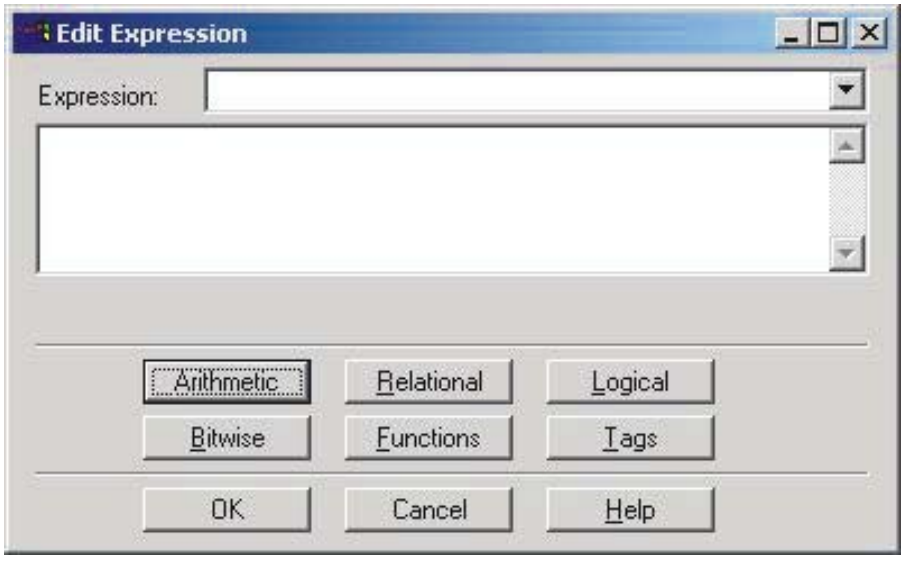
USER'S GUIDE

Expression Editor



Expression Editor

smar FIRST IN FIELDBUS

JUN / 04

Expression Editor

VERSION 7.0

FOUNDATION

PVI EWEXEME

www.smar.com

Specifications and information are subject to change without notice.
Up-to-date address information is available on our website.

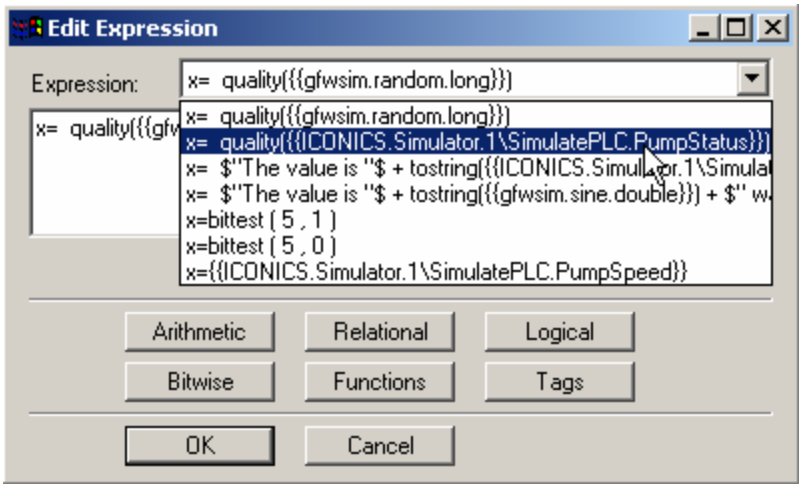
web: www.smar.com/contactus.asp

Index

Expression Editor

Creating Expressions Using the Expression Editor

The **Expression Editor,** shown below, is available to assist you in creating expressions for your ProcessView applications. The window is resizable and can be stretched as well as maximized or minimized. The drop-down list at the top of the **Edit Expression** dialog box keeps track of the last 50 expressions you have entered. The expression entered most recently is the first one in the drop-down list.



Expression Editor

Writing Expressions

An **expression** is a string that defines and evaluates a data connection between a client and an OPC server. During runtime mode, OPC servers resolve the data value for the expression. To indicate that a data connection is an expression, precede the string with the "**x=**" token, as shown below:

x={{SMAR.Simulator.1\SimulatePLC.PumpSpeed}}

You can either type your expressions directly into the text box of the **Edit Expression** dialog box, or you can use the symbols and functions provided that help you use the proper string syntax when writing expressions. The following categories are available:

- **Arithmetic**
- **Relational**
- **Logical**
- **Bitwise**
- **Functions**
- **Tags**

Strings in Expressions

Expressions allow calculations to be performed on incoming data. The OPC server can provide the data in one or more data types, such as "float," "long," "integer," "string," etc. If the numeric data are coming from the server as strings, they are compared as strings in expressions. This is done based on the alphabetical order of the letters. Therefore, an expression evaluated as TRUE "20" > "100" correctly. Of course, if there were an expectation of a numeric comparison, 20 < 100 and the above expressions might seem to be evaluated incorrectly, but they are not.

There is workaround. If you add a numerical zero to each of the tags, the logic operators will work properly. For example:

x=({{JC.N1OPC.1.0\HDQTRS\sys2\ad-3.Present Value}}+0) >
({{JC.N1OPC.1.0\HDQTRS\sys2\ad-4.Present Value}}+0)

An alternative way is to change the OPC server so that it sends the strings with a fixed number of digits with leading zeros, or to use DataWorX registers for a conversion from a string to a number.

Point Extension Syntax

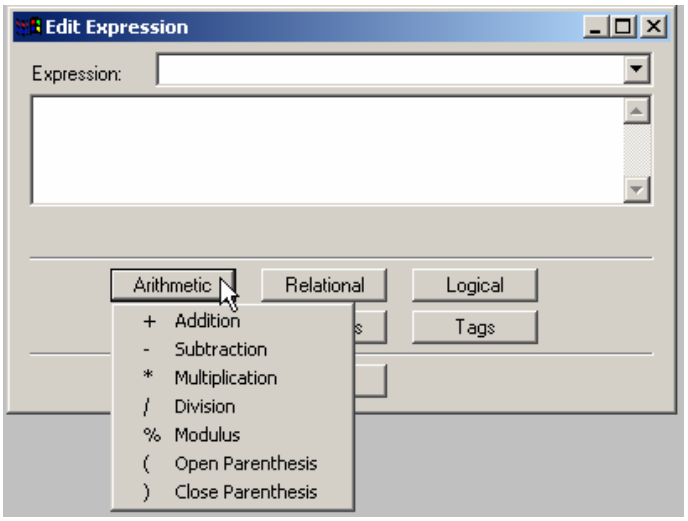
The **Point Extension Syntax (PES)** allows for retrieving additional information related to OPC tags, such as quality and timestamp. The following are example expressions using a valid PES request:

- tag:SMAR.Simulator\SimulatePLC.Ramp#timestamp
- tag:SMAR.Simulator\SimulatePLC.Ramp#quality
- tag:\\pc1\SMAR.Simulator\SimulatePLC.Ramp#timestamp
- tag:\\pc1\SMAR.Simulator\SimulatePLC.Ramp#quality

Sometimes it may be necessary to enforce the "request data type" to a specific type, such as "string," in order to display this information in a process point.

Arithmetic

The **Arithmetic** menu symbols are shown in the figure below.



Arithmetic Symbols

The symbols '+', '-', '*', '/' and '%' use the following format:
expression :: parameter **symbol** parameter

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	+ or - or * or / or %

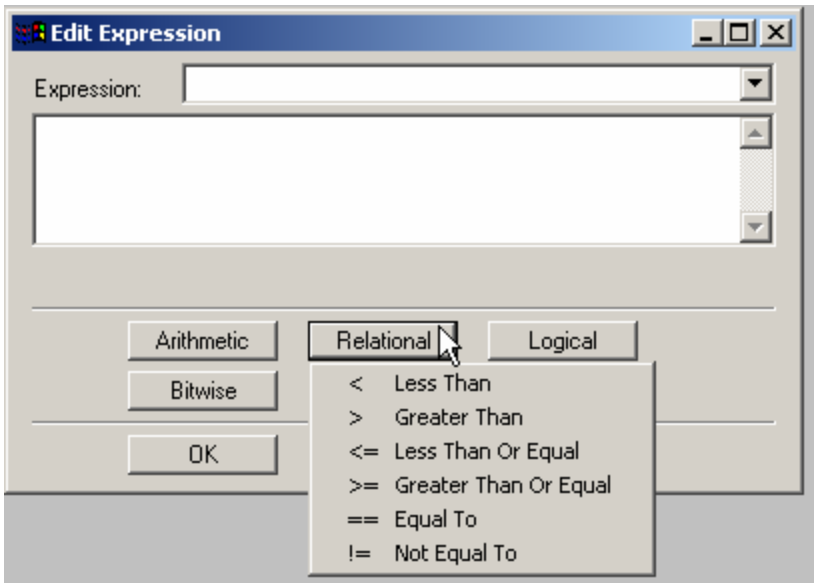
Result
The expression results in a number of any type (float, long, etc.).

Examples

Symbol	Description	Example	Result
+	Addition	~~var1~~ + ~~var2~~	8+3 = 11
-	Subtraction	~~var1~~ - ~~var2~~	8-3 = 5
*	Multiplication	~~var1~~ * ~~var2~~	8*3 = 24
/	Division	~~var1~~ / ~~var2~~	8/3 = 2.66667
%	Calculates the remainder after division	~~var1~~ % ~~var2~~	8%3 = 2
(and)	Gives precedence to parts of the calculation	~~var1~~ / (~~var2~~ + ~~var3~~)	8/(3+2) = 1.6

Relational

The **Relational** menu symbols are shown in the figure below.



Relational Symbols

The symbols '<', '>', '<=', '>=', '==' and '!=' use the following format:
expression :: parameter **symbol** parameter

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	< or > or <= or >= or == or !=

Result

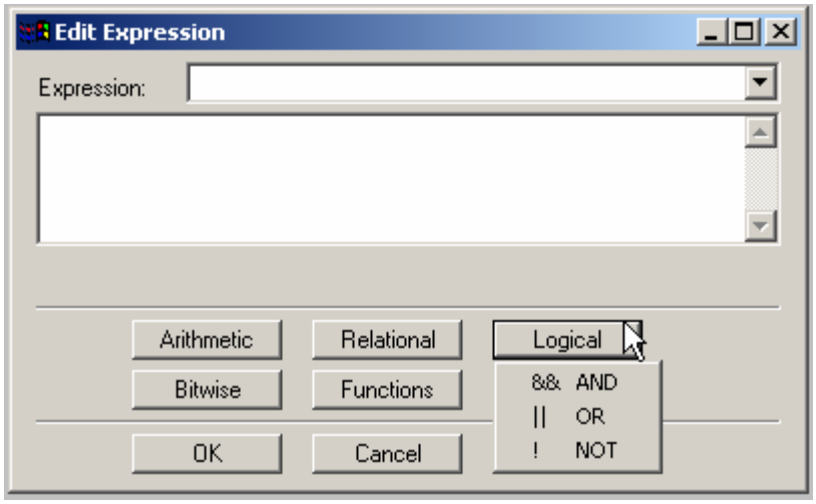
The expression results in a Boolean value (0 or 1).

Examples

Symbol	Description	Example	Result
<	Less than	~~var1~~ < ~~var2~~	8<3 = 0
>	Greater than	~~var1~~ > ~~var2~~	8>3 = 1
<=	Less than or equal to	~~var1~~ <= ~~var2~~	8<=3 = 0
>=	Greater than or equal to	~~var1~~ >= ~~var2~~	8>=3 = 1
==	Equal to	~~var1~~ == ~~var2~~	8==3 = 0
!=	Not equal to	~~var1~~ != ~~var2~~	8!=3 = 1

Logical

The **Logical** menu symbols are shown in the figure below.



Logical Symbols

The symbols '&&' and '||' use the following format:
expression :: parameter **symbol** parameter

The symbol '!' uses the following format:
expression :: **symbol** parameter

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	&& or || or !

Result
The expression results in a Boolean value (0 or 1).

Truth table

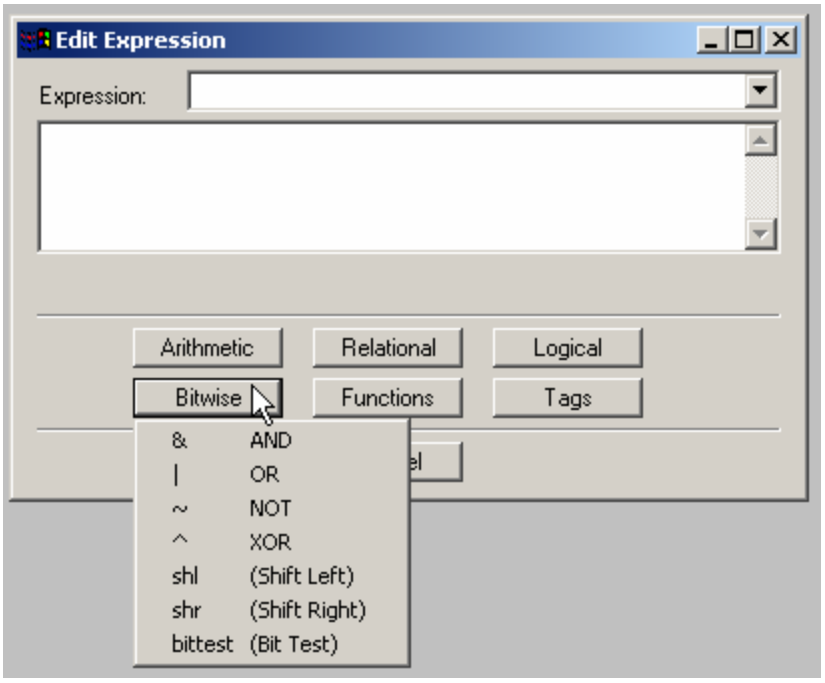
~~var1~~	0		not 0	
~~var2~~	0	not 0	0	not 0
~~var1~~ && ~~var2~~	0	0	0	1
~~var1~~ || ~~var2~~	0	1	1	1
!~~var1~~	1	1	0	0

Examples

Symbol	Description	Example	Result
&&	And	~~var1~~ && ~~var2~~	8 && 3 = 1
||	Or	~~var1~~ || ~~var2~~	8 || 3 = 1
!	Not	!~~var1~~	!8 = 0

Bitwise

The **Bitwise** menu symbols are shown in the figure below.



Bitwise Symbols

The symbols '&', '|', and '^' of the bitwise group use the following format:
expression :: parameter **symbol** parameter

The symbol '~' of the logical group uses the following format:
expression :: **symbol** parameter

The symbols 'shl' and 'shr' of the bitwise group use the following format:
expression :: **symbol** (value, shift by)

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	&& or || or ^ or shl or shr or ~

Result
The expression results in a number when the parameters used contain numbers.

Bit Table

	Binary (Decimal)	Binary (Decimal)
~~var1~~	0000.0000.0000.1000 - (8)	0000.0000.0110.0000 - (96)
~~var2~~	0000.0000.0000.1010 - (10)	0000.0000.0000.1000 - (8)
~~var1~~ & ~~var2~~	0000.0000.0000.1000 - (8)	0000.0000.0000.0000 - (0)
~~var1~~ | ~~var2~~	0000.0000.0000.1010 - (10)	0000.0000.0110.1000 - (104)
~~var1~~ ^ ~~var2~~	0000.0000.0000.0010 - (2)	0000.0000.0110.1000 - (104)
shl (~~var1~~,3)	0000.0000.0100.0000 - (64)	0000.0011.0000.0000 - (768)
shr (~~var1~~,3)	0000.0000.0000.0001 - (1)	0000.0000.0000.1100 - (12)
~(~~var1~~)	1111.1111.1111.0111 - (-9)	1111.1100.1111.1111 - (-97)
bittest(~~var1~~,3)	0000.0000.0000.0001 - (1)	0000.0000.0000.0000 - (0)

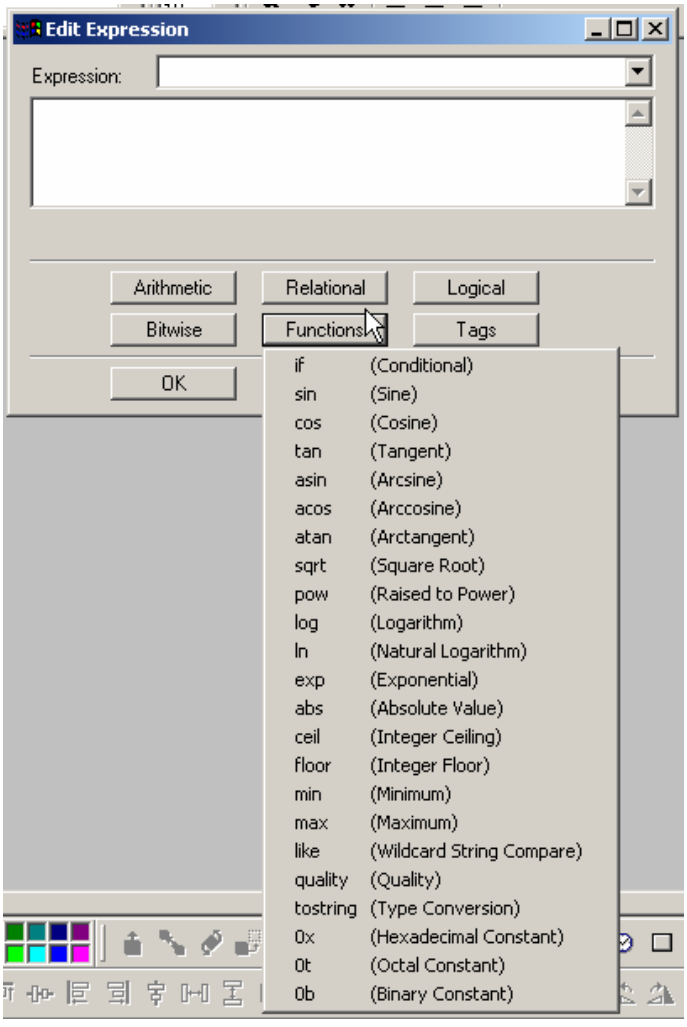
Examples

Symbol	Description	Example	Result
&	Bit And	~~var1~~ & ~~var2~~	8 && 3 = 0
|	Bit Or	~~var1~~ | ~~var2~~	8 || 3 = 11
^	Bit eXclusive Or	~~var1~~ ^ ~~var2~~	8^3=11
shl	Bit shift left	shl(~~var1~~,3)	8<<3=64
shr	Bit shift right	shr(~~var1~~,3)	8>>3=1
~	Not (two's complement)	~(~~var1~~)	!8 = -9
bittest	Bit Test	bittest (5 , 0)	1

Note: The bittest function requires you to specify the position of the bit to be tested. You must indicate that it starts from 0. In other words, a bit position of "0" indicates the "less significant" bit.

Functions

The **Functions** menu options are shown in the figure below.



Functions Menu Options

The symbols 'sin', 'asin', 'cos', 'acos', 'tan', 'atan', 'log', 'ln', 'exp', 'sqrt', 'abs', 'ceil', and 'floor' use the following format:

expression :: **symbol** (parameter)

The symbols 'pow', 'min', and 'max' use the following format:

expression :: **symbol** (parameter,parameter)

The symbol 'if' uses the following format:

expression :: **symbol** (parameter,parameter,parameter)

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	sin, asin, cos, acos, tan, atan, log, ln, exp, sqrt, abs, ceil, floor, min, max, pow, or if

Result

The expression results in a number.

Examples

Symbol	Description	Example	Result
sin	sine of an angle in radians	sin(~~var1~~)	sin(0.785)=0.71
cos	cosine of an angle in radians	cos(~~var1~~)	cos(0.785)=0.71
tan	tangent of an angle in radians	tan(~~var1~~)	tan(0.785)=1.0
asin	arc sine returns an angle in radians	asin(~~var1~~)	asin(0.5)=0.52
acos	arc cosine returns an angle in radians	acos(~~var1~~)	acos(0.5)=1.05
atan	arc tangent returns an angle in radians	atan(~~var1~~)	atan(1)=0.785
sqrt	Returns the square root	sqrt(~~var1~~)	sqrt(100)=10
pow	Returns value 1 raised to the power value 2	pow(~~var1~~,~~var2~~)	pow(100,1.5)=1000
log	10 based logarithm	log(~~var1~~)	log(100)=2
ln	*e* based logarithm	ln(~~var1~~)	ln(7.389)=2
exp	Exponential	exp(~~var1~~)	exp(2)=7.389
abs	Absolute value	abs(~~var1~~)	abs(-1)=1
ceil	Integer ceiling	ceil(~~var1~~)	ceil(7.39)=8
floor	Integer floor	floor(~~var1~~)	floor(7.39)=7
min	Lowest value of two	min(~~var1~~,~~var2~~)	min(10,5)=5
max	Highest value of two	max(~~var1~~,~~var2~~)	min(10,5)=10
if	Conditional statement	if(~~var1~~<~~var2~~, ~~var1~~,~~var2~~)	if(5<8,5,8)=5
like	Wildcard string compare	Like(string, pattern, casesensitive')	
quality	Quality of tag or expression	See below.	See below.
tostring	Type conversion	See below.	See below.
0x	Hexadecimal constant	x=0x11	17
0t	Octal constant	x=0t11	9
0b	Binary constant	x=0b11	3

Note: For the like operator: "string" equals the string to search in; "pattern" equals the string to search for (can include wildcards); nonzero for case-sensitive search; zero for case-insensitive search. String syntax is $"string"$.
You can use these special characters in pattern matches in string:

- ? Any single character.
- Zero or more characters.
- # Any single digit (0-9).
- [charlist] Any single character in charlist.
- [!charlist] Any single character not in charlist.

Quality

The **quality** option on the **Functions** menu of the **Expression Editor** is used to evaluate the quality of an OPC tag or an expression.
The following general syntax is used for quality expressions:
x=quality(expression)

Note: The "(expression)" can also be a simple expression composed of a single tag.

The **quality** function returns the OPC quality of the string between parentheses as one of the following results:

- 192: quality is GOOD
- 64: quality UNCERTAIN
- 0: quality BAD

Note: The OPC Foundation establishes the value ranges for quality. There are actually varying degrees of quality:

- GOOD: 192-252
- UNCERTAIN: 64-191
- BAD: 0-63

For more information, refer to the *OPC Data Access Custom Interface Standard* available for download at the OPC Foundation's Web site, www.opcfoundation.org/.

Example Quality Expression

Expression	Result
x=quality({{SMAR.Simulator.1\SimulatePLC.PumpStatus}})	192 (Quality GOOD)

The quality of an expression is determined through the evaluation of each single tag in the expression. Thus, if you have multiple tags in an expression (and each tag has a different quality), the result of the expression (i.e. 192 [GOOD], 64 [BAD], or 0 [UNCERTAIN]) corresponds to the quality of the tag with the lowest quality. If an expression contains a conditional statement (e.g. if, then, or else), then the result of the expression is affected only by the quality of the branch being executed.

Consider the following sample expression:
x= if (quality({{Tag1}}) == 192, {{Tag1}}, {{Tag2}})
This expression can be read as follows:
"If the quality of Tag1 is GOOD (i.e. 192), then the expression result (x) is the value of Tag1. In all other cases (i.e. the quality of Tag1 is UNCERTAIN or BAD), the expression result (x) is the value of Tag2."
We can calculate the results for this expression using different qualities for Tag1 and Tag2, as shown in the figure below.

Case	Tag1 quality	Tag2 quality	Result	Result quality
1	GOOD	GOOD	Tag1	192 (GOOD)
2	GOOD	UNCERTAIN	Tag1	192 (GOOD)
3	GOOD	BAD	Tag1	192 (GOOD)
4	UNCERTAIN	GOOD	Tag2	192 (GOOD)
5	UNCERTAIN	UNCERTAIN	Tag2	64 (UNCERTAIN)
6	UNCERTAIN	BAD	Tag2	0 (BAD)
7	BAD	GOOD	Tag2	192 (GOOD)
8	BAD	UNCERTAIN	Tag2	64 (UNCERTAIN)
9	BAD	BAD	Tag2	0 (BAD)

In cases 1-3 above, the quality of Tag1 is GOOD, and therefore the result of the expression is GOOD. Thus, the result of the expression is not affected by the quality of Tag2 (the “else” branch of the expression), which is ignored.

In cases 4-6, the quality of Tag1 is UNCERTAIN, and therefore the result of the expression is the quality of Tag2.

In cases 7-9, the quality of Tag1 is BAD, and therefore the result of the expression is the quality of Tag2.

Note: The “quality()” function returns a value that represents the quality of the expression within the parentheses but is always GOOD_QUALITY. For example, if Tag1 is BAD_QUALITY then the expression “x=quality({{Tag1}})" will return 0 with GOOD_QUALITY.

The result of an expression is the minimum quality of the evaluated tag in the expression and is affected only by the quality of the conditional (if, then, or else) branch that is executed.

Consider the following sample expression:

x= if ({{TAG_01}}>0,{{TAG_02}},{{TAG_03}})

This expression can be read as follows:

"If the value of TAG_01 is greater than 0, then the expression result (x) is TAG_02. If the value of TAG_01 is less than or equal to 0, then the expression result (x) is TAG_03."

Let's assume that the following values and qualities for these tags:

TAG_01=5 with quality GOOD

TAG_02=6 with quality UNCERTAIN

TAG_03=7 with quality BAD

Because the value of TAG_01 is 5 (greater than 0), the expression result is TAG_02. Thus, the final expression result is 6, and the final expression quality is UNCERTAIN.

Type Conversion

The **tostring** option on the **Functions** menu of the **Expression Editor** takes the value of whatever item is in parentheses and converts it into a string as follows:

The value is +(value)+unit

It can be used to convert from number to string, and it can be very useful for string concatenation.

The proper syntax for the **tostring** option is:

x=$"The value is "$ + tostring(value) + $" unit"$

Note: In the expression above, the word "unit" is placeholder text for a user-specified unit of measurement or variable (e.g. Watt, inches, meters, etc.).

Example Expressions Type Conversion

Expression	Result
x=$"The value is "$ + tostring({{gfwsim.ramp.float}}) + $" Watt"$	“The value is 543.2345152 Watt”

Constants

The **Functions** menu of the **Expression Editor** supports constant values, including hexadecimal, octal, and binary formats.

Example Expressions Using Constants

Expression	Result
x=0x11	17
x=0t11	9
x=0b11	3

The **Expression Editor** conveniently inserts the 0x and 0t and 0b prefixes for you so do not have to recall them.

Interpreting and Translating Constants

The examples below show how values are calculated for each type of constant.

- **Hexadecimal:** 0x20A = 2 * (16^2) + 0 * (16^1) + 10 * (16^0) = 2*256 + 0*16 + 10 * 1 = 512 + 0 + 10 = 522
- **Octal:** 0t36= 3 * (7^1) + 6 *(7^0) = 3* 7 + 6* 1= 21 + 6 = 27
- **Binary:** 0b110 = 1 * (2^2) + 1 * (2^1) + 0 * (2^ 0) = 1 * 4 + 1 * 2 + 0 * 1 = 4+2+0 = 6

Tags

The menu options available under the **Tags** button of the Expression Editor vary with each type of application and may include the following:

- OPC Tags
- Aliases
- Variables
- Alarm filters

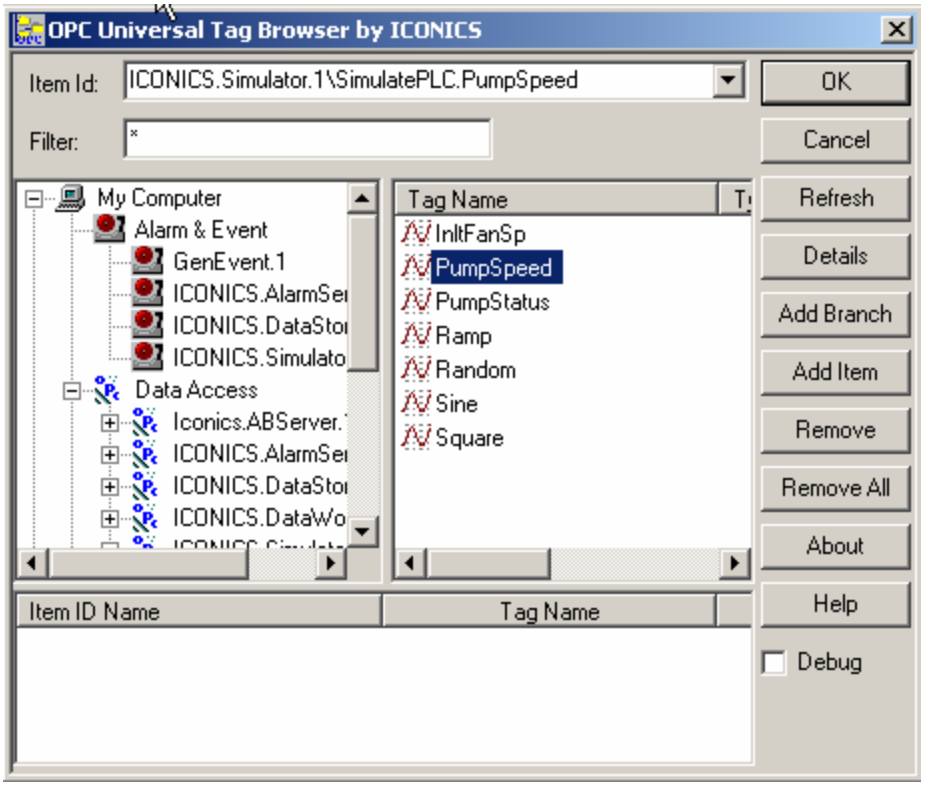
For more information about each of these options, please refer to the application's help documentation.

OPC Tags

An **OPC tag,** or data point, is a data connection between a client and an OPC server. OPC tags can be used in expressions when the tag is embedded between double brackets, as shown below:
{{tag_name**}}**

Example:
x={{SMAR.Simulator.1\SimulatePLC.PumpSpeed}}

You can use the Tag Browser, shown below, to select OPC Alarm and Event (AE), Data Access (DA), and Historical Data Access (HDA) tags to include in your expressions.



Tag Browser

Aliases

An **alias** is a string that represents or describes an object or data point in a display. Both local and global aliases can be used in expressions.

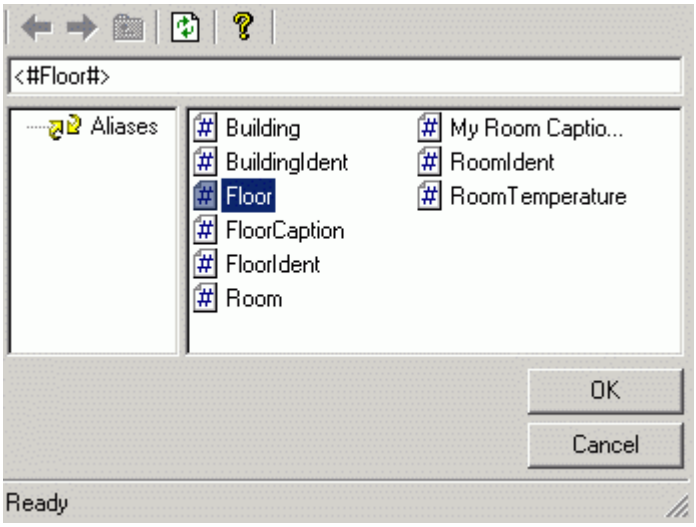
Local Aliases
For local aliases within the expression, use the following syntax:
<<local_alias_name**>>**

Example:
x=<<TankLevel>>

Global Aliases
For global aliases within the expression, use the following syntax:
<#global_alias_name**#>**

Example:
x=<#RoomTemperature#>

Selecting **Global Alias Browser** opens the Global Alias Browser, as shown in the figure below. Select a global alias from the Global Alias Browser, which includes all global aliases in the global alias database. This eliminates the need to manually type in the alias name. All global aliases that are configured in the Global Alias Engine Configurator are conveniently available to choose from inside the browser. The tree control of the Global Alias Engine Configurator is mimicked in the tree control of the Global Alias Browser. Select a global alias by double-clicking the alias name (e.g. "Floor" in the figure below). The alias name appears at the top of the browser, which automatically adds the <# and #> delimiters to the alias name. Click the **OK** button.

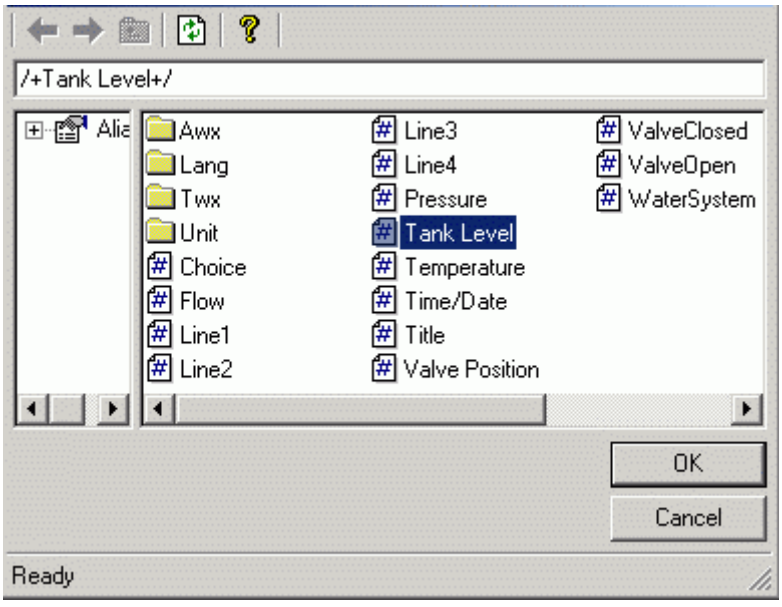


Selecting an Alias From the Global Alias Browser

Language Aliases
For language aliases within the expression, use the following syntax:
/+language_alias_name+/

Example:
x=/+WaterSystem+/

Selecting **Language Alias Browser** from the pop-up menu opens the Language Alias Browser, as shown in the figure below. The browser includes all languages aliases in the language database. All language aliases that are configured in the Language Configurator are conveniently available to choose from inside the browser. The tree control of the Language Configurator is mimicked in the tree control of the Language Alias Browser. Select a language alias by double-clicking the alias name. The alias name appears at the top of the browser, which automatically adds the /+ and +/ delimiters to the alias name. Click the **OK** button.



Selecting an Alias From the Language Alias Browser

Variables

Variables can be used in expressions. How the variable needs to be referred to depends on the type of variable. A local variable can be used in expressions when the variable is embedded between double tildes.

Local Variables
For local variables within the expression, use the following syntax:
~~local_variable_name~~

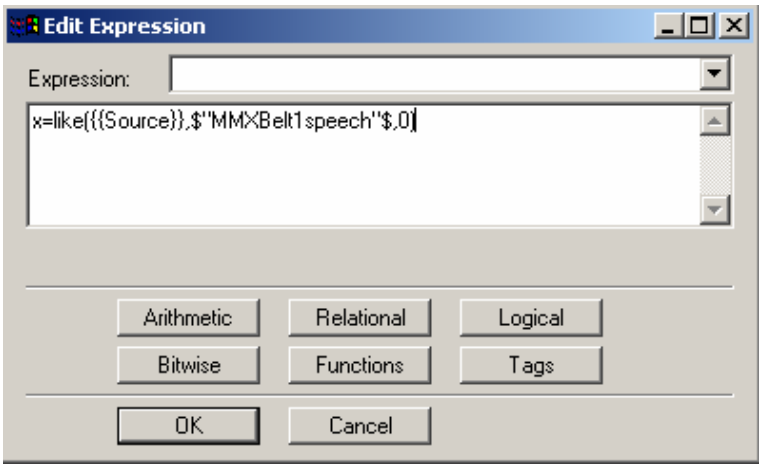
Example:
x=~~Setpoint~~

Simulation Variables
For simulation variables within the expression, use the following syntax:
{{simulation_variable_name**}}**

x={{gfwsim.random.long}}

Alarm Filters

The **Expression Editor** dialog box, shown in the figure below, can also be used to create and edit alarm filters. The Expression Editor provides a Filter Wizard and an Alarm Tag list to help you create simple alarm filters. If you want to customize your alarm filters, you can use the other functions in the Expression Editor to set up your alarm filters manually.

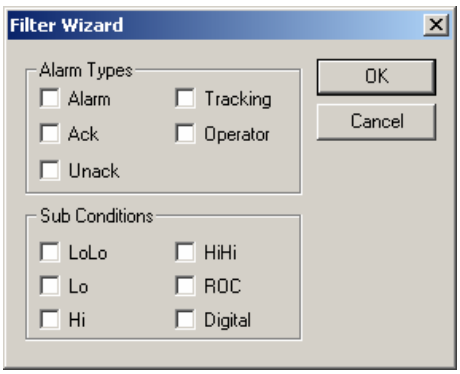


Creating Alarm Filters Using the Expression Editor

Filter Wizard

The **Filter Wizard,** shown in the figure below, allows you to choose from the following to items enter in your expression. Select one or more items, and then click **OK.** The filter string is automatically inserted into the **Edit Expression** dialog box.

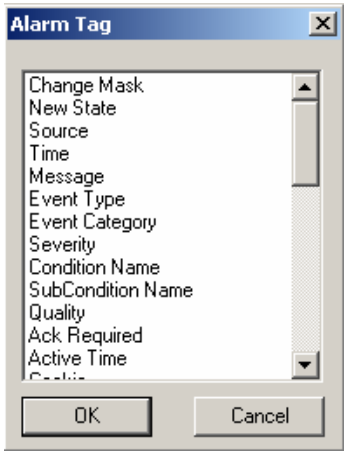
- **Alarm Types:** Alarm, Ack, Unack, Tracking, and Operator
- **Subconditions:** LoLo, Lo, Hi, HiHi, ROC, and Digital



Filter Wizard

Selecting Alarm Attributes

The **Alarm Tag** list, shown in the figure below, allows you to choose alarm attributes for your alarm filter. Select the attribute that you want to include in the filter expression and click **OK.**



Alarm Attributes List

There are two additional attributes available for use in filtering: **Alarm Type** and **Current Time.** The Alarm Type attribute allows you to filter alarms according to ALARM 1, ACK 2, UNACK 3, OPER 4, TRACK 5 or NORM 6. For example, you can set up a filter with the condition:

X = {{AlarmType}}

If the **Alarm Type** is true, then the alarms are displayed. If they are false then, the alarms are not displayed.
The **Current Time** attribute allows you to filter according to the current time. Only alarms occurring around the current time will be displayed.

Example Alarm Filters

Expression	Result
X = {{Severity}} > 500.	Only alarm messages with a severity greater than 500 will be visible.
X = Like({{Source}}, $"Tag"$,0)	Only messages with the tag in the source name will be displayed.
X = 1.	Filter displays all messages.
X = 0.	Filter does not display any messages.

All filters resolve to TRUE or FALSE. All nonzero values resolve to TRUE.

For more information, please see the AlarmWorX Server documentation.