CSSMin - Cascading Stylesheet Minifier

What does the code do:

- repaces not used whitespaces
- respect important comments like copyright (/*! ... */)
- deletes unimportat comments
- don't change anything in string literals
- handle single line rules like @import and @charset
- handle wrapper rules like @media, @supports, @document, @keyframes and @-webkit-keyframes
- handle rules like @page and @viewport like each other rule
- deletes multiple semicolons
- inserts missing semicolons in front of closing braces
- replace units after zero values
- replace zero in front of a decimal separator
- replace % behind zero, except in gradients

cssmin.pas (/code/cssmin.pas?mode=download) Pascal (7,04 kByte) 22.10.2016 16:14

```
// ******************************************************************************
//  Title..... :  Cascading Stylesheet Minify Function
//
//  Modulname ..... :  cssmin.pas
//  Type ..... :  Unit (Library)
//  Author ..... :  Udo Schmal
//  Development Status  :  21.05.2016
//  IDE ..... :  Delphi & Lazarus
// ******************************************************************************
unit cssmin;
{$ifdef fpc}
  {$mode objfpc}
{$endif}
{$H+}

interface

uses
  SysUtils, Classes;

function CSSMin(const input: RawByteString): RawByteString;
procedure CSSMinFile(cssFile: string);

implementation

resourcestring
  SErrUnterminatedComment = 'Unterminated comment.';
  SErrUnterminatedStringLiteral = 'Unterminated string literal.';
  SErrUnterminatedSection = 'Unterminated secton header.';

function CSSMin(const input: RawByteString): RawByteString;
const
  rules: array[0..6] of string = (
    'import', 'charset',
    'media', 'supports', 'document',
    'keyframes', '-webkit-keyframes'
  );
var
  line: RawByteString;
  iChar, len, o, i: integer;
  bRule: boolean;
  c: char;
begin
  result := '';
```

```
try
  len := length(input);
  if (len > 2) and (input[1] = #$ef) and (input[2] = #$bb) and (input[3] = #$bf) then
  begin
    // don't replace a utf-8 BOM
    iChar := 4;
    result := #$ef#$bb#$bf;
  end
  else
    iChar := 1;
  line := '';
  bRule := false;
  while iChar <= len do
  begin
    c := input[iChar];
    if (c = '/') and (iChar+1 <= len) and (input[iChar+1] = '*') then
    begin // comment
      if (iChar+2 <= len) and (input[iChar+2] = '!') then
      begin
        // important comment
        line := line + c;
        repeat
          inc(iChar);
          if (iChar > len) then raise Exception.Create(SErrUnterminatedComment);
          line := line + input[iChar];
        until (input[iChar-1] = '*') and (input[iChar] = '/');
        result := result + line;
        line := #10;
      end
      else
      // unimportant comment
      repeat
        inc(iChar);
        if (iChar > len) then raise Exception.Create(SErrUnterminatedComment)
      until (input[iChar-1] = '*') and (input[iChar] = '/');
    end
    else if c in ['"', ''''] then
    begin // string
      repeat
        line := line + input[iChar];
        inc(iChar);
        if (iChar > len) then raise Exception.Create(SErrUnterminatedStringLiteral)
      until input[iChar] = c;
      line := line + c;
    end
    else if (length(line) = 1) and (c = '@') then
    begin // rule
      // identify rule of type section like "@media", "@supports" or "@document"
      // or rules like "@import" or "@charset"
      for o := 0 to 6 do
      begin
        // compare rule names
        bRule := true;
        for i := 1 to length(rules[o]) do
          if (rules[o][i] <> input[iChar+i]) then
          begin
            bRule := false;
            break;
          end;
        if bRule then
        begin
          // section or rule found, copy to destination
          inc(iChar, length(rules[o]));
          line := line + c + rules[o];
          break;
        end;
      end;
      if not bRule then
```

```
    // @page and @viewport does not require any special treatment
    // else no section or rule
    line := line + c;
  end
  else if (c in [';','{']) and bRule then
  begin // rule end
    if (line[length(line)] = ' ') then
    // no space in front of closing braces or in front of a semicolon
    setLength(line, length(line)-1);
    result := result + line + c;
    line := #10;
    bRule := false;
  end
  else if c = '}' then
  begin // closing braces
    if (line[length(line)] = ' ') then
    // no space in front of closing braces
    setLength(line, length(line)-1);
    if (length(line) = 1) or (line[length(line)] <> '{') then
    begin
    // if length of line is one (#10) it must be a section
    // and be shure it is no empty definition
    if (length(line) > 1) and (line[length(line)] <> ';') then
      // semicolon after last value, in front of closing braces
      line := line + ';';
    result := result + line + c;
    end;
    line := #10;
  end
  else if (c = ';') and (line[length(line)] = ';') then
    // no multiple semicolons
  else if (c in [':', ';',',',')','{']) and (length(line) > 0) and (line[length(line)] = ' ') then
    // no space in front of some special chars
    line[length(line)] := c
  else if (c in [#9,#10,#13,' ']) then
  begin // whitespace
    if (length(line) > 2) and
      not (line[length(line)] in ['!','{',':',';',',','>','+','(','[',']',' ']) then
      // no space behind some special chars
      line := line + ' ';
  end
  else if (c = '0') then
  begin // zero
    if (length(line) > 1) and (line[length(line)] in [' ', ':']) and
       (iChar+2 < len) and
       (((input[iChar+1] = 'p') and (input[iChar+2] = 'x')) or
        ((input[iChar+1] = 'e') and (input[iChar+2] = 'm')) or
        ((input[iChar+1] = 'i') and (input[iChar+2] = 'n')) or
        ((input[iChar+1] = 'c') and (input[iChar+2] = 'm')) or
        ((input[iChar+1] = 'm') and (input[iChar+2] = 'm')) or
        ((input[iChar+1] = 'p') and (input[iChar+2] = 'c')) or
        ((input[iChar+1] = 'p') and (input[iChar+2] = 't')) or
        ((input[iChar+1] = 'e') and (input[iChar+2] = 'x'))) then
    begin
      // a zero value does not need a unit
      inc(iChar,2);
      line := line + c;
    end
    else if (length(line) > 0) and (line[length(line)] in [' ', ':']) and
      (input[iChar+1]='.') then
      // no zero in front of dot
    else if (length(line) > 0) and (line[length(line)] = ':') and
      (input[iChar+1] = '%') and (input[iChar+2] <> ',') then
    begin
      // no % behind zero, except in gradients
      inc(iChar);
      line := line + c;
    end
```

```
      else
        // if no special rule, write zero
        line := line + c;
    end
    else
      line := line + c;
    inc(iChar);
  end;
  if length(line)>1 then
    result := result + line;
 except
  // unexpected exception
  result := input;
 end;
end;

procedure CSSMinFile(cssFile: string);
var
 stream: TMemoryStream;
 sText: RawByteString;
begin
 stream := TMemoryStream.Create;
 try
  stream.LoadFromFile(cssFile);
  setLength(sText, stream.Size);
  stream.Read(sText[1], stream.Size);
  sText := CSSMin(sText);
  if length(sText) > 0 then
  begin
    stream.Clear;
    stream.Write(sText[1], length(sText));
    stream.SaveToFile(ChangeFileExt(cssFile, '.min' + ExtractFileExt(cssFile)));
  end;
 finally
  stream.Free;
 end;
end;

end.
```

Autor: Udo Schmal, veröffentlicht: 22.06.2012, letzte Änderung: 22.10.2016