KISSsoft
Drivetrain Design Solutions

SKRIPT

Documentation of the built-in language in KISSsoft

KISSsoft AG
A Gleason Company
Rosengartenstr. 4, 8608 Bubikon
Switzerland

T. +41 55 254 20 50
F. +41 55 254 20 51
info@KISSsoft.AG
www.KISSsoft.AG

Sharing Knowledge

Contents

1 Introduction

The programming language SKRIPT is intended for engineers. It has BASIC like appearance and allows for people with different styles of writing. The programming language hooks into KISSsoft and enables users of this software to access calculation functions and variables, which are displayed on the KISSsoft user interface.

SKRIPT is case insensitive. Indentation has no meaning.

2 Keywords

There are some keywords which are reserved for special purposes and must not be used for variable names:

- `number`
- `string`
- `if`
- `else`
- `for`
- `in`
- `end`
- `while`

3 Comments

For documentation embedded in the code you can use Comments, which are specially marked text in the code and are ignored on executions. Two different types exist.

3.1 Single Line Comments

Single Line Comments consist of one line only, however, there may be several lines following each other. The individual lines are marked with two slashes // as comment.

```
// This is a single line comment.
// Here is a second line with comment.
```

Single line comments can also start somewhere in a code line. Then only the part after the two slashes is ignored.

```
x = 5 // here is some comment. Everything after the two slashes is ignored.
```

3.2 Multi Line Comments

Multi Line Comments comprise a block of text. The begin is marked with /*, the end with */. Both markers may be somewhere in the code. All text between the two markers is ignored on execution.

```
x = 5 /* this is ignored text. The statement after the end marker is executed.
x = 7 */
```

4 Variables

There are variables of two different kinds of variables in SKRIPT: local variables and Meta-variables. The latter are variables of the underlying calculation module which can be accessed directly inside a script.

4.1 Meta-variables

Meta-variables are predefined variables from the underlying calculation module. The names of the variables can be found in the documentation available also for report templates and the COM interface. In opposite to the rest of SKRIPT, the names of Meta-variables **are** case sensitive.
If the name of a Meta-variable contains rectangular brackets, e.g. ZR[0].Geo.mn, the index inside the brackets can be an expression, e.g. ZR[j].Geo.mn. Of course, the value of the expression must be inside the given bounds for that specific Meat-variable, in this case $0 \leq j \leq 3$.

4.2 Local variables

A variable in SKRIPT has a fixed type and must be either declared before use or be a variable of a KISSsoft calculation module.

4.2.1 Types

Possible types are:

- number
- string

4.2.2 Identifiers

Variables (and procedures, see below) have a name, a so-called identifier. An identifier must start with a letter or an underscore _ followed by an arbitrary number of letters, digits, underscores or points.

Example:

```
Correct: number _one
Correct: string theory
Wrong: number 0value // identifier must not start with number
```

4.2.3 Declaration

A declaration of a variable consists of a type and one or multiple identifiers, separated by commas.

Examples:

```
number x
string s1, s2, s3
```

5 Strings

String constants are created starting with double quotes " and contain an arbitrary number of Unicode characters ending with another double quote ". If you need to use double quotes in your string, you must escape them with a backslash. \"
Example:

```
"This is a text which includes Umlauts like äöü, Greek βυΓΣ, and so on."
"You can also use double quotes like this \" and it will be part of the string.
The same goes with backslashes: \\"
```

6 Statements

6.1 If

An If Statement evaluates a condition and executes everything which is in the body.

```
if evaluation expression then
     // statements or variable Declarations
end

if Expression Statement {
     // statements or variable Declarations
}
```

6.2 While

A While defines a condition and executes as long the condition is fulfilled.

```
while Expression Statement do
     // statements or variable Declarations
end
while Expression Statement {
     // statements or variable Declarations
}
```

6.3 For

A for loop iterates an index variable through a give range.

```
for i=1 to 5
     // statements or variable Declarations
end
for number j=8 to 12
     // statements or variable Declarations
next
```

6.4 Procedure Call

Language native procedures

Number procedures

abs	computes absolute value of an integral value (|x|)
cbrt	computes cubic root (3vx)
ceil	nearest integer not less than the given value
copysign	Returns 1 for a negative number, 0 else
degree	Returns the given rad angle as angle in degree
div	computes quotient and remainder of integer division
erf	error function
erfc	complementary error function
exp	returns e^x
exp2	returns 2^x
expm1	returns (ex-1)
fdim	positive difference of two floating point values
floor	nearest integer not greater than the given value
fma	fused multiply-add operation
fmax	larger of two floating point values max(0, x-y)
fmin	smaller of two floating point values
fmod	remainder of the floating point division operation
fpclassify	categorizes the given floating point value
frexp	multiplies a number by
hypot	computes square root of the sum of the squares of two given numbers (vx2+y2)
ilogb	extracts exponent of the number
imaxabs	computes quotient and remainder of integer division
imaxdiv	absolute value of a floating point value
ldexp	multiplies a floating point value x by the number 2 raised to the exp power
lgamma	natural logarithm of the gamma function
log	computes natural (base)
log10	computes common (base e10)
log1p	natural logarithm (to base)
log2	base 2 logarithm of the given number (log2(x))

lround	nearest integer using current rounding mode
modf	decomposes a number into integer and fractional parts
nearbyint	nearest integer using current rounding mode with
nexttoward	returns the next representable value of from in the direction of to
nextafter	extracts exponent of the number
pow	raises a number to the given power (xy)
rad	Returns the given angle in rad
remainder	signed remainder of the division operation
remquo	signed remainder as well as the three last bits of the division operation
rint	decomposes a number into significand and a power of
round	nearest integer, rounding away from zero in halfway cases
sqrt	computes square root (vx)
tgamma	gamma function
trunc	nearest integer not greater in magnitude than the given value

Trigonmetric procedures

acos	computes arc cosine (arccos(x))
acosh	computes the inverse hyperbolic cosine (arcosh(x))
asin	computes arc sine (arcsin(x))
asinh	computes the inverse hyperbolic sine (arsinh(x))
atan	computes arc tangent (arctan(x))
atan2	arc tangent, using signs to determine quadrants
atanh	computes the inverse hyperbolic tangent (artanh(x))
cos	computes cosine (cos(x))
cosh	computes hyperbolic cosine (ch(x))
sin	computes sine (sin(x))
sinh	computes hyperbolic sine (sh(x))
tan	computes tangent (tan(x))
tanh	hyperbolic tangent
to_degrees	Converts radians to degrees
to_radians	Converts degrees to radians

Element procedures

size	returns the number of elements

Check procedures

isfinite	checks if the given number has finite value
isgreater	checks if the first floating-point argument is greater than the second
isgreaterequal	checks if the first floating-point argument is greater or equal than the second
isinf	checks if the given number is infinite
isnan	checks if the given number is NaN
isnormal	checks if the given number is normal
isunordered	checks if two floating-point values are unordered

nan	not-a-number (NaN)

String procedures

strcontains	checks whether the string contains another string or character
strcontainsat	checks whether a string is at the position of string
strempty	checks whether the string is empty
strfind	returns the position at which the given string was found -1 if it wasn't found
strfindfirstof	returns the first position of given string
strfindlastnotof	searches the string for the last character that does not match any of the characters specified in its arguments
strfindlastof	returns last position of given string
strindex	Returns index of a string inside of a string
strlength	returns the length of the string (count of characters)
strlowercase	transforms the string to lowercase
strreplace	replaces a string with another string
strtrim	removes whitespaces at both end
strtrimright	removes whitespaces from right
struppercase	transforms the string to uppercase
substr	creates a chunk from and to a specified positon

File procedures

open_file	Opens a file
read_line	Reads in a line
write_to_file	Writes a string to the file
read	Opens and reads whole file into string
write_all	Opens file, writes and closes file
close_file	Closes file handle again

Other procedures

write	Writes to console

7 Operator

Basic Arithmetic	
+	Addition, for number and string if one of the operands is a string, the result type is string
-	Subtraction, for number
*	Multiplication, for number
/	Division, for number

Comparisons	
==	True if both operands are equal
<	True if first operand is less than second
<=	True if first operand is less or equal than second
>	True if first operand is bigger than second
>=	True if first operand is bigger or equal than second

8 Expression Statements

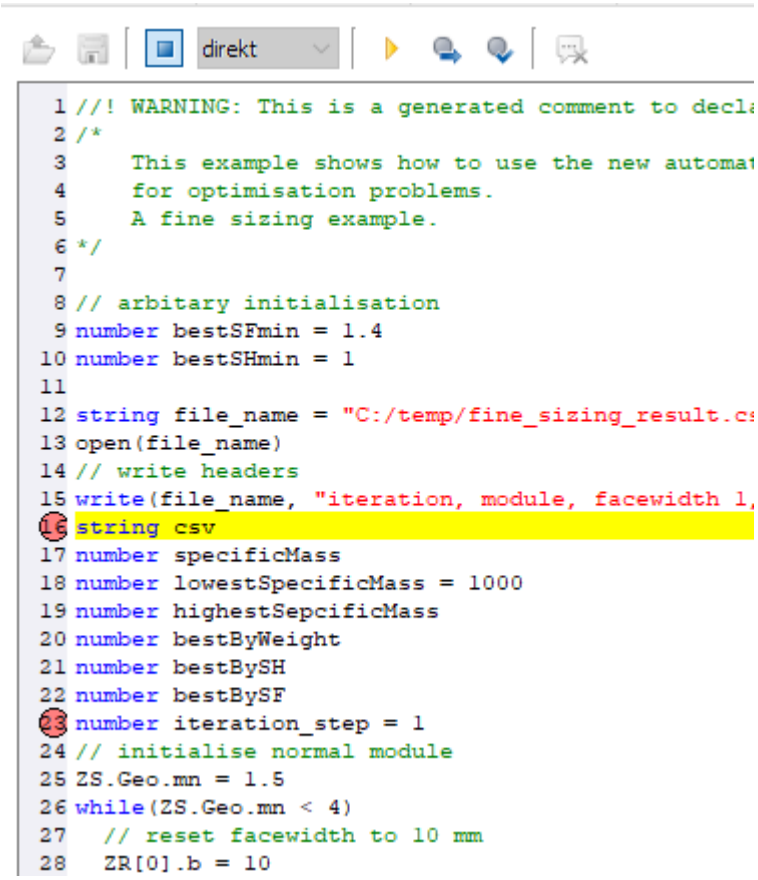
Expression Statements are statements which can be used within an expression context since they might return a value.

```
functionName(prameter,parameter,…) // calls the defined procedure above with a parameter
```

9 Editor

From left to right:

- Open file (opens file dialog)
- Save file
- Save file as (opens file dialog)
- Run Skript
- Type of skript (described below)
- Next line
- Next breakpoint
- List of breakpoints
- Remove Skript messages
- Acitivate/deactivate event script
- Delete script



When you set breakpoints and it hits one, the script halts and the buttons become active.
The line is currently active is highlighted with a yellow background.

You can execute the script until the next statement or next breakpoint.

You are always able to stop the script with the blue square.

9.1 Open Script

Opens a file dialog to load a saved script into the editor.

9.2 Save script

Opens a file dialog to save a script from the current editor.

9.3 Run script

Runs the current loaded script in the editor.

9.4 Cancel script

Cancels the running script. Useful when the script is running in an infinite loop.
You can't cancel during KISSsoft internal "callFunc" functions.

9.5 Close Dialog

The close dialog will discard changes made in the active script.

9.6 Script type

In the dropdown you have six options:

- direct
- preCalc
- postCalc
- onReport
- preSave
- postLoad

9.6.1 direct

This script is manually executed by the user.

9.6.2 preCalc

Is a script executed before the calculation.

9.6.3 postCalc

Is a script executed after the calculation.

9.6.4 onReport

Is a script executed before the report is generated.

9.6.5 preSave

Is a script executed before a calculation file is saved.

9.6.6 postLoad

Is a script executed after the user loaded a calculation file.

9.7 Scripting area

Is the scripting area where the user writes a script. Currently supports syntax highlighting, parentheses matching and autocomplete of variable and function names.
The user can set a breakpoint on the line number area.
When the script during execution reaches said breakpoint, the script halts and enters debug mode.

9.8 Next line

When in debug mode, this button is enabled and executes the next statement or statement expression.

9.9 Next breakpoint

When in debug mode, this button is enabled and continues to execute the script until the next breakpoint, the same breakpoint or the script end is reached.

9.10 Toggle breakpoints (disable/enable)

Disables or enables all the breakpoints in your script.

9.11 Clear console

Manually removes all the output written in the console area.

9.12 Console area

This area displays messages of type:

- errors
- warnings
- and manual output of the user (write() function)

10 Examples

10.1 Example1: Set values and get results from the model

A first example script shall show the basic usage of the scriping language. It shows how to set values in the calculation module, the performing of calculations, the reading of variables and how to write results to the user.

- First the user has to load the file and activate the script editor in the menu view

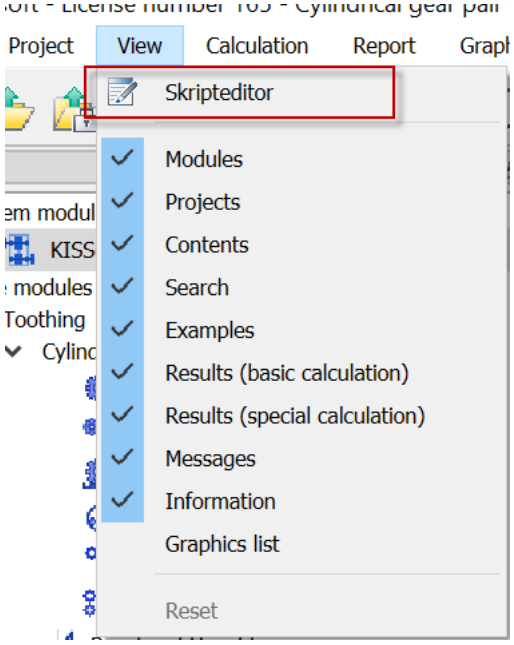


figure 1: activating the script editor

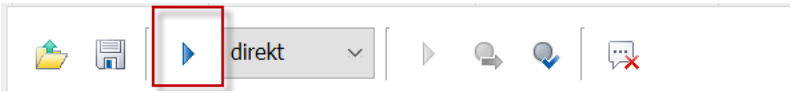
- the following example should run on a a gear pair (e.g here the example - CylGearPair1), so the script can be pasted to the script editor
- it sets the width (variable ZR[0].b and ZR[1].b) of both gears to an initial value, here 44.0 (see *line 9* and *line 10*)
- then to recalculate the method Calclulate() is called (*line 12*)

- now in a while loop the width of both gears is decremented by 1mm in each iteration. First the flag to stop/go on is defined *(line 16*)
- with write() (*line 17*) the user can write out a string to the script console. This may be used for writing results or just a status like “calculation starts now”
- the lines in the following if clauses (*line 20, line 24, line 32, line 36*) check whether the safety against root or flank breaking (ZPP[0].Fuss.SFnorm or ZPP[0].Flanke.SH) of gear1 and gear2 is less then 1. If so, the flag goOn is set to 0 and the iteration will stop in the next while loop iteration in *line 18*
- in the if clause in *line 36* the **width** of gear1 and gear2 (ZR[0].b and ZR[1].b) is decremented for the next step
- in line 42, line 51 and line 52 the current results are written to the console

```
// PURPOSE: This script shows the general usage of the scripting language by setting
// KISSsoft variables and modifying for certain results
// here in the example CylGearPair 1 the width of gear 1 and gear 2 are reduced
// until they could transfer the given load

// initialise width of teeth in the file to a certain vaule
ZR[0].b = 44
ZR[1].b = 44
// this calculate() is necessary to put the values into the file and then update all
dependendencies
Calculate()

// look how much the width can be reduced for the given power
number goOn = 1
number iteration_step = 0
write("--")
while ( goOn == 1 )
// check safety for root and flank of gear 1 and gear 2. If safety less than 1, stop iteration.
	if (ZPP[0].Fuss.SFnorm <= 1) then
		goOn = 0
		write("safety gear1, root " + ZPP[0].Fuss.SFnorm)
	end
	if (ZPP[0].Flanke.SH <= 1) then
		goOn = 0
		write("safety gear1, flank " + ZPP[0].Flanke.SH)
	end
	if (ZPP[1].Fuss.SFnorm <= 1) then
		goOn = 0
		write("safety gear2, root " + ZPP[1].Fuss.SFnorm)
	end
	if (ZPP[1].Flanke.SH <= 1) then
		goOn = 0
		write("safety gear2, flank " + ZPP[1].Flanke.SH)
	end
	if (goOn == 1) then
		ZR[0].b = ZR[0].b - 1
		ZR[1].b = ZR[1].b - 1
		Calculate()
	end

	write("safety - gear1, root " + ZPP[0].Fuss.SFnorm + " flank " + ZPP[0].Flanke.SH + " -
	gear2, root " + ZPP[1].Fuss.SFnorm + " flank " + ZPP[1].Flanke.SH)

	iteration_step = iteration_step + 1
end

ZR[0].b = ZR[0].b + 1
ZR[1].b = ZR[1].b + 1
Calculate()

write("width gear1:" + ZR[0].b + " with gear2: " + ZR[2].b)
write("number of iterations: " + iteration_step)
```

figure 2 source code of example1 - a simple script

The run of this small script by pressing the play button



will lead to the following result in the script output window:

```
27.03.2020 16:09:49: Start run skript: immediate
27.03.2020 16:09:49: --
27.03.2020 16:09:49: safety - gear1, root 2.496613883 flank 1.318490276 - gear2, root 2.406137472 flank 1.373240178
27.03.2020 16:09:49: safety - gear1, root 2.441794552 flank 1.30395656 - gear2, root 2.353304774 flank 1.358102954
27.03.2020 16:09:49: safety - gear1, root 2.386758793 flank 1.289218666 - gear2, root 2.300263491 flank 1.342753073
27.03.2020 16:09:49: safety - gear1, root 2.33089589 flank 1.274269343 - gear2, root 2.246425041 flank 1.327182984
27.03.2020 16:09:49: safety - gear1, root 2.274920737 flank 1.25910089 - gear2, root 2.192478408 flank 1.311384666
27.03.2020 16:09:49: safety - gear1, root 2.218832946 flank 1.243705116 - gear2, root 2.138423218 flank 1.295349587
27.03.2020 16:09:49: safety - gear1, root 2.162632126 flank 1.228073295 - gear2, root 2.084259096 flank 1.279068659
27.03.2020 16:09:49: safety - gear1, root 2.106317887 flank 1.212196114 - gear2, root 2.029985666 flank 1.262532184
27.03.2020 16:09:49: safety - gear1, root 2.049889835 flank 1.196063621 - gear2, root 1.975602547 flank 1.245729795
27.03.2020 16:09:49: safety - gear1, root 1.993347575 flank 1.179665154 - gear2, root 1.921109358 flank 1.228650386
27.03.2020 16:09:49: safety - gear1, root 1.93669071 flank 1.16298927 - gear2, root 1.866505718 flank 1.211282041
27.03.2020 16:09:49: safety - gear1, root 1.87991884 flank 1.146023666 - gear2, root 1.81179124 flank 1.193611946
27.03.2020 16:09:49: safety - gear1, root 1.823031564 flank 1.12875508 - gear2, root 1.756965539 flank 1.175626286
27.03.2020 16:09:49: safety - gear1, root 1.766028479 flank 1.111169182 - gear2, root 1.702028226 flank 1.15731014
27.03.2020 16:09:49: safety - gear1, root 1.708909179 flank 1.093250453 - gear2, root 1.64697891 flank 1.138647341
27.03.2020 16:09:49: safety - gear1, root 1.651673259 flank 1.074982037 - gear2, root 1.591817199 flank 1.119620335
27.03.2020 16:09:49: safety - gear1, root 1.593468854 flank 1.055938106 - gear2, root 1.535722102 flank 1.099785611
27.03.2020 16:09:49: safety - gear1, root 1.534768878 flank 1.036306364 - gear2, root 1.479149392 flank 1.079338666
27.03.2020 16:09:49: safety - gear1, root 1.476044613 flank 1.016287106 - gear2, root 1.422553274 flank 1.058488114
27.03.2020 16:09:49: safety - gear1, root 1.417296046 flank 0.9958569554 - gear2, root 1.365933734 flank 1.037209608
27.03.2020 16:09:49: safety gear1, flank 0.9958569554
27.03.2020 16:09:49: safety - gear1, root 1.417296046 flank 0.9958569554 - gear2, root 1.365933734 flank 1.037209608
27.03.2020 16:09:49: width gear1:25 with gear2: 0
27.03.2020 16:09:50: number of iterations: 21
27.03.2020 16:09:50: Finished skript: immediate
```

figure 3 output of the result window

So the user can see that it needed 21 iterations to find that with a width of 25mm the safeties of gear 1 and gear2 are just above 1.0.
With the width of 24 the flank safety of gear1 would be too low.

A list of module dependent variable names like ZR[0].b (for width), diameters, safeties and so on exists.

10.2 Example2: Export results to a csv file

In the second example a KISSsoft file is used to create several results by calling it several times with different input variables and save all into a .csv file.
Here first example - CylGearPair1 has to be loaded.
In this example (see figure 4 source code of example 2 - how to write into a .csv filefigure 4) the angle beta of both gears is changed in each iteration and in addition the normal module mn is increased.
In the first lines the Variable *ZS.ZeigerAufRadx* is set so the centre distance is adapted.

```
9 ZS.ZeigerAufRadx = -1 // enable beta input
```

With the call *open* in *line 12* a csv file is opened at the given path. There the absolute path has to be given.
close in *line 41* has to be called at the end that all data is written to the file and it is closed.
write in *line 33* takes a string as parameter which is then written to the file.
The local running variables *beta* and *mn* (defined in *line 20* and *21*) are used to set the model variables:

```
29      ZS.Geo.beta = nbeta/180*3.1415
30      ZS.Geo.mn = mn // in mm
31      Calculate()
```

```
//! WARNING: This is a generated comment to declare for which module and version this was generated. SKRIPTMODULE

// PURPOSE: This script shows the generation of a csv file from the given KISSsoft calculation file
// here from the KISSsoft file (e.g. example CylGearPair 1) beta and the modul are modified
// the resulting safeties are written with the input data into the csv file


ZS.ZeigerAufRadx = -1 // enable beta input: free axis distance

// create and open a file (to stream into)
string file_name = "C:\temp\variations.csv"
open(file_name)

// write the headers of the csv
write(file_name, "iteration, beta [°], module [mm], minimum root safety [-], minimum flank safety [-]\n")

// initialise beta and normal module
number iteration_step = 0

number nbeta = 1  // start with beta of 1 degree
number mn = 0
while (nbeta <= 26)    // run beta to 26 degrees
  alert("runnig beta:"+nbeta )

  // run the normal module from 6 to 10
  mn = 6
  while (mn <= 10)

    ZS.Geo.beta = nbeta/180*3.1415
    ZS.Geo.mn = mn // in mm
    Calculate()
    // write calculated values
    write(file_name, "" + iteration_step + "; " + nbeta + "; " + ZS.Geo.mn + "; " + ZPP[0].Fuss.SFnorm + "; " + Z

    mn = mn + 1
    iteration_step = iteration_step + 1
  end
  nbeta = nbeta + 1
end

close(file_name)
```

figure 4 source code of example 2 - how to write into a .csv file

If you run the script you will find a the location c:\temp the following .csv file:

```
Iteration; beta [°]; module [mm]; minimum root safety [-]; minimum flank safety [-]
0; 1; 6; 7.734805879; 2.413369711
1; 1; 7; 7.734805879; 2.413369711
2; 1; 8; 7.734805879; 2.413369711
3; 1; 9; 7.734805879; 2.413369711
4; 1; 10; 7.734805879; 2.413369711
5; 2; 6; 7.648818489; 2.405668419
6; 2; 7; 7.648818489; 2.405668419
7; 2; 8; 7.648818489; 2.405668419
8; 2; 9; 7.648818489; 2.405668419
9; 2; 10; 7.648818489; 2.405668419
10; 3; 6; 7.574431264; 2.399214137
11; 3; 7; 7.574431264; 2.399214137
12; 3; 8; 7.574431264; 2.399214137
13; 3; 9; 7.574431264; 2.399214137
14; 3; 10; 7.574431264; 2.399214137
15; 4; 6; 7.511207227; 2.393985257
16; 4; 7; 7.511207227; 2.393985257
…
```