



**The Expressions plugin  
PRINTED MANUAL**

# Expressions plugin

©2006-2016 AGG Software

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: 28.11.2016

## **Publisher**

*AGG Software*

## **Production**

©2006-2016 AGG Software

*<http://www.aggsoft.com>*

---

# Table of Contents

<b>Part 1 Introduction</b>	<b>1</b>
<b>Part 2 System requirements</b>	<b>1</b>
<b>Part 3 Installing Expressions</b>	<b>2</b>
<b>Part 4 Glossary</b>	<b>3</b>
<b>Part 5 User Manual</b>	<b>3</b>
1 Configuration .....	3
2 Supported functions .....	4
Math functions .....	4
Operators .....	6
String functions .....	7
Date and Time .....	8
Miscellaneous functions .....	8
<b>Part 6 Troubles?</b>	<b>10</b>
1 Possible problems .....	10

## 1 Introduction

The filter module "Expressions" for our data loggers (for example, Advanced Serial Data Logger or Advanced TCP/IP Data Logger) is an interpreter for quick and easy evaluation of expressions. It is a smart tool easy in use. Supports different data types (string, date, time, decimal, float and boolean), arithmetic and boolean operators, multiple levels of brackets, built-in math, string, boolean functions and user-defined variables.

It supports the following operators and functions:

Mathematics operations

- + : Addition
- - : Substraction
- \* : Multiplication
- / : Division
- ^ : Exponential (only positive numbers for the base)

Mathematics functions: ABS, ATAN, COS, EXP, LN, ROUND, SIN, SQRT, SQR, TRUNC and many others;

String functions: COPY, REPLACE, POS and many others;

Logical functions: AND, OR, XOR and etc.

## 2 System requirements

The following requirements must be met for "Expressions" to be installed:

**Operating system:** Windows 2000 SP4 and above, including both x86 and x64 workstations and servers. A latest service pack for the corresponding OS is required.

**Free disk space:** Not less than 5 MB of free disk space is recommended.

**Special access requirements:** You should log on as a user with Administrator rights in order to install this module.

The main application (core) must be installed, for example, Advanced Serial Data Logger.

**Notes for Microsoft Vista and above:**

Since our software saves data to the registry and installs to the Program Files folder, the following requirements must be met:

1. You need Administrator rights to run and install our software
2. The shortcut icon of our software will be located on the desktop;
3. Windows Vista will ask for your confirmation to continue the installation.

NOTE: You can configure the user account only once in order not to see the above dialog box any more. Search Google for the solution of this problem.

### 3 Installing Expressions

1. Close the main application (for example, Advanced Serial Data Logger) if it is running;
2. Copy the program to your hard drive;
3. Run the module installation file with a double click on the file name in Windows Explorer;
4. Follow the instructions of the installation software. Usually, it is enough just to click the "Next" button several times;
5. Start the main application. The name of the module will appear on the "Modules" tab of the "Settings" window if it is successfully installed.

If the module is compatible with the program, its name and version will be displayed in the module list. You can see examples of installed modules on fig.1-2. Some types of modules require additional configuration. To do it, just select a module from the list and click the "Setup" button next to the list. The configuration of the module is described below.

You can see some types of modules on the "Log file" tab. To configure such a module, you should select it from the "File type" list and click the "Advanced" button.

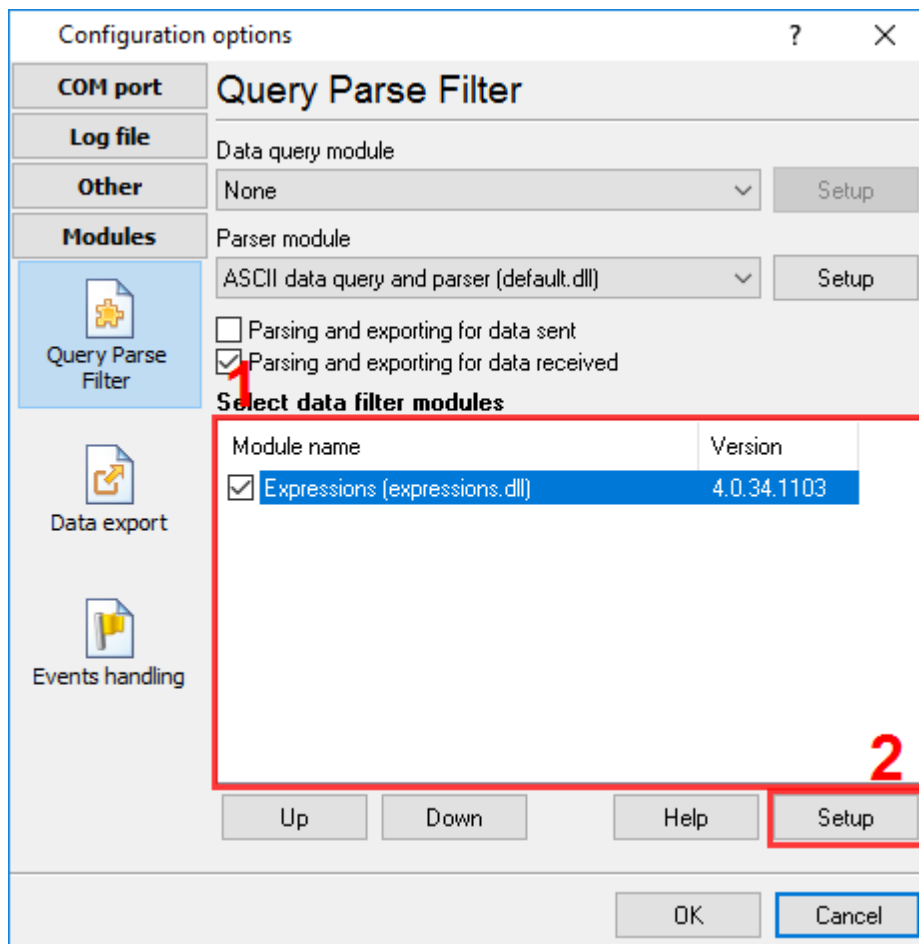


Fig.1. Example of installed module

## 4 Glossary

**Plug-in** - module

**Main program** – the program shell that uses this module. For example: Advanced Serial Data Logger

**Parser** – the module that processes the data flow singling out data packets from it and variables from data packets. These variables are used in data export modules after that.

**Core** - see "Main program".

## 5 User Manual

### 5.1 Configuration

The module configuration is very simple (pic.1). You may add one or more expressions to the edit window. One row contains one expression. Each expressions should have the following format:

VARIABLE\_NAME=EXPRESSION

Where VARIABLE\_NAME is a name of the new or existing parser variable and EXPRESSION is any valid set of functions and/or variables.

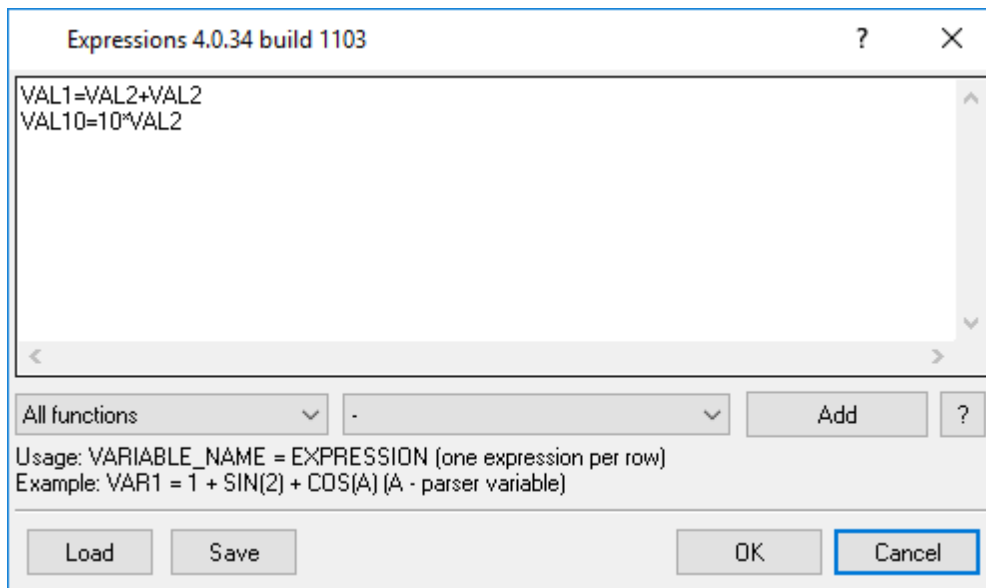
This module may use all existing parser variables (you may select these variables in the "Variables" group). You may read about parser variables in a parser's help file. Additionally you may define new variables here and assign a result of an expressions to these variables. Therefore user-defined variables may be used in data export modules as a regular parser variable.

All functions, operators and variables are exist in a drop-down list. If you want to add any function then you should:

1. Put the cursor to a position in the edit window;
2. Select the corresponding category (Date and Time for example);
3. Select a function that you need to insert to the edit window (DATE for example);
4. Click the "Add" button.

Later you may type your expressions directly, without using of drop-down lists.

All supported functions are described in following [sections](#).



Pic.1. The configuration window

## 5.2 Supported functions

### 5.2.1 Math functions

**ABS(X)** - Abs returns the absolute value of the argument, X. X is an integer-type or real-type expression.

**ARCCOS(X)** - ArcCos returns the inverse cosine of X. X must be between -1 and 1. The return value is in the range  $[0..Pi]$ , in radians.

**ARCCOSH(X)** - ArcCosh returns the inverse hyperbolic cosine of X. The value of X must be greater than or equal to 1.

**ARCSIN(X)** - ArcSin returns the inverse sine of X. X must be between -1 and 1. The return value will be in the range  $[-Pi/2..Pi/2]$ , in radians.

**ARCSINH(X)** - ArcSinh returns the inverse hyperbolic sine of X.

**ARCTAN2(X)** - ArcTan2 calculates  $\text{ArcTan}(Y/X)$ , and returns an angle in the correct quadrant. The values of X and Y must be between  $-2^{64}$  and  $2^{64}$ . In addition, the value of X can't be 0. The return value will fall in the range from -Pi to Pi radians.

**ARCTANH(X)** - ArcTanh returns the inverse hyperbolic tangent of X. The value of X must be between -1 and 1 (inclusive).

**CEIL(X)** - Call Ceil to obtain the lowest integer greater than or equal to X. The absolute value of X must be less than MaxInt.

For example:  
Ceil(-2.8) = -2  
Ceil(2.8) = 3

Ceil(-1.0) = -1

**CLIP(X,Min,Max)** - Returns Min if  $X \leq \text{Min}$ , returns Max if  $X \geq \text{Max}$ , otherwise returns X. For example:

CLIP(2, 3, 4) = 2

CLIP(3, 2, 4) = 3

CLIP(4, 2, 3) = 3

**COS(X)** - Use the COS to calculate the cosecant of X, where X is an angle in radians.

**COSH(X)** - Use the COSH to calculate the hyperbolic cosecant of X, where X is an angle in radians.

**COTAN(X),**

**COTG(X)** - Call Cotan or Cotg to obtain the cotangent of X. The cotangent is calculated using the formula  $1 / \text{Tan}(X)$ .

**DEG(X)** - Use Deg to convert angles measured in radians to degrees, where degrees = radians(180/pi).

**EXP(X)** - Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms.

**FLOOR(X)** - Call Floor to obtain the highest integer less than or equal to X. For example:

Floor(-2.8) = -3

Floor(2.8) = 2

Floor(-1.0) = -1

**Note:** The absolute value of X must be less than MaxInt.

**FRAC(X)** - Frac function returns the fractional part of the argument X. X is a real-type expression. The result is the fractional part of X; that is,  $\text{Frac}(X) = X - \text{Int}(X)$ .

**HEX(X)** - Converts a X value to a hexadecimal representation.

**LN(X)** - Ln returns the natural logarithm ( $\text{Ln}(e) = 1$ ) of the real-type expression X.

**LOG(Base, X)** - Log returns the log base Base of X.

**POW(Base, Exponent), POWER(Base, Exponent)** - Power raises Base to any power. For fractional exponents or exponents greater than 65535, Base must be greater than 0.

**POWLN2(X)** - Power raises Ln2 base to any power. For fractional exponents or exponents greater than 65535, Base must be greater than 0.

**RAD(X)** - Use DegToRad to convert angles expressed in degrees to the corresponding value in radians, where radians = degrees(pi/180).

**RANDOM(X)** - Random returns a random number within the range  $0 \leq X < \text{Range}$ . If Range is not specified, the result is a real-type random number within the range  $0 \leq X < 1$ .

**Note:** Because the implementation of the Random function may change between compiler versions, we do not recommend using Random for encryption or other purposes that require reproducible sequences of pseudo-random numbers.



**ROUND(X)** - the Round function rounds a real-type value to an integer-type value. X is a real-type expression. Round returns an Int64 value that is the value of X rounded to the nearest whole number. If X is exactly halfway between two whole numbers, the result is always the even number. This method of rounding is often called "Banker's Rounding".

**SIGN(X)** - Use Sign to test the sign of a numeric value. Sign returns

0 if AValue is zero.

1 if AValue is greater than zero.

-1 if AValue is less than zero.

**SIN(X)** - Use the SIN to calculate the sine of X, where X is an angle in radians.

**SINH(X)** - Sinh calculates the hyperbolic sine of X.

**SQR(X)** - the Sqr function returns the square of the argument. X is a floating-point expression. The result, of the same type as X, is the square of X, or X\*X.

**SQRT(X)** - X is a floating-point expression. The result is the square root of X.

**TAN(X), TG(X)** - Tan or Tg returns the tangent of X.  $\text{Tan}(X) = \text{Sin}(X) / \text{Cos}(X)$ .

**TRUNC(X)** - the Trunc function truncates a real-type value to an integer-type value. X is a real-type expression. Trunc returns an Int64 value that is the value of X rounded toward zero.

## 5.2.2 Operators

- - subtraction

\* - multiplication

/ - real division

^ or \*\* - Power raises base to any power. For fractional exponents or exponents greater than 65535, Base must be greater than 0. Example:  $x^{**}y$

+ - addition

< - less-than

<= - less-than-or-equal-to

<> - inequality

= - equality

> - greater-than

>= - greater-than-or-equal-to

**AND** - bitwise AND, Operand types: integer, Result type: integer, Example: X and Y

**DIV** - integer division. The value of  $x \text{ div } y$  is the value of  $x/y$  rounded in the direction of zero to the nearest integer.

**MOD** - remainder. The mod operator returns the remainder obtained by dividing its operands. In other words,  $x \text{ mod } y = x - (x \text{ div } y) * y$ .

**OR** - bitwise OR, Operand types: integer, Result type: integer, Example: X or Y

**SHL** - bitwise shift left, Operand types: integer, Result type: integer, Example: X shl 2

**SHR** - bitwise shift right, Operand types: integer, Result type: integer, Example: X shr 2

**XOR** - bitwise XOR, Operand types: integer, Result type: integer, Example: X xor Y

### 5.2.3 String functions

**SUBSTR(S, Index, Count), STRCOPY(S, Index, Count), COPY(S, Index, Count)** - Returns a substring of a string. Copy returns a substring or subarray containing Count characters or elements starting at S[Index]. The substring is a unique copy (that is, it does not share memory with S, although if the elements of the array are pointers or objects, these are not copied as well.). If Index is larger than the length of S, Copy returns an empty string or array. If Count specifies more characters or array elements than are available, only the characters or elements from S[Index] to the end of S are returned.

**FIRSTLINE(S)** - Copies all characters until first CR or LF character.

**TRIMLEFT(S), LTRIM(S)** - Trims leading spaces and control characters from a string.

**STRPOS(Substr, S), POS(Substr, S)** - Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions. Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns zero.

**REMOVECHAR(S, Char)** - Returns a string with occurrences of one character deleted. **REMOVECHAR** deletes occurrences of the character specified by **Char**. S is the source string, whose characters are deleted. The comparison operation is case sensitive.

**REMOVENONPRINT(S)** - Returns a string with occurrences of all non-printable characters (with ASCII code < 32) deleted.

**REPLACE(S, OldPattern, NewPattern)** - Returns a string with occurrences of one substring replaced by another substring. **REPLACE** replaces occurrences of the substring specified by OldPattern with the substring specified by NewPattern. S is the source string, whose substrings are changed. OldPattern is the substring to locate and replace with NewPattern. NewPattern is the substring to substitute for occurrences of OldPattern. The comparison operation is case insensitive.

**REPLACECHAR(S, OldChar, NewChar)** - Returns a string with occurrences of one substring replaced by another substring. **REPLACECHAR** replaces occurrences of the substring specified by OldChar with the substring specified by NewChar. S is the source string, whose substrings are

changed. OldChar is the substring to locate and replace with NewChar. NewChar is the substring to substitute for occurrences of OldChar. The comparison operation is case sensitive.

**TRIMRIGHT(S), RTRIM(S)** - Trims trailing spaces and control characters from a string.

**TRIM(S)** - Trims leading and trailing spaces and control characters from a string.

## 5.2.4 Date and Time

**DATE()** - Returns current date (value of DateTime type).

**DATE(S)** - Returns a date value of DateTime type converted from the string S. The format of the string S - 'DD.MM.YYYY'. Example: DATE('15.01.2007')

**DATE(Y,M,D)** - Returns a date value of DateTime type converted from Y (year), M (month) and D (day) integer values. Example: DATE(2007, 1, 15)

**DAY(X)** - Returns a day part from the X value. The X value should be of DateTime type.

**GOMONTH(X,Y)** - adds Y months to X date value. Y may be negative or positive integer value. X should be of DateTime type.

**MONTH(X)** - Returns a month part from the X value. The X value should be of DateTime type.

**NOW** - Returns current date and time. The return value is with DateTime type.

**TIME()** - Returns current time (value of DateTime type).

**TIME(S)** - Returns a time value of DateTime type converted from the string S. The format of the string S - 'HH:NN'. Example: TIME('15:21')

**TIME(H,M,S,MS)** - Returns a date value of DateTime type converted from H (hour), M (minutes), S (seconds) and MS (milliseconds) integer values. Example: TIME(15, 21, 0, 0)

**YEAR(X)** - Returns a year part from the X value. The X value should be of DateTime type.

## 5.2.5 Miscellaneous functions

### Conditional group

**IIF(X,Y,Z)** - if X boolean condition is True, then evaluates and returns Y, otherwise evaluates and returns Z.

**DISCARD\_DATA\_PACKET\_IF(X)** - if X boolean condition is True, then the program doesn't pass data packet to data export plugins.

**Example:** Ignore all data packets if the parser variable with the "TEMPERATURE" is over the range: -20..20.

DISCARD\_DATA\_PACKET\_IF((TEMPERATURE>20)OR(TEMPERATURE<-20))

**GENERATE\_EVENT\_IF(X, Y)** - if X boolean condition is True, then the program generates the Y event. The Y should be a string value. All data packet variables will be used as events parameters.

**Example:** Notify an user if the parser variable with the "TEMPERATURE" is over the range: -20..20.

GENERATE\_EVENT\_IF((TEMPERATURE>20)OR(TEMPERATURE<-20), 'TEMP\_ALARM')

Now, you should activate the "Events handling" plugin and add an event handler for the "TEMP\_ALARM" event with necessary options.

## General group

**MAX(A,B)** - Call Max to compare two numeric values. Max returns the greater value of the two.

**MIN(A,B)** - Call Min to compare two numeric values in Delphi. Min returns the smaller value of the two.

**SUM(A,B)** - Equal for A+B, where A and B can be string or number.

## Miscellaneous group

**BYTETOSTR(X)** - converts 1 byte in the array X to a unsigned byte value.

**DOUBLETOSTR(X)** - converts 8 bytes in the array X to a double value.

**DOUBLETOSTRBE(X)** - converts 8 bytes in "Big-endian" order in the array X to a double value.

**INT64TOSTR(X)** - converts 8 bytes in the array X to a 64 bit integer value.

**INT64TOSTRBE(X)** - converts 8 bytes in "Big-endian" order in the array X to a 64 bit integer value.

**LONGINTTOSTR(X)** - converts 4 bytes in the array X to a 32 bit integer value.

**LONGINTTOSTRBE(X)** - converts 4 bytes in "Big-endian" order in the array X to a 32 bit integer value.

**LONGWORDTOSTR(X)** - converts 4 bytes in the array X to a 32 bit unsigned decimal value.

**LONGWORDTOSTRBE(X)** - converts 4 bytes in "Big-endian" order in the array X to a 32 bit unsigned decimal value.

**SINGLETOSTR(X)** - converts 4 bytes in the array X to a float value.

**SINGLETOSTRBE(X)** - converts 4 bytes in "Big-endian" order in the array X to a float value.

**SMALLINTTOSTR(X)** - converts 2 bytes in the array X to a 16 bit signed decimal value.

**SMALLINTTOSTRBE(X)** - converts 2 bytes in "Big-endian" order in the array X to a 16 bit signed

decimal value.

**WORDTOSTR(X)** - converts 2 bytes in the array X to a 16 bit unsigned decimal value.

**WORDTOSTRBE(X)** - converts 2 bytes in "Big-endian" order in the array X to a 16 bit unsigned decimal value.

## 6 Troubles?

### 6.1 Possible problems

**No data for publication/exporting** – no data is passed for exporting. Solution: configure the parser, make sure that one or more variables are declared in the parser.

**Error on binding variable with name %s [%s]** – the error usually occurs if data does not correspond to the specified format. For example, the date and time format does not correspond to the data.

**Unable to disconnect from the database [%s]** and **Unable to connect to a database [%s]** – it is impossible to connect/disconnect to/from the database. You should check the parameters of the database connection. The analysis of the additional information will help you locate the error.

**Database access error [%s]. Stop operations with the database?** – the message appears if an error occurs during an attempt to execute an SQL query if the second variant of reacting to errors is selected. The message implies a "Yes" or "No" answer. The analysis of the additional information will help you locate the error.

**Unable to verify your SQL script [%s]** – the message appears when an attempt to analyze your SQL query fails. Check if the syntax of your SQL query is correct.

**Tested successfully** – the message appears if your database connection is successfully tested. It requires no additional actions.

**Database isn't used** – the message appears if the module is temporarily disabled (the "Temporarily disabled" check box is selected) or the database name field is empty. Check the connection parameters.

**Database isn't selected** - the message appears if the database type is not selected. Check the connection parameters.

**Database: %s** – %s contains the database name. The message appears if the database connection is successful. Usually, you see it when you call the module for the first time. It requires no additional actions.

**Invalid data block length (columns=%d,length=%d)** – an internal application error. It means that the data sent by the parser is in an invalid format. Perhaps, you are using the module incompatible with the version of the Advanced Serial Data Logger kernel. Update the versions of both the kernel and the module.

**The time of connection is not due yet (%d,%d)** – the message appears during an attempt to

connect to the database after the connection to it has been lost and the "Reconnect after" option is enabled. No additional actions are required.

**Invalid procedure call. Bad arguments** –an attempt to call the module using invalid parameters. Perhaps, you are using the module incompatible with the version of the Advanced Serial Data Logger kernel. Update the versions of both the kernel and the module.

**Writing to the database is complete** - the message appears if your queue of SQL queries is successfully executed. It requires no additional actions.

**Writing to the database is complete with errors** – the message appears if the executing your queue of SQL queries was interrupted by an error. It requires no additional actions.

**Your SQL is empty. Please, specify some SQL text first** – the message appears if you do not enter the text for your SQL query. Check if the options on the "SQL queue" tab are configured correctly.

**Invalid temporary path** – the path to the temporary file specified by you does not exist. Enter a new path in the "Temporary folder" field on the "Errors handling" tab.

%s, %d – will be replaced by additional information.