System requirements

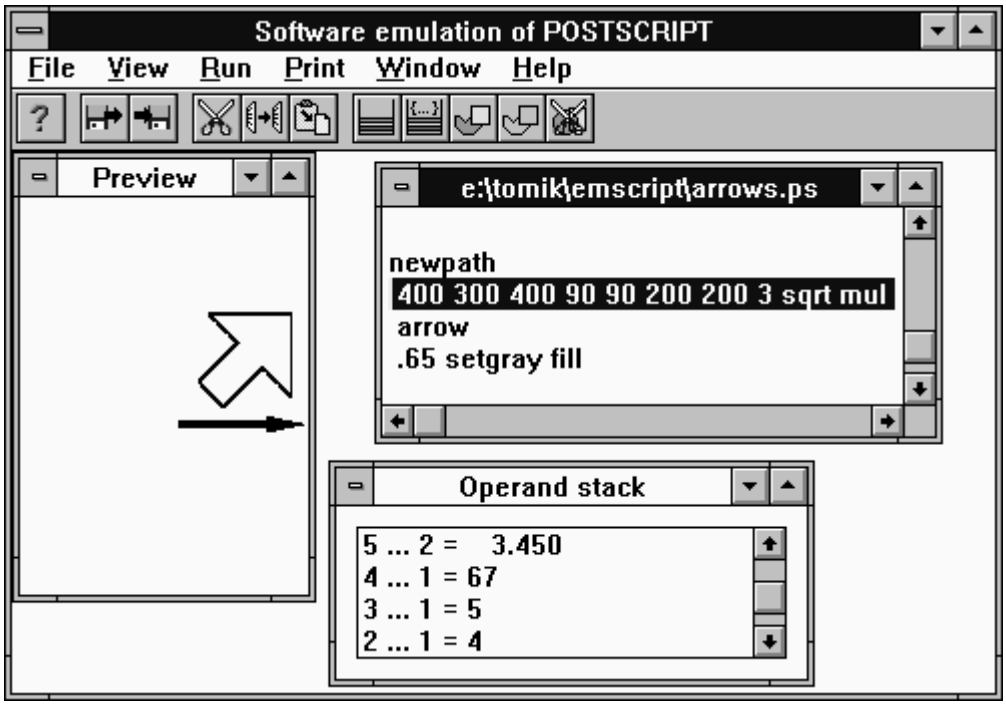
You need a 386 PC compatible, 4 MB RAM and MS WINDOWS™ 3.1.

Setup

Copy files EMSCRIPT.EXE, BASICNUM.DTA and SYSDICT.DTA into any directory. If you want to use examples, copy the files in the EXAMPLE directory. You can start the program by the EMSCRIPT.EXE command in the Run dialog under MS WINDOWS™ program manager.

Program using

EMSCRIPT allows you to interpret PostScript programs. It is also possible to view inner interpreter structures to better understand its activity. The program is useful for study of PostScript programs. You can interpret a source code either as a whole by the **Fast** command in the **Run** menu or line after line by the **Step by step** command in the **Run** menu or by the ***F8*** button. At any time you can set EMSCRIPT into the starting state by the **Reset interpret** command in the **Run** menu. This command will set the program in the state to start interpretation from the beginning. You can load the interpreted source code into the window, where you can edit and save it to the hard disk. In such a cases use the **Open** and **Save** commands in the **File** menu. You can arrange windows by the **Tile** and **Cascade** commands in the **Window** menu. It is possible to print the content of the output graphic page by the **Graph page** command in the **Print** menu. The windows which are used for checking the interpret state are described below.



EMSCRIPT main window

Operand stack window

You can open the operand stack window by the **Operand stack** command in the **View** menu. One line in the window represents just one saved operand. The first number on the line is the order of operands in the stack. The top of the stack has the highest number. The second number expresses a saved operand type:

Null (empty)	= 0
Integer	= 1
Real	= 2
Name	= 3

Mark	= 5
Start def proc	= 7
Literal Name	= 9
Logic	= 15
Dictionary	= 20
File	= 21
String	= 41
SubString	= 42
Array	= 51
SubArray	= 52
Matrix	= 53
Font	= 54
DefProc	= 60
Proc	= 61
Operator	= 70

The third value on the line depends on an operand type.

The first operand in the list is saved on the top of the stack. The last is saved on the bottom. The window is automatically set after each interpretation of a line in a source code is completed.

Executable stack window

You can open the executable stack window by the **Executable stack** command in the **View** menu. The first item is on the top of the stack. The last is on the bottom. The window is automatically set after each interpretation of a line in a source code is completed.

Virtual memory window

The virtual memory window allows you to see the complete contents of the virtual memory. The window can be opened by the **Virtual memory** command in the **View** menu. The window is automatically set when the interpreter completes the interpretation of a line in a source code. Each line in the window shows a PostScript object saved in the virtual memory. The first on the line is an order. The second number is an object type::

1 Textual string
3 Array
5 Matrix
6 Dictionary
7 Graphic state
8 Graphic path
9 Font

Additional information of the object type is written in the fourth column. This information depends on the concrete type. The third column expresses the number of sharings. If an object is shared by three other objects you can see 3 in this column. Instead of non shared objects there is the keyword **NIL** in the appropriate line.

Current graph path window

It is possible to open the window by the **Graph path** command in the **View** menu. The window displays the shape of a constructed graphic path.

Current clip path window

You can open this window by the **Clip path** command in the **View** menu. In the window you can see a black area. The area shows region where the painting operators can draw. This area is specified by the current clip path. There is also the current graphic path shown by green lines. Just a part of the graph path laying inside the black area is drawn in the output graph page (e.g. by operators **stroke** or **fill**).

Output graph page window

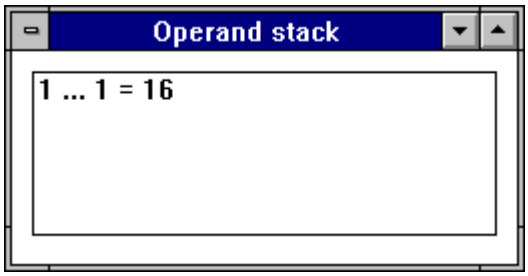
The window maintains an output picture, which is prepared for printing. It is set after each operator, that draws into the output graph page (**stroke**, **fill**, **show**). If the window is not opened you can open it by the **Output** command in the **View** menu. There is a picture shown in full size in this window (100%). There is another window in the program. It is a preview window which allows you fast orientation with the output page. You can open the window by the command **Output preview** in the **View** menu.

How to use program EMSCRIPT

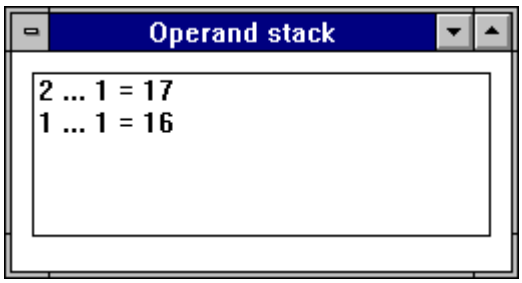
I will show you how to use my program. I will use several basic examples. The header of the examples contains the file name. This file is saved in the EXAMPLE directory. Use the ***F8*** key to interpret a source code one line at a time and keep comments on lines.

Example No.1(DEMO1.PS)

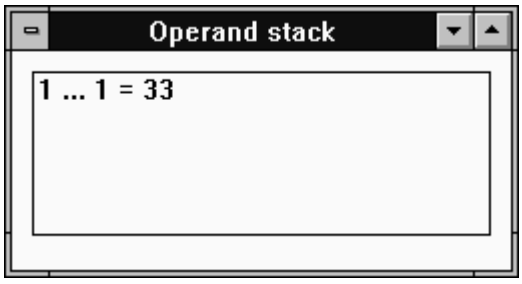
```
% Add two numbers
16                  % Open the operand stack window. Number 16 is saved inside the operand stack.
```




```
17                  % Number 17 will be saved into the stack..
```

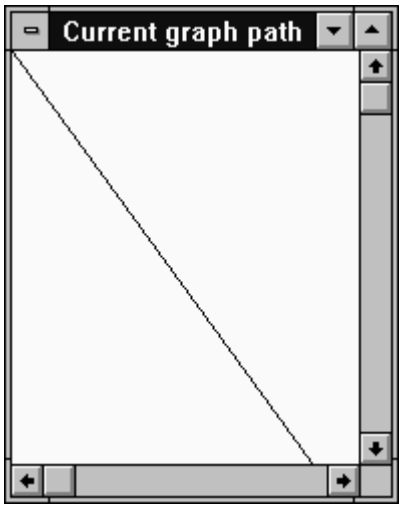



```
add                 % It will add the two numbers on the top of the stack.
                    % The result will be placed on the stack top.
```



Example No.2 (DEMO2.PS)

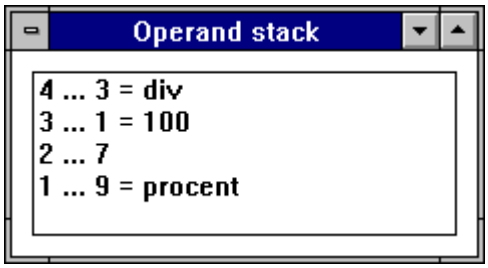
```
% Line drawing

0 792 moveto

576 0 lineto        % Open graph path window. The graph path is shown with the black line.
                    % In this case you will see a line from [0;792]  to [576;0].
```




```
stroke              % The line will be drawn into the output graph page.
```

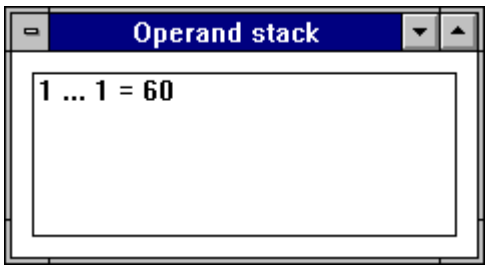
Example No.3 (DEMO3.PS)

```
% Procedure definition

/procent            % Literal name "procent" will be saved into the stack..

{                   % Start procedure definition mark will be placed into the stack.
                    % Open the executable stack window. There is a definition procedure mark
                    % in this stack.

100                 % Number 100 will be saved into the operand stack.

div                 % In this place you can see the difference between a procedure definition and
                    % direct carrying out of operators.
                    % Name will not be carried out immediately, but it will be saved into the operand stack.
```

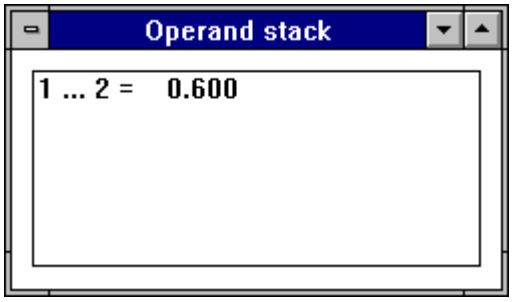



```
}                   % This line finishes the procedure definition. A procedure object
                    % will appear on the operand stack.
                    % The procedure definition mark will be removed from the executable stack.

def                 % The procedure object will be inserted into the user dictionary.
                    % The operand stack will be empty.
```

60 % Number 60 will be placed into the operand stack.

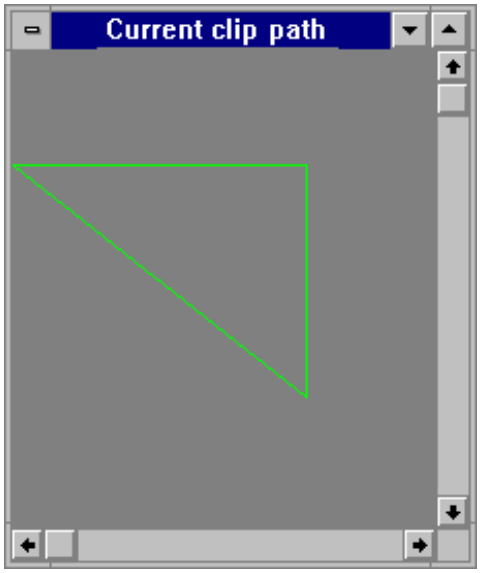


procent % Procedure *procent* will be carried out. It will save the number 0.6 (60/100)
% into the operand stack.

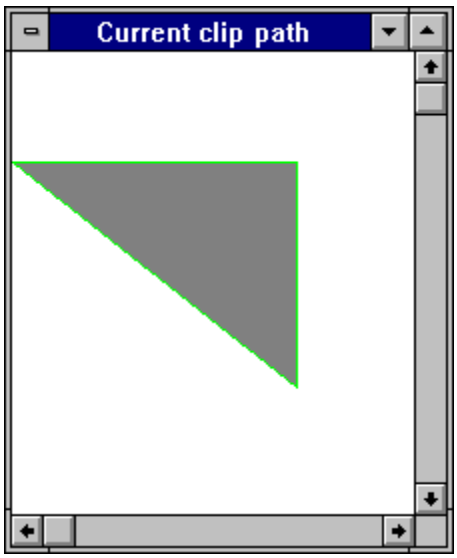


Example No. 4 (DEMO4.PS)

```
% Clip path

/cm {72 mul 2.54 div}      % This procedure allows the user to use centimeters as measure units.

newpath                    % Set the graph path to the beginning state.

0 26 cm moveto

5 cm 26 cm lineto

5 cm 22 cm lineto

closepath

% Open the clip path window. There is a black region shown where it is possible to draw
% and a green line . The line shows a current graph path.
```



clip % The black region in the clip path window will change its shape. It is a new clip path.



newpath

1 cm 25 cm moveto

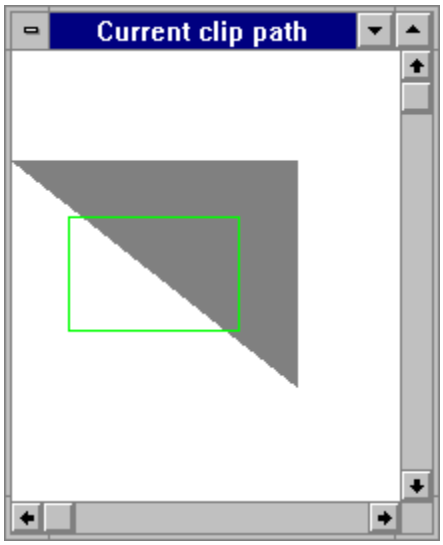
4 cm 25 cm lineto

4 cm 23 cm lineto

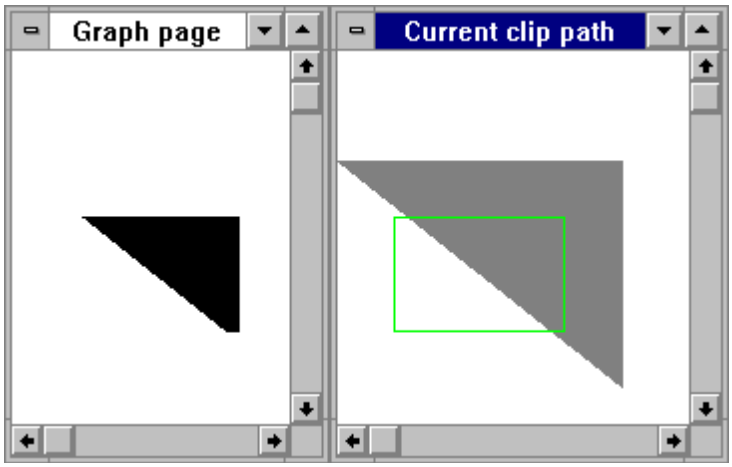
1 cm 23 cm lineto

closepath

% The graph path has the shape of a square. The square is shown with a green line.



```
fill                % By carrying out this command the current graph path will be drawn on the output
                    % graph page. Check contents of the graph page window and the clip path window.
```



Suggestions and questions

If you have some questions or suggestions, write to :

Tomáš Mayer
Branislavova 1413
266 01Beroun
Czech republic

Email: aipdev@login.cz