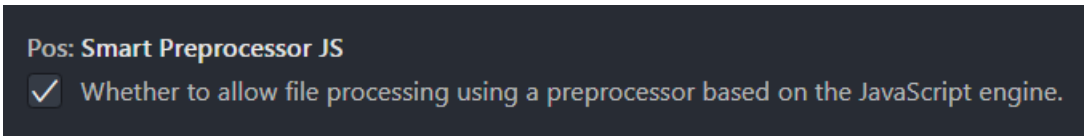
PreprocessorJS for VSCode IDE.

PreprocessorJS is based on the JavaScript engine and allows you to replace pieces of code with the necessary data in the program code before compiling. When compiling, if the code is changed in files, then such a file will be open in file panels.

To use this mode, it is necessary to do the following:

1) Turn on the setting in the positron.



2) Add to each file in which you need to use a PreprocessorJS.

```
'§enable
```

For insert symbol **§** you must press ALT + 0167 (More...)

If you write **§§**, then there will be a highlight of this code in the JavaScript style.

Types of PreprocessorJS blocks:

1) Code execution in the global context of JavaScript

```
(*§
    code javascript
*)
```

2) Code Execution using **Eval** function and replacing the specified text with a returned result

```
'§Action(param) code javascript
```

Action - is an optional.

Example	Overview
'§enable	Enable processing of PreprocessorJS blocks. Does not execute the code.
'§disable	Disable processing of PreprocessorJS blocks. Does not execute the code.
'§exit	Exit from processing of PreprocessorJS blocks. Does not execute the code.

'§region	Sets the beginning of the replaced text from the next line. Execute the code and replace the result.
'§end	Sets the end of the replaced text from the previous line. Does not execute the code.
'§=	It sets the range for the replaced text from the first found position for the symbol = and before the start of the current block. Execute the code and replace the result adding symbol space in front of it.
'§~	It sets the range for the replaced text from the beginning of the current line to the start of the current block. Execute the code asynchronously and replace the result.
'§=~	It sets the range for the replaced text from the first found position for the symbol = and before the start of the current block. Execute the code asynchronously and replace the result adding symbol space in front of it.
'§	It sets the range for the replaced text from the beginning of the current line to the start of the current block. Execute the code and replace the result.

(param) - is an optional.

Example	Overview
()	Replace a completely current line.
(-1)	Replace completely the line below the current one by **-1**.
(14)	Replace the range of characters in the current line starting from **14** position before the beginning of the block.
([14, 17])	Replace the range of characters in the current line starting from **14** position to **17** position inclusive.
([14, -2])	Replace the range of characters in the current line, starting from **14** position to (position of symbol before the start of the block **- 2**) inclusive.

("regex")	Replace the range of characters in the current line determined by the "regex" specified by the regular expression.
	Example1: Replace 123 by 90
	`Symbol my678=123 '§("=") 90`
	Example2: Replace 678 by 90
	`Symbol my678=123 '§("(?<B> my)(?<R>[^=]*)=") 90`
	For a greater understanding, see the source.

Builtin functions and constants:

Syntax	Overview
sleep(ms**)**	It is called asynchronously and allows to pause the execution for a fixed amount of time in milliseconds.
random(value, ofs**)**	Returns a random integer in the range from (0 + ofs) to (value + ofs).
chunk(value, prm**)**	Allows you to break the sequence of the species (0, 1, 2, 3 ... x) to the indicated number of parts with transfer to a new line.
repeat(cnt, value, prm**)**	Depending on the specified value, repeat it a certain number of times.
file(fName, prm**)**	Returns the contents of the file with the specified format.
pick(items, opt, def**)**	Shows a selection list and returns the chosen value. It is called asynchronously.
input(value, def**)**	Opens an input box to ask the user for input and returns the chosen value. It is called asynchronously.
Major, Minor, Release, Build	Constants contain the version number of the compiled file with the code.

For a greater understanding, see the source.