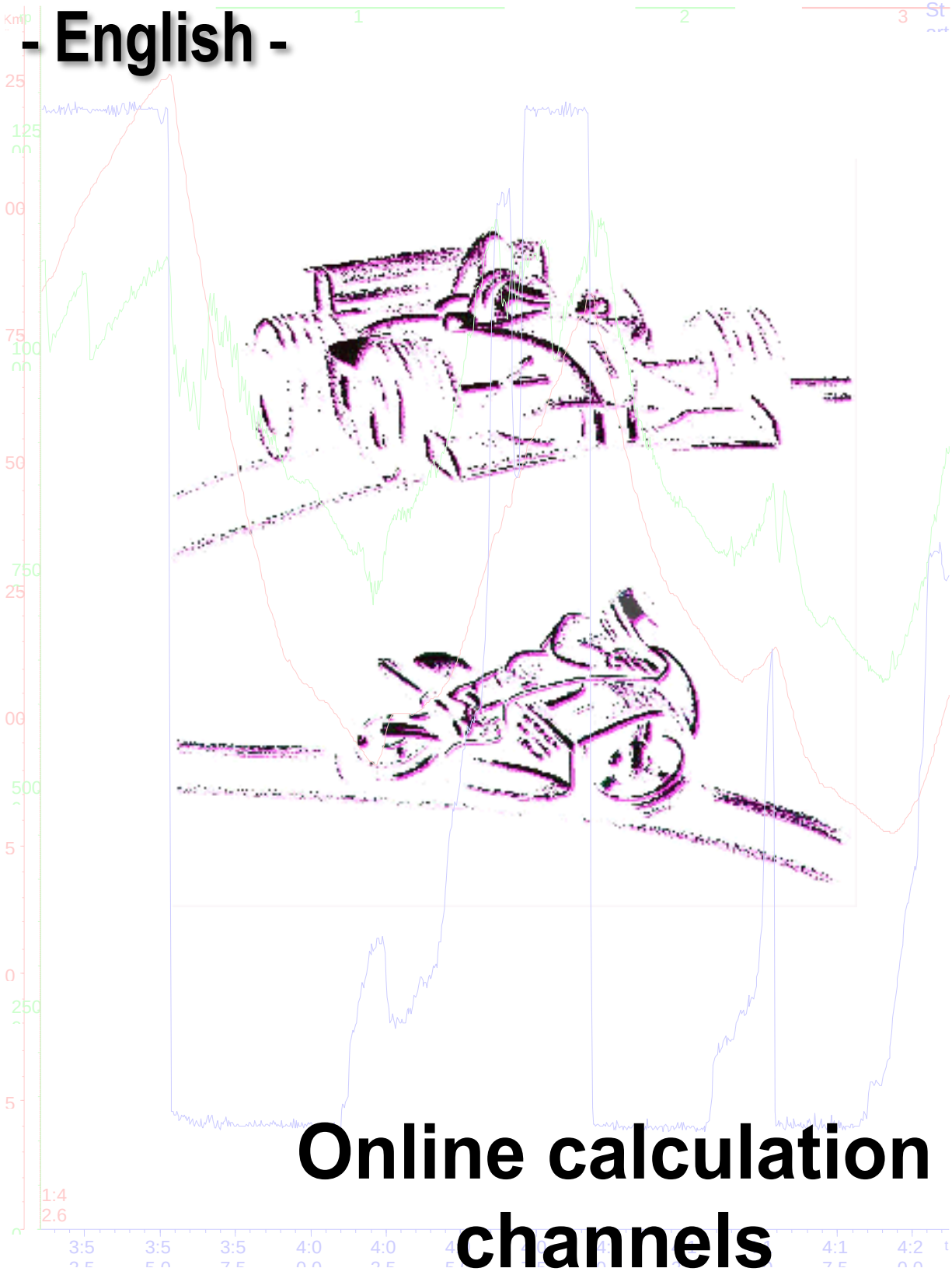
2d-datarecording.com

- English -



Online calculation channels

modified 19/04/18

Content

1 INTRODUCTION 3
2 CALCULATION FUNCTIONS OF THE CALC-MODULE 4
2.1 ARITHMETIC EXPRESSIONS 4
2.2 RELATIONAL EXPRESSIONS 5
2.3 LOGICAL EXPRESSIONS 5
2.4 BIT EXPRESSIONS 5
2.5 TRIGONOMETRIC EXPRESSIONS 5
2.6 MIN-MAX EXPRESSIONS 6
2.7 FILTER EXPRESSIONS 7
2.8 VARIABLES 7
2.9 CONDITIONAL EXPRESSIONS 7
2.10 ASSIGNMENT 7
2.11 SHIFTING 7
2.12 TABLE EXPRESSIONS 7
2.13 OTHERS 8
2.14 SYNTAX 9
3 IMPORT AND EXPORT OF CALCULATION DEFINITIONS 10

Symbols used in the text

These paragraphs contain tips and practical advice for working with the 2D system

In the paragraphs highlighted with this symbol, you will find additional information and it is very important that you follow the instructions given.

Documentation reference

- A user manual reference number is provided so the user can seek further assistance

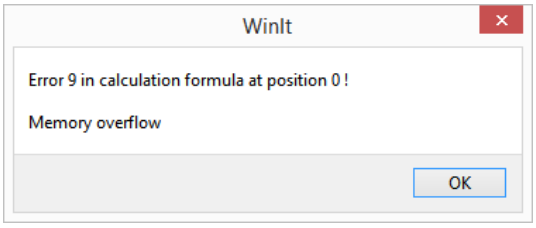
1 Introduction

In most 2D modules you can find some online calculation channels (calc channels). Calc channels can have a lot of different purposes. They can be used to put different channels together in one CAN send identifier, execute commands to get information from the module or to create new channels. Their information will be available online in your system and can be recorded or also be sent do different devices.

This manual will help you find the commands to program the calc channels and will also give a few examples on how calc channels can be used. If you want to program cal files to create calculated channels inside your measurement, please refer to the CalcTool manual[1].

Please be aware that, depending on the internal calc library, there may be differences in the modules. Therefore some commands may not work. Also in some modules the m-variables are reset after a power cycle.

If the calculation channel has the value “-1” or “65535” (depending on if the digits are signed or not) even though you expect other values, please check the channel. If you click on it and get the following message, you´re using too many calculation nodes.



You can also verify it by turning off other channels. To try to solve the problem, try to reduce the overall calculation operations.

[1] The manual can be downloaded from: 2d-datarecording.com/en/downloads/manuals/ ⇒ CalcTool

2 Calculation Functions of the Calc-Module

Evaluating Calc-formulas

The formula for a channel is evaluated in a strict time sequence given by the sampling rate of that channel. E.g., if the rate is set to 200Hz then every 1/200 seconds (cycle time) the formula is evaluated once and the result value is stored.

2.1 Arithmetic expressions

Sum	*expr1* + *expr2*	Result is the sum of *expr1* and *expr2*.	`3 + 4`
			`#4 + 10.7`
			`#speed + #4`

Difference	*expr1* – *expr2*	Result is the difference between *expr1* and *expr2*.	`3 - 4`
			`#4 - 10.7`
			`#speed - #4`

Product	*expr1* * *expr2*	Result is the product of *expr1* and *expr2.*	`3 * 4`
			`#4 * 10.7,`
			`#speed * #4`

Division	*expr1* / *expr2*	Result is *expr1* divided by *expr2*.	`3 / 4`
			`#4 / 10.7`
			`#speed / #4`

Integer division	`div(`*expr1, expr2*`)`	Result is *expr1/expr2* with the remainder discarded.	`div(3, 4)`
			`div(#4, 10.7)`
			`div(#speed, #4)`

Modulo	*expr1* % *expr2*	Result is the remainder of *expr1/expr2.*	`3 % 4`
			`#4 % 10.7`
			`#speed % #4`

Power	*expr1* ^ *expr2*	Result is *expr1* to the power of *expr2* (*expr1*^{expr2}).	`2 ^ 8`
			`#4 ^ 2`
			`2 ^ #4`

Square root	`sqrt(`*expr*`)`	Result is the square root of *expr* (*expr* must be positive).	`sqrt(2)` `sqrt(#4)`

Logarithm	`log10(`*expr*`)`	Result is the base-10 logarithm of *expr* (*expr* must be positive).	`log10(2)`
			`log10(#4)`
			`log10(#speed)`

Natural logarithm	`ln(`*expr*`)`	Result is the natural logarithm (to base *e*) of *expr* (*expr* must be positive).	`ln(2)`
			`ln(#4)`
			`ln(#speed)`

Sign	`sig(`*expr*`)`	Result is the sign of *expr*.
		1, if *expr* is positive;
		-1 if *expr* is negative;
		0, if *expr* is zero.
			`sig(-2)`
			`sig(#4)`
			`sig(#speed)`

Absolute value	`abs(`*expr*`)`	Result is the absolute value of *expr*.	`abs(-2)`
			`abs(#4)`
			`abs(#speed)`

Derivation	`der(`*expr*`)`
	`deriv(`*expr*`)`
		Result is the rate of change between the values of *expr* computed in the previous cycle and the current value. (time derivative)	`der(#4)`
			`der(#dist)`
			`der(#4-#speed)`
			`deriv(#4)`
			`deriv(#dist)`
			`deriv(#4-#speed)`

Sum over time	`sum(`*expr*`)`	Result is the sum of all the values of *expr*.	`sum(1)`
			`sum(#4)`
			`sum(#dist)`

Integration	`i(`*expr*`)`
	`integ(`*expr*`)`
		Result is the integrated values of *expr* over time.	`i(#speed)`
			`i(#acc)`
			`integ(#speed)`
			`integ(#acc)`

2.2 Relational expressions

Smaller	*expr1* `<` *expr2*	Result is 1 if *expr1* is smaller than *expr2*. 0, otherwise.	`#4 < #10`
			`#4 < 10.7`
			`10 < #speed`

Smaller or equal	*expr1* `<=` *expr2*	Result is 1 if *expr1* is smaller than or equal to *expr2*. 0, otherwise.	`#4 <= #10`
			`#4 <= 10.7`
			`10 <= #speed`

Greater	*expr1* `>` *expr2*	Result is 1 if *expr1* is greater than the value of *expr2*. 0, otherwise.	`#4 > #10`
			`#4 > 10.7`
			`10 > #speed`

Greater or equal	*expr1* `>=` *expr2*	Result is 1 if *expr1* is greater than or equal to *expr2*. 0, otherwise.	`#4 >= #10`
			`#4 >= 10.7`
			`10 >= #speed`

Equality	*expr1* `==` *expr2*	Result is 1 if *expr1* is equal to *expr2*.
		0, otherwise.
			`#4 == #10`
			`#4 == 10.7`
			`10 == #speed`

Inequality	*expr1* `!=` *expr2*	Result is 1 if *expr1* is not equal to *expr2*. 0, otherwise.	`#4 != #10`
			`#4 != 10.7`
			`10 != #speed`

2.3 Logical expressions

Logical AND	*expr1* `&&` *expr2*	Result is 1 if both *expr1* and *expr2* are not equal to 0. 0, otherwise.	`(0<#4) && (#4<10)`
Logical OR	*expr1* `||` *expr2*	Result is 1 if at least one of *expr1* and *expr2* is not equal to 0. 0, otherwise.	`(#4<0) || (#4>10)`
Logical NOT	`!` *expr*	Result is 1 if *expr* is 0. 0, otherwise.	`!(#4 > 0)`

2.4 Bit expressions

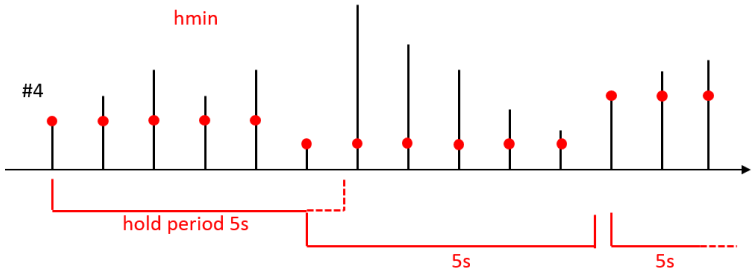
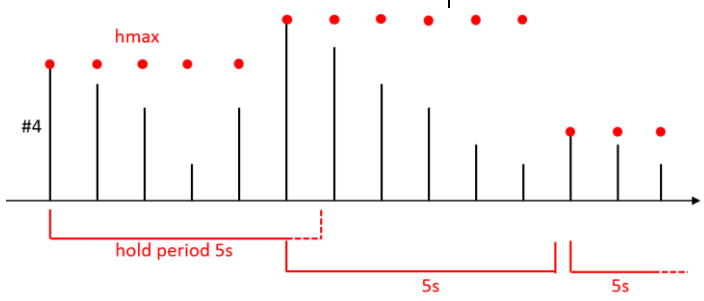
Bitwise AND	*expr1* `&` *expr2*	Result is the bitwise AND of *expr1* and *expr2*.	`#4 & 0xFF`
Bitwise OR	*expr1* `|` *expr2*	Result is the bitwise OR of *expr1* and *expr2*.	`#4 | 0x80`
Bitwise complement	`~` *expr*	Result is the bitwise complement of *expr*.	`#4 & ~0x00`

2.5 Trigonometric expressions

Rad	`rad`(*expr*)	Converts the value of *expr* from degrees to rad.	
Degree	`deg`(*expr*)	Converts the value of *expr* from rad to degrees.	
Sine	`sin`(*expr*)	Result is the sine value of *expr.* (*expr* has to be in radians).	`sin(#4)` `sin(1)`
Cosine	`cos`(*expr*)	Result is the cosine value of *expr.* (*expr* has to be in radians).	`cos(#4)` `cos(1)`
Tangent	`tan`(*expr*)	Result is the tangent value of *expr.* (*expr* has to be in radians).	`tan(#4)` `tan(1)`
Arc sine	`asin`(*expr*)	Result is the arc sine value of *expr*. (*expr* has to be in radians).	`asin(#4 / #5)`

Arc cosine	`acos`(*expr*)	Result is the arc cosine value of *expr*. (*expr* has to be in radians).	`acos(#4 / #5)`
Arc tangent	`atan`(*expr*)	Result is the arc tangent value of *expr*. (*expr* has to be in radians).	`atan(#4 / #5)`
Sine (degree)	`dsin`(*expr*) `sind`(*expr*)	Result is the sine value of *expr*. (*expr* has to be in degree).	`dsin(70)` `sind(70)`
Cosine (degree)	`dcos`(*expr*) `cosd`(*expr*)	Result is the cosine value of *expr*. (*expr* has to be in degree).	`dcos(70)` `cosd(70)`
Tangent (degree)	`dtan`(*expr*) `tand`(*expr*)	Result is the tangent value of *expr*. (*expr* has to be in degree).	`dtan(70)` `tand(70)`
Arc sine (degree)	`dasin`(*expr*) `asind`(*expr*)	Result is the arc sine value of *expr*. (*expr* has to be in degree).	`dasin(70)` `asind(70)`
Arc cosine (degree)	`dacos`(*expr*) `acosd`(*expr*)	Result is the arc cos value of *expr*. (*expr* has to be in degree).	`dacos(70)` `acosd(70)`
Arc tangent (degree)	`datan`(*expr*) `atand`(*expr*)	Result is the arc tangent value of *expr*. (*expr* has to be in degree).	`datan(70)` `atand(70)`

2.6 Min-Max expressions

Minimum	`min(`*expr1*`,` *expr2*`)`	Result is the smaller of *expr1* and *expr2*.	`min(#4,5)`
Maximum	`max(`*expr1*`,` *expr2*`)`	Result is the greater of *expr1* and *expr2*. (same as "<")	`max(#3, #speed)` `max(#speed, 10)`
Hold minimum	`hmin(`*expr1*`,` *expr2*`)`	*expr2* is called "hold-period". Result is the minimum value of *expr1*. This value is held for the hold period. If *expr1* evaluates to a smaller value during the hold period that value is the new result. (see below) 	`hmin(#4, 5)`
Hold maximum	`hmax(`*expr1*`,` *expr2*`)`	*expr2* is called "hold-period". Result is the maximum value of *expr1*. This value is held for the hold period. If *expr1* evaluates to a bigger value during the hold period that value is the new result. (see below) 	`hmax(#4, 5)`

2.7 Filter expressions

τ-filter	`flt(`*expr1*`,` *expr2*`)`	Computes a τ-filter over the values of *expr1* with a window size of *expr2* seconds.	`flt(#4, 5)`
RCLP-filter	`rclp(`*expr1*`,` *expr2*`)`	Computes an rclp-filter over the values of *expr1* with a RC of *expr2*.	`rclp(#2,0.030335)`
AVG-filter	`avg(`*expr1*`,` *expr2*`)`	Computes an avg-filter over the values of *expr1* with a window size of *expr2* seconds.	`avg(#4,0. 5)`

2.8 Variables

m1, m2, … m6	`m1 … m6`	Variables for storing computation results. Stored values survive power off/on cycles. Will be reset to 0 after data download. (Pressing Empty/F3 in WinIt.)	`m1=m1+1`
p1, p2	`p1` `p2`	Variables which can only be incremented. Cannot be reset to 0.	`p1=p1+1`
x	`x`	Holds the value from the previous computation. Each channel has its own variable x. Reset to 0 at power on.	`x+1`

2.9 Conditional expressions

Conditional execution	`if(`*expr1*`,`*expr2*`,` *expr3*`)`	The result is *expr2* If *expr1* is not zero Otherwise, the result is *expr3*.	`if (#3==0, #4, #5)`

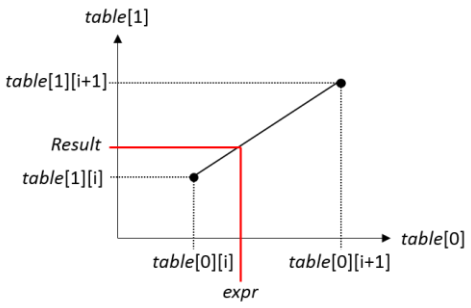
2.10 Assignment

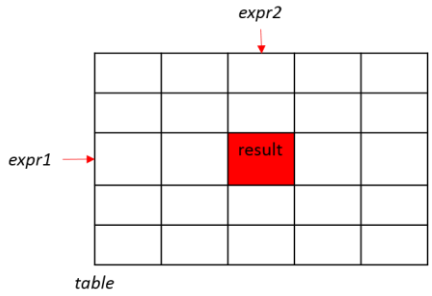
Assign	*variable* = *expr*	Stores the value of *expr* in a *variable*. The result of the assignment is the value of *expr*.	`m1 = max(#3, #4)`

2.11 Shifting

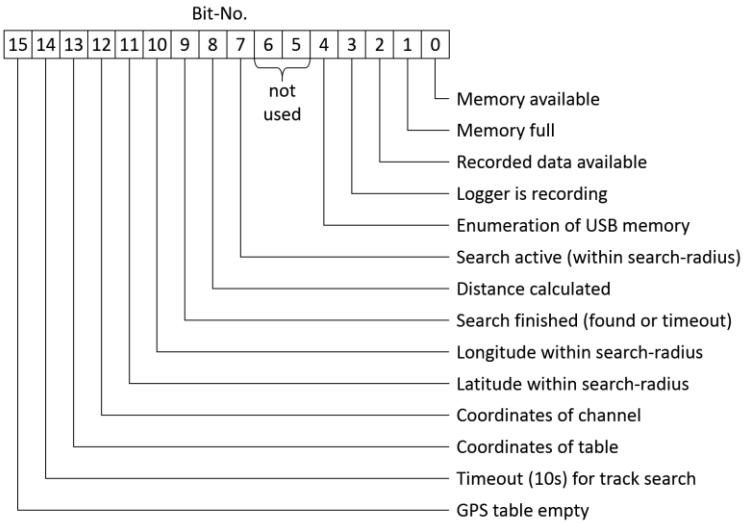
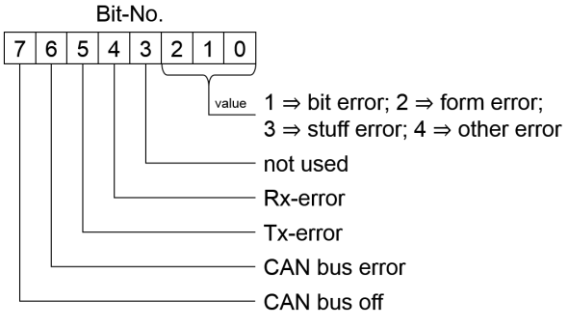
Channel delay	`delay(`*expr1*`,` *expr2*`)`	Delays the channel values of channel *expr1* for *expr2* seconds.	`delay(#5,0.25)`
Bit shift	*expr1*`>>`*expr2*	Shifts the bits of channel *expr1* by *expr2* bits to the right (lower bits).	`#3>>8`
	expr1`<<`*expr2*	Shifts the bits of channel *expr1* by *expr2* bits to the left (higher bits)	`#2<<15`

2.12 Table expressions

Table	`tab(`*table*`,`*expr*`)`	*table* is a table with two rows, *table*[0] and *table*[1]. The result of the tab-function is computed by linear interpolation as shown below: 

Linearize	`lin(`*expr1*`,`*expr2*`,`*table*`)`	*table* is a 2-dimensional table. Result is the value *table*[*expr1*][*expr2*]. 

2.13 Others

Logger status[2]	`stat(`*mask*`)`	Delivers the status of the logger. The result is a bit-coded value. *mask* can be used for masking out certain bits. If *mask* = 0 then all bits are delivered by `stat`. 
Sector pointer	`sptr()`	Delivers the pointer index
Sector number	`secnr()`	Delivers the current sector number
Sector count	`seccnt()`	Delivers the number of sectors to save
CPU load	`cpu_load_max()`	Delivers the maximum CPU load
	`Cpu_load_avg()`	Delivers the average CPU load
CAN errors	`cantxerr(CAN#)`	Error counter of the CAN line while sending, value ≥ 128 ⇒ bus off
	`canrxerr(CAN#)`	Error counter of the CAN line while receiving, value ≥ 128 ⇒ bus off
	`canerr(CAN#)`	

[2] Implemented in Logger9 and USB StickLogger

Receive-counter	`Sioirq()`	16-bit counter for incoming characters
Channel rate	`chrate(ch#)`	Delivers the sampling rate of the entered channel
Channel value	`chval(ch#)`	Delivers the value of the entered channel

2.14 Syntax

expr := *numerical constant* or *variable or channel*

expr := *expr operator expr*

expr := *function* "(" *expr* ")"

numerical constant := positive or negative decimal number

numerical constant := hexadecimal number // "0x" or "0h" followed by hexadecimal digits 0..F

numerical constant := binary number // "0b" followed by binary digits (0 and 1)

channel := "#"<channel number> or "#"<channel name>

variable := m1 or m2 or …, or m6 or p1 or p2 or x

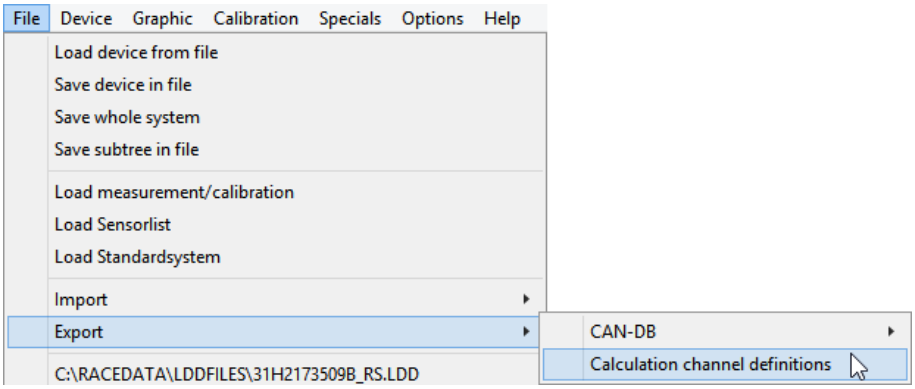
CAN# := number of the CAN line, for example "1" for CAN-1

All possible operators and functions are described in the table above.

3 Import and export of calculation definitions

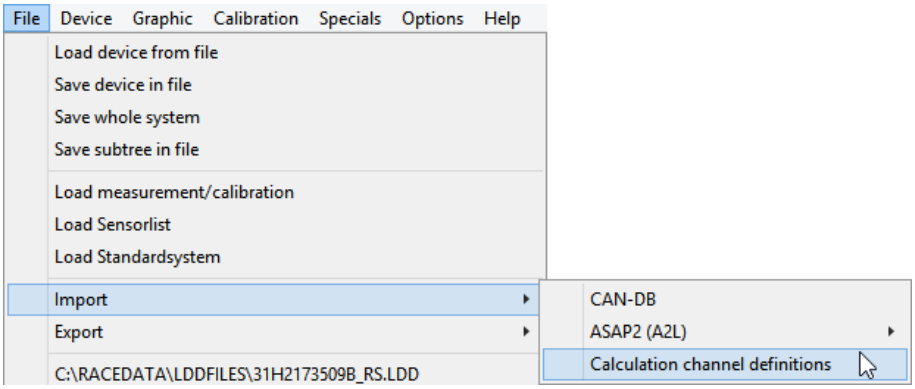
Since 2018 it is possible to export the calculation definitions of the calc channels and import exported ones as well. This might be a very helpful function, if you´re working with different data systems.

To export your calculation definitions of the calc channels, select the module you want to export the calculation definitions from. Then select the menu item “File” ⇒ “Export” ⇒ “Calculation channel definitions”:



This will export all calc channel definitions of this module. You only have to select where to store the file and maybe enter an individual file name.

To import calc channel definitions, you select the module, you want to add the calc channels to and select the menu item “File” ⇒ “Import” ⇒ “Calculation channel definitions”:



In the following window you simply have to select a calc definition file. This will import all calc channel definitions of the selected file. Confirm the changes with <**Apply**>.

This import/export function only handles the calculation definition (calculation formula). It does not contain the information about if the channel is turned on or if the channel is recorded.