Syntax of formulas and expressions of the parser SParser (V2)

Table of contents

1 General explanations. 1
2 Mathematical functions 4
2.1 Logical functions 5
3 Functions for processing character strings (string functions). 6
4 Conditional expressions 13
4.1 IF-THEN-ELSE construct 13
4.2 SWITCH-CASE-DEFAULT construct 14
5 Conversion and special functions 17
6 Bindings 19
7 Irrelevant branches 21
8 Error messages 22

1 General Explanations

The parser is used to calculate, evaluate and syntactically check formulas and expressions.
It is type-sensitive, i.e. it distinguishes between the following (data) types:

- Numbers
- Character strings (texts)

It also supports variables that can take values of both types and only act as placeholders.
An expression to be evaluated may contain both numbers and character strings.

Numbers

- The decimal separator is a point.
- Leading zeros (e.g. for 0.5) may be omitted.
- Thousands separators may not be specified
- Scientific notation is permitted, e.g:

  1.5e4→ 15000
  3e-5→ 0.00003
  5E2→ 500

Character strings

- Character strings are characterized by enclosing any characters in double quotation marks ("   "), e.g:
  "string"
- Character strings with support for "escape sequences" (\\ \n \t \r \' \") must b e enclosed in quotation marks, e.g.: 'line1\nline2'
- a distinction is made between upper and lower case (case-sensitive)

Variables

Variables are placeholders and can contain numbers or strings.

Variable names always begin with an upper or lower case letter or an underscore (_). They may be followed by any number of upper or lower case letters, numbers and underscores. The following function names or keywords are not permitted as variable names:

AND	OR	NOT	XOR		
IF	THEN	ELSE	SWITCH	CASE	DEFAULT

Functions

In principle, function names are subject to the same rules as variables. A function call can be recognized by the following brackets ().
Function parameters can be separated by a semicolon ; or a comma ,.

Further features of the parser

- With **all** functions, the specified values / parameters are **not** changed.

- All functions can be combined and nested as required while adhering to the respective syntax.

- Spaces and line breaks are ignored except in character strings.

The following symbols and fonts are used to make this document easier to read:

Symbols

Note	👉	Further information
Tip	💡	Useful tips & tricks
Notes	📌	Notes for consolidation and clarification
Example: Example	**Example of**	
Syntax	**Syntax**	

Font styles and font colors

`Text in serif font`	Expressions as they can be entered directly ("source code")

Italic text in blue color	Parameters or expressions of type **ZAHL**
Italic text in green color	Parameters or expressions of the type **character string**
italic text in grey color	Parameters or expressions of **any** type

Function parameters in square brackets [ ] are optional.

2 Mathematical functions

The most common basic mathematical functions are supported in compliance with the algebraic rules (bracket priority, dot before dash, etc.).

Basic functions	+	Addition
	-	Subtraction
	*	Multiplication
	/	division
	^	Power
	SQRT(*number*)	Square root
	+ -	Sign
Comparison functions	<	less than
	>	greater than
	<=	less than or equal to
	>=	greater than or equal to
	=	is equal to
	<>	unequal
Trigonom. Functions	SIN(*number*)	Sine (argument in degrees)
	ARCSIN(*number*)	Arc sine
	COS(*number*)	Cosine (argument in degrees)
	ARCCOS(*number*)	Arc cosine
	TAN(*number*)	Tangent (argument in degrees)
	ARCTAN(*number*)	Arc tangent
Further mathematical functions	EXP(*number*)	Exponential function (e-function)
	LN(*number*)	Natural logarithm
	MOD(*number*)	Modulo function, decimal places are omitted
	PREC(*number*)	Pre-decimal places are omitted
	ABS(*number*)	Amount

The comparison functions return the value 1 if the statement applies and the value 0 otherwise.

Examples

Expression	The result
`2+3`	5
`2-3`	-1
`2*3`	6
`2/3`	0.666667
`2^3`	8
`SQRT(2)`	1.414214
`--2`	2
`2<3`	1
`2>3`	0
`2<=3`	1

`2>=3`	0
`2=3`	0
`2<>3`	1
`SIN(30)`	0.5
`ARCSIN(.5)`	30
`COS(60)`	0.5
`ARCCOS(.5)`	60
`TAN(45)`	1
`ARCTAN(1)`	45
`EXP(1)`	2.718282
`LN(2.718282)`	1
`MOD(2.3)`	2
`PREC(2.3)`	0.3
`ABS(-2)`	2

2.1 Logical Functions

AND	And	Both conditions must be fulfilled
OR	OR	At least one of the two conditions must be fulfilled
XOR	Exclusive Or	Exactly one of the two conditions must be met
NOT	Not	Negates the argument

A condition is fulfilled if the value of the expression to be evaluated is different from 0.

Examples

Expression	Result
`0 AND 1`	0
`2 AND 3`	1
`0 OR 1`	1
`2 OR 3`	1
`0 XOR 1`	1
`2 XOR 3`	0
`NOT 0`	1
`NOT 2`	0

3 Functions for processing character strings (string functions)

In addition to numerical values, texts or character strings can also be used. The following functions are available for this purpose:

Concatenation	&
Case-sensitive	UCASE
	LCASE
substring / extraction	LEFT
	RIGHT
	MID
LENGTH	LEN
	ISEMPTY
SEARCH	FIND
	RFIND
Manipulation	REPLACE
	INSERT
	LTRIM
	RTRIM

In addition to these functions, the same comparison operators can be used for character strings as for numbers.

<   >   <=   >=   =   <>

The comparison functions are also case-sensitive, i.e. "a" is not equal to "A", but "a" is greater than "A".

Examples

Expression	Result
`"Hans"< "Hugo"`	1
`"hans"< "Hugo"`	0
`"Hans"> "Hugo"`	0
`"Hans"<= "Hugo"`	1
`"Hans">= "Hugo"`	0
`"Hans"= "Hugo"`	0
`"Hans"<> "Hugo"`	1

The following variables are defined for the following examples in this chapter:

Name	"Homag"
PrgName1	"Platte01.mpr"

&

Syntax *Text1* & *Text2*

Joins two strings together so that *text2* is appended directly to *text1* is appended.

The function can concatenate any number of texts.

Examples

Expression	Result
`Name & " Holzbearbeitungssysteme AG"`	Homag Woodworking Systems AG
`"c:\machine1\" & "A1\"`	c:\machine1\A1\
`"c:\machine1\A1\" & "mp4\" & PrgName1`	c:\machine1\A1\mp4\Platte01.mpr

UCASE

(upper **case**)

Syntax UCASE(*Text1*)

Converts all lower case letters in the string *Text1* to upper case letters. Other characters remain unchanged.

Examples

Expression	The result
`UCASE("TEST")`	TEST
`UCASE(Name)`	HOMAG
`UCASE("Part-0815-A")`	PART-0815-A

LCASE

(lower **case**)

Syntax LCASE(*Text1*)

Converts all upper case letters in the string *Text1* to lower case letters. Other characters remain unchanged.

Examples

Expression	Result
`LCASE("TEST")`	test
`LCASE(name)`	homag
`LCASE("part-0815-A")`	part-0815-a

UCASE or LCASE can be used to make the parser insensitive to upper and lower case when making comparisons: `LCASE("Homag") = LCASE("HOMAG")`

LEFT

Syntax LEFT(*Text1*; *length*)

Returns the number of characters specified by *length* from the left of the string *Text1*.

Examples

Expression	Result
`LEFT("Woodworking systems"; 4)`	Wood
`LEFT(PrgName1; 8)`	Panel01

RIGHT

Syntax RIGHT(*Text1*; *length*)

Returns the number of characters specified by *length* from the right in the string *Text1*.

Examples

Expression	Result
`RIGHT("Holzbearbeitungssysteme AG"; 2)`	PLC
`RIGHT(PrgName1; 3)`	mpr

MID

(middle)

Syntax MID(*Text1*; *start position*[; *length*])

Returns a part of the character string *Text1*. The number of characters to be skipped is specified with the *start position* parameter. If the *length* parameter is specified, it determines the length of the substring. If it is not specified, the rest of the string is returned.

Examples

Expression	Result
`MID("Homag Holzbearbeitungssysteme AG"; 6)`	Woodworking systems AG
`MID("Homag Holzbearbeitungssysteme AG"; 6; 4)`	Wood
`MID(PrgName1; 6)`	01.mpr
`MID(PrgName1; 6; 2)`	01

LEN

(length)

Syntax LEN(*Text1*)

Returns the number of characters in the string *Text1*.

Examples

Expression	Result
`LEN("")`	0
`LEN("Homag Holzbearbeitungssysteme AG")`	32
`LEN(PrgName1)`	12

ISEMPTY

Syntax ISEMPTY(*Text1*)

Checks whether the character string *Text1* is empty. If it is empty (""), 1 is returned, otherwise 0.

The function
ISEMPTY(*Text1*)
is equivalent to: LEN(*Text1*)
= 0

Examples

Expression	Result
`ISEMPTY("")`	1
`ISEMPTY("Homag Holzbearbeitungssysteme AG")`	0
`ISEMPTY(PrgName1)`	0

FIND

Syntax FIND(*haystack*; *needle* [;*start position*])

Searches the string *haystack* for occurrences of the string *needle* and returns its position. The *start position* parameter can be used to specify a number of characters to be skipped. If it is not specified, the search starts at the first character. If the character string *needle* to be searched for does not occur in *haystack*, -1 is returned.

Examples

Expression	Result
`FIND("Homag Holzbearbeitungssysteme AG"; "H")`	0
`FIND("Homag Holzbearbeitungssysteme AG"; "H"; 1)`	6
`FIND("Homag Holzbearbeitungssysteme AG"; "Wood")`	6
`FIND("Homag Holzbearbeitungssysteme AG"; "HOLZ")`	-1

RFIND

(reverse **find**)

Syntax RFIND(*haystack*; *needle* [;*start position*])

This function behaves in the same way as FIND, except that the *haystack* string is searched from back to front.

Examples

Expression	Result
`RFIND("Homag Holzbearbeitungssysteme AG"; "H")`	6
`RFIND("Homag Holzbearbeitungssysteme AG"; "H" ; 5)`	0
`RFIND("Homag Holzbearbeitungssysteme AG"; "HOLZ")`	-1
`RFIND("c:\machine1\a1\mp4"; "\")`	14

If the number of characters to be skipped is to be calculated from the last character, this can be done with the expression
LEN(*haystack*) - *number of characters*
for the *start position* parameter.

REPLACE

Syntax REPLACE(*Text1*; *Old*; *New*)

Replaces all *Old* substrings in the string *Text1* with *New*. The strings *Old* and *New* can be of any length.

Examples

Expression	Result
`REPLACE("c:\machine1\a1\mp4"; "\"; "/")`	c:/machine1/a1/mp4
`REPLACE(PrgName1; "01"; "02")`	Disk02.mpr
`REPLACE(PrgName1; "0"; "0000")`	Plate00001.mpr
`REPLACE(PrgName1; "01"; "")`	Disk.mpr

The deletion of substrings can be achieved with an empty string ("") for the parameter *New*.

INSERT

Syntax INSERT(*Text1*; *Position*; *New*)

Inserts the character string *New* into the character string *Text1* at the position *Position*.

Examples

Expression	result
`INSERT("123456"; 3; "---")`	123---456
`INSERT (PrgName1; 6; "_")`	Plate_01.mpr

LTRIM

(left **trim**)

Syntax LTRIM(*Text1*)

Removes all spaces at the beginning of the string *Text1*.

Examples

Expression	result
`LTRIM (name)`	Homag
`LTRIM ("   Text")`	Text

RTRIM

(right **trim**)

Syntax RTRIM(*Text1*)

Removes all spaces at the end of the string *Text1*.

Examples

Expression	result
`RTRIM (name)`	Homag
`RTRIM ("c:\machine1\a1\mp4        ")`	c:\machine1\a1\mp4

4 Conditional expressions

The parser supports two constructs for conditional expressions.
The IF-THEN-ELSE construct and the SWITCH-CASE-DEFAULT construct. They are described in more detail in the following two chapters.

4.1 IF-THEN-ELSE Construct

Syntax IF *Condition* THEN *IfFulfilled* ELSE *Else*

If the condition *Condition* is fulfilled, i.e. its value is not equal to 0, the value of the expression *IfFulfilled* is returned, otherwise the value of the expression *Else* is returned. The *IfFulfilled* and *Else* expressions can be of any type. The type of the returned value is identical to that of the respective expression.
The IF-THEN-ELSE construct may be nested as often as required, i.e. the expressions *IfFulfilled* or *Else* may themselves be an IF-THEN-ELSE construct.

The following variables are defined for the following examples in this chapter:

L	1200
B	800
Z	25
As a string	1
k	3

Examples

Expression	Result
`IF 0 THEN 12 ELSE 34`	34
`IF 1 THEN 12 THEN 34`	12
`IF L>800 THEN 100 THEN 200`	100
`IF B>=850 THEN B/3 ELSE B/2`	400
`IF L<B THEN "portrait format" ELSE "landscape format"`	landscape
`IF AsString THEN "123" ELSE 123`	123
`IF Z<0 THEN -1 ELSE IF Z=0 THEN 0 ELSE 1` *	1
`IF Z<=0 THEN IF Z=0 THEN 0 ELSE -1 ELSE 1` *	1
`IF k=1 THEN "A" ELSE IF k=2 THEN "B" ELSE IF k=3 THEN "C" ELSE "D"`	C

The two examples marked with an asterisk lead to the same result.

4.2 SWITCH-CASE-DEFAULT construct

With this construct, an expression can be compared multiple times without having to list it multiple times, i.e. in contrast to the IF-THEN-ELSE construct, the same expression (switch argument) is always used for the evaluation.

Syntax SWITCH *Reference* CASE *ComparisonX* THEN *ValueX* DEFAULT *Else*

The keyword SWITCH is followed by the expression *Reference* to be compared with all branches. This can be followed by any number of CASE branches, but at least one. If the value of the expression *Reference* corresponds to that of the expression *ComparisonX*, the value of the expression following the keyword THEN is returned. If none of the branches match, the value is returned after the final keyword DEFAULT.

In addition, each CASE branch may contain several CASE comparisons. This allows complex branches to be realized as in the following example:

```
SWITCH Reference
        CASE comparisonA1 CASE comparisonA2 CASE comparisonA3
        THEN ValueA
        CASE ComparisonB1 CASE ComparisonB2 CASE ComparisonB3 CASE
        ComparisonB4
        THEN ValueB
        CASE ComparisonC1 CASE ComparisonC2
        THEN ValueC
DEFAULT Else
```

When evaluating the branches, a comparison is made as to whether the value of the expression
Reference is **equal to** the value of the respective expression *ComparisonX*.

The types of the comparison values must be identical to that of the reference value.
The types of the expression values to be returned can be selected as required.

Examples

Expression	Result
`SWITCH 20 CASE 10 THEN "A" DEFAULT "XYZ"`	XYZ
`SWITCH 20 CASE 10 THEN "A" CASE 20 THEN "B" DEFAULT "XYZ"`	B
`SWITCH 20 CASE 10 CASE 20 THEN "A" DEFAULT "XYZ"`	A
`SWITCH "R" CASE "A" THEN 1 CASE "B" THEN 2 DEFAULT 9`	9

Further examples with tabular representation of the results to illustrate the branches:

Example

SWITCH k CASE 1 THEN A CASE 2 THEN B CASE 3 THEN C DEFAULT D

Depending on the value of k, the following value is determined:

k	Result
1	A
2	B
3	C
otherwise	D

Example

SWITCH k CASE 1 CASE 3 CASE 5 THEN U CASE 2 CASE 4 THEN G DEFAULT X

Depending on the value of k, the following value is determined:

k	Result
1, 3 or 5	U
2 or 4	G
otherwise	X

---

example

SWITCH k CASE 1 CASE 3 CASE 5 THEN U CASE 2 CASE 4 THEN G DEFAULT X

Depending on the value of k, the following value is determined:

k	Result
1, 3 or 5	U
2 or 4	G
otherwise	X

Intervals

If the expression *Reference* is to be compared with a whole range of values, the keyword CASE can be followed by **intervals.** An interval is specified with the following syntax (two points in direct succession):

Start ... *End*

The evaluation of an interval is positive if the expression *Reference* is equal to or between the values of the expressions *Start* and *End*.

If the expression *Start* ends with a constant integer, it must be followed by at least one space, otherwise the first point is interpreted as a decimal separator.

The expression
```
SWITCH width CASE 100 ... 500 THEN 2 DEFAULT 3
```
is equivalent to
```
IF 100<= width AND width<= 500 THEN 2 ELSE 3
```

Example:

Example

SWITCH LEFT(Name, 1) CASE "A" .. "F" THEN N1 CASE "G" ... "L" THEN N2 DEFAULT N3

The following value is determined depending on the name:

First letter of Name	Result: A to F
A to F	N1
G to L	N2
otherwise	N3

The IF-THEN-ELSE construct and the SWITCH-CASE-DEFAULT construct can be nested and mixed together as often as required.

5 Conversion and Special functions

Conversion functions

STR

Syntax STR(*expression*)

Converts the expression *Expression* into a character string.

Examples

Expression	Result string
`STR(123)`	123
`STR(4+ 5.6)`	9.6
`STR("Text")`	text

VAL

Syntax VAL(*Expression*)

Converts the expression *Expression* into a number. If it is a character string that does not begin with digits, signs or decimal separators, 0 is returned (the evaluation of the character string is aborted as soon as an invalid character occurs).

Examples

Expression	Result: VAL
`VAL("234")`	234
`VAL("-.5")`	-0.5
`VAL("Text")`	0
`VAL("15 plates")`	15

STR() and VAL() accept both character strings and numbers as parameters. This allows the value of a variable, which may accept values of different types, to be used with type safety.

Special functions

VARDEF

Syntax VARDEF (*variable name*)

Checks whether the variable *variable name* is defined. If it is defined, 1 is returned, otherwise 0.

The following variable is defined for the following examples:

X	100

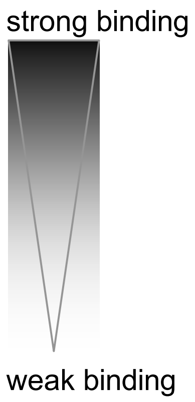
Examples

Expression	Expression result
`VARDEF("X")`	1
`VARDEF("Y")`	0
`IF VARDEF("X") THEN X ELSE 123`	100
`IF VARDEF("Y") THEN Y ELSE 123`	123

6 Bindings

To comply with the algebraic rules (e.g. "dot before dash"), the parser uses the binding levels and binding directions listed in the following table. The strongest binding level is at the top of the table.

Operators	Binding direction
Functions and brackets	
(sign) **-+** **NOT**	→
^	←
*** /**	→
+	→
< <= >= >	→
= <>	→
AND	→
XOR	→
OR	→
*Conditional expressions (*IF *and* SWITCH*)*	



Binding levels

In expressions that contain combinations of operators, the strongest bindings are calculated first. To explicitly specify the order, the expressions can be put in brackets as often as required.

Examples

Expression	Result	Binding
`1+2*3`	7	$\boxed{1+\boxed{2*3}}$
`(1+2)*3`	9	$\boxed{\boxed{1+2}*3}$
`1+2*3^4`	163	$\boxed{1+\boxed{2*\boxed{\text{3\^{}4}}}}$
`NOT a AND b`	$\bar{a}\wedge b$	$\boxed{\boxed{\text{NOT a}}\text{AND b}}$
`NOT(a AND b)`	$\overline{a\wedge b}$	$\boxed{\text{NOT }\boxed{\text{a AND b}}}$
`a OR b AND c`	$a\vee b\wedge c$	$\boxed{\text{a OR }\boxed{\text{b AND c}}}$
`a OR b AND c XOR d`	$a\vee b\wedge c\oplus d$	$\boxed{\text{a OR }\boxed{\boxed{\text{b AND c}}\text{ XOR d}}}$

`a< b= c< d`	*(the two expressions are equivalent)*	a<b= c<d
`a<b AND c<d OR a>=b AND c>=d`		a<b AND c<d OR a>=b AND c>=d
`L - p1 / 2> p_min OR p2 * -2< p3`		L- p1/2 >p_min OR p2* -2 <p3
`"c:\" & folder1 & file5`		"c:\" & folder1 & file5
`ARCSIN(SQRT(2)/2)`	45	ARCSIN(SQRT(2) /2)
`IF L/2<= min_X THEN "T123" ELSE "T" & STR(TNr)`	IF L/2<= min_X THEN "T123" ELSE "T" & STR(TNr)	

Binding direction

If several operators of the same binding level follow one another, the binding direction determines the order of evaluation. For all operators except the power ^, the binding direction is left to right. The binding direction of the power is from right to left in woodWOP6 mode (PM_WW6) and from left to right in woodWOP5 mode (PM_WW5).

Examples

Expression	Result	Binding
`1+2+3`	6	1+2 +3
`4-3-2-1`	-2	4-3 -3 -1
`5+4-3+2-1`	7	5+4 -3 +2 -1
`4^3^2`	262144	4^ 3^2
`1.1^1.2^1.3^1.4`	1.13203	1.1^ 1.2^ 1.3^1.4

7 Irrelevant branches

When evaluating some expressions, the result is known at an early stage, which means that the rest of the expression or certain parts of it no longer need to be evaluated.

Example: `IF Bed1 THEN (L-2*RX)/2 ELSE (B-2*RY)/2`

Example

If the condition Bed1 is fulfilled, only the part (L-2*RX)/2 is relevant and the part (B-2*RY)/2 does not have to be calculated. If the condition is not fulfilled, vice versa.
The parser recognizes this and only checks the irrelevant branches for syntactic correctness, but does not carry out any evaluation.
The advantage of this can be seen in the following example.

Example: x `IF X<>0 THEN 1/X ELSE 0`

If the parser were to carry out the evaluation in the 1/X branch, even if the condition X<>0 did not apply, the error "Division by zero" would be reported, although the branch is irrelevant for precisely this condition and this is the intention for this particular expression.

Irrelevant branches can occur in the following constructs and operators:

IF-THEN-ELSE	Depending on whether the condition is fulfilled or not, only the relevant expression is evaluated.
SWITCH-CASE-DEFAULT	As soon as the first comparison (CASE argument) is positive, all subsequent comparisons, both in the same CASE branch and in all subsequent branches, are no longer evaluated, are no longer evaluated.
AND	If the 1st parameter is 0, the expression can no longer be fulfilled, which is why the 2nd parameter is no longer evaluated. Parameter is no longer evaluated.
OR	If the 1st parameter is not equal to 0, then the expression is already fulfilled, regardless of the 2nd parameter, which is then no longer evaluated.

Examples

`IF VARDEF("X") THEN X ELSE 0`
`IF X<= 0 THEN 5 ELSE LN(X)`
`SWITCH X CASE 0 THEN 0 DEFAULT 1/X`
`SWITCH X CASE -1.0 .. 1.0 THEN ARCSIN(X) DEFAULT 180`
`SWITCH k CASE 1 CASE 5 THEN 1.2*L CASE 6 ..10 THEN 1.5*L2 DEFAULT 1.8*L3`
`VARDEF("X") AND X<>0`
`X>= L OR X>= B`

8 Error messages

Mathematical errors

Mathematical errors are only recognized during the calculation / evaluation of the expression, i.e. after the syntactic check.

Error text	**Division by zero**
Error number	101
Error code	SPSC_M_DIVISION_BY_ZERO
Description	The divisor is zero. (a^{-b} can also be written as $\frac{1}{a^b}$.)
Examples	`10 / 0`
	`5 / (4-2^2)`
	`0^-2`

Error text	**Negative argument of the root function**
Error number	102
Error code	SPSC_M_NONREAL_RESULT
Error description	The square root of a negative number does not yield a real number. ($a^{\frac{p}{q}}$ can also be written as $\sqrt[q]{a^p}$).
Examples	`SQRT(-2)`
	`SQRT(5-8)`
	`-10^2.5`

Error text	**Undefined result**
Error number	103
Error code	SPSC_M_UNDEFINED_RESULT
Description	The function is not defined for this value.
Examples	`TAN(90)`
	`TAN(-10*27)`

Error text	**Value outside the definition range**
Error number	104
Error code	SPSC_M_VALUE_OUT_OF_DOMAIN
Description	The function is not defined for this value. The arc functions are only defined from -1.0 to +1.0.
Examples	`ARCSIN(2)`
	`ARCCOS(-2)`

Error with variables

Error text	**Unknown variable □**
Error number	201
Error code	SPSC_V_VAR_UNKNOWN
Description	The specified variable does not exist.
Examples	`L/2`
	`0.8 * B+ Offset`

Error text	No variable list available
Error number	202
Error code	SPSC_V_NO_VARLIST_SPECIFIED
Description	The expression contains one or more variables, but no variable list is available to the parser. This error is generally not an input error by the user, but indicates a problem with the program.
Examples	`L/2`
	`0.8 * B+ Offset`

Error text	Error when determining the variable □
Error number	203
Error code	SPSC_V_INQUIRY_FAILURE
Description	The expression contains one or more variables. The determination of the value of this variable could not be successfully completed. This error is generally not an input error by the user, but indicates a problem with the program.
Examples	`L/2`
	`0.8 * B+ Offset`

Type error

Type errors occur when a function or operator encounters parameters of the wrong data type. They are also only detected during the calculation / evaluation, i.e. after the syntactic check.

Error text	Incompatible data types
Error number	301
Error code	SPSC_T_INCOMPATIBLE_TYPES
Description	Two or more parameters that correlate with each other are of different types.
Examples	`2< "A"`
	`SWITCH 12 CASE "A" THEN 0.1 DEFAULT 0.9`
	`SWITCH "Text" CASE 10 ... 20 THEN 0.1 DEFAULT 0.9`

Error text	Incorrect data type
Error number	302
Error code	SPSC_T_WRONG_TYPE
Description	The type of the parameter does not correspond to the expected type.
Examples	`1+ "x"`
	`0 OR "Text"`
	`IF "Text" THEN 2 ELSE 3`
	`SIN("x")`
	`VARDEF(0)`
	`LEFT(2, "Text")`
	`REPLACE("TextA", "A", 1)`
	`ISEMPTY (0)`

Error with functions

Error text	**Function□ unknown**
Error number	401
Error code	SPSC_F_FUNCTION_UNKNOWN
Error description	The specified function is not known.
Examples	`MYFUNCTION()`
	`ANOTHERFUN(1;2)`
	`1+ Sin(30)`

Error text	**Too many arguments**
Error number	402
Error code	SPSC_F_TOO_MANY_ARGUMENTS
Description	An attempt was made to pass more parameters to a function than permitted.
Examples	`SIN(30; 45)`
	`UCASE("Test"; 1)`
	`SQRT(8; 3)`

Error text	**Too few arguments**
Error number	403
Error code	SPSC_F_TOO_FEW_ARGUMENTS
Description	An attempt was made to pass fewer parameters to a function than permitted.
Examples	`SIN()`
	`LEFT("Text")`

Error text	**Incorrect number of arguments**
Error number	404
Error code	SPSC_F_WRONG_ARGUMENT_COUNT
Description	The number of transferred parameters does not match any function. This error can only occur if several overloads are registered for a function name.
Examples	`MYFUNC(1;2)`

	`(if the function MYFUNC had two overloads,`
	`for example with one and with three parameters)`

Error text	**The types of the arguments do not match any function**
Error number	405
Error code	SPSC_F_PARAMS_MATCH_NO_OVERLOAD
Description	The types of the transferred parameters do not match any function. This error can only occur if several overloads are registered for a function name, which only differ in the parameter types but not in the number of parameters. parameter types, but not in the number of parameters.
Examples	`MYFUNC(1; "ABC")`

	`(if the function MYFUNC had two overloads,`
	`for example with the signatures (double; double) and`
	`(string; string)`

Application-specific errors

Error text	Application-specific error
Error number	500
Error code	SPSC_CUSTOM_FAILURE
Error description	Error in application-specific implementations of the variable table and/or functions. In this case, further information can be obtained via the methods `getCustomErrorNumber()`, `getCustomErrorText` and `getCustomErrorObject() methods.`
Examples	`MYFUNCTION(1;2;3)`

Syntactic errors

Syntactic errors occur with missing or incorrect keywords, invalid characters, ambiguous input, etc.

Error text	Symbol *1* found instead of *2*
Error number	1120
Error code	SPSC_MISMATCHED_TOKEN
Description	The specified symbol *1* is not permitted at this point. Symbol *2* is expected instead. This error usually indicates incorrect keywords or incorrect bracketing.
Examples	`IF 0 ELSE 10`
	`3*((1+2)`
	`SWITCH L CASE 1 THEN 10 ELSE 20`

Error text	Unexpected symbol *1*
Error number	1140
Error code	SPSC_NO_VIABLE_ALT
Description	The specified symbol *1* is invalid at this point. This error often indicates a missing parameter or operator.
Examples	`2 ** 3`
	`0.5.0`
	`L 2`
	`SWITCH CASE 1 THEN 10 DEFAULT 0`

Error text	Incomplete expression
Error number	1141
Error code	SPSC_NO_VIABLE_ALT_EOF
Description	The expression ends without satisfying a valid expression. This error indicates a truncated expression.
Examples	`1 >`
	`B -`
	`IF`
	`IF x THEN`
	`SWITCH x CASE 1`

Error text	Unexpected character
Error number	1220
Error code	SPSC_TOKEN_STREAM_RECOGNITION_FAILURE
Description	The expression contains invalid or incorrectly placed characters.
Examples	`#12`
	`100+ Overhang`
	`0.5.`

Other errors

Error text	Unknown error
Error number	10
Error code	SPSC_UNKNOWN_ERROR
Description	Serious problems occurring during the analysis of the printout, e.g. lack of memory (stack overflow), can cause this error.

Space for notes