
name    METHOD.NAME        #000/#c0c
```

The same patch, abbreviated:

```
E SIMPLE_VARIABLE #000/#cc0 bold; N METHOD #000/#c0c
```

The patch in action:

```
::Method methodName
  len = Length("String")
  n   = Pos("x", value)
```

# The command line: `highlight.rex`

The `highlight` utility processes a file and highlights it according to a set of options. If the file has a `.md` or a `.html` extension, all Rexx fenced code blocks are processed; in all other cases, the whole file is highlighted.

Except for `.html` files, the default highlighting mode is ANSI when called from the command line, and HTML otherwise.

Options:

- choose the highlighting schema;
- specify an optional patch, or a patch file;
- determine whether lines are numbered, and, if yes, what is the number of the first line printed;
- enable support for the TUTOR-flavoured Unicode dialect;
- specify that all lines will have a minumum line width (ANSI only).

# Fenced code blocks: `FencedCode.cls`

The **FencedCode** routine is a Rexx fenced code block preprocessor. It processes code blocks enclosed between `~~~rexx` and `~~~` markers (starting on column 1, optional attributes enclosed in braces are admitted in the start marker).

It is **language-agnostic**, that is, it happily processes HTML files, but also Markdown files, etc., as long as they contain the required markers.

It is highly customizable, using an **extensive set of options**.

It can be **integrated** into any application by calling it directly; it will automatically be invoked by default by the `highlight` utility when the extension is `html` or `md`; it also can be easily integrated in your CGI infrastructure.

# Apache httpd CGI integration

The Rexx Parser package includes a sample Markdown CGI processor which automatically invokes the *FencedCode* routine, and then runs Pandoc[5] to dynamically produce HTML code. The preprocessor (which should be customized and adapted to fit your needs, should you want to use it) creates Bootstrap 3.0 code,[6] and includes optional exits for headers and footers, sidebars, etc.

The document you are reading has been served by this experimental CGI processor.

The processor installs as an Apache httpd action handler, which is invoked by the httpd daemon when certain files (normally, files with a `.md` extension) are served:

```
Action RexxCGIMarkdown /some/path/CGI.markdown.rex
# ...
<Files *.md>
  SetHandler RexxCGIMarkdown
</Files>
```

5. https://pandoc.org/.↩
6. https://getbootstrap.com/docs/3.4/.↩

# Further work

# Further work

- Based on experimentation and user feedback, collect a set of "things that work" and document them. For example,
  - which emulators display ANSI highlighting correctly, or
  - how to configure certain emulators so that they work correctly;
  - which fonts are suitable for certain forms of highlighting (if possible, for all forms of highlighting);
  - etc.
- Improve the LaTeX highlighter.
  - Possible integrations with packages other than *listings* (maybe *minted*?).
- ...

# Acknowledgements

# 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.

Some preliminary versions of the Rexx Parser and its accompanying subproject, the Rexx Highlighter, have been tested by different members of the RexxLA community, including, but not limited to Gilbert Barmwater, Jean Louis Faucher, Rony Flatscher, Ruurd-Jan Idenburg, René Jansen and Till Winkler; I want to thank them all for their observations and enhancement proposals.

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

# References

Concepts explained elsewhere, in other documents:

- Parsing **element**
- Element **category**
- Element **subcategory**

The `highlight` utility: https://rexx.epbcn.com/rexx-parser/doc/utilities/highlight/

Documentation about the Rexx Highlighter can be found at:

- The highlighter: https://rexx.epbcn.com/rexx-parser/doc/highlighter/
- The HTMLClasses routine: https://rexx.epbcn.com/rexx-parser/doc/highlighter/htmlclasses/
- The FencedCode routine https://rexx.epbcn.com/rexx-parser/doc/highlighter/fencedcode/
- The StylePatch class: https://rexx.epbcn.com/rexx-parser/doc/ref/classes/stylepatch/