

```

1| #! /usr/bin/env raku
2|
3| # Get the Pod vs. Code structure of a Raku/Pod6 file.
4| # © 2023 Shimon Bollinger. All rights reserved.
5| # Last modified: Sun 10 Sep 2023 02:08:28 PM EDT
6| # Version 0.0.1
7|
8| # begin-no-weave
9| # always use the latest version of Raku
10| use v6.*;
11| use PrettyDump;
12| use Data::Dump::Tree;
13| #end-no-weave
14| =begin pod
15| =comment 1
16|
17|
18| =TITLE A grammar to parse a file into C<Pod> and C<Code> sections.
19|
20| =head1 INTRODUCTION
21|
22| I want to create a semi-literate Raku source file with the extension
23| C<.sl>. Then, I will I<weave> it to generate a readable file in formats like
24| Markdown, PDF, HTML, and more. Additionally, I will I<tangle> it to create source
25| code without any Pod6.
26|
27| =head2 Convenient tokens
28|
29| Let's create some tokens for convenience.
30|
31| =end pod
32| #TODO Put these into a Role
33|     my token hws          {    <!ww>\h*          } # Horizontal White Space
34|     my token leading-ws   {  ^^ <hws>           } # Whitespace at start of line
35|     my token optional-chars {    \N*?           }
36|     my token rest-of-line {    \N*    [\n | $] } #no-weave-this-line
37|     my token ws-till-EOL  {    <hws> [\n | $] } #no-weave-this-line
38|     my token blank-line   {  ^^ <ws-till-EOL>    } #no-weave-this-line
39| =begin pod
40| =comment 2
41| To do this, I need to divide the file into C<Pod> and C<Code> sections by parsing
42| it. For this purpose, I will create a dedicated Grammar.
43|
44|
45| =head1 The Grammar
46|
47| =end pod
48| #use Grammar::Tracer;
49| grammar Semi::Literate is export {
50| =begin pod
51|
52| Our file will exclusively consist of C<Pod> or C<Code> sections, and nothing
53| else. The C<Code> sections are of two types, a) code that is woven into the
54| documentation, and b) code that is not woven into the documentation. The
55| C<TOP> token clearly indicates this.
56|
57| =end pod
58|     token TOP {

```

```

59|         [
60|             || <pod>
61|             || <woven-code>
62|             || <non-woven-code>
63|         ]*
64|     } # end of token TOP
65| =begin pod
66| =comment 1
67|
68| =head2 The Pod6 delimiters
69|
70| According to the L<documentation|https://docs.raku.org/language/pod>,
71|
72| =begin defn
73|
74|     Every Pod6 document has to begin with =begin pod and end with =end pod.
75| =end defn
76| So let's define those tokens.
77| =head3 The C<begin-pod> token
78|
79| =end pod
80|     token begin-pod {
81|         <leading-ws>
82|         '=' begin <hws> pod
83|         <ws-till-EOL>
84|     } # end of token begin-pod
85| =begin pod
86| =comment 1
87|
88| =head3 The C<end-pod> token
89|
90| The C<end-pod> token is much simpler.
91|
92| =end pod
93|     token end-pod { <leading-ws> '=' end <hws> pod <ws-till-EOL> }
94| =begin pod
95| =comment 1
96|
97| =head3 Replacing Pod6 sections with blank lines
98|
99| Most programming applications do not focus on the structure of the executable
100| file, which is not meant to be easily read by humans. Our tangle would replace
101| all the Pod6 blocks with a single C<\n>. That can clump code together that is
102| easier read if there were one or more blank lines.
103|
104| However, we can provide the option for users to specify the number of empty
105| lines that should replace a C<pod> block. To do this, simply add a Pod6 comment
106| immediately after the C<=begin pod> statement. The comment can say anything
107| you like, but must end with a digit specifying the number of blank lines with
108| which to replace the Pod6 section.
109|
110| =end pod
111|     token num-blank-line-comment {
112|         <leading-ws>
113|         '=' comment
114|         <optional-chars>
115|         $<num-blank-lines> = (\d+)?
116|         <ws-till-EOL>

```

```

117|     } # end of token num-blank-line-comment
118| =begin pod
119| =comment 1
120|
121| =head2 The C<Pod> token
122|
123| Within the delimiters, all lines are considered documentation. We will refer to
124| these lines as C<plain-lines>. Additionally, it is possible to have nested
125| C<Pod> sections. This allows for a hierarchical organization of
126| documentation, allowing for more structured and detailed explanations.
127|
128| It is also permissible for the block to be empty. Therefore, we will use the
129| 'zero-or-more' quantifier on the lines of documentation, allowing for the
130| possibility of having no lines in the block.
131|
132| =end pod
133|     token pod {
134|         <begin-pod>
135|         <num-blank-line-comment>?
136|         [<pod> || <plain-line>]*
137|         <end-pod>
138|     } # end of token pod
139| =begin pod
140| =comment 1
141|
142| =head2 The C<Code> tokens
143|
144| The C<Code> sections are similarly easily defined. There are two types of
145| C<Code> sections, depending on whether they will appear in the woven code. See
146| L<below> for why some code would not be included in the woven
147| code.
148|
149| =head3 Woven sections
150|
151| These sections are trivially defined.
152| They are just one or more C<plain-line>s.
153|
154| =end pod
155|     token woven-code {
156|         [
157|             || <comment> { note $/.Str }
158|             || <plain-line>
159|         ]+
160|     } # end of token woven-code
161|
162|     regex comment {
163|         $<x>=(<leading-ws> \N*?) # optional code
164|         '#'                    # comment marker
165|         <-[#]>*                  # the actual comment
166|         <ws-till-EOL>
167|     } # end of my regex comment
168| =begin pod
169| =comment 1
170|
171| =head3 Non-woven sections
172|
173| Sometimes there will be code you do not want woven into the document, such
174| as boilerplate code like C<use v6.d>|. You have two options to mark such

```

```

175| code. By individual lines or by delimited blocks of code.
176| =end pod
177|     token non-woven-code {
178|         [
179|             || <one-line-no-weave>
180|             || <delimited-no-weave>
181|         ]+
182|     } # end of token non-woven
183| =begin pod
184| =comment 1
185|
186| =head4 One line of code
187|
188| Simply append C<# begin-no-weave> at the end of the line!
189|
190| =end pod
191|     token one-line-no-weave {
192|         ^^ \N*?
193|         '#' <hws> 'no-weave-this-line'
194|         <ws-till-EOL>
195|     } # end of token one-line-no-weave
196| =begin pod
197| =comment 1
198|
199|
200|
201| =head4 Delimited blocks of code
202|
203| Simply add comments C<# begin-no-weave> and C<#end-no-weave> before and after the
204| code you want ignored in the formatted document.
205|
206| =end pod
207|     token begin-no-weave {
208|         <leading-ws>                # optional leading whitespace
209|         '#' <hws> 'begin-no-weave' # the delimiter itself (# begin-no-weave)
210|         <ws-till-EOL>              # optional trailing whitespace or comment
211|     } # end of token <begin-no-weave>
212|
213|     token end-no-weave {
214|         <leading-ws>                # optional leading whitespace
215|         '#' <hws> 'end-no-weave'   # the delimiter itself (#end-no-weave)
216|         <ws-till-EOL>              # optional trailing whitespace or comment
217|     } # end of token <end--no-weave>
218|
219|     token delimited-no-weave {
220|         <begin-no-weave>
221|         <plain-line>*
222|         <end-no-weave>
223|     } # end of token delimited-no-weave
224|
225|     token code-comments {
226|         <leading-ws>
227|         '#' <rest-of-line>
228|         <![ / <begin-no-weave> | <end-no-weave> / ]>
229|     } # end of token code-comments
230| =begin pod
231| =comment 1
232|

```

```

233| =head3 The C<plain-line> token
234|
235| The C<plain-line> token is, really, any line at all...
236| ... except for one subtlety. They it can't be one of the begin/end delimiters.
237| We can specify that with a L<Regex Boolean Condition
238| Check|https://docs.raku.org/language/regexes\#Regex_Boolean_condition_check>.
239|
240|
241| =end pod
242|     token plain-line {
243|         :my $*EXCEPTION = False;
244|         [
245|             || <begin-pod>           { $*EXCEPTION = True }
246|             || <end-pod>             { $*EXCEPTION = True }
247|             || <begin-no-weave>      { $*EXCEPTION = True }
248|             || <end-no-weave>        { $*EXCEPTION = True }
249|             || <one-line-no-weave>   { $*EXCEPTION = True }
250|             || $<plain-line> = [^^ <rest-of-line>]
251|         ]
252|         <?{ !$*EXCEPTION }>
253|     } # end of token plain-line
254| =begin pod
255| =comment 1
256|
257| And that concludes the grammar for separating C<Pod> from C<Code>!
258|
259| =end pod
260| } # end of grammar Semi::Literate
261| =begin pod
262| =comment 2
263|
264| =head1 The Tangle subroutine
265|
266| This subroutine will remove all the Pod6 code from a semi-literate file
267| (C<.sl>) and keep only the Raku code.
268|
269|
270| =end pod
271| #TODO multi sub to accept Str & IO::PatGh
272| sub tangle (
273| =begin pod
274|
275| The subroutine has a single parameter, which is the input filename. The
276| filename is required. Typically, this parameter is obtained from the command
277| line or passed from the subroutine C<MAIN>.
278| =end pod
279|     Str $input-file!,
280| =begin pod
281|
282| The subroutine will return a C<Str>, which will be a working Raku program.
283| =end pod
284|     --> Str ) is export {
285| =begin pod
286| =comment 1
287|
288| First we will get the entire Semi-Literate C<.sl> file...
289| =end pod
290|     my Str $source = $input-file.IO.slurp;

```

```

291| =begin pod
292| =comment 1
293| =head2 Clean the source
294|
295| =head3 Remove unnecessary blank lines
296|
297| Very often the C<code> section of the Semi-Literate file will have blank lines
298| that you don't want to see in the tangled working code.
299| For example:
300|
301| =begin code :lang<raku>
302|
303|                                     # <== unwanted blank lines
304|                                     # <== unwanted blank lines
305|         sub foo () {
306|             { ... }
307|         } # end of sub foo ()
308|
309|                                     # <== unwanted blank lines
310|                                     # <== unwanted blank lines
311| =end code
312| =end pod
313| =begin pod
314| =comment 1
315|
316|
317| So we'll remove the blank lines immediately outside the beginning and end of
318| the Pod6 sections.
319| =end pod
320|     my Str $cleaned-source = $source;
321|     $cleaned-source ~~ s:g{\=end (\N*)\n+} = "\=end$0\n";
322|     $cleaned-source ~~ s:g{\n+\=begin (<hws> pod) [<hws> \d]?} = "\n\=begin$0";
323| =begin pod
324| =comment 1
325| =head2 The interesting stuff
326|
327| We parse it using the C<Semi::Literate> grammar
328| and obtain a list of submatches (that's what the C<caps> method does) ...
329| =end pod
330|     my Pair @submatches = Semi::Literate.parse($cleaned-source).caps;
331| =begin pod
332| =comment 1
333|
334| ...and iterate through the submatches and keep only the C<code> sections...
335| =end pod
336| #     note "submatches.elems: {@submatches.elems}";
337|     my Str $raku-code = @submatches.map( {
338| #         note .key;
339|         when .key eq 'woven-code'|'non-woven-code' {
340|             .value;
341|         }
342|     }
343| =begin pod
344| =comment 1
345| =head3 Replace Pod6 sections with blank lines
346| =end pod
347|         when .key eq 'pod' {
348|             my $num-blank-lines =

```

```

349|         .value.hash<num-blank-line-comment><num-blank-lines>;
350|         "\n" x $num-blank-lines with $num-blank-lines;
351|     }
352|
353|     # begin-no-weave
354|     default { die "Tangle: should never get here. .key == {.key}" }
355|     #end-no-weave
356| =begin pod
357| =comment 1
358|
359| ... and we will join all the code sections together...
360| =end pod
361|     } # end of my Str $raku-code = @submatches.map(
362|     ).join;
363| =begin pod
364| =comment 1
365| =head3 Remove the I<no-weave> delimiters
366|
367| =end pod
368|     $raku-code ~~ s:g{ <leading-ws> '#' <hws> 'begin-no-weave'      <rest-of-line> }
369|     = '';
370|     $raku-code ~~ s:g{ <leading-ws> '#' <hws> 'no-weave-this-line' <rest-of-line> }
371|     = "$0\n";
372|     $raku-code ~~ s:g{ <leading-ws> '#' <hws> 'end-no-weave'      <rest-of-line> }
373|     = '';
374| =begin pod
375| =comment 1
376| =head3 remove blank lines at the end
377|
378| =end pod
379|     $raku-code ~~ s{\n <blank-line>* $ } = '';
380| =begin pod
381| =comment 1
382|
383| And that's the end of the C<tangle> subroutine!
384| =end pod
385|     return $raku-code;
386| } # end of sub tangle (
387| =begin pod
388| =comment 2
389|
390| =head1 The Weave subroutine
391|
392| The C<Weave> subroutine will I<weave> the C<.sl> file into a readable Markdown,
393| HTML, or other format. It is a little more complicated than C<sub tangle>
394| because it has to include the C<code> sections.
395|
396| =end pod
397| sub weave (
398| =begin pod
399| =comment 1
400| =head2 The parameters of Weave
401|
402| C<sub weave> will have several parameters.
403| =head3 C<$input-file>
404|
405| The input filename is required. Typically,
406| this parameter is obtained from the command line through a wrapper subroutine

```

```

407| C<MAIN>.
408|
409| =end pod
410|     Str $input-file!;
411| =begin pod
412| =comment 1
413| =head3 C<$format>
414|
415| The output of the weave can (currently) be Markdown, Text, or HTML. It
416| defaults to Markdown. The variable is case-insensitive, so 'markdown' also
417| works.
418| =end pod
419|     Str :f(:$format) is copy = 'markdown';
420|     #= The output format for the woven file.
421| =begin pod
422| =comment 1
423| =head3 C<$line-numbers>
424|
425| It can be useful to print line numbers in the code listing. It currently
426| defaults to True.
427| =end pod
428|     Bool :l(:$line-numbers) = True;
429|     #= Should line numbers be added to the embeded code?
430| =begin pod
431| C<sub weave> returns a Str.
432| =end pod
433|     --> Str ) is export {
434|
435|     my UInt $line-number = 1;
436| =begin pod
437| First we will get the entire C<.sl> file...
438| =end pod
439|     my Str $source = $input-file.IO.slurp;
440| =begin pod
441| =comment 1
442| =head3 Remove blank lines at the begining and end of the code
443|
444| B<EXPLAIN THIS!>
445|
446| =end pod
447|     my Str $cleaned-source = $source;
448|     $cleaned-source ~~ s:g{\=end (\N*)\n+} = "\=end$0\n";
449|     $cleaned-source ~~ s:g{\n+\=begin (<hws> pod) [<hws> \d]?} = "\n\=begin$0";
450| =begin pod
451| =comment 1
452|
453| =head2 Interesting stuff
454|
455| ...Next, we parse it using the C<Semi::Literate> grammar
456| and obtain a list of submatches (that's what the C<caps> method does) ...
457| =end pod
458|     my Pair @submatches = Semi::Literate.parse($cleaned-source).caps;
459| =begin pod
460| =comment 1
461|
462| ...And now begins the interesting part. We iterate through the submatches and
463| insert the C<code> sections into the Pod6...
464| =end pod

```



```

465| #    note "weave submatches.elems: {@submatches.elems}";
466| #    note "submatches keys: {@submatches».keys}";
467| my Str $weave = @submatches.map( {
468|     when .key eq 'pod' {
469|         .value
470|     } # end of when .key
471|
472|     when .key eq 'woven-code' {qq:to/EOCB/; }
473|         \=begin pod
474|         \=begin code :lang<raku>
475|             { my $fmt = ($line-numbers ?? "%3s| " !! ' ') ~ "%s\n";
476|                 .value
477|                 .lines
478|                 .map($line-numbers
479|                     ?? {"%4s| %s\n".sprintf($line-number++, $_) }
480|                     !! {      "%s\n".sprintf(          $_) }
481|                 )
482|                 .chomp # get rid of the last \n
483|             }
484|         \=end code
485|         \=end pod
486|         EOCB
487|
488|     when .key eq 'non-woven-code' {
489|         ''; # do nothing
490|         #TODO don't insert a newline here.
491|     } # end of when .key eq 'non-woven-code'
492|
493|     # begin-no-weave
494|     default {
495|         die "Weave: should never get here. .key == {.key}" }
496|     # end-no-weave
497| } # end of my Str $weave = @submatches.map(
498| ).join;
499| =begin pod
500| =comment 1
501| =head3 remove blank lines at the end
502|
503| =end pod
504| $weave ~~ s{\n <blank-line>* $ } = '';
505| =begin pod
506| =comment 1
507|
508| And that's the end of the C<tangle> subroutine!
509| =end pod
510| return $weave
511| } # end of sub weave (
512| =begin pod
513| =comment 1
514| =head1 NAME
515|
516| C<Semi::Literate> - A semi-literate way to weave and tangle Raku/Pod6 source code.
517| =head1 VERSION
518|
519| This documentation refers to C<Semi-Literate> version 0.0.1
520|
521| =head1 SYNOPSIS
522|

```

```

523| =begin code :lang<raku>
524|
525| use Semi::Literate;
526| # Brief but working code example(s) here showing the most common usage(s)
527|
528| # This section will be as far as many users bother reading
529| # so make it as educational and exemplary as possible.
530|
531| =end code
532| =head1 DESCRIPTION
533|
534| C<Semi::Literate> is based on Daniel Sockwell's Pod::Literate module
535|
536| A full description of the module and its features.
537| May include numerous subsections (i.e. =head2, =head2, etc.)
538|
539| =head1 BUGS AND LIMITATIONS
540|
541| There are no known bugs in this module.
542| Patches are welcome.
543|
544| =head1 AUTHOR
545|
546| Shimon Bollinger (deoac.bollinger@gmail.com)
547|
548| =head1 LICENSE AND COPYRIGHT
549|
550| © 2023 Shimon Bollinger. All rights reserved.
551|
552| This module is free software; you can redistribute it and/or
553| modify it under the same terms as Raku itself.
554| See L<The Artistic License 2.0|https://opensource.org/licenses/Artistic-2.0>.
555|
556| This program is distributed in the hope that it will be useful,
557| but WITHOUT ANY WARRANTY; without even the implied warranty of
558| MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
559|
560| =end pod
561| # begin-no-weave
562| my %*SUB-MAIN-OPTS =
563|     :named-anywhere,          # allow named variables at any location
564|     :bundling,                # allow bundling of named arguments
565|     # :coerce-allomorphs-to(Str), # coerce allomorphic arguments to given type
566|     :allow-no,                # allow --no-foo as alternative to --/foo
567|     :numeric-suffix-as-value, # allow -j2 as alternative to --j=2
568| ;
569|
570| #| Run with option '--pod' to see all of the Pod6 objects
571| multi MAIN(Bool :$pod!) is hidden-from-USAGE {
572|     for $=pod -> $pod-item {
573|         for $pod-item.contents -> $pod-block {
574|             $pod-block.raku.say;
575|         }
576|     }
577| } # end of multi MAIN (:$pod)
578|
579| #| Run with option '--doc' to generate a document from the Pod6
580| #| It will be rendered in Text format

```

```
581| #| unless specified with the --format option.  e.g.
582| #|      --doc --format=HTML
583| multi MAIN(Bool :$doc!, Str :$format = 'Text') is hidden-from-USAGE {
584|     run $*EXECUTABLE, "--doc=$format", $*PROGRAM;
585| } # end of multi MAIN(Bool :$man!)
586|
587| my $semi-literate-file =
'/Users/jimbollinger/Documents/Development/raku/Projects/Semi-Literate/source/Literate.sl';
588| multi MAIN(Bool :$testt!) {
589|     say tangle($semi-literate-file);
590| } # end of multi MAIN(Bool :$test!)
591|
592| multi MAIN(Bool :$testw!) {
593|     say weave($semi-literate-file);
594| } # end of multi MAIN(Bool :$test!)
595|
596| #end-no-weave
```