RCI/RSE

Regex Search and Replace for XyWrite 4/Windows

C.L. Distefano

rev. May 13, 2018

1. Introducing RCI/RSE and REGEX

`RCI` and its companion routine `RSE` use a Perl-compatible regular expression (regex) engine to perform search-and-replace operations analogous to XyWrite's `CI` (Change Invisible) and `SE`arch commands.

Another utility, `REGEX`, takes a regular expression and lists all matches in the subject file.

Regular expressions add power and flexibility to XyWrite's native `SEarch` and `CHange` commands. Among many other advantages, you'll be able to reorder capturing groups—the regex equivalent of XyWrite's `SE`arch wildcards—on the replacement side of the change. You can also search for arbitrarily large blocks of text and replace them with other blocks of text. And much more . . .

1.1. The Power of Regular Expressions

Although regexes have a well-deserved reputation for geeky complexity, the basics are straight-forward—and the rewards immediate. To ease the learning curve, RCI and RSE make regexes *Xy-friendly*: XyWrite wildcards can be used in the search string, and are translated into their regex equivalents. Also, once you've perfected a regex, you can save it for future re-use. See "Canned Searches" (Section 5, below).

If you're frustrated by XyWrite's error message *Wildcards must be in the same order on both sides of a change*, RCI is for you. Text captured by wildcards (regex capturing groups) on the search side can be reshuffled in whatever order you like on the replace side. For example—and this only hints at the possibilities—you can transpose all dates in the form `MM/DD/YYYY` to `YYYY-MM-DD`. Simple, you say—but try doing it with XyWrite wildcards: you can't.

1.2. Installation

RCI is bundled with the XyWWWeb Jumbo U2. Required external programs are included in `U2EXTRAS.ZIP`. Unzip all files into Editor's directory.

1.3. RCI Constituent Files

AI3.EXE AutoIt v3 EXEcutable for Win32 (compressed)

RCI.A3X RCI (Change module) .A3X executable

RCI.BAT Batch file for sequential search-and-replace operations

RCI.EXE RCI (Change module) EXEcutable

RCI.FRM frames RCI, RSE, REGEX and related U2 routines

RSE.A3X RSE (SEarch module) .A3X executable

REGEX.EXE Regex search utility EXEcutable

REGEX.A3X Regex search utility .A3X executable

REGEX.PDF Excerpt from AutoIt documentation on regexes

RCI.PDF This documentation file[*].

[*] Written in XyWrite 4 and typeset with LaTeX

1.4. Two “Flavors”: U2 and Windows

RCI, RSE, and REGEX are U2 frames for XyWrite 4 + the XyWWWeb Jumbo U2. RCI.EXE and REGEX.EXE are Windows command-prompt EXEcutables (compiled AutoIt scripts). The EXEcutables, as standalone Windows programs, are relevant for users of XyWrite III+ or Nota Bene for DOS or Windows, as well as for XyWrite 4 users.

1.5. Useful Information

Appendix A has a list of XyWrite search wildcards available for use with RCI/RSE/REGEX, along with their regex equivalents. **Appendix B** is a cheat sheet for Perl-compatible regular expressions.

2. Setting SEARCH.TXT

Before running RCI.EXE or REGEX.EXE, you must set a search string/regex—and, for RCI.EXE, a replacement string—in a separate file, SEARCH.TXT, which must be located in the same directory as RCI.EXE and REGEX.EXE. The format for SEARCH.TXT is:

```
search_string [regex]
|---|
replacement_string
```

Search String (Regex). The regex on the search side of the separator may include newlines, represented by actual CrLfs, the XyWrite carriage-return wildcard $\boxed{\texttt{<=}}$, wildcard function WC, or the regex tokens \r\n.

Separator. The separator consists of the string |---| with a newline fore and aft.

If the separator is omitted, the *entire contents* of SEARCH.TXT will be the search string and, if found, will be replaced by nothing—*i.e.,* deleted.

Replacement String. In general, the replacement string is a string literal. There are, however, three important exceptions:

- Regex special characters must be “escaped” with \. See subsection 2.2, below.

- Text captured by capturing groups on the search side is represented by $1, $2, $3 . . . (or \1, \2, \3, . . .) on the replacement side. See subsection 3.3.2, below.
- XyWrite's CrLf wildcard `<=` or function `WC` may be used instead of an actual newline.

2.1. Case Sensitivity

All searches are case-sensitive, unless the search string contains regex option `(?i)`. Case-sensitivity starts at the point in the search string where `(?i)` appears. Thus, `abc(?i)DEF` matches *abcDEF* and *abcdef*, but not *ABCDEF*; while `(?i)abcDEF` matches all case variants of *abcdef*.

2.2. Regex Special Characters

The following characters have special meanings in regular expressions:

```
    \ . ^ $ | [ ( { * + ? #
```

To include any of them in your search string, you must "escape," or "quote," them—*i.e.,* prepend `\`, the regex escape character. Thus, `d:\path\myfile.txt` becomes `d:\\path\\myfile.txt`; `$5.00` becomes `\$5\.00`; and so forth.

2.3. Replacing Blocks of Literal Text

The search regex may consist of a block of text of any length. Be careful, though: longer text passages will likely contain one or more regex special characters (such as `.`), which will have to be escaped with \. In the alternative, long blocks of literal text can be enclosed between the regex quote and end-quote tokens, `\Q` . . . `\E`, thus avoiding the need to escape special characters contained in the text:

```
\Q[existing block of text]\E
|---|
[replacement text]
```

The length of the search or replace string is virtually unlimited. I've successfully performed search and replace with strings the size of *Moby-Dick* in its entirety—that's 1.2 megabytes—on either side of the search.

3. Windows Command-Prompt Syntax

This section assumes that search and replacement strings have been set in `SEARCH.TXT` (Section 2, above). `RCI.EXE` requires both search and replacement strings in `SEARCH.TXT`. `REGEX.EXE` requires only a search string; any replacement string in `SEARCH.TXT` is ignored.

3.1. RCI.EXE: Search and Replace

`RCI.EXE` accepts four arguments, all optional. The usage is:

```
RCI.EXE file_in file_out start_pos number_of_changes
```

3.1.1. Default arguments

If you omit arguments, `RCI.EXE` defaults to the following:

`file_in: ORIGINAL.TXT`, in same directory as `RCI.EXE`
`file_out: CHANGED.TXT`, in same directory as `RCI.EXE`
`start_pos: 0` (search from top of file_in)
`number_of_changes: 0` (change all instances)

To alter some defaults but not others, use a dot (.) to represent the default value of an argument. For example:

```
RCI.EXE d:\path\my.txt . . 10
```

Translation: Use `d:\path\my.txt` as the subject file, save the changed file as `CHANGED.TXT`, start search from top of `my.txt`, make a maximum of `10` changes.

Note

RCI does not modify the subject file unless you instruct it to do so, by making the first and second arguments point to the same filename:

```
RCI.EXE my.txt my.txt
```

3.2. REGEX.EXE: List All Regex Matches

`REGEX.EXE` accepts three arguments, all optional. The usage is:

```
REGEX.EXE file_in file_out start_pos
```

3.2.1. Default arguments

If you omit arguments, `REGEX.EXE` defaults to the following:

`file_in`: `ORIGINAL.TXT`, in same directory as `REGEX.EXE`
`file_out`: `RESULTS.TXT`, in same directory as `REGEX.EXE`
`start_pos`: `0` (search from top of `file_in`)

To alter some defaults but not others, use a dot (.) to represent the default value of an argument. For example:

```
REGEX.EXE d:\path\my.txt . 3599
```

Translation: Use `d:\path\my.txt` as the subject file, save the changed file as `RESULTS.TXT`, start search from character position `3599` in `my.txt`.

3.3. Examples

3.3.1. Search and Replace Using Big Blocks of Text

`SEARCH.TXT:`

```
\QI've had a perfectly wonderful evening. But this wasn't it.\E
|---|
I didn't attend the funeral, but I sent a nice letter saying
I approved of it.
```

☞Note the use of `\Q ... \E` to enable literal quoting.

Command:

```
RCI.EXE
```

Changes are in `CHANGED.TXT`.

3.3.2. Search and Replace Using XyWrite Wildcards

`SEARCH.TXT`:

In XyWrite 4 the search string may contain search wildcards $\boxed{\texttt{A}}$, $\boxed{\texttt{L}}$, $\boxed{\texttt{N}}$, $\boxed{\texttt{S}}$, $\boxed{\texttt{W}}$, $\boxed{\texttt{X}}$, $\boxed{\texttt{<=}}$, $\boxed{\texttt{10}}$, $\boxed{\texttt{13}}$, $\boxed{\ \cdot\ }$, $\boxed{\ll}$, and $\boxed{\gg}$. The first seven of these are available in XyWrite III+; alternatively, the search string may use wildcard functions `WA`, `WL`, `WN`, `WS`, `WW`, `WX`, and `WC`, in lieu of the corresponding wildcards.

On the replacement side, "back-references" to the text matched by wildcards are made with $1, $2, $3 ... (or \1, \2, \3 ...), where $1 (\1) is the first wildcard, $2 (\2) is the second wildcard, etc.

So, to make the date-format change from `MM/DD/YYYY` to `YYYY-MM-DD`, you could set `SEARCH.TXT` as follows:

$\boxed{\texttt{N}}\boxed{\texttt{N}}/\boxed{\texttt{N}}\boxed{\texttt{N}}/\boxed{\texttt{N}}\boxed{\texttt{N}}\boxed{\texttt{N}}\boxed{\texttt{N}}$

```
|---|
$5$6$7$8-$1$2-$3$4
```

Wildcard functions can be used instead of the wildcards themselves (for example, if you're using XyWrite III+):

```
WN WN /WN WN /WN WN WN WN /
|---|
$5$6$7$8-$1$2-$3$4
```

You could also use an actual regex instead of wildcards, with the capturing groups (in parens) corresponding to \1, \2, \3... or $1, $2, $3, etc.:

```
(\d{2})/(\d{2})/(\d{4})
|---|
$3-$1-$2
```

Or, more precisely and narrowly, covering the years from 1900 to 2099, inclusive:

```
([01]?\d)/([0-3]?\d)/((19|20)\d\d)
|---|
$3-$1-$2
```

(In the second and third examples, the numbered references, \1, \2 . . . ($1, $2, . . .) refer to the capturing groups in parentheses, which is why there are fewer than in the wildcard example.)

Note 1

If XyWrite wildcards (or wildcard functions) are used in the search string, regex matching becomes "lazy" (regex default is "greedy") and searches span newlines—XyWrite's native behavior. Appendix B has further information.

Note 2

`RCI/RSE/REGEX` define wildcard $\boxed{\texttt{W}}$ as matching *any number* of characters, including newlines. As XyWrite's 79-character limit is removed, there is never a need to use multiple $\boxed{\texttt{W}}$s.

Note 3

The search/replace examples can be run with `REGEX.EXE` instead of `RCI.EXE`. With `REGEX.EXE`, search "hits" and their character positions in the subject file are listed in `RESULTS.TXT`. The subject file is not modified.

4. XyWrite 4 (U2) Usage

4.1. RCI: Regex Change Invisible

4.1.1. Commnd Line (“CMline”)

```
RCI[/#][/S|/T] [sep]search_regex[sep]replacement_string[sep]<Helpkey>
[/#]  = maximum number of changes to make (default = change all)
[/S]  = search within Selected text (DeFined block) only
[/T]  = search from top_of_file regardless of cursor position
        default = search from cursor position to end of file
[sep] = separator character not contained in search or replace strings
```

If the search and replace are set in SEARCH.TXT (see Section 2, above), then the usage is simply:

```
RCI[/#][/S|/T]<Helpkey>
```

Note 1

RCI does not modify the subject file unless you instruct it to do so, by making the first and second arguments point to the same filename:

```
RCI.EXE my.txt my.txt
```

Note 2

The number of changes reported by frame RCI is often higher than the number that would be reported by XyWrite’s native CI command. This is because the regex engine counts the number of *captured groups* matched by the regex, not the number of matching strings. Thus, for example, with the date format change

RCI /$\boxed{\texttt{N}}\boxed{\texttt{N}}$-$\boxed{\texttt{N}}\boxed{\texttt{N}}$-$\boxed{\texttt{N}}\boxed{\texttt{N}}\boxed{\texttt{N}}\boxed{\texttt{N}}$/$5$6$7$8-$1$2-$3$4/<Helpkey>

if the search string matches once, the number of changes reported will be 8, because each $\boxed{\texttt{N}}$ wildcard represents a captured group and there are eight $\boxed{\texttt{N}}$ wildcards in the match. XyWriters will find this strange—but that’s the way it is.

4.1.2. XPL Usage

For example:

```
«SV50,"Merry"Happy"»JM 2.rci/tQ2 ;*;
```

4.2. RSE: Regex SEarch

```
RSE[/S|/T] [sep]search_regex[sep]<Helpkey>
/S    = search within Selected text (DeFined block) only
/T    = search from top_of_file regardless of cursor position
        default = search from cursor position to end of file
[sep] = separator character not contained in search or replace
          strings (optional -- see note)
```

Operation is from cursor position to end of file, or within a `DeFined` block.
☞ The search string (regex) may—but need not—be enclosed in separators.

If the regex is set in `SEARCH.TXT`, the usage is:

```
RSE[/S|/T]<Helpkey>
```

4.3. REGEX: Regex Match Lister

```
REGEX[/S|/T] [sep]search_regex[sep]<Helpkey>
/S    = search within Selected text (DeFined block) only
/T    = search from top_of_file regardless of cursor position
        default = search from cursor position to end of file
[sep] = separator character not contained in search or replace
          strings (optional -- see note)
```

Operation is from cursor position to end of file, or within a `DeFined` block.

Results are listed in `RESULTS.TXT`, located in Editor's directory. Use Helpkey to execute the `CMline` eXtended Macro to jump to any search result in the subject file.

☞The search string (regex) may—but need not—be enclosed in separators.

If the regex is set in SEARCH.TXT, the usage is:

```
REGEX[/S|/T]<Helpkey>
```

5. “Canned” Searches

Frame setRCI* supports “canned” search/replace strings (write once, use many times). Search/replace pairs are stored in your U2 file as separate frames. Each must have its own frame, and the frame name must start with Rx-. In the frame, the search/replace string must be saved to Save/Get 49 in SEARCH.TXT format—see Section 2. For example:

```
{{5Rx-Merry}} "Merry" to "Happy"
{002}«SV49,Merry
|---|
Happy»{002}
```

Then, to set the canned search/replace pair and execute it in your XPL program:

```
«SV50,Rx-Merry»JM 2.setrciQ2 ;*; Set search
JM 2.rciQ2 ;*; Execute search
```

or

```
«SV50,Merry»JM 2.setrciQ2 ;*; Set search
JM 2.rciQ2 ;*; Execute search
```

or

```
JM 2.setrci:Rx-MerryQ2 ;*; Set search
JM 2.rciQ2 ;*; Execute search
```

Canned searches can also be set from the CMline. In CMline usage, the initial Rx- can be dropped from the search name. For example:

```
SETRCI Merry<Helpkey>; then
```

```
RCI<Helpkey>
```

Canned search frames for use with `RSE` or `REGEX` need not contain replacement strings. See the examples below.

5.1. Some Useful Canned Searches, for use with RSE or REGEX (Improve/Adjust to Taste)

```
{{5Rx-Email}} Email address (non-Unicode, non-quoted)
{002}«SV49,(?i)(?:[-\w!#$%&'*+/=?^_‘{\|}~][-\w.!#$%&'*+/=?
^_‘{\|}~]*)@(?:[\w.-]*)\.(?:[a-z.]{2,6})»{002}

{{5Rx-URL}} URL
{002}«SV49,(?i)(?:(?:https{0,1}|ftp|file|news|telnet|nttp)
:/{2,3}(?:[-\w:/%\\\.]+)+(?::\d+)?(?:/([\w/_\.]*(\?:/%\S+)
?)?)?)»{002}

{{5Rx-IPaddress}} Numeric IP address (quick and dirty)
{002}«SV49,(?:[0-2]{0,1}\d{0,1}\d\.){3}[0-2]{0,1}\d{0,1}\d»{002}

{{5Rx-Username}} Username 3-16 chars (alphnum, underscore & hyphen)
{002}«SV49,(?i)[-\w_]{3,16}»{002}

{{5Rx-Password}} Password 6-25 chars
{002}«SV49,(?i)[-\w.,;&$#_]{6,25}»{002}
```

6. Utilities

The RCI package includes the following search tools:

RCITE List all lines/paragraphs matching a string or regex (U2 frame)
RCI.BAT Run sequential canned searches against a single file
RCIBAT Run `RCI.BAT` against the current file (U2 frame)

6.1. Frame RCITE

U2 frame RCITE lists all lines/paragraphs in the current file containing a (case-*in*sensitive) string, which may be a regular expression.
Usage:

```
RCITE [string/regex]<Helpkey>
```

6.2. RCI.BAT

RCI.BAT runs up to 999 sequential preset RCI search-and-replace operations against a single subject file. Before running RCI.BAT, set the search-and-replace strings in files named SEARCH.1, SEARCH.2, . . . , all located in the directory that contains RCI.A3X. The format for each SEARCH.# file is the same as for SEARCH.TXT; see Section 2, above.
Usage:

```
RCI.BAT file_in file_out start_pos number_of_changes
```

The arguments, all optional, are the same as for RCI.EXE; see Subsection 3.1.1, above.

6.3. Frame RCIBAT

U2 frame RCIBAT runs RCI.BAT against the file in the current window, and displays the changes (without saving the file). Before running RCIBAT, prepare search-and-replace strings in files named SEARCH.1, SEARCH.2, . . . , as per the instructions in the preceding subsection.
Usage:

```
RCIBAT<Helpkey>
```

☞RCIBAT operates on the *entire file*, regardless of cursor position.

7. Adding Comments to Regular Expressions

Option (?x) allows working regex patterns to be formatted and commented. Starting at the point at which (?x) appears, white space outside of groups is ignored and everything from # to the end of the line is treated as a comment.

Here is a sample SEARCH.TXT that works with RCI:

```
(?x) # Change phone number format from (###) ###-#### to ###-###-####
(?U)             # Ungreedy (lazy) matching
\(               # open parens (escaped)
([2-9][0-8]\d)   # area code (\1)
\)               # close parens (escaped)
[-\s\.]??        # first separator
([2-9]\d{2})     # 3-digit exchange (\2)
[\s-\.]??        # second separator
(\d{4})          # 4-digit number (\3)
|---|
\1-\2-\3
```

Enjoy!

—CLD

Appendices

A. XyWrite 4 Wildcards and Their Regular Expression Equivalents

```
[A] = [A-Za-z0-9ÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÖÜáíóúñÑ]
[L] = [A-Za-zÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÖÜáíóúñÑ]
[N] = \d
[S] = \r\n|[\n\r\s\x00\x01\x02\x03\x08\x09\x1A !"()+,-./:;
      <=>?[\]^‘{|}\x7E\x7F£Ptƒ¿¡ AE AF\xC2\xC3\xB0\xB1\xB2\xB3
      \xB4\xB9\xBA\xBB\xBC\xBE\xBF\xC0\xC1\xC4\xC5\xC8\xC9
      \xCA\xCB\x5D]|[\xFE\xFF]..
[W] = .*
[X] = .
[.] = [!.?¿¡]
[10] = \x0A
[13] = \x0D
[CrLf] = \r\n
[<<] = «
[>>] = »
```

Negation [-] ^
The XyWrite wildcard [-] negates the next character: "walk[-]s" matches "walk" or "walked" but not "walks". Regex negation flag ^ can be used to negate a single character (walk^s matches "walk" but not "walks"), or a set of characters ("die[^st]" matches "die" but not "dies" or "diet").

Disjunction (Alternation) [O] |
The regex equivalent of Xy’s "boy[O]girl" is "boy|girl". The | operator is more flexible. For example, "b(o|uo)y|gir[dl]" matches "boy", or "buoy" or "gird" or "girl". XyWrite equivalent is "boy[O]buoy[O]gird[O]girl". The regex can also be written as

"boy|buoy|gir[dl]" or, even more succinctly, "bu??oy|gir[dl]".

Repetition

Numeric wildcards [0]...[9] are rendered with the number in curly braces after the character or group in question. Thus, [100X] = .{0,100}, meaning anywhere from 0 to 100 of any character. In regex, you can also specify a different lower limit. For example, .{5,10} means a minimum of 5 and a maximum of 10 of any character.

Searching for Whole Words (SE/W)

Regex token "\b" denotes a zero-length word boundary. The regex equivalent of XyWrite’s SE/W "walk" is "\bwalk\b". Of course, other regex special characters can be used with "\b". Thus, "\b[Ww]\D\S{2,}\b" means any word form (broadly defined) beginning with "W" or "w" and having at least 2 characters other than digits (\D = ^\d) or separators (\S = ^\s). "\b[A-Za-z]{1,}\b" means any alphabetic word form having one or more letters. And so forth.

Searching for Whole Lines

With option (?m) (multiline matching), ^ and $ match line beginnings and endings. For example, to find any line containing "regex":

```
(?m)^.*regex.*$
```

B. Perl-Compatible Regular Expressions: Quick Reference

(See also included file REGEX.PDF—excerpt from AutoIt documentation)

\	Escape a regex special character
\n	Newline (line feed)
\r	Carriage return
\r\n	CrLf
\t	Tab character
[...]	Character set
[a-z]	Character range
[^...]	Character set negation
.	Any character
\d, \D	Any digit, any non-digit
\w, \W	Any word char, any non-word char
\s, \S	Any whitespace char, any non-whitespace char
^	Start of string or line
$	End of string or line
\A	Start of string
\Z	End of string
\z	End string, no line breaks
\b	Word boundary (zero-length)
\B	Non-word boundary
\033	Octal char
\x1B	Hex char
\l	Lowercase next
\u	Uppercase next
\L	Lowercase until \E
\U	Uppercase until \E
\E	End quote/modification
\Q	Quote until \E
|	OR (disjunctiion)
?	Optional (greedy)
??	Optional (lazy)
*	Repeat 0+ (greedy)

*?	Repeat 0+ (lazy)
+	Repeat 1+ (greedy)
+?	Repeat 1+ (lazy)
{n}	Repeat n times
{n,m}	Repeat n to m (greedy)
{n,m}?	Repeat n to m (lazy)
{n,}	Repeat at least n (greedy)
{n,}?	Repeat at least n (lazy)
(...)	Capturing group
(?:...)	Non-capturing group
$1-$9	Group backreferences
(?>...)	Atomic group
?+, *+	Possessive quantifiers
(?=...)	Positive lookahead
(?!...)	Negative lookahead
(?<=..)	Positive lookbehind
(?<!..)	Negative lookbehind
\G	Continious Match
(?(?=.).|.)	If. then . else .
(?#...)	Comment
\033	Octal char
(?x)	Extended (ignore whitespace, comment from # to end of line)
\x1B	Hex char
\X	Unicode character
\uFFFF	Specific unicode char
\p{x}	Char with x property
\P{x}	Char without x property

Options:

- Are enclosed in (?) sequences;
- Can be grouped together: "(?imx)";
- Following a hyphen are negated: "(?im-sx)";
- Outside a group, affect the pattern from that point onwards;
- Inside a group, affect that group only.

Options lose their special meaning inside a character class [...], where they are treated literally.

Common options:

(?i) Caseless: matching becomes case-insensitive from the point where (?i) appears onward. By default, matching is case-sensitive. By default, casing applies to ASCII letters A-Z and a-z only. If Unicode Category Properties (*UCP) is enabled, casing applies to the entire Unicode plane 0.

(?m) Multiline: ^ and $ match the beginning and end of lines. By default, multiline is off. Multiline matching is enabled automatically in RCI when XyWrite wildcards appear in the search string.

(?s) Single-line or DotAll: . matches anything including a newline sequence. By default, DotAll is off; hence . does not match a newline sequence.

(?U) Ungreedy: quantifiers become "lazy" from the point where (?U) appears. By default, patterns are "greedy", meaning that quantifiers * + ? {...} will match the longest string that doesn't cause the rest of the pattern to fail. Non-greedy (lazy) patterns will match the smallest string that still allows the rest of the pattern to match. XyWrite SEarch wildcards are lazy.

(?x) eXtended: whitespaces outside character classes are ignored, and # starts a comment up to the next solid newline in pattern. Meaningless whitespaces between components make regular expressions much more readable. By default, whitespaces match themselves and # is a literal character.