JSMin - JavaScript Minifier

jsmin.pas (/code/jsmin.pas?mode=download) Pascal (7,76 kByte) 18.10.2016 14:55

```
// ***************************************************************************
// Title..... : Javascript Minify Function
//
// Modulname ..... : jsmin.pas
// Type ..... : Unit (Library)
// Author ..... : Udo Schmal
// Development Status : 18.05.2016
// Operating System .. : Win32/64
// IDE ..... : Delphi & Lazarus
// ***************************************************************************
unit jsmin;
{$ifdef FPC}
  {$mode objfpc}
{$endif}
{$H+}

interface

uses
  SysUtils, Classes;

function JSMin(const input: RawByteString): RawByteString;
procedure JSMinFile(jsFile: string);

implementation

Resourcestring
  SErrUnterminatedComment = 'Unterminated comment.';
  SErrUnterminatedStringLiteral = 'Unterminated string literal.';
  SErrUnterminatedSetInRegexp = 'Unterminated set in Regular Expression literal.';
  SerrUnterminatedRegexp = 'Unterminated Regular Expression literal.';

function JSMin(const input: RawByteString): RawByteString;
var
  c1, c2: char;
  iChar, iPos, len: integer;

  procedure add(c: char);
  begin
    if c<>#0 then
    begin
      inc(iPos);
      result[iPos] := c;
    end;
  end;

  // isAlphanum -- return true if the character is a letter, digit, underscore,
  // dollar sign, or non-ASCII character.
  function isAlphanum(ch: char): boolean;
  begin
    result := (ch in ['a'..'z','0'..'9','A'..'Z','_', '$', '\']) or (ord(ch)>126);
  end;

  // getChar -- return the next character
  // convert control characters, translate it to a space or linefeed.
  function getChar(movePtr: boolean = true): char;
  begin
    if (iChar > len) then
      // eof
      result := #$1A
    else
    begin
      result := input[iChar];
      if movePtr then
        inc(iChar); // move reading pointer
      if (result = #13) then
      begin
        // only unix linefeed
        if movePtr then
        begin
          if (input[iChar] = #10) then
            inc(iChar);
        end
        else
        begin
          if (input[iChar+1] = #10) then
            inc(iChar);
```

```
      end;
      result := #10
    end
    else if (result < ' ') and (result <> #10) then
      // change other control characters for example the tab #9 to space
      result := ' ';
  end;
end;

// getSigChar -- get the next significant character
// excluding comments but respect important comments
procedure getSigChar();
var bLoop: boolean;
begin
  c2 := getChar();
  if c2 = '/' then
    case getChar(false) of
    '/': while (c2 > #10) do
         // comment until eol
         c2 := getChar();
    '*': begin
         // comment until */
         inc(iChar); // move reading pointer
         if (getChar(false) = '!') then
         begin
           // important comment
           add(c2);
           add('*');
           inc(iChar); // move reading pointer
           add('!');
           bLoop := true;
           while bLoop do
           begin
             c2 := getChar();
             case c2 of
             '*': if (getChar(false) = '/') then
                  begin
                    add(c2);
                    inc(iChar); // move reading pointer
                    add('/');
                    c2 := getChar();
                    if c2 = #10 then
                      // save linefeed afer important comment
                      add(c2);
                    bLoop := false;
                  end
                  else
                    add(c2);
             #$1A: raise Exception.Create(SErrUnterminatedComment);
             else
               add(c2);
             end;
           end;
         end
         else
           // unimportant comment
           while c2 <> #$20 do
             case getChar() of
             '*': if (getChar(false) = '/') then
                  begin
                    inc(iChar); // move reading pointer
                    c2 := #$20; // handle comment as a space
                  end;
             #$1A: raise Exception.Create(SErrUnterminatedComment);
             end;
       end;
    end;
end;

// getNextChar -- get next significant char
// solve regular expressions
procedure getNextChar();
begin
  getSigChar();
  if (c2 = '/') and (c1 in ['(',',','=',':','[','!','&','|','?','{','}',';',#$0A]) then
  begin
    // it is a regular expression
    add(c1);
    add(c2);
    while true do
    begin
```

```
      c1 := getChar();
      if c1 = '[' then
       // bracket - range of characters
       while true do
       begin
         add(c1);
         c1 := getChar();
         if c1 = ']' then
          // end of rage of characters
           break
         else if c1 = '\' then
         begin
          // metacharacter
          add(c1);
          c1 := getChar();
         end;
         if c1 = #$1A then
          raise Exception.Create(SerrUnterminatedRegexp);
       end
      else if c1 = '/' then
       // end of regular expression
       break
      else if c1 = '\' then
      begin
       // metacharacter
       add(c1);
       c1 := getChar();
      end
      else if(c1 <= #10) then
       raise Exception.Create(SErrUnterminatedSetInRegexp);
      add(c1);
    end;
    getSigChar();
  end;
 end;

var space: boolean;
begin
  result := '';
  try
    len := length(input);
    setLength(result, len);
    iPos := 0;
    if (len > 2) and (input[1] = #$ef) and (input[2] = #$bb) and (input[3] = #$bf) then
    begin
     // don't replace utf-8 BOM
     iChar := 4;
     add(#$ef);
     add(#$bb);
     add(#$bf);
    end
    else
     iChar := 1;

    c1 := #0;
    getNextChar();
    while(c1 <> #$1A) do
    begin
     space := false;
     case c1 of
     ' ': if isAlphanum(c2) then add(c1);
     #10: case c2 of
          '{', '[', '(', '+', '-': add(c1);
          ' ': space := true;
         else if isAlphanum(c2) then add(c1);
         end
     else
       case c2 of
       ' ': if isAlphanum(c1) then add(c1) else space := true;
       #10: case c1 of
            '}', ']', ')', '+', '-', '"', '''': add(c1);
            else if isAlphanum(c1) then add(c1) else space := true;
            end;
       else add(c1);
       end;
     end;
     if not space then
     begin
       c1 := c2;
       if (c1 = '"') or (c1 = '''') then
        // it is a string literal
```

```
      while true do
      begin
       add(c1);
       c1 := getChar();
       if (c1 = c2) then
        break
       else if(c1 <= #10) then
        raise Exception.Create(SErrUnterminatedStringLiteral)
       else if(c1 = '\') then
       begin
        // it has a masked character
        add(c1);
        c1 := getChar();
       end;
      end;
    end;
    getNextChar();
  end;
  setLength(result, iPos);
 except
  result := input;
 end;
end;

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

end.
```

Autor: Udo Schmal (https://plus.google.com/107378996518617284564?rel=author) , veröffentlicht: 22.06.2012, letzte Änderung: 18.10.2016