

UL Meets Regular Expressions

SUG 06: Dave Evans
Monday 16:30



Regular expression, and operations made available in UL with Sirius Mods 6.9

- Also called *regex*; made popular in Perl
- Test that a string matches a regex
 - \$RegexMatch
- Extract or *capture* matching portions of a string
 - RegexCapture method of Stringlist class
- Replace matching portion(s) of one string with another
 - \$RegexReplace
 - RegexReplaceCorresponding method of Stringlist class



Regular expression operations (*continued*)

- Find matching Stringlist item(s)
 - RegexLocate, RegexLocateUp, and RegexSubset functions in Stringlist class



Implemented in Sirius Mods 6.9

- \$Regex... Longstring-capable \$functions
- Stringlist Regex... methods
- Perl functionality
- And more
- Some differences (restrictions - mostly small issues) relative to Perl
- Mastering Regular Expressions, Jeffrey E. F. Friedl, O'Reilly Media, Inc. (2nd edition, July 15, 2002)
- Note: complex regex may require larger LPDLST



Start with small example of \$RegexMatch

```
%str = 'Your humble speaker is Dave Evans'  
If $RegexMatch(%str, 'Dave +Evans') Then  
    Print 'You are getting sleepy...'  
End If
```

- '+' is a metacharacter - it means "one or more of the preceding"
 - In this case, the preceding is one blank character
- Non-metacharacters match "as is"
- Matching is done "within" the string - this is called **unanchored** matching
- \$RegexMatch is the "basic" operation: test if string is matched by the regex



Sirius Software,
Inc.

Many metasequences start with '\'

```
%str = 'Dear Dave Evans, you owe $14,000'
If $RegexMatch(%str, '^Dear +Dave Evans\C') Then
    Print 'You have mail'
End If
```

- '^' is a metacharacter which means "match only at start of string"
 - Makes the match *anchored*
 - As usual in M204, characters are EBCDIC; '^' is shift-6 on 3270 = X'5F'
- '\C' is a two-character metasequence - it means "any character other than a character which can be part of a name"
 - Name characters are '-', '_', ':', '.', 'A'-'Z', 'a'-'z', and '0'-'9'
 - Called a "character class"
 - Many predefined classes have positive (\c - "any legal name char") and negative (\C) forms, denoted by lower/upper case



Sirius Software,
Inc.

End anchor - '\$'

```
%str = 'Dave, You owe $14,000. Sincerely, Wolf '  
If $RegexMatch(%str, 'Wolf$') Then  
    Print 'Huffing and Puffing'  
End If
```

- Important general rule - to use any metacharacter to match "as is", escape with '\'
- If \$RegexMatch(%str, '\\$')) Then
 Print 'Money matters'
End If



Did we need the \$ to Huff & Puff...?

```
%str = 'Wolf, You owe $14,000. Sincerely, Bear'  
If $RegexMatch(%str, 'Wolf') Then  
    Print 'Huffing and Puffing Two'  
End If  
If $RegexMatch(%str, 'Wolf$') Then  
    Print 'Huffing and Puffing Three'  
End If
```

- What do you think happens?



Sirius Software,
Inc.

\$RegexReplace: replace matched substring(s)

```
%str = 'fuzzy wuzzy wuz a bear'  
%new = $RegexReplace(%str, 'z', 'x')  
Print %new  
Result:  
fuxzy wuxzy wuz a bear
```

- 2nd arg is regex
- 3rd arg is string to use in place of portion of input matched by regex
- To replace **all** matched substrings, use **G** option:
 - \$RegexReplace(%str, 'z', 'x', 'G') -> fuxxy wuxxy wux a bear



Sirius Software,
Inc.

Stringlist RegexCapture: extract substrings

```
%str = 'fuzzy wuzzy wuz a bear'
%sl Stringlist Object Auto New
%sl:RegexCapture( '(.*)wuz(.*)' )
%sl:Print
  Result:
  fuzzy wuzzy
  a bear
```

- '.' is a metacharacter meaning "any character except CR or LF"
 - Harkens to regex roots in line-oriented sed/grep; can change '.' to "match any character" by using S option ("DotAll mode")
- Haven't seen parens yet, but in general used to group sub-expression
 - For RegexCapture, string portion matched by 1st parens is 1st item appended to Stringlist'; portion matched by 2nd parens is 2nd item, etc.



Sirius Software,
Inc.

Capitalization: Thank you, M204-L!

```
%str = 'fuzzy wuzzy wuz a bear'  
%new = $RegexReplace(%str, ' (.)', -  
    ' $U1', 'G')  
Print %new  
Result:  
fuzzy Wuzzy Wuz A Bear
```

- \$1 in replacement string means "put portion matched by 1st paren here"
- \$U1 means "first translate portion matched by 1st paren"
- Little trickier to do leading character:
 - \$RegexReplace, %str, '(^|)(,)', '(\$1)(\$U2)', 'G')



Sirius Software,
Inc.

Rules of regex

- A regex can be a series of the following items:
 - normal character, which matches ("as is") exactly that character
 - A metasequence, which matches any one of a set of characters
 - ▶ Have seen "shortcuts", e.g., `\c` and `\C`
 - ▶ Create your own: ***character class***: `[class]`
 - ▶ Inside the brackets is list of single chars and/or ranges of chars, e.g., `[\-_.:A-Za-z0-9]` (that's `\c`)
 - ▶ Also can negate a char class by starting with `^`, e.g., `[^\-_.:A-Za-z0-9]` (that's `\C`)



Rules of regex (*continued*)

- Any valid regex can be enclosed in parens, for grouping
 - Any (?...) is non-capturing; simple form (?:exp)
- After a character or metasequence, you can place a quantifier, indicating the min and max repetitions of that match which are required
 - {min,max} or {min}
 - '*' = {0}
 - '+' = {1}
 - '?' = {0,1}



Rules of regex (*continued*)

- Finally, and with lowest precedence, you can place a '|' between two regexs, indicating that either one of the regexs can match
- ^ matches (0 chars at) start of string, \$ matches (0 chars at) end of string
- Lookahead matches 0 chars at a position:
 - (?=exp) if exp matches starting after the position
 - (?!exp) if exp does not match starting after the position



Rules of regex (*continued*)

- ? after quantifier for non-greedy matching
 - Implemented?
- Different escaping rules inside char class
 - TBD (differ from Perl?)
- Only metachars and "defined shortcuts" can be escaped, e.g., \@ is illegal
 - Different from Perl
- Match options: S: dotall; C: Xml Schema; I: case-insensitive match; M: ^ also matches before line boundary, \$ also matches after line boundary



More interesting \$RegexMatch?

```
* Validate store code: Uppercase letter, 1-3
* digits, hyphen, and 2 letter US state code:
%us = '(AK|AL|AZ|tbd|WY)'
%regex = '[A-Z]\d{1,3}-' With %us
If $RegexMatch(%code, %regex) Then
    CHANGE STORCD TO %code
Else
    Jump to BADSTORCD
End If
```

- Is this OK?



Sirius Software,
Inc.

More interesting \$RegexMatch? (*answer*)

- What if %cod = 'ju A951-AL nk'?
- Use %regex = '^ [A-Z] \d{1,3} -' With %us With '\$'



Replace multiple cases; switch 'x' and 'z':

```
%str = 'Drink some xxx and get some zzzs'
```

```
%str = $RegexReplace(%str, 'x', '!', 'G')
```

```
%str = $RegexReplace(%str, 'z', 'x', 'G')
```

```
Print $RegexReplace(%str, '!', 'z', 'G')
```

```
* ----- Or -----
```

```
%regex Object Stringlist Auto New
```

```
%rep Object Stringlist Auto New
```

```
%regex:Add('x')
```

```
%rep:Add('z')
```

```
%regex:Add('z')
```

```
%rep:Add('x')
```

```
Print %regex:RegexReplaceCorresponding(%str, %rep, 'G')
```



Fixup HTML to use LoadXml

- I could use HttpRequest to fetch my credit card data
- Then if I could parse it into an XmlDocument, I could checkoff my receipts once a week instead of once a year...
- But the HTML page is not quite a well-formed XML document
- ..RegexReplace.. to the rescue!

```
%resp = %req:Post
%str = %resp:Content(1)
..fix with $RegexReplace or RegexReplaceCorresponding
%doc Object XmlDocument
%doc = New
%doc:LoadXml(%str)
```



Sirius Software,
Inc.

HTML first problem:

- First try results in **XML doc parse error: expecting rest of EntityRef or CharRef after "&"**, pointing to
- So let's change to &
- The 'G' option says to replace all matches; there are several sequences to be replaced

```
...  
%str = %resp:Content(1)  
%s = $RegexReplace(%s, '&nbsp;', ' ', 'G')  
...
```



HTML next problem: no ETag

- Next result is XML doc parse error: expecting ETag to match STag (img), pointing to </td> after end of img tag (width="22">)
- Need to insert after end if
- Our first try: Match ''
 - (<img .*) parens to capture that part of the match
- Replace match with the part matched in parens, then /> to create empty ("self-ending") element tag

```

...
%str = %resp:Content(1)
...
%s = $RegexReplace(%s, '(<img .*>', '$1/>', 'G')
...

```



Still no end img: '.' doesn't match linefeed

- Getting same result; notice more of the last part of the img tag: `= "0"? height="90" width="22"></td>`
- This is because in the middle of the `` tag is a linefeed character (x'25', which displays in the message as "?"), and the `.` metacharacter doesn't really match any character - it matches any character except linefeed or carriage return
- Let's try the `S` (Dotall) option, which means `'.'` truly matches any character

```

...
%str = %resp:Content(1)
...
%s = $RegexReplace(%s, '(<img .*>', '$1/>', 'GS')
...

```



Sirius Software,
Inc.

Still missing img end: .* greedy

- Same result; adding a diagnostic to print %str after the (replacement, we see this at the very end of the string: </html/>
- That's because (with Dotall) the match succeeds in a greedy fashion, matching every character to the last '>' and inserting '/' before it.
- There is syntax for non-greedy matching, but another way, which also doesn't need Dotall, is a negated character class to match the content of the img tag, getting all characters which are not '>'

```
...
%str = %resp:Content(1)
```

```
...
%s = $RegexReplace(%s, '(<img [^>]*)>', '$1/>', 'G')
```

```
...
```



Sirius Software,
Inc.

Another missing ETag:

- Next is another XML doc parse error: expecting ETag to match STag (br)
- Need to change
 to
:

```
...  
%str = %resp:Content(1)  
...  
%s = $RegexReplace(%s, '<br>', '<br/>', 'G')  
...
```



Now have attribute with no value

- Next result is **XML doc parse error: expecting apostrophe (') or quote (") to start Attribute value**, pointing to `alt= border="0"`
- Need to insert "" here; notice that all of this is learning about the data, and the following is good enough for our purposes"

```
...  
%str = %resp:Content(1)  
...  
%s = $RegexReplace(%s, 'alt= b', 'alt="" b', 'G')  
...
```



Sirius Software,
Inc.

Another missing ETag: <head>

- Yet another XML doc parse error: expecting ETag to match STag (head)
- Looking at the document, we notice there's no </head> before <body>, so we'll add one
 - Unlike the others, notice this only happens once, so we don't need the G flag

```
...  
%str = %resp:Content(1)
```

```
...  
%s = $RegexReplace(%s, '<body ', '</head><body ')  
...
```



Sirius Software,
Inc.

Optional output argument for status

- All of the regex methods/\$functions will, by default, cancel the request if an invalid string is passed as a regex argument
- If you specify a %var as the output argument which reflects status, then instead of cancelling the request, this %var is set to a negative value indicating an invalid regex; to 0 if the regex was good but there is no match; to a positive number if there is a match. See specific method or \$function documentation.
- If you know your regex is good, probably you should use the default action (i.e., do not supply a status %var arg)

```
$RegexMatch(%s, %rx, , %stat)  
If %stat EQ 0 Then Print 'No match'  
ElseIf %stat LT 0 Then Print 'Bad regex'  
End If
```



Sirius Software,
Inc.