

./ - lempate

- [a.lua](#)
- [lib/](#)
- [src/](#)
- [t/](#)

a.lua - lemlate

Functions defined

- [_M.process](#)
- [context_meta.include](#)
- [context_meta.plugin](#)
- [context_meta.process](#)
- [filter](#)
- [if type\(v\)](#)
- [if type\(value\)](#)
- [stash_get](#)
- [stash_set](#)
- [template_map\['tmpQQS k.tt2'\]](#)
- [tt2_not](#)
- [tt2_true](#)

Source code

```
1  --[[
2      This Lua code was generated by Lemplate, the Lua
3      Template Toolkit. Any changes made to this file will be lost the next
4      time the templates are compiled.
5
6      Copyright 2016 - Yichun Zhang (agentzh) - All rights reserved.
7
8      Copyright 2006-2014 - Ingy döt Net - All rights reserved.
9  ]]
10
11  local gsub = ngx.re.gsub
12  local concat = table.concat
13  local type = type
14  local math_floor = math.floor
15  local table_maxn = table.maxn
16
17  local _M = {
18      version = '0.01'
19  }
20
21  local template_map = {}
22
23  local function tt2_true(v)
24      return v and v ~= 0 and v ~= "" and v ~= '0'
25  end
26
27  local function tt2_not(v)
28      return not v or v == 0 or v == "" or v == '0'
29  end
30
31  local context_meta = {}
32
33  function context_meta.plugin(context, name, args)
34      if name == "iterator" then
35          local list = args[1]
36          local count = table_maxn(list)
37          return { list = list, count = 1, max = count - 1, index = 0, size = count, first = true, last = false,
prev = "" }
```

```

38     else
39         return error("unknown iterator: " .. name)
40     end
41 end
42
43 function context_meta.process(context, file)
44     local f = template_map[file]
45     if not f then
46         return error("file error - " .. file .. ": not found")
47     end
48     return f(context)
49 end
50
51 function context_meta.include(context, file)
52     local f = template_map[file]
53     if not f then
54         return error("file error - " .. file .. ": not found")
55     end
56     return f(context)
57 end
58
59 context_meta = { __index = context_meta }
60
61 local function stash_get(stash, k)
62     local v
63     if type(k) == "table" then
64         v = stash
65         for i = 1, #k, 2 do
66             local key = k[i]
67             local typ = k[i + 1]
68             if type(typ) == "table" then
69                 local value = v[key]
70                 if type(value) == "function" then
71                     return value()
72                 end
73                 if value then
74                     return value
75                 end
76                 if key == "size" then
77                     if type(v) == "table" then
78                         return #v
79                     else
80                         return 1
81                     end
82                 else
83                     return error("virtual method " .. key .. " not supported")
84                 end
85             end
86             if type(key) == "number" and key == math_floor(key) and key >= 0 then
87                 key = key + 1
88             end
89             if type(v) ~= "table" then
90                 return nil
91             end
92             v = v[key]
93         end
94     else
95         v = stash[k]
96     end
97     if type(v) == "function" then
98         return v()
99     end
100     return v
101 end
102
103 local function stash_set(stash, k, v, default)
104     if default then
105         local old = stash[k]
106         if old == nil then
107             stash[k] = v
108         end
109     else
110         stash[k] = v
111     end
112 end
113

```

```

114 function _M.process(file, params)
115     local stash = params
116     local context = {
117         stash = stash,
118         filter = function (bits, name, params)
119             local s = concat(bits)
120             if name == "html" then
121                 s = gsub(s, "&", '&amp;', "jo")
122                 s = gsub(s, "<", '&lt;', "jo");
123                 s = gsub(s, ">", '&gt;', "jo");
124                 s = gsub(s, "'", '&quot;', "jo"); -- " end quote for emacs
125                 return s
126             end
127         end
128     }
129     context = setmetatable(context, context_meta)
130     local f = template_map[file]
131     if not f then
132         return error("file error - " .. file .. ": not found")
133     end
134     return f(context)
135 end
136 -- tmpQQS_k.tt2
137 template_map['tmpQQS_k.tt2'] = function (context)
138     if not context then
139         return error("Lemplate function called without context\n")
140     end
141     local stash = context.stash
142     local output = {}
143     local i = 0
144
145     i = i + 1 output[i] = 'Hello, '
146     -- line 1 "tmpQQS_k.tt2"
147     i = i + 1 output[i] = stash_get(stash, 'world')
148     i = i + 1 output[i] = '!\n'
149
150     return output
151 end
152
153 return _M

```

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

lib/ - lempate

- [Lemplate/](#)
- [Lemplate.pm](#)

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

lib/Lemplate/ - lempate

- [Directive.pm](#)
- [Parser.pm](#)
- [Runtime/](#)
- [Runtime.pm](#)

[One Level Up](#)

[Top Level](#)

lib/Lemplate/Directive.pm - leemplate

Data types defined

- [_attempt_range_expand_val](#)
- [args](#)
- [assign](#)
- [block](#)
- [break](#)
- [call](#)
- [capture](#)
- [clear](#)
- [default](#)
- [filenames](#)
- [filter](#)
- [foreach](#)
- [get](#)
- [ident](#)
- [if](#)
- [include](#)
- [javascript](#)
- [macro](#)
- [multi_wrapper](#)
- [new](#)
- [next](#)
- [no_javascript](#)
- [process](#)
- [quoted](#)
- [raw](#)
- [return](#)
- [set](#)
- [stop](#)
- [switch](#)
- [template](#)

- [text](#)
- [textblock](#)
- [throw](#)
- [use](#)
- [while](#)
- [wrapper](#)

Source code

```

1 package Lemplate::Directive;
2 use strict;
3 use warnings;
4
5 # VERSION
6
7 our $OUTPUT = 'i = i + 1 output[i] =';
8 our $WHILE_MAX = 1000;
9
10 # parser state variable
11 # only true when inside JAVASCRIPT blocks
12 our $INJAVASCRIPT = 0;
13
14 sub new {
15     my $class = shift;
16
17     return bless {}, $class
18 }
19
20 sub template {
21     my ($class, $block) = @_;
22
23     return "function() return '' end" unless $block =~ /\S/;
24
25     return <<"...";
26 function (context)
27     if not context then
28         return error("Lemplate function called without context\\n")
29     end
30     local stash = context.stash
31     local output = {}
32     local i = 0
33
34     $block
35
36     return output
37 end
38 ...
39 }
40
41 # Try to do 1 .. 10 expansions
42 sub _attempt_range_expand_val ($) {
43     my $val = shift;
44     return $val unless
45         my ( $from, $to ) = $val =~ m/\s*[\s*(\S+)\s*\.\.\s*(\S+)\s*]/;
46
47     die "Range expansion is current supported for positive/negative integer values only (e.g. [ 1 .. 10
48 ])\nCannot expand: $val" unless $from =~ m/^-?\d+$/ && $to =~ m/^-?\d+$/;
49
50     return join '', '[', join( ',', $from .. $to ), ']';
51 }
52
53 #-----
54 # textblock\(\$text\)
55 #-----
56
57 sub textblock {
58     my ($class, $text) = @_;
59     return $text if $INJAVASCRIPT;

```

```

59     return "$OUTPUT " . $class->text($text);
60 }
61
62 #-----
63 # text($text)
64 #-----
65
66 sub text {
67     my ($class, $text) = @_;
68     for ($text) {
69         s/(['\\])/\\$1/g;
70         s/\\n/\\n/g;
71         s/\\r/\\r/g;
72     }
73     return "" . $text . "";
74 }
75
76 #-----
77 # ident(\@ident)                                foo.bar(baz)
78 #-----
79
80 sub ident {
81     my ($class, $ident) = @_;
82     return "" unless @$ident;
83     my $ns;
84
85     # does the first element of the identifier have a NAMESPACE
86     # handler defined?
87     if (ref $class && @$ident > 2 && ($ns = $class->{ NAMESPACE })) {
88         my $key = $ident->[0];
89         $key =~ s/^(.+)$/1/s;
90         if ($ns = $ns->{ $key }) {
91             return $ns->ident($ident);
92         }
93     }
94
95     if (scalar @$ident <= 2 && ! $ident->[1]) {
96         $ident = $ident->[0];
97     }
98     else {
99         $ident = '{' . join(' ', @$ident) . ' ';
100     }
101     return "stash_get(stash, $ident)";
102 }
103
104
105 #-----
106 # assign(\@ident, $value, $default)                foo = bar
107 #-----
108
109 sub assign {
110     my ($class, $var, $val, $default) = @_;
111
112     if (ref $var) {
113         if (scalar @$var == 2 && ! $var->[1]) {
114             $var = $var->[0];
115         }
116         else {
117             $var = '{' . join(' ', @$var) . ' ';
118         }
119     }
120     $val = attempt_range_expand_val $val;
121     $val .= ', 1' if $default;
122     return "stash_set(stash, $var, $val)";
123 }
124
125
126 #-----
127 # args(\@args)                                    foo, bar, baz = qux
128 #-----
129
130 sub args {
131     my ($class, $args) = @_;
132     my $hash = shift @$args;
133     push(@$args, '{ ' . join(' ', @$hash) . ' }')
134         if @$hash;

```

```

135         return '{}' unless @$args;
136         return ' ' . join(' ', @$args) . ' ';
137     }
138 }
139
140
141 #-----
142 # filenames(\@names)
143 #-----
144
145 sub filenames {
146     my ($class, $names) = @_;
147     if (@$names > 1) {
148         $names = ' ' . join(' ', @$names) . ' ';
149     }
150     else {
151         $names = shift @$names;
152     }
153     return $names;
154 }
155
156
157 #-----
158 # get($expr)                                     [% foo %]
159 #-----
160
161 sub get {
162     my ($class, $expr) = @_;
163     return "$OUTPUT $expr";
164 }
165
166 sub block {
167     my ($class, $block) = @_;
168     return join "\n", map {
169         s/^(?=line \d+)/-- /gm;
170         $_;
171     } @{$block} || [];
172 }
173
174 #-----
175 # call($expr)                                     [% CALL bar %]
176 #-----
177
178 sub call {
179     my ($class, $expr) = @_;
180     $expr .= ' ';
181     return $expr;
182 }
183
184
185 #-----
186 # set(\@setlist)                                 [% foo = bar, baz = qux %]
187 #-----
188
189 sub set {
190     my ($class, $setlist) = @_;
191     my $output;
192     while (my ($var, $val) = splice(@$setlist, 0, 2)) {
193         $output .= $class->assign($var, $val) . ";\n";
194     }
195     chomp $output;
196     return $output;
197 }
198
199
200 #-----
201 # default(\@setlist)                             [% DEFAULT foo = bar, baz = qux %]
202 #-----
203
204 sub default {
205     my ($class, $setlist) = @_;
206     my $output;
207     while (my ($var, $val) = splice(@$setlist, 0, 2)) {
208         $output .= &assign($class, $var, $val, 1) . ";\n";
209     }
210     chomp $output;

```

```

211     return $output;
212 }
213
214
215 #-----
216 # include(\@nameargs)                [% INCLUDE template foo = bar %]
217 #           # => [ [ $file, ... ], \@args ]
218 #-----
219
220 sub include {
221     my ($class, $nameargs) = @_;
222     my ($file, $args) = @$nameargs;
223     my $hash = shift @$args;
224     $file = $class->filenames($file);
225     (my $raw_file = $file) =~ s/^\|'$/g;
226     $Lemplate::ExtraTemplates{$raw_file} = 1;
227     my $file2 = "$Lemplate::TemplateName/$raw_file";
228     my $str_args = (@$hash ? ', { ' . join(', ', @$hash) . ' }' : '');
229     return "$OUTPUT context.include(context, template_map['$Lemplate::TemplateName/$raw_file'] and $file2 or
$files$str_args)";
230 }
231
232
233 #-----
234 # process(\@nameargs)                [% PROCESS template foo = bar %]
235 #           # => [ [ $file, ... ], \@args ]
236 #-----
237
238 sub process {
239     my ($class, $nameargs) = @_;
240     my ($file, $args) = @$nameargs;
241     my $hash = shift @$args;
242     $file = $class->filenames($file);
243     (my $raw_file = $file) =~ s/^\|'$/g;
244     $Lemplate::ExtraTemplates{$raw_file} = 1;
245     $file .= @$hash ? ', { ' . join(', ', @$hash) . ' }' : '';
246     return "$OUTPUT context.process(context, $file)";
247 }
248
249
250 #-----
251 # if($expr, $block, $else)                [% IF foo < bar %]
252 #                                     ...
253 #                                     [% ELSE %]
254 #                                     ...
255 #                                     [% END %]
256 #-----
257
258 sub if {
259     my ($class, $expr, $block, $else) = @_;
260     my @else = $else ? @$else : ();
261     $else = pop @else;
262
263     my $output = "if tt2_true($expr) then\n$block\n";
264
265     foreach my $elsif (@else) {
266         ($expr, $block) = @$elsif;
267         $output .= "elseif tt2_true($expr) then\n$block\n";
268     }
269     if (defined $else) {
270         $output .= "else\n$else\nend\n";
271     } else {
272         $output .= "end\n";
273     }
274
275     return $output;
276 }
277
278 #-----
279 # foreach($target, $list, $args, $block)    [% FOREACH x = [ foo bar ] %]
280 #                                     ...
281 #                                     [% END %]
282 #-----
283
284 sub foreach {
285     my ($class, $target, $list, $args, $block) = @_;

```

```

286 $args = shift @$args;
287 $args = @$args ? ', { ' . join(', ', @$args) . ' }' : '';
288
289 my ($loop_save, $loop_set, $loop_restore, $setiter);
290 if ($target) {
291     $loop_save =
292         'local oldloop = ' . $class->ident(['loop']);
293     $loop_set = "stash['$target'] = value";
294     $loop_restore = "stash_set(stash, 'loop', oldloop)";
295 }
296 else {
297     die "XXX - Not supported yet";
298     $loop_save = 'stash = context.localise()';
299     $loop_set =
300         "stash.get(['import', [value]]) if typeof(value) == 'object'";
301     $loop_restore = 'stash = context.delocalise()';
302 }
303
304 $list = _attempt_range_expand_val $list;
305
306 return <<EOF;
307
-- FOREACH
308 do
309     local list = $list
310     local iterator
311     if list.list then
312         iterator = list
313         list = list.list
314     end
315     $loop_save
316     local count
317     if not iterator then
318         count = table_maxn(list)
319         iterator = { count = 1, max = count - 1, index = 0, size = count, first = true, last = false, prev =
320 "" }
321     else
322         count = iterator.size
323     end
324     stash.loop = iterator
325     for idx, value in ipairs(list) do
326         if idx == count then
327             iterator.last = true
328         end
329         iterator.index = idx - 1
330         iterator.count = idx
331         iterator.next = list[idx + 1]
332         $loop_set
333     $block
334         iterator.first = false
335         iterator.prev = value
336     end
337     $loop_restore
338 end
339 EOF
340 }
341
342
343 #-----
344 # next() [% NEXT %]
345 #
346 # Next iteration of a FOREACH loop (experimental)
347 #-----
348
349 sub next {
350     return <<EOF;
351     return error("NEXT not implemented yet")
352 EOF
353 }
354
355 #-----
356 # wrapper(\@nameargs, $block) [% WRAPPER template foo = bar %]
357 # # => [ [$file,...], \@args ]
358 #-----
359 sub wrapper {
360     my ($class, $nameargs, $block) = @_;
```

```

361     my ($file, $args) = @$nameargs;
362     my $hash = shift @$args;
363
364     s/ => /: / for @$hash;
365     return $class->multi_wrapper($file, $hash, $block)
366         if @$file > 1;
367     $file = shift @$file;
368     push(@$hash, "'content': output");
369     $file .= @$hash ? ', { ' . join(', ', @$hash) . ' }' : '';
370
371     return <<EOF;
372
373 // WRAPPER
374 $OUTPUT (function() {
375     var output = '';
376     $block;
377     return context.include($file);
378 })();
379 EOF
380 }
381
382 sub multi_wrapper {
383     my ($class, $file, $hash, $block) = @_;
384
385     push(@$hash, "'content': output");
386     $hash = @$hash ? ', { ' . join(', ', @$hash) . ' }' : '';
387
388     $file = join(', ', reverse @$file);
389     # print STDERR "multi_wrapper: $file\n";
390
391     return <<EOF;
392
393 // WRAPPER
394 $OUTPUT (function() {
395     var output = '';
396     $block;
397     var files = new Array($file);
398     for (var i = 0; i < files.length; i++) {
399         output = context.include(files[i]$hash);
400     }
401     return output;
402 })();
403 EOF
404 }
405
406
407 #-----
408 # while($expr, $block)                                [% WHILE x < 10 %]
409 #                                                         ...
410 #                                                         [% END %]
411 #-----
412
413 sub while {
414     my ($class, $expr, $block) = @_;
415
416     return <<EOF;
417
418 -- WHILE
419 do
420     local failsafe = $WHILE_MAX;
421     while $expr do
422         failsafe = failsafe - 1
423         if failsafe <= 0 then
424             break
425         end
426     $block
427     end
428     if not failsafe then
429         return error("WHILE loop terminated (> $WHILE_MAX iterations)\n")
430     end
431 end
432 EOF
433 }
434
435 #-----
436 # javascript($script)                                [% JAVASCRIPT %]

```

```

437 #                                     ...
438 #                                     [% END %]
439 #-----
440 sub javascript {
441     my ( $class, $javascript ) = @_;
442     return $javascript;
443 }
444
445 sub no_javascript {
446     my ( $class ) = @_;
447     die "EVAL_JAVASCRIPT has not been enabled, cannot process [% JAVASCRIPT %] blocks";
448 }
449
450 #-----
451 # switch($expr, \@case)                                     [% SWITCH %]
452 #                                                         [% CASE foo %]
453 #                                                         ...
454 #                                                         [% END %]
455 #-----
456
457 sub switch {
458     my ( $class, $expr, $case ) = @_;
459     my @case = @$case;
460     my ( $match, $block, $default );
461     my $caseblock = '';
462
463     $default = pop @case;
464
465     foreach $case (@case) {
466         $match = $case->[0];
467         $block = $case->[1];
468         # $block = pad($block, 1) if $PRETTY;
469         $caseblock .= <<EOF;
470     case $match:
471     $block
472     break;
473
474 EOF
475     }
476
477     if (defined $default) {
478         $caseblock .= <<EOF;
479     default:
480     $default
481     break;
482 EOF
483     }
484     # $caseblock = pad($caseblock, 2) if $PRETTY;
485
486     return <<EOF;
487
488     switch($expr) {
489     $caseblock
490     }
491 EOF
492 }
493
494
495 #-----
496 # throw(\@nameargs)                                     [% THROW foo "bar error" %]
497 # # => [ [$type], \@args ]
498 #-----
499
500
501 sub throw {
502     my ( $class, $nameargs ) = @_;
503     my ( $type, $args ) = @$nameargs;
504     my $hash = shift(@$args);
505     my $info = shift(@$args);
506     $type = shift @$type;
507
508     return qq{return error($type, $info)};
509 }
510
511
512 #-----

```



```

589
590 sub filter {
591     my ($class, $lnameargs, $block) = @_;
592     my ($name, $args, $alias) = @$lnameargs;
593     $name = shift @$name;
594     $args = &args($class, $args);
595     $args = $args ? "$args, $alias" : ", null, $alias"
596         if $alias;
597     $name .= ", $args" if $args;
598     return <<EOF;
599
600 -- FILTER
601 local value
602 do
603     local output = {}
604     local i = 0
605
606     $block
607
608     value = context.filter(output, $name)
609 end
610 $OUTPUT value
611 EOF
612 }
613
614 sub quoted {
615     my $class = shift;
616     if ( @_ && ref($_[0]) ) {
617         return join( " .. ", @{$_[0]} );
618     }
619     return "return error('QUOTED called with unknown arguments in Lemplate')";
620 }
621
622 #-----
623 # macro($name, $block, \@args)
624 #-----
625
626 sub macro {
627     my ($class, $ident, $block, $args) = @_;
628
629     if ($args) {
630         $args = join(';', map { "args['$_'] = fargs.shift()" } @$args);
631
632         return <<EOF;
633
634 //MACRO
635 stash.set('$ident', function () {
636     var output = '';
637     var args = {};
638     var fargs = Array.prototype.slice.call(arguments);
639     $args;
640     args.arguments = Array.prototype.slice.call(arguments);
641
642     var params = fargs.shift() || {};
643
644     for (var key in params) {
645         args[key] = params[key];
646     }
647
648     context.stash.clone(args);
649     try {
650 $block
651     }
652     catch(e) {
653         var error = context.set_error(e, output);
654         throw(error);
655     }
656
657     context.stash.declone();
658     return output;
659 });
660
661 EOF
662
663 }
664 else {

```

```

665         return <<EOF;
666
667 //MACRO
668
669 stash.set('$_ident', function () {
670     var output = '';
671     var args = {};
672
673     var fargs = Array.prototype.slice.call(arguments);
674     args.arguments = Array.prototype.slice.call(arguments);
675
676     if (typeof arguments[0] == 'object') args = arguments[0];
677
678     context.stash.clone(args);
679     try {
680 $block
681     }
682     catch(e) {
683         var error = context.set_error(e, output);
684         throw(error);
685     }
686
687     context.stash.declone();
688     return output;});
689
690 EOF
691 }
692 }
693
694 sub capture {
695     my ($class, $name, $block) = @_ ;
696
697     if (ref $name) {
698         if (scalar @$name == 2 && ! $name->[1]) {
699             $name = $name->[0];
700         }
701         else {
702             $name = '[' . join(' ', @$name) . ']';
703         }
704     }
705
706     return <<EOF;
707
708 // CAPTURE
709 (function() {
710     var output = '';
711     $block
712     stash.set($name, output);
713 })();
714 EOF
715
716 }
717
718 BEGIN {
719     return; # Comment out this line to get callback traces
720     no strict 'refs';
721     my $pkg = __PACKAGE__ . '::';
722     my $stash = \ %$pkg;
723     use strict 'refs';
724     for my $name (keys %$stash) {
725         my $glob = $stash->{$name};
726         if (*$glob{CODE}) {
727             my $code = *$glob{CODE};
728             no warnings 'redefine';
729             $stash->{$name} = sub {
730                 warn "Calling $name(@_)\n";
731                 &$code(@_);
732             };
733         }
734     }
735 }
736
737
738 1;
739
740 __END__

```

```
741
742 =encoding UTF-8
743
744 =head1 NAME
745
746 Lemplate::Directive - Lemplate Code Generating Backend
747
748 =head1 SYNOPSIS
749
750     use Lemplate::Directive;
751
752 =head1 DESCRIPTION
753
754 Lemplate::Directive is the analog to Template::Directive, which is the
755 module that produces that actual code that templates turn into. The
756 Lemplate version obviously produces Lua code rather than Perl.
757 Other than that the two modules are almost exactly the same.
758
759 =head1 BUGS
760
761 Unfortunately, some of the code generation seems to happen before
762 Lemplate::Directive gets control. So it currently has heuristical code
763 to rejigger Perl code snippets into Lua. This processing needs to
764 happen upstream once I get more clarity on how Template::Toolkit works.
765
766 =head1 AUTHOR
767
768 Ingy döt Net <ingy@cpan.org>
769
770 =head1 COPYRIGHT
771
772 Copyright (c) 2016. Yichun Zhang (agentzh). All rights reserved.
773
774 Copyright (c) 2006-2014. Ingy döt Net. All rights reserved.
775
776 This program is free software; you can redistribute it and/or modify it
777 under the same terms as Perl itself.
778
779 See L<http://www.perl.com/perl/misc/Artistic.html>
780
781 =cut
```

[One Level Up](#)

[Top Level](#)

lib/Lemplate/Parser.pm - leemplate

Data types defined

- [new](#)

Source code

```
1 package Lemplate::Parser;
2 use strict;
3 use warnings;
4 use base 'Template::Parser';
5
6 # VERSION
7
8 use Lemplate::Grammar;
9 use Lemplate::Directive;
10
11 sub new {
12     my $class = shift;
13     my $parser = $class->SUPER::new(
14         GRAMMAR => Lemplate::Grammar->new(),
15         FACTORY => 'Lemplate::Directive',
16         @_,
17     );
18
19     # flags passed from Lemplate object
20     my %args = @_;
21
22     # eval-javascript is default "on"
23     $parser->{EVAL_JAVASCRIPT} = exists $args{EVAL_JAVASCRIPT}
24         ? $args{EVAL_JAVASCRIPT} : 1;
25
26     # tie the parser state-variable to the global Directive var
27     $parser->{INJAVASCRIPT} = \$Lemplate::Directive::INJAVASCRIPT;
28
29     return $parser;
30 }
31
32 1;
33
34 __END__
35
36 =encoding UTF-8
37
38 =head1 NAME
39
40 Lemplate::Parser - Lemplate Parser Subclass
41
42 =head1 SYNOPSIS
43
44     use Lemplate::Parser;
45
46 =head1 DESCRIPTION
47
48 Lemplate::Parser is a simple subclass of Template::Parser. Not much
49 to see here.
50
51 =head1 AUTHOR
52
53 Ingy döt Net <ingy@cpan.org>
54
55 =head1 COPYRIGHT
56
57 Copyright (c) 2006-2014. Ingy döt Net. All rights reserved.
58
59 This program is free software; you can redistribute it and/or modify it
60 under the same terms as Perl itself.
61
62 See L<http://www.perl.com/perl/misc/Artistic.html>
```

63
64 =cut

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

lib/Lemplate/Runtime/ - lemlate

- [Compact.pm](#)

[One Level Up](#)

[Top Level](#)

lib/Lemplate/Runtime/Compact.pm - leemplate

Data types defined

- [ajax_jquery](#)
- [ajax_xhr](#)
- [ajax_yui](#)
- [json2](#)
- [json_json2](#)
- [json_json2_internal](#)
- [json_yui](#)
- [kernel](#)
- [main](#)
- [xhr_gregory](#)
- [xhr_ilinsky](#)
- [xxx](#)

Source code

```
1 package Lemplate::Runtime::Compact;
2 use strict;
3 use warnings;
4
5 # VERSION
6
7 sub main { return &kernel }
8 sub kernel {
9     <<'...';
10     ...
11 }
12
13 sub ajax_jquery {
14     <<'...';
15     ...
16 }
17
18 sub ajax_xhr {
19     <<'...';
20     ...
21 }
22
23 sub ajax_yui {
24     <<'...';
25     ...
26 }
27
28 sub json_json2 {
29     <<'...';
30     ...
31 }
32
33 sub json_json2_internal {
34     <<'...';
35     ;(function(){
36
37     var JSON;
38
```

```

39
40
41 }();
42 ...
43 }
44
45 sub json_yui {
46     <<'...';
47     ...
48 }
49
50 sub json2 {
51     <<'...';
52     ...
53 }
54
55 sub xhr_gregory {
56     <<'...';
57     ...
58 }
59
60 sub xhr_ilinsky {
61     <<'...';
62     ...
63 }
64
65 sub xxx {
66     <<'...';
67     ...
68 }
69
70 1;
71
72 __END__
73
74 =encoding UTF-8
75
76 =head1 NAME
77
78 Lemplate::Runtime - Perl Module containing the Lemplate JavaScript Runtime
79
80 =head1 SYNOPSIS
81
82     use Lemplate::Runtime;
83     print Lemplate::Runtime->main;
84
85 =head1 DESCRIPTION
86
87 This module is auto-generated and used internally by Lemplate. It
88 contains subroutines that simply return various parts of the Lemplate
89 JavaScript Runtime code.
90
91 =head1 METHODS
92
93 head2 kernel
94
95 head2 ajax\_jquery
96
97 head2 ajax\_xhr
98
99 head2 ajax\_yui
100
101 head2 json\_json2
102
103 head2 json\_yui
104
105 head2 json2
106
107 head2 xhr\_gregory
108
109 head2 xhr\_ilinsky
110
111 head2 xxx
112
113 =head1 COPYRIGHT
114

```

115 *Copyright (c) 2014. Ingy döt Net.*
116
117 *This program is free software; you can redistribute it and/or modify it*
118 *under the same terms as Perl itself.*
119
120 See L<<http://www.perl.com/perl/misc/Artistic.html>>
121
122 =cut

[One Level Up](#)

[Top Level](#)

src/lib/Lemplate/Runtime/Compact.pm - leemplate

Data types defined

- [ajax_jquery](#)
- [ajax_xhr](#)
- [ajax_yui](#)
- [json2](#)
- [json_json2](#)
- [json_json2_internal](#)
- [json_yui](#)
- [kernel](#)
- [main](#)
- [xhr_gregory](#)
- [xhr_ilinsky](#)
- [xxx](#)

Source code

```
1 package Lemplate::Runtime::Compact;
2 use strict;
3 use warnings;
4
5 sub main { return &kernel }
6 sub kernel {
7     <<'...';
8     ...
9 }
10
11 sub ajax_jquery {
12     <<'...';
13     ...
14 }
15
16 sub ajax_xhr {
17     <<'...';
18     ...
19 }
20
21 sub ajax_yui {
22     <<'...';
23     ...
24 }
25
26 sub json_json2 {
27     <<'...';
28     ...
29 }
30
31 sub json_json2_internal {
32     <<'...';
33     ;(function(){
34
35     var JSON;
36
37
38
```

```

39 }();
40 ...
41 }
42
43 sub json_yui {
44     <<'...';
45     ...
46 }
47
48 sub json2 {
49     <<'...';
50     ...
51 }
52
53 sub xhr_gregory {
54     <<'...';
55     ...
56 }
57
58 sub xhr_ilinsky {
59     <<'...';
60     ...
61 }
62
63 sub xxx {
64     <<'...';
65     ...
66 }
67
68 1;
69
70 __END__
71
72 =encoding UTF-8
73
74 =head1 NAME
75
76 Lemplate::Runtime - Perl Module containing the Lemplate JavaScript Runtime
77
78 =head1 SYNOPSIS
79
80     use Lemplate::Runtime;
81     print Lemplate::Runtime->main;
82
83 =head1 DESCRIPTION
84
85 This module is auto-generated and used internally by Lemplate. It
86 contains subroutines that simply return various parts of the Lemplate
87 JavaScript Runtime code.
88
89 =head1 METHODS
90
91 head2 kernel
92
93 head2 ajax\_jquery
94
95 head2 ajax\_xhr
96
97 head2 ajax\_yui
98
99 head2 json\_json2
100
101 head2 json\_yui
102
103 head2 json2
104
105 head2 xhr\_gregory
106
107 head2 xhr\_ilinsky
108
109 head2 xxx
110
111 =head1 COPYRIGHT
112
113 Copyright (c) 2014. Ingy döt Net.
114

```

115 *This program is free software; you can redistribute it and/or modify it*
116 *under the same terms as Perl itself.*

117
118 See [L<http://www.perl.com/perl/misc/Artistic.html>](http://www.perl.com/perl/misc/Artistic.html)

119
120 =cut

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

src/lib/Lemplate/Runtime/ - lempate

- [Compact.pm](#)

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

src/lib/Lemplate/ - lempate

- [Runtime/](#)
- [Runtime.pm](#)

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

src/lib/ - lempate

- [Lemplate/](#)

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

src/ - lemlate

- [Makefile](#)
- [bin/](#)
- [lib/](#)
- [parser/](#)

[One Level Up](#)

[Top Level](#)

src/Makefile - lemlate

```
1 .PHONY: fetch runtime
2
3 .DELETE_ON_ERROR: $(RUNTIME_MODULE) $(RUNTIME_COMPACT_MODULE)
4
5 JEMPLATE_SCRIPT=../bin/lemlate
6 JEMPLATE_STANDALONE_SCRIPT=../lemlate
7 GRAMMAR_MODULE=../lib/Lemplate/Grammar.pm
8 RUNTIME_MODULE=../lib/Lemplate/Runtime.pm
9 RUNTIME_COMPACT_MODULE=../lib/Lemplate/Runtime/Compact.pm
10 JEMPLATE_MODULES=$(GRAMMAR_MODULE) $(RUNTIME_MODULE) $(RUNTIME_COMPACT_MODULE)
11
12 all: $(JEMPLATE_STANDALONE_SCRIPT)
13
14 $(JEMPLATE_STANDALONE_SCRIPT): $(JEMPLATE_MODULES) _force
15     ./bin/make-standalone-script $(JEMPLATE_SCRIPT) > $@
16     chmod +x $@
17
18 $(GRAMMAR_MODULE): parser _force
19     (cd parser; ./yc)
20     mv parser/Grammar.pm $@
21     rm parser/Parser.output
22
23 $(RUNTIME_MODULE): lib/Lemplate/Runtime.pm _force
24     bin/tpage $< > $@
25
26 $(RUNTIME_COMPACT_MODULE): lib/Lemplate/Runtime/Compact.pm _force
27     bin/tpage $< > $@
28
29 _force:
```

[One Level Up](#)

[Top Level](#)

src/bin/ - lemlate

- [make-standalone-script](#)

[One Level Up](#)

[Top Level](#)

src/bin/make-standalone-script - leemplate

Data types defined

- [disable_libs](#)
- [get_module](#)
- [guts](#)

Source code

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5  use FindBin qw($Bin);
6  use lib "$Bin/../../lib", "$Bin/../lib";
7  use Template;
8  use IO::All;
9
10 {
11     my $script = io(shift)->all;
12     $script =~ s{^#!/usr/bin/perl$}{#!/usr/bin/env perl}m;
13
14     $script =~ /(.*\n#BOOTSTRAP-BEGIN\n)..*\n(#BOOTSTRAP-END\n.*)/s
15         or die;
16
17     print $1 . guts() . $2;
18 }
19
20 sub guts {
21     my $output = '';
22     for (qw(
23         Number::Compare
24         Text::Glob
25         File::Find::Rule
26         Template::Constants
27         Template::Base
28         Template::Config
29         Template::Document
30         Template::Exception
31         Template::Service
32         Template::Provider
33         Template
34         Template::Grammar
35         Template::Directive
36         Template::Parser
37         Lemplate::Directive
38         Lemplate::Grammar
39         Lemplate::Parser
40         Lemplate::Runtime
41         Lemplate::Runtime::Compact
42         Lemplate
43     )) {
44         $output .= get_module($_);
45     }
46     return disable_libs() . $output;
47 }
48
49 sub disable_libs {
50     return <<'...';
51     # This is the standalone Lemplate compiler.
52     #
53     # All you need is this program and the program called `perl`. You don't need
54     # to install any Perl modules.
55     #
56     # If you downloaded this program from the internet, don't forget to put it in
57     # your path and make sure it is executable. Like this:
58     #
```

```

59 # mv lempate /usr/local/bin/
60 # chmod +x /usr/local/bin/lempate
61 #
62 # Try this command to make sure it works:
63 #
64 # lempate --help
65
66 use Config;
67 BEGIN {
68     @INC = (
69         $Config::Config{archlib},
70         $Config::Config{privlib},
71     );
72 }
73 use strict;
74 use warnings;
75
76 ...
77 }
78
79 sub get_module {
80     my $module = shift;
81     eval "require $module; 1" or die "$module not found";
82     $module =~ s{::}{/}g;
83     $module .= '.pm';
84     my $content = io($INC{$module})->all;
85     # Get rid of DATA section
86     $content =~ s/^__(END|DATA)__.*/sm;
87     # Remove POD
88     $content =~ s/^=\w+.*?(\n=cut\n|\n$)//msg;
89     # Remove comments
90     $content =~ s/^#.*\n//gm;
91
92     # Return the concatenation of prerequisite modules
93     return
94         "\n# Inline include of $module\n#\n" .
95         "BEGIN { \ $INC{'$module'} = 'dummy/$module'; }\n" .
96         "BEGIN {\n" .
97         "#line 0 \"$module\"\n" .
98         $content .
99         "\n}\n" .
100         "";
101 }

```

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

src/parser/ - lemlite

- [Grammar.pm.skel](#)
- [Parser.y](#)
- [yc](#)

[One Level Up](#)

[Top Level](#)

src/parser/Grammar.pm.skel - lemlate

```
1  #===== -*-Perl-*-
2  #
3  # Lemplate::Grammar
4  #
5  # DESCRIPTION
6  #   Grammar file for the Template Toolkit language containing token
7  #   definitions and parser state/rules tables generated by Parse::Yapp.
8  #
9  # AUTHOR
10 #   Ingy döt Net   <ingy@cpan.org>
11 #
12 # ORIGINAL AUTHOR
13 #   Andy Wardley   <abw@kfs.org>
14 #
15 # COPYRIGHT
16 #   Copyright (C) 2006-2008 Ingy döt Net.
17 #   Copyright (C) 1996-2000 Andy Wardley.
18 #   Copyright (C) 1998-2000 Canon Research Centre Europe Ltd.
19 #
20 #   This module is free software; you can redistribute it and/or
21 #   modify it under the same terms as Perl itself.
22 #
23 #-----
24 #
25 # NOTE: this module is constructed from the parser/Grammar.pm.skel
26 # file by running the parser/yc script. You only need to do this if
27 # you have modified the grammar in the parser/Parser.yp file and need
28 # to-recompile it. See the README in the 'parser' directory for more
29 # information (sub-directory of the Lemplate distribution).
30 #
31 #=====
32
33 package Lemplate::Grammar;
34
35 require 5.004;
36
37 use strict;
38 use vars qw( $VERSION );
39
40 $VERSION = sprintf("%d.%02d", q$Revision: 2.10 $ =~ /\d+\.\d+/);
41
42 my (@RESERVED, %CMPPOP, $LEXTABLE, $RULES, $STATES);
43 my ($factory, $rawstart);
44
45
46 #=====
47 # Reserved words, comparison and binary operators
48 #=====
49
50 @RESERVED = qw(
51     GET CALL SET DEFAULT INSERT INCLUDE PROCESS WRAPPER BLOCK END
52     USE RAW PLUGIN FILTER MACRO JAVASCRIPT TO STEP AND OR NOT DIV MOD
53     IF UNLESS ELSE ELSIF FOR NEXT WHILE SWITCH CASE META IN
54     TRY THROW CATCH FINAL LAST RETURN STOP CLEAR VIEW DEBUG
55 );
56
57 # for historical reasons, != and == are converted to ne and eq to perform
58 # stringwise comparison (mainly because it doesn't generate "non-numerical
59 # comparison" warnings which != and == can) but the others (e.g. < > <= >=)
60 # are not converted to their stringwise equivalents. I added 'gt' et al,
61 # briefly for v2.04d and then took them out again in 2.04e.
62
63 %CMPPOP = qw(
64     != ~=
65     == ==
66     < <
67     > >
68     >= >=
69     <= <=
70 );
71
```

```

72
73 #=====
74 # Lexer Token Table
75 #=====
76
77 # lookup table used by lexer is initialised with special-cases
78 $LEXTABLE = {
79     'FOREACH' => 'FOR',
80     'BREAK'   => 'LAST',
81     '&&'       => 'AND',
82     '||'      => 'OR',
83     '!'       => 'NOT',
84     '|'       => 'FILTER',
85     '.'       => 'DOT',
86     '_'       => 'CAT',
87     '..'      => 'TO',
88     # ':'      => 'MACRO',
89     '='       => 'ASSIGN',
90     '=>'      => 'ASSIGN',
91     # '->'    => 'ARROW',
92     ','       => 'COMMA',
93     '\\\\'    => 'REF',
94     'and'     => 'AND',          # explicitly specified so that qw( and or
95     'or'      => 'OR',          # not ) can always be used in lower case,
96     not      => 'NOT',          # regardless of ANYCASE flag
97     'mod'     => 'MOD',
98     'div'     => 'DIV',
99 };
100
101 # localise the temporary variables needed to complete lexer table
102 {
103     # my @tokens = qw< ( ) [ ] { } ${ $ / ; : ? >;
104     my @tokens = qw< ( ) [ ] { } ${ $ + / ; : ? >;
105     my @cmpop  = keys %CMPOP;
106     # my @binop = qw( + - * % );          # '/' above, in @tokens
107     my @binop  = qw( - * % );            # '+' and '/' above, in @tokens
108
109     # fill lexer table, slice by slice, with reserved words and operators
110     @$LEXTABLE{ @RESERVED, @cmpop, @binop, @tokens }
111     = ( @RESERVED, ('CMPOP') x @cmpop, ('BINOP') x @binop, @tokens );
112 }
113
114
115 #=====
116 # CLASS METHODS
117 #=====
118
119 sub new {
120     my $class = shift;
121     bless {
122         LEXTABLE => $LEXTABLE,
123         STATES   => $STATES,
124         RULES    => $RULES,
125     }, $class;
126 }
127
128 # update method to set package-scoped $factory lexical
129 sub install_factory {
130     my ($self, $new_factory) = @_;
131     $factory = $new_factory;
132 }
133
134
135 #=====
136 # States
137 #=====
138
139 $STATES = <<$states>>;
140
141
142 #=====
143 # Rules
144 #=====
145
146 $RULES = <<$rules>>;
147

```

[One Level Up](#)

[Top Level](#)

src/parser/Parser.y - lemlate

```
1  #===== -*-Perl-*-*
2  #
3  # Parser.y
4  #
5  # DESCRIPTION
6  #   Definition of the parser grammar for the Template Toolkit language.
7  #
8  # AUTHOR
9  #   Ingy döt Net   <ingy@cpan.org>
10 #
11 # ORIGINAL AUTHOR
12 #   Andy Wardley   <abw@kfs.org>
13 #
14 # HISTORY
15 #   Totally re-written for version 2, based on Doug Steinwand's
16 #   implementation which compiles templates to Perl code. The generated
17 #   code is considerably faster, more portable and easier to process.
18 #
19 # WARNINGS
20 #   Expect 1 reduce/reduce conflict. This can safely be ignored.
21 #   Now also expect 1 shift/reduce conflict, created by adding a rule
22 #   to 'args' to allow assignments of the form 'foo.bar = baz'. It
23 #   should be possible to fix the problem by rewriting some rules, but
24 #   I'm loathed to hack it up too much right now. Maybe later.
25 #
26 # COPYRIGHT
27 #   Copyright (C) 2006,2008 Ingy döt Net.
28 #   Copyright (C) 1996-2004 Andy Wardley.
29 #   Copyright (C) 1998-2004 Canon Research Centre Europe Ltd.
30 #
31 #   This module is free software; you can redistribute it and/or
32 #   modify it under the same terms as Perl itself.
33 #
34 #-----
35 #
36 # NOTE: this module is constructed from the parser/Grammar.pm.skel
37 # file by running the parser/yc script. You only need to do this if
38 # you have modified the grammar in the parser/Parser.y file and need
39 # to recompile it. See the README in the 'parser' directory for more
40 # information (sub-directory of the Template distribution).
41 #
42 #-----
43 #
44 # $Id: Parser.y,v 2.20 2004/01/13 15:32:22 abw Exp $
45 #
46 #=====
47
48 %right ASSIGN
49 %right '?' ':'
50 %left COMMA
51 %left AND OR
52 %left NOT
53 %left CAT
54 %left DOT
55 %left CPOP
56 %left BINOP
57 %left '+'
58 %left '/'
59 %left DIV
60 %left MOD
61 %left TO
62 %%
63
64 #-----
65 # START AND TOP-LEVEL RULES
66 #-----
67
68 template:   block                { $factory->template($_[1])          }
69 ;
70
71 block:      chunks                { $factory->block($_[1])          }
```

```

72 | /* NULL */ { $factory->block() }
73 ;
74
75 chunks:      chunks chunk { push(@{$_[1]}, $_[2])
76             if defined $_[2]; $_[1] }
77 | chunk      { defined $_[1] ? [ $_[1] ] : [ ] }
78 ;
79
80 chunk: TEXT { $factory->textblock($_[1]) }
81 | statement ';' { return '' unless $_[1];
82                 $_[0]->location() . $_[1];
83                 }
84 ;
85
86 statement: directive
87 | defblock
88 | anonblock
89 | capture
90 | macro
91 | use
92 | raw
93 | view
94 | rawperl
95 | expr { $factory->get($_[1]) }
96 | META metadata { $_[0]->add_metadata($_[2]); }
97 | /* empty statement */
98 ;
99
100 directive: setlist { $factory->set($_[1]) }
101 | atomdir
102 | condition
103 | switch
104 | loop
105 | try
106 | | javascript
107 | perl
108 ;
109
110
111 #-----
112 # DIRECTIVE RULES
113 #-----
114
115 atomexpr: expr { $factory->get($_[1]) }
116 | atomdir
117 ;
118
119 atomdir: GET expr { $factory->get($_[2]) }
120 | CALL expr { $factory->call($_[2]) }
121 | SET setlist { $factory->set($_[2]) }
122 | DEFAULT setlist { $factory->default($_[2]) }
123 | INSERT nameargs { $factory->insert($_[2]) }
124 | INCLUDE nameargs { $factory->include($_[2]) }
125 | PROCESS nameargs { $factory->process($_[2]) }
126 | THROW nameargs { $factory->throw($_[2]) }
127 | RETURN { $factory->return() }
128 | STOP { $factory->stop() }
129 | CLEAR { $factory->clear() }
130 | LAST { $factory->break() }
131 | NEXT { $factory->next() }
132 | DEBUG nameargs { if ($_[2]->[0]->[0] =~ /\^(on|off)'$/ ) {
133                     $_[0]->{ DEBUG_DIRS } = ($1 eq 'on');
134                     $factory->debug($_[2]);
135                 }
136                 else {
137                     $_[0]->{ DEBUG_DIRS } ? $factory->debug($_[2]) : '';
138                 }
139             }
140 | wrapper
141 | filter
142 ;
143
144 condition: IF expr ';'
145             block else END { $factory->if(@_[2, 4, 5]) }
146 | atomexpr IF expr { $factory->if(@_[3, 1]) }
147 | UNLESS expr ';'

```

```

148     block else END      { $factory->if("tt2_not($_[2])", @_[4, 5]) }
149 |   atomexpr UNLESS expr { $factory->if("tt2_not($_[3])", $_[1]) }
150 ;
151
152 else:      ELSIF expr ';'
153     block else      { unshift(@{$_[5]}, [ @_[2, 4] ]);
154         $_[5];      }
155 |   ELSE ';' block    { [ $_[3] ] }
156 |   /* NULL */      { [ undef ] }
157 ;
158
159 switch:      SWITCH expr ';'
160     block case END      { $factory->switch(@_[2, 5]) }
161 ;
162
163 case:      CASE term ';' block
164     case      { unshift(@{$_[5]}, [ @_[2, 4] ]);
165         $_[5];      }
166 |   CASE DEFAULT ';' block { [ $_[4] ] }
167 |   CASE ';' block      { [ $_[3] ] }
168 |   /* NULL */      { [ undef ] }
169 ;
170
171 loop:      FOR loopvar ';'      { $_[0]->{ INFOR }++ }
172     block END      { $_[0]->{ INFOR }--;
173         $factory->foreach(@{$_[2]}, $_[5]) }
174 #loop:      FOR loopvar ';'
175 #     block END      { $factory->foreach(@{$_[2]}, $_[4]) }
176 |   atomexpr FOR loopvar { $factory->foreach(@{$_[3]}, $_[1]) }
177 |   WHILE expr ';'      { $_[0]->{ INWHILE }++ }
178     block END      { $_[0]->{ INWHILE }--;
179         $factory->while(@_[2, 5]) }
180 |   atomexpr WHILE expr { $factory->while(@_[3, 1]) }
181 ;
182
183 loopvar:      IDENT ASSIGN term args { [ @_[1, 3, 4] ] }
184 |   IDENT IN term args { [ @_[1, 3, 4] ] }
185 |   term args { [ 0, @_[1, 2] ] }
186 ;
187
188 wrapper:      WRAPPER nameargs ';'
189     block END      { $factory->wrapper(@_[2, 4]) }
190 |   atomexpr
191     WRAPPER nameargs { $factory->wrapper(@_[3, 1]) }
192 ;
193
194 try:      TRY ';'
195     block final END      { $factory->try(@_[3, 4]) }
196 ;
197
198 final:      CATCH filename ';'
199     block final      { unshift(@{$_[5]}, [ @_[2,4] ]);
200         $_[5];      }
201 |   CATCH DEFAULT ';'
202     block final      { unshift(@{$_[5]}, [ undef, $_[4] ]);
203         $_[5];      }
204 |   CATCH ';'
205     block final      { unshift(@{$_[4]}, [ undef, $_[3] ]);
206         $_[4];      }
207 |   FINAL ';' block    { [ $_[3] ] }
208 |   /* NULL */      { [ 0 ] } # no final
209 ;
210
211 use:      USE lnameargs      { $factory->use($_[2]) }
212 ;
213
214 raw:      RAW lnameargs      { $factory->raw($_[2]) }
215 ;
216
217 view:      VIEW nameargs ';'      { $_[0]->push\_defblock(); }
218     block END      { $factory->view(@_[2,5],
219         $_[0]->pop\_defblock) }
220 ;
221
222 javascript: JAVASCRIPT ';'      { ${$_[0]}->{ INJAVASCRIPT } }++
223     block END      { ${$_[0]}->{ INJAVASCRIPT } }--;

```

```

224         $_[0]->{ EVAL_JAVASCRIPT }
225         ? $factory->javascript($_[4])
226         : $factory->no_javascript();
227     }
228
229     filter:      FILTER lnameargs ';'
230         block END      { $factory->filter(@_[2,4]) }
231     | atomexpr FILTER
232         lnameargs      { $factory->filter(@_[3,1]) }
233
234
235     defblock: defblockname
236         blockargs ';'
237         template END      { my $name = join('/', @{$_[0]->{ DEFBLOCKS } });
238             pop(@{ $_[0]->{ DEFBLOCKS } });
239             $_[0]->define_block($name, $_[4]);
240             undef
241         }
242
243
244     defblockname: BLOCK blockname      { push(@{ $_[0]->{ DEFBLOCKS } }, $_[2]);
245         $_[2];
246     }
247
248
249     blockname: filename
250         | LITERAL      { $_[1] =~ s/^(.*)'$/1/; $_[1] }
251
252
253     blockargs: metadata
254         | /* NULL */
255
256
257     anonblock: BLOCK blockargs ';' block END
258         { local $" = ', ';
259             print STDERR "experimental block args: [@{ $_[2] }]\n"
260             if $_[2];
261             $factory->anon_block($_[4]) }
262
263
264     capture: ident ASSIGN mdir      { $factory->capture(@_[1, 3]) }
265
266
267     macro:      MACRO IDENT '(' margs ')'
268         mdir      { $factory->macro(@_[2, 6, 4]) }
269     | MACRO IDENT mdir      { $factory->macro(@_[2, 3]) }
270
271
272     mdir:      directive
273     | BLOCK ';' block END      { $_[3] }
274
275
276     margs:      margs IDENT      { push(@{ $_[1] }, $_[2]); $_[1] }
277     | margs COMMA      { $_[1] }
278     | IDENT      { [ $_[1] ] }
279
280
281     metadata: metadata meta      { push(@{ $_[1] }, @{ $_[2] }); $_[1] }
282     | metadata COMMA
283     | meta
284
285
286     meta:      IDENT ASSIGN LITERAL      { for ($_[3]) { s/^\//; s/'$//;
287         s/\\\'/\'/g };
288         [ @_[1,3] ] }
289     | IDENT ASSIGN TEXT TEXT      { [ @_[1,4] ] }
290     | IDENT ASSIGN NUMBER      { [ @_[1,3] ] }
291
292
293
294     #-----
295     # FUNDAMENTAL ELEMENT RULES
296     #-----
297
298     term:      lterm
299     | sterm

```

```

300 ;
301
302 lterm:      '[' list      ']'      { "{ $_[2] }" }
303 | '[' range ']'      { "{ $_[2] }" }
304 | '['      ']'      { "{ }" }
305 | '[' hash  ']'      { "{ $_[2] }" }
306 ;
307
308 sterm:      ident      { $factory->ident($_[1]) }
309 | REF ident      { $factory->identref($_[2]) }
310 | '"" quoted '""      { $factory->quoted($_[2]) }
311 | LITERAL
312 | NUMBER
313 ;
314
315 list:      list term      { "$_[1], $_[2]" }
316 | list COMMA
317 | term
318 ;
319
320 range:      sterm TO sterm      { "$_[1] . '...' . $_[3]" }
321 ;
322
323
324 hash:      params
325 | /* NULL */      { "" }
326 ;
327
328 params:      params param      { "$_[1], $_[2]" }
329 | params COMMA
330 | param
331 ;
332
333 param:      LITERAL ASSIGN expr      { "$_[1]] = $_[3]" }
334 | item ASSIGN expr      { "$_[1]] = $_[3]" }
335 ;
336
337 ident:      ident DOT node      { push(@{$_[1]}, @{$_[3]}); $_[1] }
338 | ident DOT NUMBER      { push(@{$_[1]},
339 | map {($_, 0)} split(/\./, $_[3]));
340 | $_[1]; }
341 | node
342 ;
343
344 node:      item      { [ $_[1], 0 ] }
345 | item '(' args ')'      { [ $_[1], $factory->args($_[3]) ] }
346 ;
347
348 item:      IDENT      { "'$_[1]'" }
349 | '${' sterm '}'      { $_[2] }
350 | '$' IDENT      { $_[0]->{ V1DOLLAR }
351 | ? "'$_[2]'"
352 | : $factory->ident("$_[2]", 0) }
353 ;
354
355 expr:      expr BINOP expr      { "$_[1] $_[2] $_[3]" }
356 | expr '/' expr      { "$_[1] $_[2] $_[3]" }
357 | expr '+' expr      { "$_[1] $_[2] $_[3]" }
358 | expr DIV expr      { "math_floor($_[1] / $_[3])" }
359 | expr MOD expr      { "$_[1] % $_[3]" }
360 | expr CMPOP expr      { "$_[1] $CMPOP{ $_[2] } $_[3]" }
361 | expr CAT expr      { "$_[1] .. $_[3]" }
362 | expr AND expr      { "tt2 true($_[1]) and tt2 true($_[3])" }
363 | expr OR expr      { "tt2 true($_[1]) or tt2 true($_[3])" }
364 | NOT expr      { "tt2 not($_[2])" }
365 | expr '?' expr ':' expr      { "tt2 true($_[1]) and $_[3] or $_[5]" }
366 | '(' assign ')'      { $factory->assign(@{$_[2]}) }
367 | '(' expr ')'      { "($_[2])" }
368 | term
369 ;
370
371 setlist:      setlist assign      { push(@{$_[1]}, @{$_[2]}); $_[1] }
372 | setlist COMMA
373 | assign
374 ;
375

```

```

376
377 assign:      ident ASSIGN expr      { [ $_[1], $_[3] ] }
378 |      LITERAL ASSIGN expr      { [ @_[1,3] ] }
379 ;
380
381 # The 'args' production constructs a list of named and positional
382 # parameters. Named parameters are stored in a list in element 0
383 # of the args list. Remaining elements contain positional parameters
384
385 args:      args expr      { push(@{$_[1]}, $_[2]); $_[1] }
386 |      args param      { push(@{$_[1]->[0]}, $_[2]); $_[1] }
387 |      args ident ASSIGN expr { push(@{$_[1]->[0]}, "'", " .
388 |                               $factory->assign(@_[2,4])); $_[1] }
389 |      args COMMA      { $_[1] }
390 |      /* init */      { [ [ ] ] }
391 ;
392
393
394 # These are special case parameters used by INCLUDE, PROCESS, etc., which
395 # interpret barewords as quoted strings rather than variable identifiers;
396 # a leading '$' is used to explicitly specify a variable. It permits '/',
397 # '.' and ':' characters, allowing it to be used to specify filenames, etc.
398 # without requiring quoting.
399
400 lnameargs: lvalue ASSIGN nameargs { push(@{$_[3]}, $_[1]); $_[3] }
401 |      nameargs
402 ;
403
404 lvalue:      item
405 |      "' quoted '"      { $factory->quoted($_[2]) }
406 |      LITERAL
407 ;
408
409 nameargs:    '$' ident args      { [ [$factory->ident($_[2])], $_[3] ] }
410 |      names args      { [ @_[1,2] ] }
411 |      names '(' args ')'      { [ @_[1,3] ] }
412 ;
413
414 names:      names '+' name      { push(@{$_[1]}, $_[3]); $_[1] }
415 |      name      { [ $_[1] ] }
416 ;
417
418 name:      "' quoted '"      { $factory->quoted($_[2]) }
419 |      filename      { "$_[1]" }
420 |      LITERAL
421 ;
422
423 #nameargs:  literal args      { [ @_[1,2] ] }
424 # |      literal '(' args ')'      { [ @_[1,3] ] }
425 # |      '$' ident
426 #;
427
428 #namesargs: names args      { [ @_[1,2] ] }
429 #;
430
431 filename:   filename DOT filepart { "$_[1].$_[3]" }
432 |      filepart
433 ;
434
435 filepart: FILENAME | IDENT | NUMBER
436 ;
437
438
439 # The 'quoted' production builds a list of 'quotable' items that might
440 # appear in a quoted string, namely text and identifiers. The lexer
441 # adds an explicit ';' after each directive it finds to help the
442 # parser identify directive/text boundaries; we're not interested in
443 # them here so we can simply accept and ignore by returning undef
444
445 quoted:      quoted quotable      { push(@{$_[1]}, $_[2])
446 |                               if defined $_[2]; $_[1] }
447 |      /* NULL */      { [ ] }
448 ;
449
450 quotable:    ident      { $factory->ident($_[1]) }
451 |      TEXT      { $factory->text($_[1]) }

```

```
452 | ';'
453 ;
454
455
456 %%
457
458
459
```

{ undef }

src/parser/yc - lemlate

```
1  #!/bin/sh
2  #=====
3  #
4  # yc - yapp compile
5  #
6  # This calls 'yapp', distributed with the Parse::Yapp module, to
7  # compile the parser grammar and construct the ../Template/Grammar.pm
8  # file. The grammar is defined in ./Parser.y. The skeleton file
9  # Grammar.pm.skel is used as a template for creating the grammar file.
10 # An output file 'Parser.output' is generated containing a summary of
11 # the rule and state tables.
12 #
13 # You only need to run this script if you have changed the grammar and
14 # wish to recompile it.
15 #
16 # Andy Wardley <abw@kfs.org>
17 #
18 #=====
19
20 : ${GRAMMAR:="Parser.y"}
21 # : ${OUTPUT:="../lib/Lemplate/Grammar.pm"}
22 : ${OUTPUT:="Grammar.pm"}
23 : ${TEMPLATE:="Grammar.pm.skel"}
24
25 echo "Compiling parser grammar (${GRAMMAR} -> ${OUTPUT})"
26
27 yapp -v -s -o ${OUTPUT} -t ${TEMPLATE} ${GRAMMAR}
28
```

src/lib/Lemplate/Runtime.pm - leemplate

Data types defined

- [ajax_jquery](#)
- [ajax_xhr](#)
- [ajax_yui](#)
- [json2](#)
- [json_json2](#)
- [json_json2_internal](#)
- [json_yui](#)
- [kernel](#)
- [main](#)
- [xhr_gregory](#)
- [xhr_ilinsky](#)
- [xxx](#)

Source code

```
1 package Lemplate::Runtime;
2 use strict;
3 use warnings;
4
5 sub main { return &kernel }
6 sub kernel {
7     <<'...';
8     ...
9 }
10
11 sub ajax_jquery {
12     <<'...';
13     ...
14 }
15
16 sub ajax_xhr {
17     <<'...';
18     ...
19 }
20
21 sub ajax_yui {
22     <<'...';
23     ...
24 }
25
26 sub json_json2 {
27     <<'...';
28     ...
29 }
30
31 sub json_json2_internal {
32     <<'...';
33     ;(function(){
34
35     var JSON;
36
37     }());
38     ...
```

```

39 }
40
41 sub json_yui {
42     <<'...';
43     ...
44 }
45
46 sub json2 {
47     <<'...';
48     ...
49 }
50
51 sub xhr_gregory {
52     <<'...';
53     ...
54 }
55
56 sub xhr_ilinsky {
57     <<'...';
58     ...
59 }
60
61 sub xxx {
62     <<'...';
63     ...
64 }
65
66 1;
67
68 __END__
69
70 =encoding UTF-8
71
72 =head1 NAME
73
74 Lemlate::Runtime - Perl Module containing the Lemplate JavaScript Runtime
75
76 =head1 SYNOPSIS
77
78     use Lemplate::Runtime;
79     print Lemplate::Runtime->main;
80
81 =head1 DESCRIPTION
82
83 This module is auto-generated and used internally by Lemplate. It
84 contains subroutines that simply return various parts of the Lemplate
85 JavaScript Runtime code.
86
87 =head1 METHODS
88
89 head2 kernel
90
91 head2 ajax\_jquery
92
93 head2 ajax\_xhr
94
95 head2 ajax\_yui
96
97 head2 json\_json2
98
99 head2 json\_yui
100
101 head2 json2
102
103 head2 xhr\_gregory
104
105 head2 xhr\_ilinsky
106
107 head2 xxx
108
109 =head1 COPYRIGHT
110
111 Copyright (c) 2014. Ingy döt Net.
112
113 This program is free software; you can redistribute it and/or modify it
114 under the same terms as Perl itself.

```

115
116 See L<<http://www.perl.com/perl/misc/Artistic.html>>
117
118 =cut

[One Level Up](#)

[Top Level](#)

lib/Lemplate/Runtime.pm - leemplate

Data types defined

- [ajax_jquery](#)
- [ajax_xhr](#)
- [ajax_yui](#)
- [json2](#)
- [json_json2](#)
- [json_json2_internal](#)
- [json_yui](#)
- [kernel](#)
- [main](#)
- [xhr_gregory](#)
- [xhr_ilinsky](#)
- [xxx](#)

Source code

```
1 package Lemplate::Runtime;
2 use strict;
3 use warnings;
4
5 # VERSION
6
7 sub main { return &kernel }
8 sub kernel {
9     <<'...';
10     ...
11 }
12
13 sub ajax_jquery {
14     <<'...';
15     ...
16 }
17
18 sub ajax_xhr {
19     <<'...';
20     ...
21 }
22
23 sub ajax_yui {
24     <<'...';
25     ...
26 }
27
28 sub json_json2 {
29     <<'...';
30     ...
31 }
32
33 sub json_json2_internal {
34     <<'...';
35     ;(function(){
36
37     var JSON;
38
```

```

39 }();
40 ...
41 }
42
43 sub json_yui {
44     <<'...';
45     ...
46 }
47
48 sub json2 {
49     <<'...';
50     ...
51 }
52
53 sub xhr_gregory {
54     <<'...';
55     ...
56 }
57
58 sub xhr_ilinsky {
59     <<'...';
60     ...
61 }
62
63 sub xxx {
64     <<'...';
65     ...
66 }
67
68 1;
69
70 __END__
71
72 =encoding UTF-8
73
74 =head1 NAME
75
76 Lemplate::Runtime - Perl Module containing the Lemplate Lua Runtime
77
78 =head1 SYNOPSIS
79
80     use Lemplate::Runtime;
81     print Lemplate::Runtime->main;
82
83 =head1 DESCRIPTION
84
85 This module is auto-generated and used internally by Lemplate. It
86 contains subroutines that simply return various parts of the Lemplate
87 Lua Runtime code.
88
89 =head1 METHODS
90
91 head2 kernel
92
93 head2 ajax\_jquery
94
95 head2 ajax\_xhr
96
97 head2 ajax\_yui
98
99 head2 json\_json2
100
101 head2 json\_yui
102
103 head2 json2
104
105 head2 xhr\_gregory
106
107 head2 xhr\_ilinsky
108
109 head2 xxx
110
111 =head1 COPYRIGHT
112
113 Copyright (c) 2014. Ingy döt Net.
114

```

115 *This program is free software; you can redistribute it and/or modify it*
116 *under the same terms as Perl itself.*

117
118 See L<<http://www.perl.com/perl/misc/Artistic.html>>
119

120 =cut

[One Level Up](#)

[Top Level](#)

lib/Lemplate.pm - leemplate

Data types defined

- [_preamble](#)
- [compile_module](#)
- [compile_module_cached](#)
- [compile_template_content](#)
- [compile_template_files](#)
- [get_options](#)
- [main](#)
- [make_file_list](#)
- [new](#)
- [print_usage_and_exit](#)
- [recurse_dir](#)
- [runtime_source_code](#)
- [slurp](#)
- [usage](#)

Source code

```
1  # ToDo:
2  # - Use TT:Simple in Makefiles
3
4  # ABSTRACT: compiles Perl TT2 templates to standalone Lua modules for OpenResty
5
6  package Lemplate;
7  use strict;
8  use warnings;
9  use Template 2.14;
10 use Getopt::Long;
11
12 # VERSION
13
14 use Lemplate::Parser;
15
16 #-----
17
18 our %ExtraTemplates;
19 our %ProcessedTemplates;
20 our $TemplateName;
21
22 sub usage {
23     <<'...';
24     Usage:
25
26     leemplate --runtime [runtime-opt]
27
28     leemplate --compile [compile-opt] <template-list>
29
30     leemplate --runtime [runtime-opt] --compile [compile-opt] <template-list>
31
32     leemplate --list <template-list>
33
```

```

34 Where "--runtime" and "runtime-opt" can include:
35
36     --runtime           Equivalent to --ajax=ilinsky --json=json2
37     --runtime=standard
38
39     --runtime=lite      Same as --ajax=none --json=none
40     --runtime=jquery    Same as --ajax=jquery --json=none
41     --runtime=yui       Same as --ajax=yui --json=yui
42     --runtime=legacy    Same as --ajax=gregory --json=json2
43
44     --json              By itself, equivalent to --json=json2
45     --json=json2          Include http://www.json.org/json2.js for parsing/stringifying
46     --json=yui          Use YUI: YAHOO.lang.JSON (requires external YUI)
47     --json=none         Doesn't provide any JSON functionality except a warning
48
49     --ajax              By itself, equivalent to --ajax=xhr
50     --ajax=jquery       Use jQuery for Ajax get and post (requires external jQuery)
51     --ajax=yui          Use YUI: yui/connection/connection.js (requires external YUI)
52     --ajax=xhr          Use XMLHttpRequest (will automatically use --xhr=ilinsky if --xhr is not set)
53     --ajax=none         Doesn't provide any Ajax functionality except a warning
54
55     --xhr               By itself, equivalent to --xhr=ilinsky
56     --xhr=ilinsky       Include http://code.google.com/p/xmlhttprequest/
57     --xhr=gregory       Include http://www.scss.com.au/family/andrew/webdesign/xmlhttprequest/
58
59     --xxx               Include XXX and JJJ helper functions
60
61     --compact           Use the YUICompressor compacted version of the runtime
62

```

63 Where "compile-opt" can [include](#):

```

64
65     --include\_path=DIR  Add directory to INCLUDE_PATH
66
67     --start-tag
68     --end-tag
69     --pre-chomp
70     --post-chomp
71     --trim
72     --any-case
73     --eval
74     --noeval
75     -s, --source
76     --exclude
77

```

78 For more information [use](#):

```

79     perldoc leemplate
80     ...
81 }
82

```

```

83 sub main {
84     my $class = shift;
85
86     my @argv = @_;
87
88     my ($template_options, $leemplate_options) = get\_options(@argv);
89     my ($runtime, $compile, $list) = @$template_options{qw/runtime compile list/};
90
91     if ($runtime) {
92         print runtime source code(@$leemplate_options{qw/runtime ajax json xhr xxx compact/});
93         return unless $compile;
94     }
95
96     my $templates = make\_file\_list($leemplate_options->{exclude}, @argv);
97     print usage and exit() unless @$templates;
98
99     if ($list) {
100         foreach (@$templates) {
101             print STDOUT $_->{short} . "\n";
102         }
103         return;
104     }
105
106     if ($compile) {
107         my $leemplate = Lemplate->new(%$leemplate_options);
108         print STDOUT $leemplate->preamble;
109         for (my $i = 0; $i < @$templates; $i++) {

```

```

110 my $template = $templates->[$i];
111 #warn "processing $template->{short}";
112 my $content = slurp($template->{full});
113 if ($content) {
114     %ExtraTemplates = ();
115     print STDOUT $template->compile_template_content(
116         $content,
117         $template->{short}
118     );
119     my @new_files;
120     for my $new_template (keys %ExtraTemplates) {
121         if (!$ProcessedTemplates{$new_template}) {
122             if (!-f $new_template) {
123                 $new_template = "t/data/" . $new_template;
124             }
125             #warn $new_template;
126             if (-f $new_template) {
127                 #warn "adding new template $new_template";
128                 push @new_files, $new_template;
129             }
130         }
131     }
132     push @$templates, @{ make_file_list({}, @new_files) };
133 }
134 }
135 print STDOUT "return _M\n";
136 return;
137 }
138
139 print usage_and_exit();
140 }
141
142 sub get_options {
143     local @ARGV = @_;
144
145     my $runtime;
146     my $compile = 0;
147     my $list = 0;
148
149     my $start_tag = exists $ENV{LEMPATE_START_TAG}
150         ? $ENV{LEMPATE_START_TAG}
151         : undef;
152     my $end_tag = exists $ENV{LEMPATE_END_TAG}
153         ? $ENV{LEMPATE_END_TAG}
154         : undef;
155     my $pre_chomp = exists $ENV{LEMPATE_PRE_CHOMP}
156         ? $ENV{LEMPATE_PRE_CHOMP}
157         : undef;
158     my $post_chomp = exists $ENV{LEMPATE_POST_CHOMP}
159         ? $ENV{LEMPATE_POST_CHOMP}
160         : undef;
161     my $trim = exists $ENV{LEMPATE_TRIM}
162         ? $ENV{LEMPATE_TRIM}
163         : undef;
164     my $anycase = exists $ENV{LEMPATE_ANYCASE}
165         ? $ENV{LEMPATE_ANYCASE}
166         : undef;
167     my $eval_javascript = exists $ENV{LEMPATE_EVAL_JAVASCRIPT}
168         ? $ENV{LEMPATE_EVAL_JAVASCRIPT}
169         : 1;
170
171     my $source = 0;
172     my $exclude = 0;
173     my ($ajax, $json, $xxx, $xhr, $compact, $minify);
174
175     my $help = 0;
176     my @include_paths;
177
178     GetOptions(
179         "compile|c"    => \$compile,
180         "list|l"       => \$list,
181         "runtime|r:s"  => \$runtime,
182
183         "start-tag=s"  => \$start_tag,
184         "end-tag=s"    => \$end_tag,
185         "trim=s"       => \$trim,

```

```

186     "pre-chomp"      => \$pre_chomp,
187     "post-chomp"     => \$post_chomp,
188     "any-case"       => \$anycase,
189     "eval!"          => \$eval_javascript,
190
191     "source|s"        => \$source,
192     "exclude=s"       => \$exclude,
193
194     "ajax:s"          => \$ajax,
195     "json:s"          => \$json,
196     "xxx"             => \$xxx,
197     "xhr:s"           => \$xhr,
198
199     "include_path"    => \@include_paths,
200     "compact"         => \$compact,
201     "minify:s"        => \$minify,
202
203     "help|?"          => \$help,
204 ) or print_usage_and_exit();
205
206 if ($help) {
207     print_usage_and_exit();
208 }
209
210 ($runtime, $ajax, $json, $xxx, $xhr, $minify) = map { defined $_ && ! length $_ ? 1 : $_ } ($runtime,
$ajax, $json, $xxx, $xhr, $minify);
211 $runtime = "standard" if $runtime && $runtime eq 1;
212
213 print_usage_and_exit("Don't understand '--runtime $runtime'") if defined $runtime && ! grep { $runtime =~
m/$_/ } qw/standard lite jquery yui legacy/;
214 print_usage_and_exit("Can't specify --list with a --runtime and/or the --compile option") if $list &&
($runtime || $compile);
215 print_usage_and_exit() unless $list || $runtime || $compile;
216
217 my $command =
218     $runtime ? 'runtime' :
219     $compile ? 'compile' :
220     $list ? 'list' :
221     print_usage_and_exit();
222
223 my $options = {};
224 $options->{START_TAG} = $start_tag if defined $start_tag;
225 $options->{END_TAG} = $end_tag if defined $end_tag;
226 $options->{PRE_CHOMP} = $pre_chomp if defined $pre_chomp;
227 $options->{POST_CHOMP} = $post_chomp if defined $post_chomp;
228 $options->{TRIM} = $trim if defined $trim;
229 $options->{ANYCASE} = $anycase if defined $anycase;
230 $options->{EVAL_JAVASCRIPT} = $eval_javascript if defined $eval_javascript;
231 $options->{INCLUDE_PATH} = \@include_paths;
232
233 return (
234     $options,
235     { compile => $compile, runtime => $runtime, list => $list,
236       source => $source,
237       exclude => $exclude,
238       ajax => $ajax, json => $json, xxx => $xxx, xhr => $xhr,
239       compact => $compact, minify => $minify },
240 );
241 }
242
243
244 sub slurp {
245     my $filepath = shift;
246     open(F, '<', $filepath) or die "Can't open '$filepath' for input:\n$!";
247     my $contents = do {local $/; <F>};
248     close(F);
249     return $contents;
250 }
251
252 sub recurse_dir {
253     require File::Find::Rule;
254
255     my $dir = shift;
256     my @files;
257     foreach ( File::Find::Rule->file->in( $dir ) ) {
258         if ( m{/\.[^\.]+"} ) {} # Skip ".hidden" files or directories

```

```

259         else {
260             push @files, $_;
261         }
262     }
263     return @files;
264 }
265
266 sub make_file_list {
267     my ($exclude, @args) = @_;
268
269     my @list;
270
271     foreach my $arg (@args) {
272         unless (-e $arg) { next; } # file exists
273         unless (-s $arg or -d $arg) { next; } # file size > 0 or directory (for Win platform)
274         if ($exclude and $arg =~ m/$exclude/) { next; } # file matches exclude regex
275
276         if (-d $arg) {
277             foreach my $full ( recurse_dir($arg) ) {
278                 $full =~ /$arg(\\|/)(.*)/;
279                 my $short = $2;
280                 push(@list, {full=>$full, short=>$short} );
281             }
282         }
283         else {
284             my $full = $arg;
285             my $short = $full;
286             $short =~ s/[\\\/]//;
287             push(@list, {full=>$arg, short=>$short} );
288         }
289     }
290
291     return [ sort { $a->{short} cmp $b->{short} } @list ];
292 }
293
294 sub print_usage_and_exit {
295     print STDOUT join "\n", "", @_, "Aborting!", "\n" if @_;
296     print STDOUT usage();
297     exit;
298 }
299
300 sub runtime_source_code {
301     require Lemplate::Runtime;
302     require Lemplate::Runtime::Compact;
303
304     unshift @_, "standard" unless @_;
305
306     my ($runtime, $ajax, $json, $xhr, $xxx, $compact) = map { defined $_ ? lc $_ : "" } @_[0 .. 5];
307
308     my $Lemplate_Runtime = $compact ? "Lemplate::Runtime::Compact" : "Lemplate::Runtime";
309
310     if ($runtime eq "standard") {
311         $ajax ||= "xhr";
312         $json ||= "json2";
313         $xhr ||= "ilinsky";
314     }
315     elsif ($runtime eq "jquery") {
316         $ajax ||= "jquery";
317     }
318     elsif ($runtime eq "yui") {
319         $ajax ||= "yui";
320         $json ||= "yui";
321     }
322     elsif ($runtime eq "legacy") {
323         $ajax ||= "xhr";
324         $json ||= "json2";
325         $xhr ||= "gregory";
326         $xxx = 1;
327     }
328     elsif ($runtime eq "lite") {
329     }
330
331     $ajax = "xhr" if $ajax eq 1;
332     $xhr ||= 1 if $ajax eq "xhr";
333     $json = "json2" if $json eq 1;
334     $xhr = "ilinsky" if $xhr eq 1;

```

```

335 my @runtime;
336
337
338 push @runtime, $Lemplantime_Runtine->kernel if $runtime;
339
340 push @runtime, $Lemplantime_Runtine->json2 if $json =~ m/^json2?$/i;
341
342 push @runtime, $Lemplantime_Runtine->ajax_xhr if $ajax eq "xhr";
343 push @runtime, $Lemplantime_Runtine->ajax_jquery if $ajax eq "jquery";
344 push @runtime, $Lemplantime_Runtine->ajax_yui if $ajax eq "yui";
345
346 push @runtime, $Lemplantime_Runtine->json_json2 if $json =~ m/^json2?$/i;
347 push @runtime, $Lemplantime_Runtine->json_json2_internal if $json =~ m/^json2?[_-]?internal$/i;
348 push @runtime, $Lemplantime_Runtine->json_yui if $json eq "yui";
349
350 push @runtime, $Lemplantime_Runtine->xhr_ilinsky if $xhr eq "ilinsky";
351 push @runtime, $Lemplantime_Runtine->xhr_gregory if $xhr eq "gregory";
352
353 push @runtime, $Lemplantime_Runtine->xxx if $xxx;
354
355 return join ";", @runtime;
356 }
357
358 #-----
359
360 sub new {
361     my $class = shift;
362     return bless { @_ }, $class;
363 }
364
365 sub compile_module {
366     my ($self, $module_path, $template_file_paths) = @_;
367     my $result = $self->compile_template_files(@$template_file_paths)
368         or return;
369     open MODULE, "> $module_path"
370         or die "Can't open '$module_path' for output:\n$!";
371     print MODULE $result;
372     close MODULE;
373     return 1;
374 }
375
376 sub compile_module_cached {
377     my ($self, $module_path, $template_file_paths) = @_;
378     my $m = -M $module_path;
379     return 0 unless grep { -M($_) < $m } @$template_file_paths;
380     return $self->compile_module($module_path, $template_file_paths);
381 }
382
383 sub compile_template_files {
384     my $self = shift;
385     my $output = $self->preamble;
386     for my $filepath (@_) {
387         my $filename = $filepath;
388         $filename =~ s/[\\\/]/_/;
389         open FILE, $filepath
390             or die "Can't open '$filepath' for input:\n$!";
391         my $template_input = do {local $/; <FILE>};
392         close FILE;
393         $output .=
394             $self->compile_template_content($template_input, $filename);
395     }
396     return $output;
397 }
398
399 sub compile_template_content {
400     die "Invalid arguments in call to Lemplantime->compile_template_content"
401         unless @_ == 3;
402     my ($self, $template_content, $template_name) = @_;
403     $TemplateName = $template_name;
404     my $parser = Lemplantime::Parser->new( ref($self) ? %$self : () );
405     my $parse_tree = $parser->parse(
406         $template_content, {name => $template_name}
407     ) or die $parser->error;
408     my $output =
409         "-- $template_name\n" .
410         "template_map['$template_name'] = " .

```

```

411     $parse_tree->{BLOCK} .
412     "\n";
413     for my $function_name (sort keys %{$parse_tree->{DEFBLOCKS}}) {
414         my $name = "$template_name/$function_name";
415         next if $ProcessedTemplates{$name};
416         #warn "seen $name";
417         $ProcessedTemplates{$name} = 1;
418         $output .=
419             "template_map['$name'] = " .
420             $parse_tree->{DEFBLOCKS}{$function_name} .
421             "\n";
422     }
423     return $output;
424 }
425
426 sub _preamble {
427     return <<'...';
428 --[[
429     This Lua code was generated by Lemplate, the Lua
430     Template Toolkit. Any changes made to this file will be lost the next
431     time the templates are compiled.
432
433     Copyright 2016 - Yichun Zhang (agentzh) - All rights reserved.
434
435     Copyright 2006-2014 - Ingy döt Net - All rights reserved.
436 ]]
437
438 local gsub = ngx.re.gsub
439 local concat = table.concat
440 local type = type
441 local math_floor = math.floor
442 local table_maxn = table.maxn
443
444 local _M = {
445     version = '0.02'
446 }
447
448 local template_map = {}
449
450 local function tt2_true(v)
451     return v and v ~= 0 and v ~= "" and v ~= '0'
452 end
453
454 local function tt2_not(v)
455     return not v or v == 0 or v == "" or v == '0'
456 end
457
458 local context_meta = {}
459
460 function context_meta.plugin(context, name, args)
461     if name == "iterator" then
462         local list = args[1]
463         local count = table_maxn(list)
464         return { list = list, count = 1, max = count - 1, index = 0, size = count, first = true, last =
false, prev = "" }
465     else
466         return error("unknown iterator: " .. name)
467     end
468 end
469
470 function context_meta.process(context, file)
471     local f = template_map[file]
472     if not f then
473         return error("file error - " .. file .. ": not found")
474     end
475     return f(context)
476 end
477
478 function context_meta.include(context, file)
479     local f = template_map[file]
480     if not f then
481         return error("file error - " .. file .. ": not found")
482     end
483     return f(context)
484 end
485

```

```

486 context_meta = { __index = context_meta }
487
488 local function stash_get(stash, k)
489     local v
490     if type(k) == "table" then
491         v = stash
492         for i = 1, #k, 2 do
493             local key = k[i]
494             local typ = k[i + 1]
495             if type(typ) == "table" then
496                 local value = v[key]
497                 if type(value) == "function" then
498                     return value()
499                 end
500                 if value then
501                     return value
502                 end
503                 if key == "size" then
504                     if type(v) == "table" then
505                         return #v
506                     else
507                         return 1
508                     end
509                 else
510                     return error("virtual method " .. key .. " not supported")
511                 end
512             end
513             if type(key) == "number" and key == math_floor(key) and key >= 0 then
514                 key = key + 1
515             end
516             if type(v) ~= "table" then
517                 return nil
518             end
519             v = v[key]
520         end
521     else
522         v = stash[k]
523     end
524     if type(v) == "function" then
525         return v()
526     end
527     return v
528 end
529
530 local function stash_set(stash, k, v, default)
531     if default then
532         local old = stash[k]
533         if old == nil then
534             stash[k] = v
535         end
536     else
537         stash[k] = v
538     end
539 end
540
541 function _M.process(file, params)
542     local stash = params
543     local context = {
544         stash = stash,
545         filter = function (bits, name, params)
546             local s = concat(bits)
547             if name == "html" then
548                 s = gsub(s, "&", '&amp;', "jo")
549                 s = gsub(s, "<", '&lt;', "jo");
550                 s = gsub(s, ">", '&gt;', "jo");
551                 s = gsub(s, "'", '&quot;', "jo"); -- " end quote for emacs
552                 return s
553             end
554         end
555     }
556     context = setmetatable(context, context_meta)
557     local f = template_map[file]
558     if not f then
559         return error("file error - " .. file .. ": not found")
560     end
561     return f(context)

```

```

562 end
563 ...
564 }
565
566 1;
567
568 __END__
569
570 =encoding utf8
571
572 =head1 Name
573
574 Lemplate - OpenResty/Lua template framework implementing Perl's TT2 templating language
575
576 =head1 Status
577
578 This is still under early development. Check back often.
579
580 =head1 Synopsis
581
582     local templates = require "myapp.templates"
583     ngx.print(templates.process("homepage.tt2", { var1 = 32, var2 = "foo" }))
584
585 From the command-line:
586
587     leemplate --compile path/to/leemplate/directory/ > myapp/templates.lua
588
589 =head1 Description
590
591 Lemplate is a templating framework for OpenResty/Lua that is built over
592 Perl's Template Toolkit (TT2).
593
594 Lemplate parses TT2 templates using the TT2 Perl framework, but with a
595 twist. Instead of compiling the templates into Perl code, it compiles
596 them into Lua that can run on OpenResty.
597
598 Lemplate then provides a Lua runtime module for processing
599 the template code. Presto, we have full featured Lua
600 templating language!
601
602 Combined with OpenResty, Lemplate provides a really simple
603 and powerful way to do web stuff.
604
605 =head1 HowTo
606
607 Lemplate comes with a command line tool call C<leemplate> that you use to
608 precompile your templates into a Lua module file. For example if you have
609 a template directory called C<templates> that contains:
610
611     $ ls templates/
612     body.tt2
613     footer.tt2
614     header.tt2
615
616 You might run this command:
617
618     $ leemplate --compile template/* > myapp/templates.lua
619
620 This will compile all the templates into one Lua module file which can be loaded in your
621 main OpenResty/Lua application as the module C<myapp.templates>.
622
623 Now all you need to do is load the Lua module file in your OpenResty app:
624
625     local templates = require "myapp.templates"
626
627 and do the HTML page rendering:
628
629     local results = templates.process("some-page.tt2",
630                                     { var1 = val1, var2 = val2, ...})
631
632 Now you have Lemplate support for these templates in your OpenResty application.
633
634 =head1 Public API
635
636 The Lemplate Lua runtime module has the following API method:
637

```

```

638 =over
639
640 =item process(template-name, data)
641
642 The C<template-name> is a string like C<'body.tt2'> that is the name of
643 the top level template that you wish to process.
644
645 The optional C<data> specifies the data object to be used by the
646 templates. It can be an object, a function or a url. If it is an object,
647 it is used directly. If it is a function, the function is called and the
648 returned object is used.
649
650 =back
651
652 =head1 Current Support
653
654 The goal of Lemplate is to support all of the Template Toolkit features
655 that can possibly be supported.
656
657 Lemplate now supports almost all the TT directives, including:
658
659 * Plain text
660 * [% [GET] variable %]
661 * [% CALL variable %]
662 * [% [SET] variable = value %]
663 * [% DEFAULT variable = value ... %]
664 * [% INCLUDE [arguments] %]
665 * [% PROCESS [arguments] %]
666 * [% BLOCK name %]
667 * [% FILTER filter %] text... [% END %]
668 * [% WRAPPER template [variable = value ...] %]
669 * [% IF condition %]
670 * [% ELSIF condition %]
671 * [% ELSE %]
672 * [% SWITCH variable %]
673 * [% CASE [{value|DEFAULT}] %]
674 * [% FOR x = y %]
675 * [% WHILE expression %]
676 * [% RETURN %]
677 * [% THROW type message %]
678 * [% STOP %]
679 * [% NEXT %]
680 * [% LAST %]
681 * [% CLEAR %]
682 * [%# this is a comment %]
683 * [% MACRO name(param1, param2) BLOCK %] ... [% END %]
684
685 ALL of the string virtual functions are supported.
686
687 ALL of the array virtual functions are supported:
688
689 ALL of the hash virtual functions are supported:
690
691 MANY of the standard filters are implemented.
692
693 The remaining features will be added very soon. See the DESIGN document
694 in the distro for a list of all features and their progress.
695
696 =head1 Community
697
698 =head2 English Mailing List
699
700 The L<openresty-en|https://groups.google.com/group/openresty-en> mailing list is for English speakers.
701
702 =head2 Chinese Mailing List
703
704 The L<openresty|https://groups.google.com/group/openresty> mailing list is for Chinese speakers.
705
706 =head1 Code Repository
707
708 The bleeding edge code is available via Git at
709 git://github.com/openresty/lemplate.git
710
711 =head1 Bugs and Patches
712
713 Please submit bug reports, wishlists, or patches by

```

714
715 =over
716
717 =item 1.
718
719 *creating a ticket on the [L<GitHub Issue Tracker|https://github.com/openresty/lua-nginx-module/issues>](https://github.com/openresty/lua-nginx-module/issues),*
720
721 =item 2.
722
723 *or posting to the [L</Community>](#).*
724
725 =back
726
727 =head1 CREDIT
728
729 *This project is based on Ingy dot Net's excellent [L<Jemplate>](#) project.*
730
731 =head1 AUTHOR
732
733 *Yichun Zhang (agentzh), [E<lt>agentzh@gmail.comE<gt>](mailto:agentzh@gmail.com), CloudFlare Inc.*
734
735 =head1 Copyright
736
737 *Copyright (C) 2016 Yichun Zhang (agentzh). All Rights Reserved.*
738
739 *Copyright (C) 1996-2014 Andy Wardley. All Rights Reserved.*
740
741 *Copyright (c) 2006-2014. Ingy döt Net. All rights reserved.*
742
743 *Copyright (C) 1998-2000 Canon Research Centre Europe Ltd*
744
745 *This module is free software; you can redistribute it and/or modify it under the same terms as Perl itself.*
746
747 =head1 See Also
748
749 =over
750
751 =item *
752
753 *Perl TT2 Reference Manual: [http://www.\[template\]\(#\)-toolkit.org/docs/manual/index.html](http://www.template-toolkit.org/docs/manual/index.html)*
754
755 =item *
756
757 *Jemplate for compiling TT2 templates to client-side JavaScript: <http://www.jemplate.net/>*
758
759 =back

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

t/ - lempplate

- [TestLemplate.pm](#)
- [binop.t](#)
- [block.t](#)
- [blocks.t](#)
- [iterator.t](#)
- [pod.t](#)
- [sanity.t](#)

[One Level Up](#)

[Top Level](#)

t/TestLemplate.pm - leemplate

Data types defined

- [run_test](#)
- [run_tests](#)

Source code

```
1 package t::TestLemplate;
2
3 use lib 'inc';
4 use Test::Base -Base;
5 use File::Temp qw( tempfile );
6 use File::Copy qw( copy );
7 use IPC::Run3 qw( run3 );
8 use Lemplate;
9
10 our @EXPORT = qw( run_tests );
11
12 sub run_tests {
13     for my $block (blocks()) {
14         run_test($block);
15     }
16 }
17
18 sub run_test ($) {
19     my $block = shift;
20     #print $json_xs->pretty->encode(\@new_rows);
21     #my $res = #print $json_xs->pretty->encode($res);
22     my $name = $block->name;
23
24     my $tt2 = $block->tt2;
25     if (!defined $tt2) {
26         die "No --- tt2 specified for test $name\n";
27     }
28
29     my ($out_fh, $tt2file) = tempfile("tmpXXXXX", SUFFIX => '.tt2', UNLINK => 1);
30     print $out_fh $tt2;
31     close $out_fh;
32
33     my @cmd = ($^X, "./bin/leemplate", "--compile", $tt2file);
34
35     my ($comp_out, $comp_err);
36
37     run3(\@cmd, undef, \$comp_out, \$comp_err);
38
39     #warn "res:$res\nerr:$comp_err\n";
40
41     if (defined $block->comp_err) {
42         if (ref $block->comp_err) {
43             like $comp_err, $block->comp_err, "$name - comp_err expected";
44         } else {
45             is $comp_err, $block->comp_err, "$name - comp_err expected";
46         }
47     } elsif ($?) {
48         if (defined $block->fatal) {
49             if (defined $block->fatal) {
50                 pass("failed as expected");
51             } else {
52                 fail("failed to compile TT2 source for test $name: $comp_err\n");
53                 return;
54             }
55         } else {
56             if ($comp_err) {
57                 if (!defined $block->comp_err) {
58                     warn "$comp_err\n";
59                 }
60             }
61         }
62     }
```

```

61         } else {
62             is $comp_err, $block->comp_err, "$name - err ok";
63         }
64     }
65 }
66 }
67
68 my $expected_lua = $block->lua;
69 if (defined $expected_lua) {
70     if (ref $expected_lua) {
71         like $comp_out, $expected_lua, "$name - lua expected";
72     } else {
73         is $comp_out, $expected_lua, "$name - lua expected";
74     }
75 }
76
77 my $luafile;
78 ($out_fh, $luafile) = tempfile("tmpXXXXX", SUFFIX => '.lua', UNLINK => 1);
79 print $out_fh $comp_out;
80 close $out_fh;
81
82 copy($luafile, "a.lua") or die $!;
83
84 (my $luamod = $luafile) =~ s/\.lua$//;
85
86 my $define = $block->define // '';
87 my $init = $block->init // '';
88
89 @cmd = ("resty", "-e", qq{$init ngx.print(require("$luamod").process("$tt2file", {$define})));
90 #warn "cmd: @cmd";
91
92 my ($run_out, $run_err);
93
94 run3(\@cmd, undef, \$run_out, \$run_err);
95
96 if (defined $block->lua_err) {
97     $run_err =~ s/^\s+\.lua:\d+:\s*//;
98     if (ref $block->lua_err) {
99         like $run_err, $block->lua_err, "$name - run_err expected";
100     } else {
101         is $run_err, $block->lua_err, "$name - run_err expected";
102     }
103 }
104 } elsif ($?) {
105     if (defined $block->fatal) {
106         pass("failed as expected");
107     }
108     } else {
109         fail("failed to run Lua code for test $name: $run_err\n");
110         return;
111     }
112 }
113 } else {
114     if ($run_err) {
115         if (!defined $block->lua_err) {
116             warn "$run_err\n";
117         }
118         } else {
119             is $run_err, $block->lua_err, "$name - err ok";
120         }
121     }
122 }
123
124 my $expected_out = $block->out;
125 if (defined $expected_out) {
126     if (defined $run_out) {
127         $run_out =~ s/^\n+//gs;
128         $run_out =~ s/\n\n+$/\n/g;
129     }
130     if (ref $expected_out) {
131         like $run_out, $expected_out, "$name - out expected";
132     } else {
133         is $run_out, $expected_out, "$name - out expected";
134     }
135 }
136 }

```

137

138 1;

[One Level Up](#)

[Top Level](#)

t/binop.t - lemlate

```
1 # vim:set ft= ts=4 sw=4 et fdm=marker:
2
3 use t::TestLemlate;
4
5 plan tests => 1 * blocks();
6
7 $ENV{LEMLATE_POST_CHOMP} = 1;
8
9 run_tests;
10
11 __DATA__
12
13 === TEST 1:
14 --- tt2
15 maybe
16 [% IF yes %]
17 yes
18 [% END %]
19
20 --- define: yes = 1
21 --- out
22 maybe
23 yes
24
25
26
27 === TEST 2:
28 --- tt2
29 [% IF yes %]
30 yes
31 [% ELSE %]
32 no
33 [% END %]
34
35 --- define: yes = 1
36 --- out
37 yes
38
39
40
41 === TEST 3:
42 --- tt2
43 [% IF yes %]
44 yes
45 [% ELSE %]
46 no
47 [% END %]
48
49 --- define: yes = 1
50 --- out
51 yes
52
53
54
55 === TEST 4:
56 --- tt2
57 [% IF yes and true %]
58 yes
59 [% ELSE %]
60 no
61 [% END %]
62
63 --- define: yes = 1, ['true'] = 'this is true'
64 --- out
65 yes
66
67
68
69 === TEST 5:
70 --- tt2
71 [% IF yes && true %]
```

```
72 yes
73 [% ELSE %]
74 no
75 [% END %]
76
77 --- define: yes = 1, ['true'] = 'this is true'
78 --- out
79 yes
80
81
82
83 === TEST 6:
84 --- tt2
85 [% IF yes && sad || happy %]
86 yes
87 [% ELSE %]
88 no
89 [% END %]
90
91 --- define: yes = 1, sad = '', happy = 'yes'
92 --- out
93 yes
94
95
96
97 === TEST 7:
98 --- tt2
99 [% IF yes AND ten && true and twenty && 30 %]
100 yes
101 [% ELSE %]
102 no
103 [% END %]
104
105 --- define: yes = 1, ten = 10, ['true'] = 'this is true', twenty = 20
106 --- out
107 yes
108
109
110
111 === TEST 8:
112 --- tt2
113 [% IF ! yes %]
114 no
115 [% ELSE %]
116 yes
117 [% END %]
118
119 --- define: yes = 1
120 --- out
121 yes
122
123
124
125 === TEST 9:
126 --- tt2
127 [% UNLESS yes %]
128 no
129 [% ELSE %]
130 yes
131 [% END %]
132
133 --- define: yes = 1
134 --- out
135 yes
136
137
138
139 === TEST 10:
140 --- tt2
141 [% "yes" UNLESS no %]
142
143 --- define: yes = 1, no = 0
144 --- out chomp
145 yes
146
147
```

```
148
149 === TEST 11:
150 --- tt2
151 [% IF ! yes %]
152 no
153 [% ELSE %]
154 yes
155 [% END %]
156
157 --- define: yes = 1, no = 0
158 --- out
159 yes
160
161
162
163 === TEST 12:
164 --- tt2
165 [% IF yes || no %]
166 yes
167 [% ELSE %]
168 no
169 [% END %]
170
171 --- define: yes = 1, no = 0
172 --- out
173 yes
174
175
176
177 === TEST 13:
178 --- tt2
179 [% IF yes || no || true || false %]
180 yes
181 [% ELSE %]
182 no
183 [% END %]
184
185 --- define: yes = 1, no = 0, ['true'] = 'this is true', ['false'] = '0'
186 --- out
187 yes
188
189
190
191 === TEST 14:
192 --- tt2
193 [% IF yes or no %]
194 yes
195 [% ELSE %]
196 no
197 [% END %]
198
199 --- define: yes = 1, no = 0
200 --- out
201 yes
202
203
204
205 === TEST 15:
206 --- tt2
207 [% IF not false and not sad %]
208 yes
209 [% ELSE %]
210 no
211 [% END %]
212
213 --- define: ['false'] = '0', sad = ''
214 --- out
215 yes
216
217
218
219 === TEST 16:
220 --- tt2
221 [% IF ten == 10 %]
222 yes
223 [% ELSE %]
```

```

224 no
225 [% END %]
226
227 --- define: ten = 10
228 --- out
229 yes
230
231
232
233 === TEST 17:
234 --- tt2
235 [% IF ten == twenty %]
236 I canna break the laws of mathematics, Captain.
237 [% ELSIF ten > twenty %]
238 Your numerical system is inverted. Please reboot your Universe.
239 [% ELSIF twenty < ten %]
240 Your inverted system is numerical. Please universe your reboot.
241 [% ELSE %]
242 Normality is restored. Anything you can't cope with is your own problem.
243 [% END %]
244
245 --- define: ten = 10, twenty = 20
246 --- out
247 Normality is restored. Anything you can't cope with is your own problem.
248
249
250
251 === TEST 18:
252 --- tt2
253 [% IF ten >= twenty or false %]
254 no
255 [% ELSIF twenty <= ten %]
256 nope
257 [% END %]
258 nothing
259
260 --- define: ten = 10, twenty = 20, ['false'] = '0'
261 --- out
262 nothing
263
264
265
266 === TEST 19:
267 --- tt2
268 [% IF ten >= twenty or false %]
269 no
270 [% ELSIF twenty <= ten %]
271 nope
272 [% END %]
273 nothing
274
275 --- define: ten = 10, twenty = 20, ['false'] = '0'
276 --- out
277 nothing
278
279
280
281 === TEST 20:
282 --- tt2
283 [% IF ten > twenty %]
284 no
285 [% ELSIF ten < twenty %]
286 yep
287 [% END %]
288
289 --- define: ten = 10, twenty = 20, ['false'] = '0'
290 --- out
291 yep
292
293
294
295 === TEST 21:
296 --- tt2
297 [% IF ten != 10 %]
298 no
299 [% ELSIF ten == 10 %]

```

```

300  yup
301  [% END %]
302
303  --- define: ten = 10
304  --- out
305  yup
306
307
308
309  === TEST 22:
310  --- tt2
311  [% IF alpha AND omega %]
312  alpha and omega are true
313  [% ELSE %]
314  alpha and/or omega are not true
315  [% END %]
316  count: [% count %]
317
318  --- init
319  local counter = 0
320  --- define
321  alpha = function () counter = counter + 1 return counter end,
322  omega = function () counter = counter + 10 return 0 end,
323  count = function () return counter end,
324  reset = function () return counter == 0 end
325  --- out chomp
326  alpha and/or omega are not true
327  count: 11
328
329
330
331  === TEST 23:
332  --- tt2
333  [% IF omega AND alpha %]
334  omega and alpha are true
335  [% ELSE %]
336  omega and/or alpha are not true
337  [% END %]
338  count: [% count %]
339
340  --- init: local counter = 11
341  --- define
342  ['true'] = 'this is true',
343  alpha = function () counter = counter + 1 return counter end,
344  omega = function () counter = counter + 10 return 0 end,
345  count = function () return counter end,
346  reset = function () return counter == 0 end
347
348  --- out chomp
349  omega and/or alpha are not true
350  count: 21
351
352
353
354  === TEST 24:
355  --- tt2
356  [% IF alpha OR omega %]
357  alpha and/or omega are true
358  [% ELSE %]
359  neither alpha nor omega are true
360  [% END %]
361  count: [% count %]
362
363  --- init: local counter = 21
364  --- define
365  ['true'] = 'this is true',
366  alpha = function () counter = counter + 1 return counter end,
367  omega = function () counter = counter + 10 return 0 end,
368  count = function () return counter end,
369  reset = function () return counter == 0 end
370
371  --- out chomp
372  alpha and/or omega are true
373  count: 22
374
375

```

```

376
377 === TEST 25:
378 --- tt2
379 [% IF omega OR alpha %]
380 alpha and/or omega are true
381 [% ELSE %]
382 neither alpha nor omega are true
383 [% END %]
384 count: [% count %]
385
386 --- init: local counter = 22
387 --- define
388 alpha = function () counter = counter + 1 return counter end,
389 omega = function () counter = counter + 10 return 0 end,
390 count = function () return counter end,
391 --- out chomp
392 alpha and/or omega are true
393 count: 33
394
395
396
397 === TEST 26:
398 --- tt2
399 [% small = 5
400     mid   = 7
401     big   = 10
402     both  = small + big
403     less  = big - mid
404     half  = big / small
405     left  = big % mid
406     mult  = big * small
407 %]
408 both: [% both +%]
409 less: [% less +%]
410 half: [% half +%]
411 left: [% left +%]
412 mult: [% mult +%]
413 maxi: [% mult + 2 * 2 +%]
414 mega: [% mult * 2 + 2 * 3 %]
415
416 --- out chomp
417 both: 15
418 less: 3
419 half: 2
420 left: 3
421 mult: 50
422 maxi: 54
423 mega: 106
424

```

[One Level Up](#)

[Top Level](#)

t/block.t - lemlate

```
1 # vim:set ft= ts=4 sw=4 et fdm=marker:
2
3 use t::TestLemlate;
4
5 plan tests => 1 * blocks();
6
7 $ENV{LEMLATE_POST_CHOMP} = 1;
8
9 run_tests;
10
11 __DATA__
12
13 === TEST 1: line 1
14 --- tt2
15 [% BLOCK block1 %]
16 This is the original block1
17 [% END %]
18 [% INCLUDE block1 %]
19 [% INCLUDE blockdef %]
20 [% INCLUDE block1 %]
21
22 --- out
23 This is the original block1
24 start of blockdef
25 end of blockdef
26 This is the original block1
27 --- LAST
28
29
30
31 === TEST 2: line 60
32 --- tt2
33 [% BLOCK block1 %]
34 This is the original block1
35 [% END %]
36 [% INCLUDE block1 %]
37 [% PROCESS blockdef %]
38 [% INCLUDE block1 %]
39
40 --- out
41 This is the original block1
42 start of blockdef
43 end of blockdef
44 This is block 1, defined in blockdef, a is alpha
45
46
47
48 === TEST 3: line 74
49 --- tt2
50 [% INCLUDE block_a +%]
51 [% INCLUDE block_b %]
52
53 --- out
54 this is block a
55 this is block b
56
57
58
59 === TEST 4: line 81
60 --- tt2
61 [% INCLUDE header
62     title = 'A New Beginning'
63     +%]
64 A long time ago in a galaxy far, far away...
65 [% PROCESS footer %]
66
67 --- out
68 <html><head><title>A New Beginning</title></head><body>
69 A long time ago in a galaxy far, far away...
70 </body></html>
71
```

```
72
73
74 === TEST 5: line 93
75 --- tt2
76 [% BLOCK foo:bar %]
77 blah
78 [% END %]
79 [% PROCESS foo:bar %]
80
81 --- out
82 blah
83
84
85
86 === TEST 6: line 101
87 --- tt2
88 [% BLOCK 'hello html' -%]
89 Hello World!
90 [% END -%]
91 [% PROCESS 'hello html' %]
92
93 --- out
94 Hello World!
95
```

[One Level Up](#)

[Top Level](#)

t/blocks.t - lempate

```
1 # vim:set ft= ts=4 sw=4 et fdm=marker:
2
3 use t::TestLemplate;
4
5 plan tests => 1 * blocks();
6
7 $ENV{LEMPATE_POST_CHOMP} = 1;
8
9 run_tests;
10
11 __DATA__
12
13 === TEST 1: line 1
14 --- tt2
15 [% INCLUDE blockdef/block1 %]
16
17 --- lua_err
18 file error - blockdef/block1: not found
19 --- LAST
20
21
22
23 === TEST 2: line 61
24 --- tt2
25 [% INCLUDE blockdef/block1 %]
26
27 --- out
28 This is block 1, defined in blockdef, a is alpha
29
30
31
32 === TEST 3: line 68
33 --- tt2
34 [% INCLUDE blockdef/block1 a='amazing' %]
35
36 --- out
37 This is block 1, defined in blockdef, a is amazing
38
39
40
41 === TEST 4: line 74
42 --- tt2
43 [% TRY; INCLUDE blockdef/none; CATCH; error; END %]
44
45 --- out
46 file error - blockdef/none: not found
47
48
49
50 === TEST 5: line 79
51 --- tt2
52 [% INCLUDE "$dir/blockdef/block1" a='abstract' %]
53
54 --- out
55 This is block 1, defined in blockdef, a is abstract
56
```

t/iterator.t - lemlate

```
1 # vim:set ft= ts=4 sw=4 et fdm=marker:
2
3 use t::TestLemlate;
4
5 plan tests => 1 * blocks();
6
7 $ENV{LEMLATE_POST_CHOMP} = 1;
8
9 run_tests;
10
11 __DATA__
12
13 === TEST 1: line 1
14 --- tt2
15 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
16 [% FOREACH i = items %]
17     * [% i +%]
18 [% END %]
19
20 --- out
21     * foo
22     * bar
23     * baz
24     * qux
25
26
27
28 === TEST 2: line 99
29 --- tt2
30 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
31 [% FOREACH i = items %]
32     #[% loop.index %]/[% loop.max %] [% i +%]
33 [% END %]
34
35 --- out
36     #0/3 foo
37     #1/3 bar
38     #2/3 baz
39     #3/3 qux
40
41
42
43 === TEST 3: line 110
44 --- tt2
45 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
46 [% FOREACH i = items %]
47     #[% loop.count %]/[% loop.size %] [% i +%]
48 [% END %]
49
50 --- out
51     #1/4 foo
52     #2/4 bar
53     #3/4 baz
54     #4/4 qux
55
56
57
58 === TEST 4: line 121
59 --- SKIP
60 --- tt2
61 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
62 [% FOREACH i = items %]
63     #[% loop.number %]/[% loop.size %] [% i +%]
64 [% END %]
65
66 --- out
67     #1/4 foo
68     #2/4 bar
69     #3/4 baz
70     #4/4 qux
71
```

```

72
73
74 === TEST 5: line 134
75 --- tt2
76 [% USE iterator(data) %]
77 [% FOREACH i = iterator %]
78 [% IF iterator.first %]
79 List of items:
80 [% END %]
81     * [% i +%]
82 [% IF iterator.last %]
83 End of list
84 [% END %]
85 [% END %]
86
87 --- define
88 data = {'foo', 'bar', 'baz', 'qux', 'wiz', 'woz', 'waz'}
89 --- out
90 List of items:
91     * foo
92     * bar
93     * baz
94     * qux
95     * wiz
96     * woz
97     * waz
98 End of list
99
100
101
102 === TEST 6: line 157
103 --- tt2
104 [% FOREACH i = [ 'foo' 'bar' 'baz' 'qux' ] %]
105 [% "$loop.prev<-" IF loop.prev -%][[% i -%]][% "->$loop.next" IF loop.next +%]
106 [% END %]
107
108 --- out
109 [foo]->bar
110 foo<-[bar]->baz
111 bar<-[baz]->qux
112 baz<-[qux]
113

```

[One Level Up](#)

[Top Level](#)

[One Level Up](#)

[Top Level](#)

t/pod.t - lempate

```
1  use Test::More;
2
3  eval "use Test::Pod";
4  plan skip_all => "Test::Pod required for testing POD" if $@;
5  all_pod_files_ok();
```

[One Level Up](#)

[Top Level](#)

t/sanity.t - lempate

```
1 # vim:set ft= ts=4 sw=4 et fdm=marker:
2
3 use t::TestLemplate;
4
5 plan tests => 1 * blocks();
6
7 run_tests;
8
9 __DATA__
10
11 === TEST 1: simple varaible interpolation
12 --- tt2
13 Hello, [% world %]!
14 --- define: world = "Lua"
15 --- out
16 Hello, Lua!
```