

```

1 1  /**
2 2  * @fileoverview microlight - syntax highlightning library
3 3  * @version 0.0.1
4 4  *
5 5  * @license MIT, see http://github.com/asvd/microlight
6 6  * @copyright 2016 asvd <heliosframework@gmail.com>
7 7  *
8 8  * Code structure aims at minimizing the compressed library size
9 9  */
10 10
11 11
12 12  (function (root, factory) {
13 13  if (typeof define === 'function' && define.amd) {
14 14  define(['exports'], factory);
15 15  } else if (typeof exports !== 'undefined') {
16 16  factory(exports);
17 17  } else {
18 18  factory((root.microlight = {}));
19 19  }
20 20  }(this, function (exports) {
21 21  // for better compression
22 22  var _window      = window,
23 23  _document       = document,
24 24  appendChild     = 'appendChild',
25 25  test            = 'test',
26 26  // style and color templates
27 27  textShadow      = ';text-shadow:',
28 28  opacity         = 'opacity:.',
29 29  _0px_0px       = ' 0px 0px ',
30 30  _3px_0px_5     = '3px 0px 5',
31 31  brace           = ')',
32 32
33 33  el, // current microlighted element to run through
34 34
35 35  // dynamic set of nodes to highlight
36 36  microlighted = _document.getElementsByClassName('microlight');
37 37
38 38
39 39  - var reset = function(i) {
40 40  - for (i = 0; el = microlighted[i++];) {
41 41  - var text = el.textContent,
39 39  + var process = function(text, color) {
40 40  + var el = document.createElement('div'),
42 42  pos = 0, // current position
43 43  next1 = text[0], // next character
44 44  chr = 1, // current character
45 45  prev1, // previous character
46 46  prev2, // the one before the previous
47 47  - token = // current token content
48 48  - el.innerHTML = '', // (and cleaning the node)
46 46  + token = '', // current token content
49 49
50 50  // current token type:
51 51  // 0: anything else (whitespaces / newlines)
52 52  // 1: operator or brace
53 53  // 2: closing braces (after which '/' is division not regex)
54 54  // 3: (key)word
55 55  // 4: regex
56 56  // 5: string starting with "
57 57  // 6: string starting with '
58 58  // 7: xml comment <!-- -->
59 59  // 8: multiline comment /* */
60 60  // 9: single-line comment starting with two slashes //
61 61  // 10: single-line comment starting with hash #
62 62  tokenType = 0,
63 63
64 64  // kept to determine between regex and division
65 65  lastTokenType,
66 66  // flag determining if token is multi-character

```

```

67 65 multichar,
68 66 node,
69 67
70 68 // calculating the colors for the style templates
71 - colorArr = /(\d*\, \d*\, \d*)(, ([.\d]*)?)?/g.exec(
72 - _window.getComputedStyle(e1).color
73 - ),
74 - pxColor = 'px rgba('+colorArr[1]+' ,',
75 - alpha = colorArr[3]||1;
69 + colorArr = /(\d*\, \d*\, \d*)(, ([.\d]*)?)?/g.exec(color),
70 + pxColor = 'px rgba('+colorArr[1]+' ,',
71 + alpha = colorArr[3]||1;
72
73 // running through characters and highlighting
74 while (prev2 = prev1,
75 // escaping if needed (with except for comments)
76 // pervious character will not be therefore
77 // recognized as a token finalize condition
78 prev1 = tokenType < 7 && prev1 == '\\ ' ? 1 : chr
79 ) {
80 chr = next1;
81 next1=text[++pos];
82 multichar = token.length > 1;
83
84 // checking if current token should be finalized
85 if (!chr || // end of content
86 // types 9-10 (single-line comments) end with a
87 // newline
88 (tokenType > 8 && chr == '\n') ||
89 [ // finalize conditions for other token types
90 // 0: whitespaces
91 /\S/[test](chr), // merged together
92 // 1: operators
93 1, // consist of a single character
94 // 2: braces
95 1, // consist of a single character
96 // 3: (key)word
97 !/[ $\w ]/[test](chr),
98 // 4: regex
99 (prev1 == '/' || prev1 == '\n') && multichar,
100 // 5: string with "
101 prev1 == '"' && multichar,
102 // 6: string with '
103 prev1 == "'" && multichar,
104 // 7: xml comment
105 text[pos-4]+prev2+prev1 == '-->',
106 // 8: multiline comment
107 prev2+prev1 == '*/'
108 ][tokenType]
109 ) {
110 // appending the token to the result
111 if (token) {
112 // remapping token type into style
113 // (some types are highlighted similarly)
114 el.appendChild(
115 node = _document.createElement('span')
116 ).setAttribute('style', [
117 // 0: not formatted
118 '',
119 // 1: keywords
120 textShadow + _0px_0px+9+pxColor + alpha * .7 + '),'+
121 _0px_0px+2+pxColor + alpha * .4 + brace,
122 // 2: punctuation
123 opacity + 6 +
124 textShadow + _0px_0px+7+pxColor + alpha / 4 + '),'+
125 _0px_0px+3+pxColor + alpha / 4 + brace,
126 // 3: strings and regexps
127 opacity + 7 +
128 textShadow + _3px_0px_5+pxColor + alpha / 5 + '),-' +

```

```

133 129  _3px_0px_5+pxColor + alpha / 5 + brace,
134 130  // 4: comments
135 131  'font-style:italic;'+
136 132  opacity + 5 +
137 133  textShadow + _3px_0px_5+pxColor + alpha / 4 + '),-' +
138 134  _3px_0px_5+pxColor + alpha / 4 + brace
139 135  ]{
140 136  // not formatted
141 137  !tokenType ? 0 :
142 138  // punctuation
143 139  tokenType < 3 ? 2 :
144 140  // comments
145 141  tokenType > 6 ? 4 :
146 142  // regex and strings
147 143  tokenType > 3 ? 3 :
148 144  // otherwise tokenType == 3, (key)word
149 145  // (1 if regex matches, 0 otherwise)
150 146  + /^(a(bstract|lias|nd|rguments|rray|s(m|sert)?|uto)|b(ase|egin|ool(ean)?|reak|yte)|c(ase|atch|har|hecked|lass|lone|ompl|onst|ontinue)|de(bugger|cimal|clare|f(ault|er)?|init|l(egate|ete)?)|dc
151 147  ]);
152 148
153 149  node.appendChild(_document.createTextNode(token));
154 150  }
155 151
156 152  // saving the previous token type
157 153  // (skipping whitespaces and comments)
158 154  lastTokenType =
159 155  (tokenType && tokenType < 7) ?
160 156  tokenType : lastTokenType;
161 157
162 158  // initializing a new token
163 159  token = '';
164 160
165 161  // determining the new token type (going up the
166 162  // list until matching a token type start
167 163  // condition)
168 164  tokenType = 11;
169 165  while (![
170 166  1, // 0: whitespace
171 167  // 1: operator or braces
172 168  /[\/\{\}]{\-\+*<>:;|\.\.?!&@-}/[test](chr),
173 169  /\}\}]/[test](chr), // 2: closing brace
174 170  /[\$\\w]/[test](chr), // 3: (key)word
175 171  chr == '/' && // 4: regex
176 172  // previous token was an
177 173  // opening brace or an
178 174  // operator (otherwise
179 175  // division, not a regex)
180 176  (lastTokenType < 2) &&
181 177  // workaround for xml
182 178  // closing tags
183 179  prevl != '<',
184 180  chr == '"', // 5: string with "
185 181  chr == "'", // 6: string with '
186 182  // 7: xml comment
187 183  chr+nextl+text[pos+1]+text[pos+2] == '<!--',
188 184  chr+nextl == '/*', // 8: multiline comment
189 185  chr+nextl == '//', // 9: single-line comment
190 186  chr == '#', // 10: hash-style comment
191 187  ][--tokenType]);
192 188  }
193 189
194 190  token += chr;
195 191  }
192 +
193 + return el.innerHTML;
196 194  }
195 +
196 + var reset = function(i) {

```

```
197 + if (isNaN(i)) {
198 + i = 0;
199 }
200
199 - exports.reset = reset;
201 + for (;el = microlighted[i++];) {
202 + el.innerHTML = process(
203 + el.textContent,
204 + _window.getComputedStyle(el).color
205 + );
206 + }
207 + };
208 +
209 + exports.process = process;
210 + exports.reset = reset;
200
201 211
201 212 if (_document.readyState == 'complete') {
202 213 reset();
203 214 } else {
204 215 _window.addEventListener('load', reset, 0);
205 216 }
206 217 }));
```