

[51ce10f4da](#) ▾

...

[html-minifier](#) / [src](#) / [htmlminifier.js](#) / [Jump to](#) ▾

alexlams handle custom fragments within CSS/JS correctly (#1015) ...

[History](#)[48 contributors](#)

+25

1344 lines (1240 sloc) | 44.8 KB

...

```
1  'use strict';
2
3  var CleanCSS = require('clean-css');
4  var decode = require('he').decode;
5  var HTMLParser = require('./htmlparser').HTMLParser;
6  var RelateUrl = require('relateurl');
7  var TokenChain = require('./tokenchain');
8  var UglifyJS = require('uglify-js');
9  var utils = require('./utils');
10
11 function trimWhitespace(str) {
12   return str && str.replace(/^[ \n\r\t\f]+/, '').replace(/[ \n\r\t\f]+$/, '');
13 }
14
15 function collapseWhitespaceAll(str) {
16   // Non-breaking space is specifically handled inside the replacer function here:
17   return str && str.replace(/[ \n\r\t\f\xA0]+/g, function(spaces) {
18     return spaces === '\t' ? '\t' : spaces.replace(/(^|\xA0+)[\xA0]+/g, '$1 ');
19   });
20 }
21
22 function collapseWhitespace(str, options, trimLeft, trimRight, collapseAll) {
23   var lineBreakBefore = '', lineBreakAfter = '';
24
25   if (options.preserveLineBreaks) {
26     str = str.replace(/^[ \n\r\t\f]*?[\n\r][ \n\r\t\f]*$/, function() {
27       lineBreakBefore = '\n';
28       return '';
29     }).replace(/[ \n\r\t\f]*?[\n\r][ \n\r\t\f]*$/, function() {
```

```

30     lineBreakAfter = '\n';
31     return '';
32 });
33 }
34
35 if (trimLeft) {
36     // Non-breaking space is specifically handled inside the replacer function here:
37     str = str.replace(/^[ \n\r\t\f\xA0]+/, function(spaces) {
38         var conservative = !lineBreakBefore && options.conservativeCollapse;
39         if (conservative && spaces === '\t') {
40             return '\t';
41         }
42         return spaces.replace(/^[^\xA0]+/, '').replace(/(\xA0+)[^\xA0]+/g, '$1 ') || (conservative ?
43     });
44 }
45
46 if (trimRight) {
47     // Non-breaking space is specifically handled inside the replacer function here:
48     str = str.replace(/[ \n\r\t\f\xA0]+$/, function(spaces) {
49         var conservative = !lineBreakAfter && options.conservativeCollapse;
50         if (conservative && spaces === '\t') {
51             return '\t';
52         }
53         return spaces.replace(/^[^\xA0]+(\xA0+)/g, ' $1').replace(/^[^\xA0]+$/, '') || (conservative ?
54     });
55 }
56
57 if (collapseAll) {
58     // strip non space whitespace then compress spaces to one
59     str = collapseWhitespaceAll(str);
60 }
61
62 return lineBreakBefore + str + lineBreakAfter;
63 }
64
65 var createMapFromString = utils.createMapFromString;
66 // non-empty tags that will maintain whitespace around them
67 var inlineTags = createMapFromString('a,abbr,acronym,b,bdi,bdo,big,button,cite,code,del,dfn,em,font,
68 // non-empty tags that will maintain whitespace within them
69 var inlineTextTags = createMapFromString('a,abbr,acronym,b,big,del,em,font,i,ins,kbd,mark,nobr,rp,
70 // self-closing tags that will maintain whitespace around them
71 var selfClosingInlineTags = createMapFromString('comment,img,input,wbr');
72
73 function collapseWhitespaceSmart(str, prevTag, nextTag, options) {
74     var trimLeft = prevTag && !selfClosingInlineTags(prevTag);
75     if (trimLeft && !options.collapseInlineTagWhitespace) {
76         trimLeft = prevTag.charAt(0) === '/' ? !inlineTags(prevTag.slice(1)) : !inlineTextTags(prevTag
77     }
78     var trimRight = nextTag && !selfClosingInlineTags(nextTag);

```

```

79     if (trimRight && !options.collapseInlineTagWhitespace) {
80         trimRight = nextTag.charAt(0) === '/' ? !inlineTextTags(nextTag.slice(1)) : !inlineTags(nextTa
81     }
82     return collapseWhitespace(str, options, trimLeft, trimRight, prevTag && nextTag);
83 }
84
85 function isConditionalComment(text) {
86     return /^\[if\s[^\]]+\]\[\endif\]$/.test(text);
87 }
88
89 function isIgnoredComment(text, options) {
90     for (var i = 0, len = options.ignoreCustomComments.length; i < len; i++) {
91         if (options.ignoreCustomComments[i].test(text)) {
92             return true;
93         }
94     }
95     return false;
96 }
97
98 function isEventAttribute(attrName, options) {
99     var patterns = options.customEventAttributes;
100     if (patterns) {
101         for (var i = patterns.length; i--;) {
102             if (patterns[i].test(attrName)) {
103                 return true;
104             }
105         }
106         return false;
107     }
108     return /^on[a-z]{3,}$/ .test(attrName);
109 }
110
111 function canRemoveAttributeQuotes(value) {
112     // https://mathiasbynens.be/notes/unquoted-attribute-values
113     return /^[^ \t\n\f\r"'`=<>]+$/.test(value);
114 }
115
116 function attributesInclude(attributes, attribute) {
117     for (var i = attributes.length; i--;) {
118         if (attributes[i].name.toLowerCase() === attribute) {
119             return true;
120         }
121     }
122     return false;
123 }
124
125 function isAttributeRedundant(tag, attrName, attrValue, attrs) {
126     attrValue = attrValue ? trimWhitespace(attrValue.toLowerCase()) : '';
127

```

```

128     return (
129         tag === 'script' &&
130         attrName === 'language' &&
131         attrValue === 'javascript' ||
132
133         tag === 'form' &&
134         attrName === 'method' &&
135         attrValue === 'get' ||
136
137         tag === 'input' &&
138         attrName === 'type' &&
139         attrValue === 'text' ||
140
141         tag === 'script' &&
142         attrName === 'charset' &&
143         !attributesInclude(attrs, 'src') ||
144
145         tag === 'a' &&
146         attrName === 'name' &&
147         attributesInclude(attrs, 'id') ||
148
149         tag === 'area' &&
150         attrName === 'shape' &&
151         attrValue === 'rect'
152     );
153 }
154
155 // https://mathiasbynens.be/demo/javascript-mime-type
156 // https://developer.mozilla.org/en/docs/Web/HTML/Element/script#attr-type
157 var executableScriptsMimetypes = utils.createMap([
158     'text/javascript',
159     'text/ecmascript',
160     'text/jscript',
161     'application/javascript',
162     'application/x-javascript',
163     'application/ecmascript'
164 ]);
165
166 function isScriptTypeAttribute(attrValue) {
167     attrValue = trimWhitespace(attrValue.split(/;/, 2)[0]).toLowerCase();
168     return attrValue === '' || executableScriptsMimetypes(attrValue);
169 }
170
171 function isExecutableScript(tag, attrs) {
172     if (tag !== 'script') {
173         return false;
174     }
175     for (var i = 0, len = attrs.length; i < len; i++) {
176         var attrName = attrs[i].name.toLowerCase();

```

```

177     if (attrName === 'type') {
178         return isScriptTypeAttribute(attrs[i].value);
179     }
180 }
181 return true;
182 }
183
184 function isStyleLinkTypeAttribute(attrValue) {
185     attrValue = trimWhitespace(attrValue).toLowerCase();
186     return attrValue === '' || attrValue === 'text/css';
187 }
188
189 function isStyleSheet(tag, attrs) {
190     if (tag !== 'style') {
191         return false;
192     }
193     for (var i = 0, len = attrs.length; i < len; i++) {
194         var attrName = attrs[i].name.toLowerCase();
195         if (attrName === 'type') {
196             return isStyleLinkTypeAttribute(attrs[i].value);
197         }
198     }
199     return true;
200 }
201
202 var isSimpleBoolean = createMapFromString('allowfullscreen,async,autofocus,autoplay,checked,compact');
203 var isBooleanValue = createMapFromString('true,false');
204
205 function isBooleanAttribute(attrName, attrValue) {
206     return isSimpleBoolean(attrName) || attrName === 'draggable' && !isBooleanValue(attrValue);
207 }
208
209 function isUriTypeAttribute(attrName, tag) {
210     return (
211         /^(?:a|area|link|base)$/.test(tag) && attrName === 'href' ||
212         tag === 'img' && /^(?:src|longdesc|usemap)$/.test(attrName) ||
213         tag === 'object' && /^(?:classid|codebase|data|usemap)$/.test(attrName) ||
214         tag === 'q' && attrName === 'cite' ||
215         tag === 'blockquote' && attrName === 'cite' ||
216         (tag === 'ins' || tag === 'del') && attrName === 'cite' ||
217         tag === 'form' && attrName === 'action' ||
218         tag === 'input' && (attrName === 'src' || attrName === 'usemap') ||
219         tag === 'head' && attrName === 'profile' ||
220         tag === 'script' && (attrName === 'src' || attrName === 'for')
221     );
222 }
223
224 function isNumberTypeAttribute(attrName, tag) {
225     return (

```

```

226     /^(?:a|area|object|button)$/.test(tag) && attrName === 'tabindex' ||
227     tag === 'input' && (attrName === 'maxlength' || attrName === 'tabindex') ||
228     tag === 'select' && (attrName === 'size' || attrName === 'tabindex') ||
229     tag === 'textarea' && /^(?:rows|cols|tabindex)$/.test(attrName) ||
230     tag === 'colgroup' && attrName === 'span' ||
231     tag === 'col' && attrName === 'span' ||
232     (tag === 'th' || tag === 'td') && (attrName === 'rowspan' || attrName === 'colspan')
233 );
234 }
235
236 function isLinkType(tag, attrs, value) {
237     if (tag !== 'link') {
238         return false;
239     }
240     for (var i = 0, len = attrs.length; i < len; i++) {
241         if (attrs[i].name === 'rel' && attrs[i].value === value) {
242             return true;
243         }
244     }
245 }
246
247 function isMediaQuery(tag, attrs, attrName) {
248     return attrName === 'media' && (isLinkType(tag, attrs, 'stylesheet') || isStyleSheet(tag, attrs))
249 }
250
251 var srcsetTags = createMapFromString('img,source');
252
253 function isSrcset(attrName, tag) {
254     return attrName === 'srcset' && srcsetTags(tag);
255 }
256
257 function cleanAttributeValue(tag, attrName, attrValue, options, attrs) {
258     if (isEventAttribute(attrName, options)) {
259         attrValue = trimWhitespace(attrValue).replace(/^javascript:\s*/i, '');
260         return options.minifyJS(attrValue, true);
261     }
262     else if (attrName === 'class') {
263         attrValue = trimWhitespace(attrValue);
264         if (options.sortClassName) {
265             attrValue = options.sortClassName(attrValue);
266         }
267         else {
268             attrValue = collapseWhitespaceAll(attrValue);
269         }
270         return attrValue;
271     }
272     else if (isUriTypeAttribute(attrName, tag)) {
273         attrValue = trimWhitespace(attrValue);
274         return isLinkType(tag, attrs, 'canonical') ? attrValue : options.minifyURLs(attrValue);

```

```

275     }
276     else if (isNumberTypeAttribute(attrName, tag)) {
277         return trimWhitespace(attrValue);
278     }
279     else if (attrName === 'style') {
280         attrValue = trimWhitespace(attrValue);
281         if (attrValue) {
282             if (/;$/.test(attrValue) && !/&#?[0-9a-zA-Z]+;$/.test(attrValue)) {
283                 attrValue = attrValue.replace(/\s*$/, '');
284             }
285             attrValue = options.minifyCSS(attrValue, 'inline');
286         }
287         return attrValue;
288     }
289     else if (isSrcset(attrName, tag)) {
290         // https://html.spec.whatwg.org/multipage/embedded-content.html#attr-img-srcset
291         attrValue = trimWhitespace(attrValue).split(/\s+,\s*|\s*\s+/).map(function(candidate) {
292             var url = candidate;
293             var descriptor = '';
294             var match = candidate.match(/\s+([1-9][0-9]*w|[0-9]+(?:\.[0-9]+)?x)$/);
295             if (match) {
296                 url = url.slice(0, -match[0].length);
297                 var num = +match[1].slice(0, -1);
298                 var suffix = match[1].slice(-1);
299                 if (num !== 1 || suffix !== 'x') {
300                     descriptor = ' ' + num + suffix;
301                 }
302             }
303             return options.minifyURLs(url) + descriptor;
304         }).join(', ');
305     }
306     else if (isMetaViewport(tag, attrs) && attrName === 'content') {
307         attrValue = attrValue.replace(/\s+/g, '').replace(/[0-9]+\.[0-9]+/g, function(numString) {
308             // "0.90000" -> "0.9"
309             // "1.0" -> "1"
310             // "1.0001" -> "1.0001" (unchanged)
311             return (+numString).toString();
312         });
313     }
314     else if (isContentSecurityPolicy(tag, attrs) && attrName.toLowerCase() === 'content') {
315         return collapseWhitespaceAll(attrValue);
316     }
317     else if (options.customAttrCollapse && options.customAttrCollapse.test(attrName)) {
318         attrValue = attrValue.replace(/\n+|\r+|\s{2,}/g, '');
319     }
320     else if (tag === 'script' && attrName === 'type') {
321         attrValue = trimWhitespace(attrValue.replace(/\s*\s*/g, ''));
322     }
323     else if (isMediaQuery(tag, attrs, attrName)) {

```

```
324     attrValue = trimWhitespace(attrValue);
325     return options.minifyCSS(attrValue, 'media');
326 }
327 return attrValue;
328 }
329
330 function isMetaViewport(tag, attrs) {
331     if (tag !== 'meta') {
332         return false;
333     }
334     for (var i = 0, len = attrs.length; i < len; i++) {
335         if (attrs[i].name === 'name' && attrs[i].value === 'viewport') {
336             return true;
337         }
338     }
339 }
340
341 function isContentSecurityPolicy(tag, attrs) {
342     if (tag !== 'meta') {
343         return false;
344     }
345     for (var i = 0, len = attrs.length; i < len; i++) {
346         if (attrs[i].name.toLowerCase() === 'http-equiv' && attrs[i].value.toLowerCase() === 'content-
347             return true;
348         }
349     }
350 }
351
352 function ignoreCSS(id) {
353     return '/* clean-css ignore:start */' + id + '/* clean-css ignore:end */';
354 }
355
356 // Wrap CSS declarations for CleanCSS > 3.x
357 // See https://github.com/jakubpawlowicz/clean-css/issues/418
358 function wrapCSS(text, type) {
359     switch (type) {
360         case 'inline':
361             return '*{' + text + '}';
362         case 'media':
363             return '@media ' + text + '{a{top:0}}';
364         default:
365             return text;
366     }
367 }
368
369 function unwrapCSS(text, type) {
370     var matches;
371     switch (type) {
372         case 'inline':
```



```

373     matches = text.match(/^*\{([\s\S]*)\}$/);
374     break;
375     case 'media':
376         matches = text.match(/^@media ([\s\S]*?)\s*{([\s\S]*)}$/);
377         break;
378     }
379     return matches ? matches[1] : text;
380 }
381
382 function cleanConditionalComment(comment, options) {
383     return options.processConditionalComments ? comment.replace(/^(\[if\s[^\]]+\>)([\s\S]*?)(<!\[end
384         return prefix + minify(text, options, true) + suffix;
385     }) : comment;
386 }
387
388 function processScript(text, options, currentAttrs) {
389     for (var i = 0, len = currentAttrs.length; i < len; i++) {
390         if (currentAttrs[i].name.toLowerCase() === 'type' &&
391             options.processScripts.indexOf(currentAttrs[i].value) > -1) {
392             return minify(text, options);
393         }
394     }
395     return text;
396 }
397
398 // Tag omission rules from https://html.spec.whatwg.org/multipage/syntax.html#optional-tags
399 // with the following deviations:
400 // - retain <body> if followed by <noscript>
401 // - </rb>, </rt>, </rtc>, </rp> & </tfoot> follow https://www.w3.org/TR/html5/syntax.html#optiona
402 // - retain all tags which are adjacent to non-standard HTML tags
403 var optionalStartTags = createMapFromString('html,head,body,colgroup,tbody');
404 var optionalEndTags = createMapFromString('html,head,body,li,dt,dd,p,rb,rt,rtc,rp,optgroup,option,
405 var headerTags = createMapFromString('meta,link,script,style,template,noscript');
406 var descriptionTags = createMapFromString('dt,dd');
407 var pBlockTags = createMapFromString('address,article,aside,blockquote,details,div,dl,fieldset,fig
408 var pInlineTags = createMapFromString('a,audio,del,ins,map,noscript,video');
409 var rubyTags = createMapFromString('rb,rt,rtc,rp');
410 var rtcTag = createMapFromString('rb,rtc,rp');
411 var optionTag = createMapFromString('option,optgroup');
412 var tableContentTags = createMapFromString('tbody,tfoot');
413 var tableSectionTags = createMapFromString('thead,tbody,tfoot');
414 var cellTags = createMapFromString('td,th');
415 var topLevelTags = createMapFromString('html,head,body');
416 var compactTags = createMapFromString('html,body');
417 var looseTags = createMapFromString('head,colgroup,caption');
418 var trailingTags = createMapFromString('dt,thead');
419 var htmlTags = createMapFromString('a,abbr,acronym,address,applet,area,article,aside,audio,b,base,
420
421 function canRemoveParentTag(optionalStartTag, tag) {

```

```
422     switch (optionalStartTag) {
423         case 'html':
424         case 'head':
425             return true;
426         case 'body':
427             return !headerTags(tag);
428         case 'colgroup':
429             return tag === 'col';
430         case 'tbody':
431             return tag === 'tr';
432     }
433     return false;
434 }
435
436 function isStartTagMandatory(optionalEndTag, tag) {
437     switch (tag) {
438         case 'colgroup':
439             return optionalEndTag === 'colgroup';
440         case 'tbody':
441             return tableSectionTags(optionalEndTag);
442     }
443     return false;
444 }
445
446 function canRemovePrecedingTag(optionalEndTag, tag) {
447     switch (optionalEndTag) {
448         case 'html':
449         case 'head':
450         case 'body':
451         case 'colgroup':
452         case 'caption':
453             return true;
454         case 'li':
455         case 'optgroup':
456         case 'tr':
457             return tag === optionalEndTag;
458         case 'dt':
459         case 'dd':
460             return descriptionTags(tag);
461         case 'p':
462             return pBlockTags(tag);
463         case 'rb':
464         case 'rt':
465         case 'rp':
466             return rubyTags(tag);
467         case 'rtc':
468             return rtcTag(tag);
469         case 'option':
470             return optionTag(tag);
```

```
471     case 'thead':
472     case 'tbody':
473         return tableContentTags(tag);
474     case 'tfoot':
475         return tag === 'tbody';
476     case 'td':
477     case 'th':
478         return cellTags(tag);
479 }
480 return false;
481 }
482
483 var reEmptyAttribute = new RegExp(
484     '^(?:class|id|style|title|lang|dir|on(?:focus|blur|change|click|dblclick|mouse(' +
485     '?:down|up|over|move|out)|key(?:press|down|up)))$');
486
487 function canDeleteEmptyAttribute(tag, attrName, attrValue, options) {
488     var isEmptyValue = !attrValue || /^s*$/i.test(attrValue);
489     if (!isEmptyValue) {
490         return false;
491     }
492     if (typeof options.removeEmptyAttributes === 'function') {
493         return options.removeEmptyAttributes(attrName, tag);
494     }
495     return tag === 'input' && attrName === 'value' || reEmptyAttribute.test(attrName);
496 }
497
498 function hasAttrName(name, attrs) {
499     for (var i = attrs.length - 1; i >= 0; i--) {
500         if (attrs[i].name === name) {
501             return true;
502         }
503     }
504     return false;
505 }
506
507 function canRemoveElement(tag, attrs) {
508     switch (tag) {
509         case 'textarea':
510             return false;
511         case 'audio':
512         case 'script':
513         case 'video':
514             if (hasAttrName('src', attrs)) {
515                 return false;
516             }
517             break;
518         case 'iframe':
519             if (hasAttrName('src', attrs) || hasAttrName('srcdoc', attrs)) {
```

```

520         return false;
521     }
522     break;
523     case 'object':
524         if (hasAttrName('data', attrs)) {
525             return false;
526         }
527         break;
528     case 'applet':
529         if (hasAttrName('code', attrs)) {
530             return false;
531         }
532         break;
533 }
534 return true;
535 }
536
537 function canCollapseWhitespace(tag) {
538     return !/^(?:script|style|pre|textarea)$/i.test(tag);
539 }
540
541 function canTrimWhitespace(tag) {
542     return !/^(?:pre|textarea)$/i.test(tag);
543 }
544
545 function normalizeAttr(attr, attrs, tag, options) {
546     var attrName = options.name(attr.name),
547         attrValue = attr.value;
548
549     if (options.decodeEntities && attrValue) {
550         attrValue = decode(attrValue, { isAttributeValue: true });
551     }
552
553     if (options.removeRedundantAttributes &&
554         isAttributeRedundant(tag, attrName, attrValue, attrs) ||
555         options.removeScriptTypeAttributes && tag === 'script' &&
556         attrName === 'type' && isScriptTypeAttribute(attrValue) ||
557         options.removeStyleLinkTypeAttributes && (tag === 'style' || tag === 'link') &&
558         attrName === 'type' && isStyleLinkTypeAttribute(attrValue)) {
559         return;
560     }
561
562     if (attrValue) {
563         attrValue = cleanAttributeValue(tag, attrName, attrValue, options, attrs);
564     }
565
566     if (options.removeEmptyAttributes &&
567         canDeleteEmptyAttribute(tag, attrName, attrValue, options)) {
568         return;

```

```

569     }
570
571     if (options.decodeEntities && attrValue) {
572         attrValue = attrValue.replace(/(&#?[0-9a-zA-Z]+);/g, '&amp;$1');
573     }
574
575     return {
576         attr: attr,
577         name: attrName,
578         value: attrValue
579     };
580 }
581
582 function buildAttr(normalized, hasUnarySlash, options, isLast, uidAttr) {
583     var attrName = normalized.name,
584         attrValue = normalized.value,
585         attr = normalized.attr,
586         attrQuote = attr.quote,
587         attrFragment,
588         emittedAttrValue;
589
590     if (typeof attrValue !== 'undefined' && (!options.removeAttributeQuotes ||
591         ~attrValue.indexOf(uidAttr) || !canRemoveAttributeQuotes(attrValue))) {
592         if (!options.preventAttributesEscaping) {
593             if (typeof options.quoteCharacter === 'undefined') {
594                 var apos = (attrValue.match(/'/g) || []).length;
595                 var quot = (attrValue.match(/"/g) || []).length;
596                 attrQuote = apos < quot ? '\'' : '""';
597             }
598             else {
599                 attrQuote = options.quoteCharacter === '\'' ? '\'' : '""';
600             }
601             if (attrQuote === '""') {
602                 attrValue = attrValue.replace(/"/g, '&#34;');
603             }
604             else {
605                 attrValue = attrValue.replace(/'/g, '&#39;');
606             }
607         }
608         emittedAttrValue = attrQuote + attrValue + attrQuote;
609         if (!isLast && !options.removeTagWhitespace) {
610             emittedAttrValue += ' ';
611         }
612     }
613     // make sure trailing slash is not interpreted as HTML self-closing tag
614     else if (isLast && !hasUnarySlash && !/\$/g.test(attrValue)) {
615         emittedAttrValue = attrValue;
616     }
617     else {

```

```

618     emittedAttrValue = attrValue + ' ';
619 }
620
621 if (typeof attrValue === 'undefined' || options.collapseBooleanAttributes &&
622     isBooleanAttribute(attrName.toLowerCase(), attrValue.toLowerCase())) {
623     attrFragment = attrName;
624     if (!isLast) {
625         attrFragment += ' ';
626     }
627 }
628 else {
629     attrFragment = attrName + attr.customAssign + emittedAttrValue;
630 }
631
632 return attr.customOpen + attrFragment + attr.customClose;
633 }
634
635 function identity(value) {
636     return value;
637 }
638
639 function processOptions(values) {
640     var options = {
641         name: function(name) {
642             return name.toLowerCase();
643         },
644         canCollapseWhitespace: canCollapseWhitespace,
645         canTrimWhitespace: canTrimWhitespace,
646         html5: true,
647         ignoreCustomComments: [/^!/,
648         ignoreCustomFragments: [
649             /<%(\\s\\S)*?%>/,
650             /<\\?(\\s\\S)*?\\?>/
651         ],
652         includeAutoGeneratedTags: true,
653         log: identity,
654         minifyCSS: identity,
655         minifyJS: identity,
656         minifyURLs: identity
657     };
658     Object.keys(values).forEach(function(key) {
659         var value = values[key];
660         if (key === 'caseSensitive') {
661             if (value) {
662                 options.name = identity;
663             }
664         }
665         else if (key === 'log') {
666             if (typeof value === 'function') {

```

```

667     options.log = value;
668 }
669 }
670 else if (key === 'minifyCSS' && typeof value !== 'function') {
671     if (!value) {
672         return;
673     }
674     if (typeof value !== 'object') {
675         value = {};
676     }
677     options.minifyCSS = function(text, type) {
678         text = text.replace(/(url\s*(\s*)("'|")(.*)\2(\s*\s*))/ig, function(match, prefix, quote,
679             return prefix + quote + options.minifyURLs(url) + quote + suffix;
680         ));
681         var cleanCssOutput = new CleanCSS(value).minify(wrapCSS(text, type));
682         if (cleanCssOutput.errors.length > 0) {
683             cleanCssOutput.errors.forEach(options.log);
684             return text;
685         }
686         return unwrapCSS(cleanCssOutput.styles, type);
687     };
688 }
689 else if (key === 'minifyJS' && typeof value !== 'function') {
690     if (!value) {
691         return;
692     }
693     if (typeof value !== 'object') {
694         value = {};
695     }
696     (value.parse || (value.parse = {})).bare_returns = false;
697     options.minifyJS = function(text, inline) {
698         var start = text.match(/^s*<!--.*\/);
699         var code = start ? text.slice(start[0].length).replace(/\n\s*-->\s*$/, '') : text;
700         value.parse.bare_returns = inline;
701         var result = UglifyJS.minify(code, value);
702         if (result.error) {
703             options.log(result.error);
704             return text;
705         }
706         return result.code.replace(/;$/, '');
707     };
708 }
709 else if (key === 'minifyURLs' && typeof value !== 'function') {
710     if (!value) {
711         return;
712     }
713     if (typeof value === 'string') {
714         value = { site: value };
715     }

```

```

716     else if (typeof value !== 'object') {
717         value = {};
718     }
719     options.minifyURLs = function(text) {
720         try {
721             return RelateUrl.relate(text, value);
722         }
723         catch (err) {
724             options.log(err);
725             return text;
726         }
727     };
728 }
729 else {
730     options[key] = value;
731 }
732 });
733 return options;
734 }
735
736 function uniqueId(value) {
737     var id;
738     do {
739         id = Math.random().toString(36).replace(/^0\.[0-9]*/, '');
740     } while (~value.indexOf(id));
741     return id;
742 }
743
744 var specialContentTags = createMapFromString('script,style');
745
746 function createSortFns(value, options, uidIgnore, uidAttr) {
747     var attrChains = options.sortAttributes && Object.create(null);
748     var classChain = options.sortClassName && new TokenChain();
749
750     function attrNames(attrs) {
751         return attrs.map(function(attr) {
752             return options.name(attr.name);
753         });
754     }
755
756     function shouldSkipUID(token, uid) {
757         return !uid || token.indexOf(uid) === -1;
758     }
759
760     function shouldSkipUIDs(token) {
761         return shouldSkipUID(token, uidIgnore) && shouldSkipUID(token, uidAttr);
762     }
763
764     function scan(input) {

```



```

765     var currentTag, currentType;
766     new HTMLParser(input, {
767         start: function(tag, attrs) {
768             if (attrChains) {
769                 if (!attrChains[tag]) {
770                     attrChains[tag] = new TokenChain();
771                 }
772                 attrChains[tag].add(attrNames(attrs).filter(shouldSkipUIDs));
773             }
774             for (var i = 0, len = attrs.length; i < len; i++) {
775                 var attr = attrs[i];
776                 if (classChain && attr.value && options.name(attr.name) === 'class') {
777                     classChain.add(trimWhitespace(attr.value).split(/[ \t\n\f\r]+/).filter(shouldSkipUIDs))
778                 }
779                 else if (options.processScripts && attr.name.toLowerCase() === 'type') {
780                     currentTag = tag;
781                     currentType = attr.value;
782                 }
783             }
784         },
785         end: function() {
786             currentTag = '';
787         },
788         chars: function(text) {
789             if (options.processScripts && specialContentTags(currentTag) &&
790                 options.processScripts.indexOf(currentType) > -1) {
791                 scan(text);
792             }
793         }
794     });
795 }
796
797 var log = options.log;
798 options.log = identity;
799 options.sortAttributes = false;
800 options.sortClassName = false;
801 scan(minify(value, options));
802 options.log = log;
803 if (attrChains) {
804     var attrSorters = Object.create(null);
805     for (var tag in attrChains) {
806         attrSorters[tag] = attrChains[tag].createSorter();
807     }
808     options.sortAttributes = function(tag, attrs) {
809         var sorter = attrSorters[tag];
810         if (sorter) {
811             var attrMap = Object.create(null);
812             var names = attrNames(attrs);
813             names.forEach(function(name, index) {

```

```

814         (attrMap[name] || (attrMap[name] = [])).push(attrs[index]);
815     });
816     sorter.sort(names).forEach(function(name, index) {
817         attrs[index] = attrMap[name].shift();
818     });
819     }
820 };
821 }
822 if (classChain) {
823     var sorter = classChain.createSorter();
824     options.sortClassName = function(value) {
825         return sorter.sort(value.split(/[ \n\f\r]+/)).join(' ');
826     };
827 }
828 }
829
830 function minify(value, options, partialMarkup) {
831     if (options.collapseWhitespace) {
832         value = collapseWhitespace(value, options, true, true);
833     }
834
835     var buffer = [],
836         charsPrevTag,
837         currentChars = '',
838         hasChars,
839         currentTag = '',
840         currentAttrs = [],
841         stackNoTrimWhitespace = [],
842         stackNoCollapseWhitespace = [],
843         optionalStartTag = '',
844         optionalEndTag = '',
845         ignoredMarkupChunks = [],
846         ignoredCustomMarkupChunks = [],
847         uidIgnore,
848         uidAttr,
849         uidPattern;
850
851     // temporarily replace ignored chunks with comments,
852     // so that we don't have to worry what's there.
853     // for all we care there might be
854     // completely-horribly-broken-alien-non-html-emoji-cthulhu-filled content
855     value = value.replace(/<!-- htmlmin:ignore -->([\s\S]*?)<!-- htmlmin:ignore -->/g, function(matc
856         if (!uidIgnore) {
857             uidIgnore = uniqueId(value);
858             var pattern = new RegExp('^' + uidIgnore + '([0-9]+)$');
859             if (options.ignoreCustomComments) {
860                 options.ignoreCustomComments = options.ignoreCustomComments.slice();
861             }
862             else {

```

```

863     options.ignoreCustomComments = [];
864 }
865 options.ignoreCustomComments.push(pattern);
866 }
867 var token = '<!--' + uidIgnore + ignoredMarkupChunks.length + '-->';
868 ignoredMarkupChunks.push(group1);
869 return token;
870 });
871
872 var customFragments = options.ignoreCustomFragments.map(function(re) {
873     return re.source;
874 });
875 if (customFragments.length) {
876     var reCustomIgnore = new RegExp('\\s*(?:' + customFragments.join('|') + ')+\\s*', 'g');
877     // temporarily replace custom ignored fragments with unique attributes
878     value = value.replace(reCustomIgnore, function(match) {
879         if (!uidAttr) {
880             uidAttr = uniqueId(value);
881             uidPattern = new RegExp('(\\s*)' + uidAttr + '([0-9]+)' + uidAttr + '(\\s*)', 'g');
882             if (options.minifyCSS) {
883                 options.minifyCSS = (function(fn) {
884                     return function(text, type) {
885                         text = text.replace(uidPattern, function(match, prefix, index) {
886                             var chunks = ignoredCustomMarkupChunks[+index];
887                             return chunks[1] + uidAttr + index + uidAttr + chunks[2];
888                         });
889                         var ids = [];
890                         new CleanCSS().minify(wrapCSS(text, type)).warnings.forEach(function(warning) {
891                             var match = uidPattern.exec(warning);
892                             if (match) {
893                                 var id = uidAttr + match[2] + uidAttr;
894                                 text = text.replace(id, ignoreCSS(id));
895                                 ids.push(id);
896                             }
897                         });
898                         text = fn(text, type);
899                         ids.forEach(function(id) {
900                             text = text.replace(ignoreCSS(id), id);
901                         });
902                         return text;
903                     };
904                 })(options.minifyCSS);
905             }
906             if (options.minifyJS) {
907                 options.minifyJS = (function(fn) {
908                     return function(text, type) {
909                         return fn(text.replace(uidPattern, function(match, prefix, index) {
910                             var chunks = ignoredCustomMarkupChunks[+index];
911                             return chunks[1] + uidAttr + index + uidAttr + chunks[2];

```

```

912         )), type);
913     };
914     })(options.minifyJS);
915 }
916 }
917 var token = uidAttr + ignoredCustomMarkupChunks.length + uidAttr;
918 ignoredCustomMarkupChunks.push(/^(\\s*)(\\s\\S)*?(\\s*)$/).exec(match));
919 return '\\t' + token + '\\t';
920 });
921 }
922
923 if (options.sortAttributes && typeof options.sortAttributes !== 'function' ||
924     options.sortClassName && typeof options.sortClassName !== 'function') {
925     createSortFns(value, options, uidIgnore, uidAttr);
926 }
927
928 function _canCollapseWhitespace(tag, attrs) {
929     return options.canCollapseWhitespace(tag, attrs, canCollapseWhitespace);
930 }
931
932 function _canTrimWhitespace(tag, attrs) {
933     return options.canTrimWhitespace(tag, attrs, canTrimWhitespace);
934 }
935
936 function removeStartTag() {
937     var index = buffer.length - 1;
938     while (index > 0 && !/^<[^\!]/.test(buffer[index])) {
939         index--;
940     }
941     buffer.length = Math.max(0, index);
942 }
943
944 function removeEndTag() {
945     var index = buffer.length - 1;
946     while (index > 0 && !/^<\\/.test(buffer[index])) {
947         index--;
948     }
949     buffer.length = Math.max(0, index);
950 }
951
952 // look for trailing whitespaces, bypass any inline tags
953 function trimTrailingWhitespace(index, nextTag) {
954     for (var endTag = null; index >= 0 && _canTrimWhitespace(endTag); index--) {
955         var str = buffer[index];
956         var match = str.match(/^<\\([\\w:-]+>$/);
957         if (match) {
958             endTag = match[1];
959         }
960         else if (/>$/.test(str) || (buffer[index] = collapseWhitespaceSmart(str, null, nextTag, opti

```

```

961         break;
962     }
963 }
964 }
965
966 // look for trailing whitespaces from previously processed text
967 // which may not be trimmed due to a following comment or an empty
968 // element which has now been removed
969 function squashTrailingWhitespace(nextTag) {
970     var charsIndex = buffer.length - 1;
971     if (buffer.length > 1) {
972         var item = buffer[buffer.length - 1];
973         if (/^(?:<|!|$)/.test(item) && item.indexOf(uidIgnore) === -1) {
974             charsIndex--;
975         }
976     }
977     trimTrailingWhitespace(charsIndex, nextTag);
978 }
979
980 new HTMLParser(value, {
981     partialMarkup: partialMarkup,
982     continueOnParseError: options.continueOnParseError,
983     customAttrAssign: options.customAttrAssign,
984     customAttrSurround: options.customAttrSurround,
985     html5: options.html5,
986
987     start: function(tag, attrs, unary, unarySlash, autoGenerated) {
988         if (tag.toLowerCase() === 'svg') {
989             options = Object.create(options);
990             options.caseSensitive = true;
991             options.keepClosingSlash = true;
992             options.name = identity;
993         }
994         tag = options.name(tag);
995         currentTag = tag;
996         charsPrevTag = tag;
997         if (!inlineTextTags(tag)) {
998             currentChars = '';
999         }
1000         hasChars = false;
1001         currentAttrs = attrs;
1002
1003         var optional = options.removeOptionalTags;
1004         if (optional) {
1005             var htmlTag = htmlTags(tag);
1006             // <html> may be omitted if first thing inside is not comment
1007             // <head> may be omitted if first thing inside is an element
1008             // <body> may be omitted if first thing inside is not space, comment, <meta>, <link>, <scr
1009             // <colgroup> may be omitted if first thing inside is <col>

```

```

1010 // <tbody> may be omitted if first thing inside is <tr>
1011 if (htmlTag && canRemoveParentTag(optionalStartTag, tag)) {
1012     removeStartTag();
1013 }
1014 optionalStartTag = '';
1015 // end-tag-followed-by-start-tag omission rules
1016 if (htmlTag && canRemovePrecedingTag(optionalEndTag, tag)) {
1017     removeEndTag();
1018     // <colgroup> cannot be omitted if preceding </colgroup> is omitted
1019     // <tbody> cannot be omitted if preceding </tbody>, </thead> or </tfoot> is omitted
1020     optional = !isStartTagMandatory(optionalEndTag, tag);
1021 }
1022 optionalEndTag = '';
1023 }
1024
1025 // set whitespace flags for nested tags (eg. <code> within a <pre>)
1026 if (options.collapseWhitespace) {
1027     if (!stackNoTrimWhitespace.length) {
1028         squashTrailingWhitespace(tag);
1029     }
1030     if (!unary) {
1031         if (!_canTrimWhitespace(tag, attrs) || stackNoTrimWhitespace.length) {
1032             stackNoTrimWhitespace.push(tag);
1033         }
1034         if (!_canCollapseWhitespace(tag, attrs) || stackNoCollapseWhitespace.length) {
1035             stackNoCollapseWhitespace.push(tag);
1036         }
1037     }
1038 }
1039
1040 var openTag = '<' + tag;
1041 var hasUnarySlash = unarySlash && options.keepClosingSlash;
1042
1043 buffer.push(openTag);
1044
1045 if (options.sortAttributes) {
1046     options.sortAttributes(tag, attrs);
1047 }
1048
1049 var parts = [];
1050 for (var i = attrs.length, isLast = true; --i >= 0;) {
1051     var normalized = normalizeAttr(attrs[i], attrs, tag, options);
1052     if (normalized) {
1053         parts.unshift(buildAttr(normalized, hasUnarySlash, options, isLast, uidAttr));
1054         isLast = false;
1055     }
1056 }
1057 if (parts.length > 0) {
1058     buffer.push(' ');

```

```

1059     buffer.push.apply(buffer, parts);
1060 }
1061 // start tag must never be omitted if it has any attributes
1062 else if (optional && optionalStartTags(tag)) {
1063     optionalStartTag = tag;
1064 }
1065
1066 buffer.push(buffer.pop() + (hasUnarySlash ? '/' : '') + '>');
1067
1068 if (autoGenerated && !options.includeAutoGeneratedTags) {
1069     removeStartTag();
1070     optionalStartTag = '';
1071 }
1072 },
1073 end: function(tag, attrs, autoGenerated) {
1074     if (tag.toLowerCase() === 'svg') {
1075         options = Object.getPrototypeOf(options);
1076     }
1077     tag = options.name(tag);
1078
1079     // check if current tag is in a whitespace stack
1080     if (options.collapseWhitespace) {
1081         if (stackNoTrimWhitespace.length) {
1082             if (tag === stackNoTrimWhitespace[stackNoTrimWhitespace.length - 1]) {
1083                 stackNoTrimWhitespace.pop();
1084             }
1085         }
1086         else {
1087             squashTrailingWhitespace('/') + tag);
1088         }
1089         if (stackNoCollapseWhitespace.length &&
1090             tag === stackNoCollapseWhitespace[stackNoCollapseWhitespace.length - 1]) {
1091             stackNoCollapseWhitespace.pop();
1092         }
1093     }
1094
1095     var isElementEmpty = false;
1096     if (tag === currentTag) {
1097         currentTag = '';
1098         isElementEmpty = !hasChars;
1099     }
1100
1101     if (options.removeOptionalTags) {
1102         // <html>, <head> or <body> may be omitted if the element is empty
1103         if (isElementEmpty && topLevelTags(optionalStartTag)) {
1104             removeStartTag();
1105         }
1106         optionalStartTag = '';
1107         // </html> or </body> may be omitted if not followed by comment

```





```

1157         return '';
1158     });
1159 }
1160 }
1161 }
1162 if (prevTag) {
1163     if (prevTag === '/noabr' || prevTag === 'wbr') {
1164         if (/^\s/.test(text)) {
1165             var tagIndex = buffer.length - 1;
1166             while (tagIndex > 0 && buffer[tagIndex].lastIndexOf('<' + prevTag) !== 0) {
1167                 tagIndex--;
1168             }
1169             trimTrailingWhitespace(tagIndex - 1, 'br');
1170         }
1171     }
1172     else if (inlineTextTags(prevTag.charAt(0) === '/' ? prevTag.slice(1) : prevTag)) {
1173         text = collapseWhitespace(text, options, /(?:^|\s)$/.test(currentChars));
1174     }
1175 }
1176 if (prevTag || nextTag) {
1177     text = collapseWhitespaceSmart(text, prevTag, nextTag, options);
1178 }
1179 else {
1180     text = collapseWhitespace(text, options, true, true);
1181 }
1182 if (!text && /\s$/.test(currentChars) && prevTag && prevTag.charAt(0) === '/') {
1183     trimTrailingWhitespace(buffer.length - 1, nextTag);
1184 }
1185 }
1186 if (!stackNoCollapseWhitespace.length && nextTag !== 'html' && !(prevTag && nextTag)) {
1187     text = collapseWhitespace(text, options, false, false, true);
1188 }
1189 }
1190 if (options.processScripts && specialContentTags(currentTag)) {
1191     text = processScript(text, options, currentAttrs);
1192 }
1193 if (isExecutableScript(currentTag, currentAttrs)) {
1194     text = options.minifyJS(text);
1195 }
1196 if (isStyleSheet(currentTag, currentAttrs)) {
1197     text = options.minifyCSS(text);
1198 }
1199 if (options.removeOptionalTags && text) {
1200     // <html> may be omitted if first thing inside is not comment
1201     // <body> may be omitted if first thing inside is not space, comment, <meta>, <link>, <scr
1202     if (optionalStartTag === 'html' || optionalStartTag === 'body' && !/^\s/.test(text)) {
1203         removeStartTag();
1204     }
1205     optionalStartTag = '';

```

```

1206 // </html> or </body> may be omitted if not followed by comment
1207 // </head>, </colgroup> or </caption> may be omitted if not followed by space or comment
1208 if (compactTags(optionalEndTag) || looseTags(optionalEndTag) && !/^s/.test(text)) {
1209     removeEndTag();
1210 }
1211 optionalEndTag = '';
1212 }
1213 charsPrevTag = /^s*$/.test(text) ? prevTag : 'comment';
1214 if (options.decodeEntities && text && !specialContentTags(currentTag)) {
1215     // Escape any `&` symbols that start either:
1216     // 1) a legacy named character reference (i.e. one that doesn't end with `;`)
1217     // 2) or any other character reference (i.e. one that does end with `;`)
1218     // Note that `&` can be escaped as `&amp`, without the semi-colon.
1219     // https://mathiasbynens.be/notes/ambiguous-ampersands
1220     text = text.replace(/&((?:Iacute|aacute|uacute|plusmn|Otilde|otilde|agrave|Agrave|Yacute|y
1221 )
1222 if (uidPattern && options.collapseWhitespace && stackNoTrimWhitespace.length) {
1223     text = text.replace(uidPattern, function(match, prefix, index) {
1224         return ignoredCustomMarkupChunks[+index][0];
1225     });
1226 }
1227 currentChars += text;
1228 if (text) {
1229     hasChars = true;
1230 }
1231 buffer.push(text);
1232 },
1233 comment: function(text, nonStandard) {
1234     var prefix = nonStandard ? '<!' : '<!--';
1235     var suffix = nonStandard ? '>' : '-->';
1236     if (isConditionalComment(text)) {
1237         text = prefix + cleanConditionalComment(text, options) + suffix;
1238     }
1239     else if (options.removeComments) {
1240         if (isIgnoredComment(text, options)) {
1241             text = '<!--' + text + '-->';
1242         }
1243         else {
1244             text = '';
1245         }
1246     }
1247     else {
1248         text = prefix + text + suffix;
1249     }
1250     if (options.removeOptionalTags && text) {
1251         // preceding comments suppress tag omissions
1252         optionalStartTag = '';
1253         optionalEndTag = '';
1254     }

```

```

1255     buffer.push(text);
1256 },
1257 doctype: function(doctype) {
1258     buffer.push(options.useShortDoctype ? '<!doctype' +
1259         (options.removeTagWhitespace ? ' ' : ' ') + 'html>' :
1260         collapseWhitespaceAll(doctype));
1261 }
1262 });
1263
1264 if (options.removeOptionalTags) {
1265     // <html> may be omitted if first thing inside is not comment
1266     // <head> or <body> may be omitted if empty
1267     if (topLevelTags(optionalStartTag)) {
1268         removeStartTag();
1269     }
1270     // except for </dt> or </thead>, end tags may be omitted if no more content in parent element
1271     if (optionalEndTag && !trailingTags(optionalEndTag)) {
1272         removeEndTag();
1273     }
1274 }
1275 if (options.collapseWhitespace) {
1276     squashTrailingWhitespace('br');
1277 }
1278
1279 return joinResultSegments(buffer, options, uidPattern ? function(str) {
1280     return str.replace(uidPattern, function(match, prefix, index, suffix) {
1281         var chunk = ignoredCustomMarkupChunks[+index][0];
1282         if (options.collapseWhitespace) {
1283             if (prefix !== '\t') {
1284                 chunk = prefix + chunk;
1285             }
1286             if (suffix !== '\t') {
1287                 chunk += suffix;
1288             }
1289             return collapseWhitespace(chunk, {
1290                 preserveLineBreaks: options.preserveLineBreaks,
1291                 conservativeCollapse: !options.trimCustomFragments
1292             }, /^[ \n\r\t\f]/.test(chunk), /[ \n\r\t\f]$/.test(chunk));
1293         }
1294         return chunk;
1295     });
1296 } : identity, uidIgnore ? function(str) {
1297     return str.replace(new RegExp('<!--' + uidIgnore + '([0-9]+)-->', 'g'), function(match, index)
1298         return ignoredMarkupChunks[+index];
1299     });
1300 } : identity);
1301 }
1302
1303 function joinResultSegments(results, options, restoreCustom, restoreIgnore) {

```

```

1304     var str;
1305     var maxLineLength = options.maxLineLength;
1306     if (maxLineLength) {
1307         var line = '', lines = [];
1308         while (results.length) {
1309             var len = line.length;
1310             var end = results[0].indexOf('\n');
1311             if (end < 0) {
1312                 line += restoreIgnore(restoreCustom(results.shift()));
1313             }
1314             else {
1315                 line += restoreIgnore(restoreCustom(results[0].slice(0, end)));
1316                 results[0] = results[0].slice(end + 1);
1317             }
1318             if (len > 0 && line.length > maxLineLength) {
1319                 lines.push(line.slice(0, len));
1320                 line = line.slice(len);
1321             }
1322             else if (end >= 0) {
1323                 lines.push(line);
1324                 line = '';
1325             }
1326         }
1327         if (line) {
1328             lines.push(line);
1329         }
1330         str = lines.join('\n');
1331     }
1332     else {
1333         str = restoreIgnore(restoreCustom(results.join('')));
1334     }
1335     return options.collapseWhitespace ? collapseWhitespace(str, options, true, true) : str;
1336 }
1337
1338 exports.minify = function(value, options) {
1339     var start = Date.now();
1340     options = processOptions(options || {});
1341     var result = minify(value, options);
1342     options.log('minified in: ' + (Date.now() - start) + 'ms');
1343     return result;
1344 };

```