Table of Contents

Expression Syntax

Formula editor is used in the following cases:

- Setting Column Formula
- Custom Fit Curve
- Entering value in any numeric field and in tables
- MagicPlot Calculator

MagicPlot uses standard IEEE 754 double precision floating-point arithmetic. Double precision floating point takes 8 bytes per number and provides a relative precision of about 16 decimal digits and magnitude range from about 10^{-308} to about 10^{+308}.

Syntax Highlighting

MagicPlot formula editor highlights expression syntax. It also marks matching brackets and highlights variables:

```
x = $A;
a = 1.4499;
x0 = -1.232;
dx = 0.6936;
(a * exp(-ln(2) * (x-x0)^2 / dx^2))
+ (a * exp(-ln(2) * (x-(x0*-1))^2 / dx^2))
```

General Rules

Case Sensitivity

MagicPlot formula translator is generally **case sensitive**, i.e. you can write `sin` but not `Sin`. Note that x and X are different variables. You can use this feature when naming `Custom Equation` Fit Curve parameters.

Entering Numbers

- You can use **dot (`.`) only** as decimal separator, and separate function arguments with a comma (`,`) in:
  - Setting Column Formula
  - Custom Fit Curve
  - Using MagicPlot Calculator
- You can use **dot (`.`) or comma (`,`)** as decimal separator, and separate function arguments with a semicolon (`;`) in the following cases:
  - Cell editing in Tables

- Entering value in any numeric field

You can use e or E for scientific notation: `1.45e-3` or `1.45E-3`.

Using Spaces and Line Breaks

You can freely insert space characters and line breaks in formula, but do not break function names, numbers, operators. You do not need to enter special characters to indicate line break.

Local variables in expression

You can set a local variables in expression. Use semicolon to separate variable assignments and the result expression: `a=5; a*a + 2*a + 1`. The expression after the last semicolon is the result expression. The variables are calculated in the present order so you need to assign the variable before usage.

Functions

You can see a list of all available functions and their descriptions in `Functions` tab in `Set Column Formula` window and in `Help on Functions` window which can be opened from menu in calculator window.

MagicPlot uses functions of Java programming language library `StrictMath` to evaluate `sin`, `cos`, `exp`, etc. These functions are available from the well-known network library `netlib` as a “Freely Distributable Math Library”, `fdlibm` package. The same library is widely used in many scientific computing applications.

Special functions (Bessel, Erf, Gamma, Beta) calculation is based on Colt 1.2 library.

Trigonometric Functions

MagicPlot supports all standard trigonometric functions (`sin`, `cos`, etc.). All angles are always measured in radians for clarity.

You can use the following functions to convert angles units:

- `deg(a)` — converts angles input in radians to an equivalent measure in degrees.
- `rad(a)` — converts angles input in degrees to an equivalent measure in radians.

Examples

- `sin(rad(90))`
- `deg(asin(1))`

Constants

The predefined constants are:

- `pi, Pi, PI` — π = 3.1416... value (the ratio of circumference of a circle to its diameter).
- e — e = 2.7183... value (the base of the natural logarithms). **Note:** expression e^a is evaluated as `exp(a)`.
- nan, NaN, NAN — Not-a-Number value.
- `inf`, `Inf`, `infinity`, `Infinity` — positive infinity value which may be used in some calculations. **Note:** write `-inf` for negative infinity.
- `eps` — machine epsilon, gives an upper bound of the relative error due to rounding in floating point arithmetic. **Note:** `eps = ulp(1) = 2^(-52) = 2,2204E-16`. (52 is the number of bits used to store fractional part of a number.)

Boolean Logic

MagicPlot can interpret boolean logic expressions. Zero is interpreted as `false` and non-zero values are interpreted as `true` similarly to C programming language (Note: legacy MagicPlot versions 2.9 and older interpret zero *and negative* values as `false`, this was changed in the 3.0 version). You can use simple logical operators which are described below. Use `1` as `true` and `0` as `false`.

The conditional function 'if'

The conditional logical function is `if(condition, a, b)`. If `condition` argument is `true` (non-zero) it returns the second argument (a), otherwise it returns the third argument (b).

Examples

- `if(col(A) >= 0, col(A), -col(A))` — evaluates absolute value of column A (you can use `abs(col(A))` for that, of course).
- `if(col(B) >= 0, col(B), NaN)` — returns only positive values from column B. Negative values are replaced with NaN value (empty cell). You can use this expression to filter negative values if you do not want to use them in future calculations. Note that "*Not-a-Number returned at row #*" warning can be shown for such expression.
- `if(col(A) > 0 & col(B) > 0, max(col(A), col(B)), NaN)`
- `a * if(x >= 0, x, -x)` – custom fit function example for abs.

Equality Checking

You have to be careful if you need to check equality of two values. Due to inaccuracy of computer floating-point calculations the result of evaluation is always approximate. For example, result of `sqrt(3)^2` is the number `2.9999999999999996`, not exactly 3. The expression `sqrt(3)^2 == 3` is `false` (it returns `0`). Keep in mind that for convenience MagicPlot rounds numbers when showing on the screen, so this value will be shown as 3 in table if the number of shown fractional digits in MagicPlot preferences is not big enough.

Generally, if you want to check equality of two values you need to use some equality threshold for relative difference. That is, you should compare the modulus of relative difference of two values a and b with threshold `t`: `if(abs((a-b)/a) < t, …, …)`.

Examples

- `sqrt(3)^2 - 3` results something about `-4,4409E-16`
- `if(abs(sqrt(3)^2 - 3) / 3 < 1e-10, …, …)` — checks equality of `sqrt(3)^2` and `3` with a threshold of `1e-10`.

Operators

Operator	Description	Operator	Description
+	addition	==	equal to
-	subtraction	!=	not equal to
*	multiplication	!	logical negation
/	division	<	less than
%	remainder after division	>	greater than
^	exponentiation	<=	less than or equal to
|	logical or	>=	greater than or equal to
&	logical and		

Operations Priority

Operations	Precedence	Associativity
`function()`	1 (is evaluated first)	—
^	2	Right-to-left
!, - (unary minus)	3	—
*, /, %	4	Left-to-right
+, -	5	Left-to-right
<, >, <=, >=	6	Left-to-right
==, !=	7	Left-to-right
&	8	Left-to-right
|	9	Left-to-right
= (assignment)	10 (is evaluated last)	Left-to-right

Operators with **lower** precedence value are evaluated **earlier**. You can use brackets to change calculation sequence.

Expression is evaluated left-to-right, excluding repeated exponentiation operator ^. The ^ operator is Wright-associative like in Fortran language (evaluated right-to-left; note that in general case `a^(b^c)` ≠ `(a^b)^c`). Hence `a^b^c` is evaluated as `a^(b^c)`. The reason for exponentiation being right-associative is that a repeated left-associative exponentiation operation would be less useful: Multiple appearances could (and would) be rewritten with multiplication: `(a^b)^c = a^(b*c)`.

Examples

- `1 + 2 * 3` returns 7.
- `(1 + 2) * 3` returns 9.
- `2*-3` returns `-6`.
- `-3^2` is equal to `-(3^2)`, because ^ priority is higher than that of unary minus. The result is `-9`.
- `(-3)^2` returns 9.
- `2^2^3` is equal to `2^(2^3)`, because ^ is right-associative operator. The result is 256.