ibaPDA

Expression Builder

Manual Part 4

Issue 8.13

Measurement Systems for Industry and Energy

www.iba-ag.com

Manufacturer

iba AG

Gebhardtstrasse 10-20

90762 Fuerth

Germany

Contacts

Main office	+49 911 97282-0
Support	+49 911 97282-14
Engineering	+49 911 97282-13
E-mail	iba@iba-ag.com
Web	www.iba-ag.com

The content of this publication has been checked for compliance with the described hardware and software. Nevertheless, discrepancies cannot be ruled out, and we do not provide guarantee for complete conformity. However, the information furnished in this publication is updated regularly. Required corrections are contained in the following regulations or can be downloaded on the Internet.

The current version is available for download on our web site www.iba-ag.com and can be found in the iba help center docs.iba-ag.com.

Version	Date	Revision	Author	Version SW
8.13	02-2026	New: MMaxDynamic, MMinDynamic; extended: ConvertFromText and ConvertToText	rm	8.13.0

Windows® is a brand and registered trademark of Microsoft Corporation. Other product and company names mentioned in this manual can be labels or registered trademarks of the corresponding owners.

Contents

1 About this documentation

This documentation describes the function and application of the software

ibaPDA.

1.1 Target group and previous knowledge

This documentation is aimed at qualified professionals who are familiar with handling electrical and electronic modules as well as communication and measurement technology. A person is regarded as professional if they are capable of assessing safety and recognizing possible consequences and risks on the basis of their specialist training, knowledge and experience and knowledge of the standard regulations.

1.2 Notations

In this manual, the following notations are used:

Action	Notation
Menu command	Menu *Logic diagram*
Calling the menu command	*Step 1 – Step 2 – Step 3 – Step x* Example: Select the menu *Logic diagram – Add – New function block*.
Keys	<Key name> Example: <Alt>; <F1>
Press the keys simultaneously	<Key name> + <Key name> Example: <Alt> + <Ctrl>
Buttons	<Key name> Example: <OK>; <Cancel>
Filenames, paths	`Filename`, `Path` Example: `Test.docx`

1.3   Used symbols

If safety instructions or other notes are used in this manual, they mean:

Danger!

⚠️ **The non-observance of this safety information may result in an imminent risk of death or severe injury!**

Observe the specified measures.

Warning!

⚠️ **The non-observance of this safety information may result in a potential risk of death or severe injury!**

Observe the specified measures.

Caution!

⚠️ **The non-observance of this safety information may result in a potential risk of injury or material damage!**

Observe the specified measures.

Note

ℹ️ A note specifies special requirements or actions to be observed.

Tip

💡 Tip or example as a helpful note or insider tip to make the work a little bit easier.

Other documentation

📖 Reference to additional documentation or further reading.

1.4   Documentation structure

This documentation fully describes the functionality of the *ibaPDA* system. It is designed both as a tutorial as well as a reference document. The sections and chapters essentially follow the procedure for configuring the system.

In addition to this documentation, you can examine the version history in the main menu, *Help – Version history* (file `versions.htm`) for the latest information about the installed version of the program. This file not only lists the bugs that have been eliminated, but also refers to extensions of the system in note form.

In addition, special "NewFeatures…" documentation comes with any software update that includes significant new features, which provides a more detailed description of the new features.

The state of the software to which the respective part of this documentation refers is listed in the revision table on page 2.

The *ibaPDA* system documentation (PDF and printed version) is divided into seven separate parts. Each part has its own section and page numbering beginning at 1, and is updated independently.

Part 1	Introduction and installation	General notes, license policy and add-ons
		Installation and program start
		User interface, system architecture, client server
		User management, printing
Part 2	I/O Manager	Basic information about the I/O Manager, general settings
		Groups and vector signals, text signals, outputs, configuration files
Part 3	Data interfaces and modules	Interfaces for the measurement data acquisition
		Standard interfaces, ibaFOB, Ethernet-based interfaces and more. For interfaces for which there are separate manuals, these are referred to.
Part 4	Expression builder	All functions for calculating virtual signals
Part 5	Data storage	Types of data recording, recording profiles, signal selection
Part 6	Data visualization	All display modes for live data, their operation and settings
Part 7	Appendix	Various additions, error listings, etc.

2   Expression builder (virtual signals)

With arithmetical and Boolean operations, you can create "virtual signals". These virtual signals can be recorded like regular input signals and/or used for creating complex trigger conditions. Using virtual signals, you can even calculate, e.g., sums, differences or check for limit violations and much more during measurement. Or you create reference signals or characteristics for further comparison in analysis.

Note

The datatype of the analog results of a computation in a virtual module is always Float, no matter what the datatype of the input signals is, which are used in the expression.

Note

In the expression builder and in all expressions and formulas the dot should be used as decimal seperator!

Note

A note about the notation of the function parameters (arguments) in tooltips and help texts:
If a value is assigned to an argument, this is the default value that is assumed if the parameter is omitted. You can only omit parameters which have a default value, e.g. *GenerateSignal('Type',* **'Amplitude=10'**, **'T1=1'**, **'T2=1'**).

In this case you may omit the arguments 2 (Amplitude), 3 (T1) and 4 (T2), if you agree to use the default values. *GenerateSignal (3,10,1,1)* means *GenerateSignal (3)*.

However, a notation like *GenerateSignal (3,,2,5)* is not permitted. If one argument is omitted all following (optional) arguments should be omitted as well. No gaps are allowed. But if following arguments are needed, the argument before must get a value or expression too. In this example the correct notation would be *GenerateSignal (3,10,2,5)*.

Note

If you want to use double quotes in a static text, then you should write two double quotes after each other.

2.1   Encryption of certain text arguments

When entering certain text arguments in expressions you can choose to encrypt the text in order to prevent readability. This mainly applies to arguments like username and password, e.g. as for the ExecuteCommand function.

When you start to type in an argument which is supported by the encryption feature, a pane opens showing the command "Encrypt text". If you want to encrypt the text, then just click on "Encrypt text". Another dialog will open with an entry field for the argument. Enter the argument and click on <OK>.

Instead of the readable text a generated string of characters with the prefix `encrypted_` will be entered as argument in the function.

2.2   Logical functions

2.2.1   Comparison functions >, >=, <, <=, <>, =

The comparative operations > (greater than), >= (greater than / equal to), < (smaller than), <= (smaller than / equal to), <> (unequal) and = (equal) enable comparisons of the values of two expressions (operands). The result of such an operation is the Boolean value TRUE or FALSE. Original signals, calculated expressions or constant values can be entered as operands. The result can be presented and evaluated as a new expression, such as a signal. So, binary signals can easily be generated and can then be used as conditions for other features.

Note.

If the crossing point of two charts is located between two measuring points, the result of the comparative operation of the last two measured values is retained until the next measuring point. I.e. that any change from TRUE to FALSE (or vice versa) is always entered in the grid of the measuring points. The line which connects two measuring points in the presentation of analog values is just a graphic approximation.

2.2.2   Boolean functions

`e.g. ('Expression1') AND ('Expression2')`

AND	Logical AND
OR	Logical OR
XOR	Logical exclusive OR
NOT	Logical NOT, negation

Description

The Boolean functions AND (logical AND), OR (logical OR), NOT (logical NOT, negation) and XOR (logical exclusive OR) can be used to connect binary expressions, such as digital signals. According to the rules of Boolean logic, the functions return the value TRUE or FALSE as their result. Digital signals, calculated (binary) expressions or the numerical values 0 or 1 can be entered as parameters.

The result can be presented and evaluated as a new expression, such as a signal. So, binary signals can easily be generated and can then be used as conditions for other features.

AND				OR				XOR				NOT	
A	B	f (A,B)		A	B	f (A,B)		A	B	f (A,B)		A	f (A)
0	0	0		0	0	0		0	0	0		0	1
1	0	0		1	0	1		1	0	1		1	0
0	1	0		0	1	1		0	1	1			
1	1	1		1	1	1		1	1	0			

Table 1: Logical functions, truth tables

Example

Graphical presentation of Boolean functions

Solution
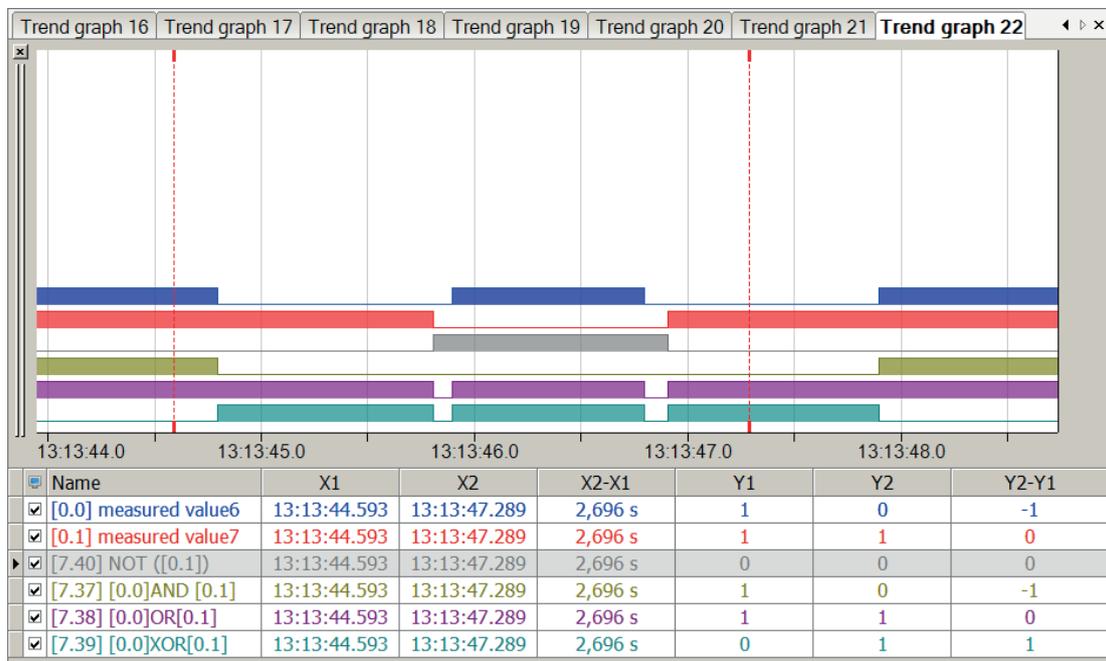
2.2.3   Bitwise logical combinations

```
e.g. 'Expression1' bw_AND 'Expression2'
```

Overview

Operation	Function
bw_AND	Bitwise AND
bw_OR	Bitwise OR
bw_XOR	Bitwise XOR
bw_NOT	Bitwise NOT

Description

The result of this function is the bitwise combination of two analog values. Preferably, this function is to be applied to datatypes like BYTE, WORD, INTEGER etc.

With floating point values (FLOAT), only the integer part will be considered, because the value is converted internally into a 32 bit integer, not rounded.

Examples

By means of a combination of two decimal numbers the following table shows the result of the operations.

Bitwise AND (bw_AND)

```
[Value A] bw_AND [Wert B]
```

Value A = 25	0	0	0	1	1	0	0	1
Value B = 14	0	0	0	0	1	1	1	0
Result bw_AND (decimal = 8)	0	0	0	0	1	0	0	0

Bitwise OR (bw_OR)

```
[Value A] bw_OR [Wert B]
```

Value A = 25	0	0	0	1	1	0	0	1
Value B = 14	0	0	0	0	1	1	1	0
Result bw_OR (decimal = 31)	0	0	0	1	1	1	1	1

Bitwise XOR (bw_XOR)

```
[Value A] bw_XOR [Wert B]
```

Value A = 25	0	0	0	1	1	0	0	1
Value B = 14	0	0	0	0	1	1	1	0
Result bw_XOR (decimal = 23)	0	0	0	1	0	1	1	1

Bitwise NOT (bw_NOT)

Returns the complement of the bits of a value.

```
bw_NOT([Value A])
```

Value A = 25	0	0	0	1	1	0	0	1
Result bw_NOT (decimal = -26)	1	1	1	0	0	1	1	0

2.2.4   ExtendPulse

```
ExtendPulse('Input','Time*')
```

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function extends a pulse on 'Input' to the minimum length of 'Time' seconds. A further rising edge on 'Input' restarts the time.

Example

Extending the pulses of a measured value to a minimum length of 0.3.

Solution

In the figure below the blue bar shows the measured value and the red bar shows the extended pulses of the measured value.
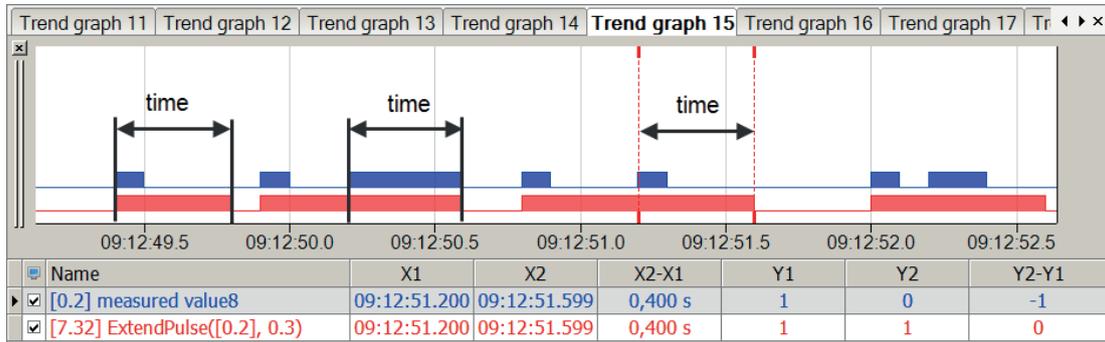
Note

By switching the edges in the sampling grid, the graphic presentation and calculation of the minimum length can be one sample above the minimum length specified under 'Time'.

2.2.5   F_TRIG

```
F_TRIG('Expression')
```

Arguments

'Expression'	Digital input signal or expression

Description

This function returns the value TRUE for 1 sample, if the change from TRUE to FALSE has been detected on 'Expression'.

Specifically, falling edges can be detected and indicated by this function.

Example

Each time the signal of a temperature alert turns off, a signal shall be generated.

Solution

Generate a digital signal *Temp_down* by using the function F_TRIG, which only turns TRUE for a short time, if there is a falling edge on the temperature alert signal *Temperature_alert*.

2.2.6   FALSE

```
FALSE()
```

Description

Returns the logical expression, FALSE or zero (0).

2.2.7   If

```
If('Condition','Expression1','Expression2')
```

Arguments

'Condition'	Condition as an operation with the Boolean results, TRUE or FALSE	
'Expression1'	Operation is performed if 'Condition' is TRUE	
'Expression2'	Operation is performed if 'Condition' is FALSE	

Description

The If-function can be used for a conditioned execution of further calculations. Depending on the Boolean result of a 'Condition', which can itself be an operation, the operation 'Expression1' will be executed if the result is TRUE and the operation 'Expression2' if the result is FALSE.

In conclusion, different process-controlled calculations can be performed. You can use this function of course in an interlaced form to realize further branches. Text signals are supported.

Tip

> If only one value is entered for 'Condition', as a condition it will be checked whether the value is greater than (TRUE) or less than (FALSE) 0.5.

Note

> Instead of "If" function you may use the "Switch" function. Moreover, the "Switch" function offers the advantage to configure more than two cases.

Example

Detecting when a measurement signal is over a set limit

Solution

The limit is formulated in 'Condition' as a condition with logical operands. If 'Condition' is TRUE, the value of 4 is returned, if FALSE then the value of 1.

In the figure below the condition is met above the dotted line (marked in green). Below the dotted line the condition is not met.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:14] measured value12	16:05:54.196	16:05:54.308	0,112 s	1,90	3,03	1,12
[7:82] If([0:14]>=2, 4, 1)	16:05:54.196	16:05:54.308	0,112 s	1,000	4,000	3,000
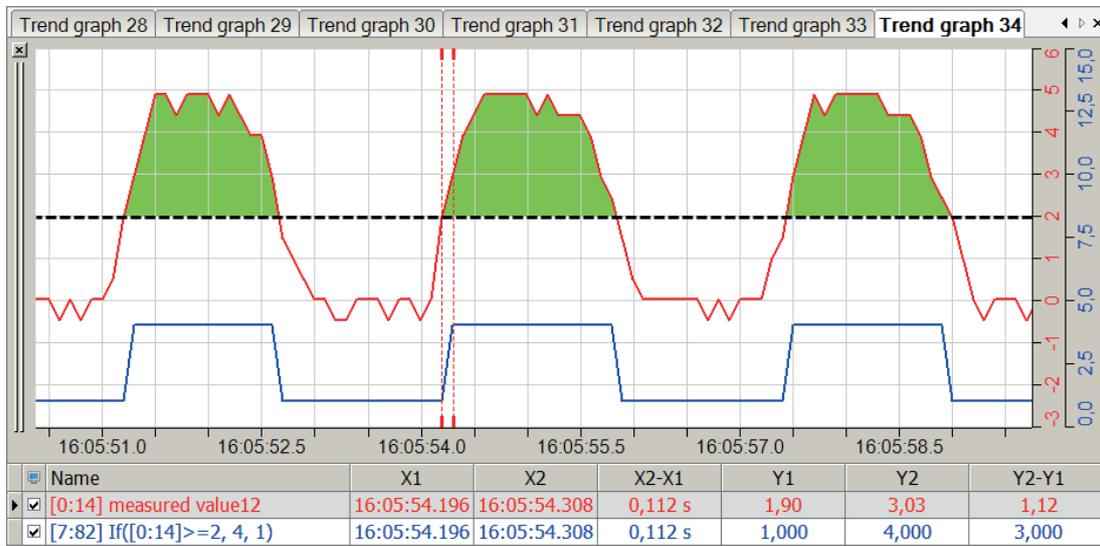
2.2.8   OneShot

```
OneShot('Expression')
```

Description

This function returns the result TRUE, if the current measured value of 'Expression' is not equal to the previous one. It returns the result FALSE, if the current measured value does equal the previous one. The function supports text signals.

Example 1

Detecting value changes

Task description

For a real signal waveform, value changes should be displayed.

Solution

In the figure below the blue curve shows the original signal and the red bar shows the area with value changes of the signal.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:2] measured value3	12:04:28.252	12:04:27.184	-1,068 s	14,00	13,00	-1,00
[7.7] OneShot ([0:2])	12:04:28.252	12:04:27.184	-1,068 s	1	0	-1

Example 2

Detecting edges

Task description

The positive edges of a digital signal should be formed.

Solution

In the figure below the blue bar shows the original signal and the red bar shows the positive edges.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0.2] measured value8	10:39:20.273	10:40:00.296	40,024 s	-	1	-
[7.7] OneShot ([0.2])AND[0.2]	10:39:20.273	10:40:00.296	40,024 s	-	1	-
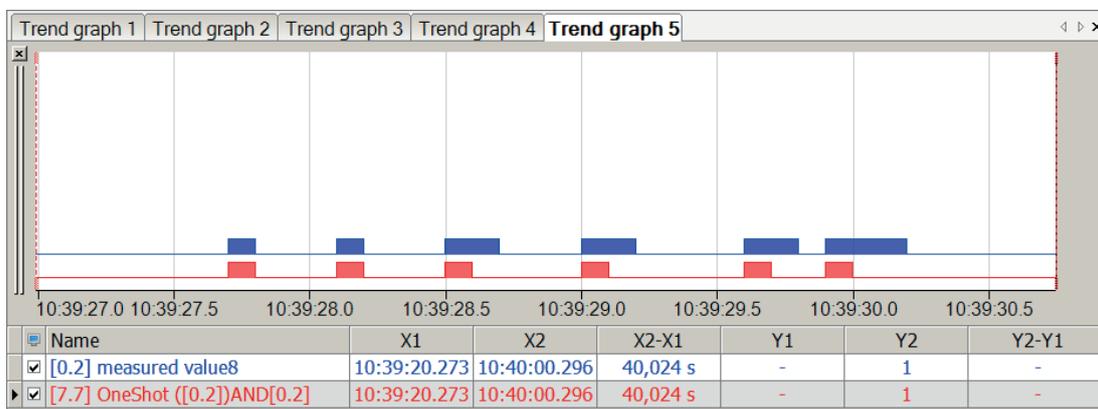
2.2.9   R_TRIG

```
R_TRIG('Expression')
```

Arguments

'Expression'	Digital input signal or expression

Description

This function returns the value TRUE for 1 sample, if the change from FALSE to TRUE has been detected on 'Expression'.

Specifically, rising edges can be detected and indicated by this function.

Example

Each time the signal of a temperature alert turns on, a signal shall be generated.

Solution

Generate a digital signal *Temp_up* by using the function R_TRIG, which only turns TRUE for a short time, if there is a rising edge on the temperature alert signal *Temperature_alert*.
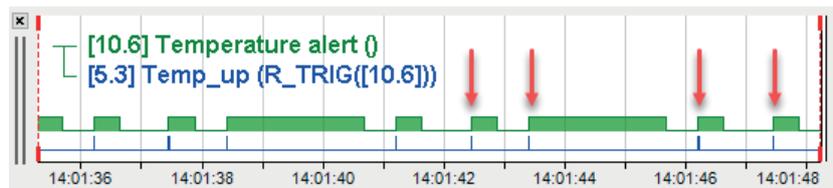
2.2.10   SetReset

```
SetReset('Set','Reset','SetDominant=1*')
```

Arguments

'Set'	Positive edge sets function to TRUE	
'Reset'	Positive edge sets function to FALSE	
'Setdominant*'	Optional parameter (default = 1), which controls which input argument is dominant if both arguments simultaneously receive a positive edge.	
	'Setdominant' = 1	Set takes precedence over Reset
	'Setdominant' = 0	Reset takes precedence over Set

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function can be used to control a digital result (TRUE/FALSE) with the help of positive edges (transition from 0 to 1) of the arguments 'Set' and 'Reset'.

A positive edge of 'Set' returns a static TRUE as result. A positive edge of 'Reset' resets the result to FALSE. The argument *'SetDominant'* is optional and determines the dominance of 'Set' or 'Reset'.

Tip

For an analog signal, exceeding the value 0.5 corresponds to a positive edge.

Example

This function can be used to enable conditional calculations with a signal and to disable them with another signal (e.g. combined with IF function).

The figure below shows the function SetReset with digital and analog signals. For simultaneous edges, the parameter determined under 'SetDominant' is set. For staggered edges, the 'Set-Dominant' parameter has no effect.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0.0] measured value6	11:21:38.396	11:21:38.396	0,000 s	1	1	0
[0.1] measured value7	11:21:38.396	11:21:38.396	0,000 s	1	1	0
[7.10] SetReset ([0.0], [0.1])	11:21:38.396	11:21:38.396	0,000 s	1	1	0
[7.11] SetReset ([0.0], [0.1], 0)	11:21:38.396	11:21:38.396	0,000 s	1	1	0

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:4] measured value4	11:34:32.586	11:34:53.400	20,814 s	0,7540	0,8956	0,1416
[0:5] measured value5	11:34:32.586	11:34:53.400	20,814 s	0,7170	0,7524	0,0354
[7.8] SetReset ([0:4], [0:5])	11:34:32.586	11:34:53.400	20,814 s	0	0	0
[7.9] SetReset ([0:4], [0:5],0)	11:34:32.586	11:34:53.400	20,814 s	0	0	0

2.2.11   Switch

```
Switch('Selector', 'Case1', 'Value1', 'Case2', 'Value2',..., 'Default')
```

Arguments

'Selector'	Expression, which is to be checked against different conditions
'CaseN'	Expression, which is to be compared to 'Selector'
'ValueN'	Result, if 'Selector' and 'CaseN' match
'Default'	Result, if none of the 'CaseN' matches 'Selector'

Description

This command compares a 'Selector' expression to any number of 'CaseN', referring to the SQL statement CASE. At least 3 arguments are required. In case of an even number of arguments the last one will be considered automatically as 'Default', which will be taken if no 'CaseN' expression matches the 'Selector' expression.

If 'Selector' matches 'CaseN' the respective 'ValueN' will be returned as a result. If more than one 'CaseN' match 'Selector', the first 'CaseN' expression will be taken automatically.

The following signals are permitted to be used as 'Selector':

■ A numerical constant

■ A text constant

■ An equidistant or non-equidistant sampled channel

■ A text channel

Basically, the types of the expressions to be compared must match, otherwise the respective case will not be selected.

2.2.12   TOF

```
TOF('IN','PT*')
```

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

Off Delay Timer. The output is switched off 'PT' seconds after switching off the 'IN' input.

Example

Delaying switching off by one second

Solution

In the figure below the blue bar shows the measured value and the red bar shows the output value with output delayed by one second.
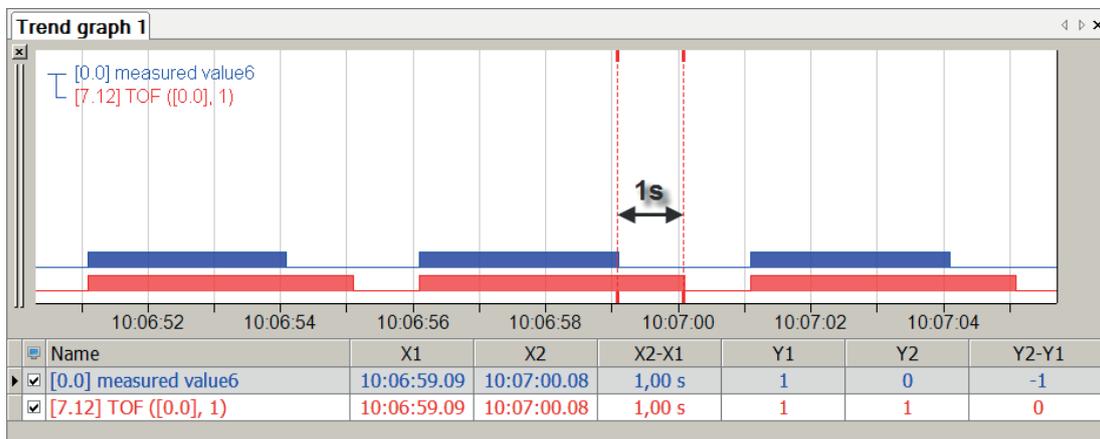
2.2.13   Toggle

```
Toggle('Trigger', 'Initial=0')
```

Arguments

'Trigger'	Trigger signal (rising edge), which toggles the output
'Initial=0'	Value at start of acquisition
	If this argument is omitted, the start value is 0.

Description

This function toggles the output with each rising edge on 'Trigger'. The last parameter 'Initial' is optional and determines the value at start of acquisition.

Example

The rising edges of a temperature alert signal *Temperature alert* should switch a digital signal *NewRisingEdge* on and off.

Solution
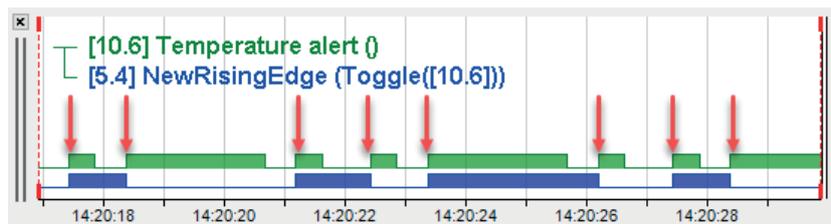
2.2.14   TON

`TON('IN','PT*')`

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

On Delay Timer. The output is switched on 'PT' seconds after switching on the 'IN' input.

Example

Delaying switching on by one second

Solution

In the figure below the blue bar shows the measured value and the red bar shows the input value with input delayed by one second.
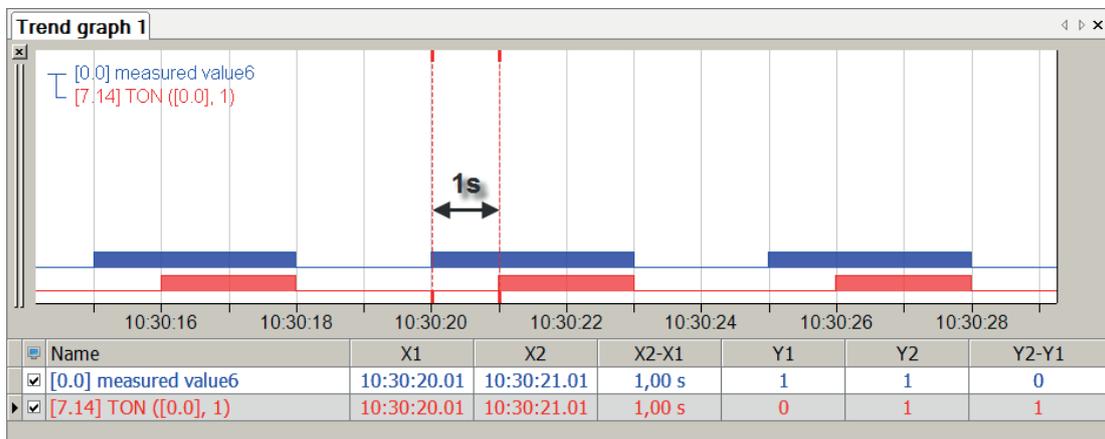
2.2.15 TP

```
TP('in','pt*')
```

Parameters ending with * are only evaluated once at the start of the acquisition..

Description

Pulse Timer. The output is switched on for 'PT' seconds after rising edge at the 'IN' input.

Tip.	
💡	Another rising edge during the output pulse does not extend the output pulse and does not restart the pulse.

Example

Generating a 0.5 second pulse from a random signal

Solution

In the figure below the blue bar shows the measured value and the red bar shows the output value in 0.5 second pulses.
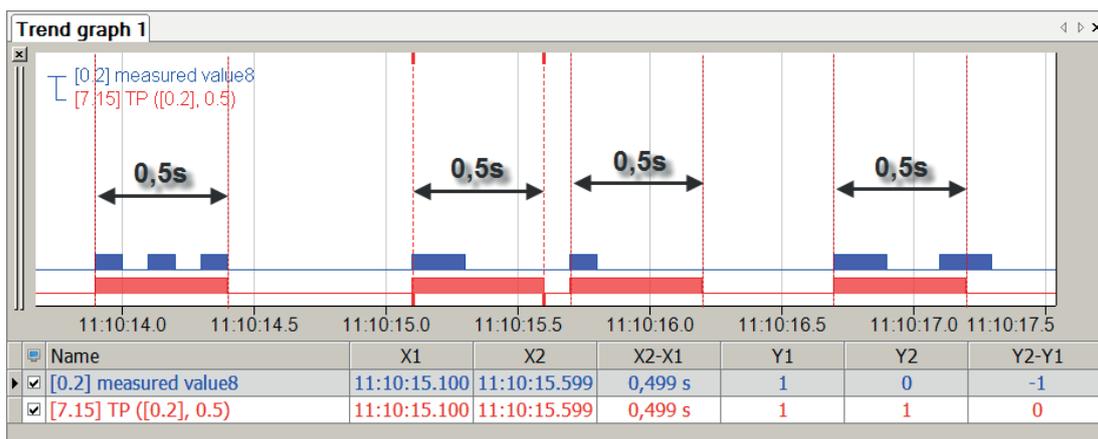
2.2.16 TRUE

```
TRUE()
```

Description

Returns the logical expression, TRUE or 1.

2.3   Mathematical functions

2.3.1   Fundamental arithmetic operations +, -, *, /

```
('Expression1')+ ('Expression2')
```

Description

All signals and expressions can be processed by fundamental arithmetic operations (addition, subtraction, multiplication and division). If digital signals or expressions are used as operands in fundamental arithmetic operations, the program translates the TRUE value as 1.0 and FALSE as 0.0. The result of a fundamental arithmetic operation is always an analog expression.

Example

Graphical presentation of fundamental arithmetic operations

Solution

The figure below shows the resulting charts for the different fundamental arithmetic operations.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [0:0] measured value1	15:40:12.548	15:40:16.577	4,029 s	18,11	20,68	2,57
☑ [0:1] measured value2	15:40:12.548	15:40:16.577	4,029 s	25,16	4,00	-21,16
☑ [7:68] [0:0]-[0:1]	15:40:12.548	15:40:16.577	4,029 s	-7,05	16,68	23,73
☑ [7:67] [0:0]+[0:1]	15:40:12.548	15:40:16.577	4,029 s	43,27	24,68	-18,59
☑ [7:69] [0:0]*[0:1]	15:40:12.548	15:40:16.577	4,029 s	455,60	82,69	-372,91
☑ [7:70] [0:0]/[0:1]	15:40:12.548	15:40:16.577	4,029 s	0,72	5,17	4,45
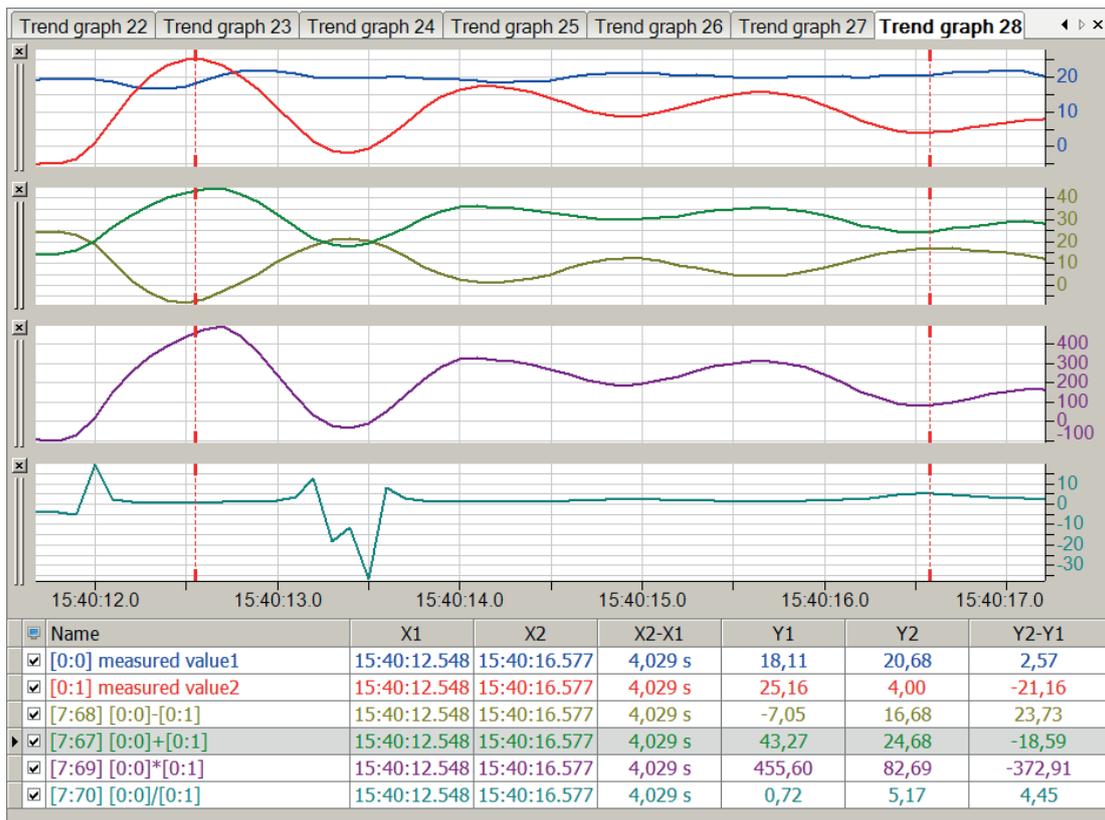
Blue	Measured value 1	Red	Measured value 2
Green	Addition	Yellow	Subtraction
Purple	Multiplication	Light blue	Division

2.3.2   Abs

```
Abs('Expression')
```

Description

The absolute function returns the absolute value (= |value|) of 'Expression.'

Example

Only the absolute value of the value from a measured signal is relevant

Solution

In the figure below the blue curve shows the original signal and the red curve is the absolute value of the signal.

Tip

> Interpolated values in the case of a change of sign between two samples may differ in value.

2.3.3   Add

```
Add('Expression', 'Enable', 'Reset=0')
```

Arguments

'Expression'	Analog input signal or expression	
'Enable'	Digital input signal or expression; *'Enable'* = 1 enables the addition	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation:	
	' *Reset*' = 0	Perform calculation
	' *Reset*' = 1	Stop calculation and set result to 0

Description

This function adds the value of 'Expression' and its own actual value every cycle if 'Enable' = TRUE or 1. If 'Enable' = FALSE or 0, the last actual value will be held. 'Reset' = TRUE or 1 resets the actual value to 0.

Example

Each time a pushbutton is pressed, an analog value should be increased by 10 every cycle. With a reset-pushbutton the value should be reset to 0.

Solution
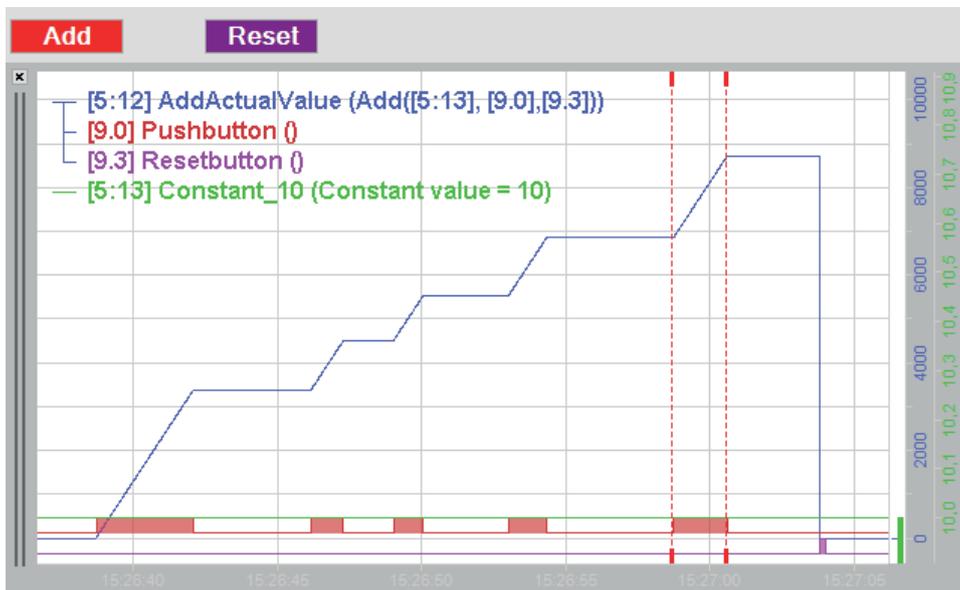
2.3.4   Ceiling

```
Ceiling('Expression')
```

Description

This function returns the smallest integer value that is greater than or equal to 'Expression'.

Example

Rounding up values of a signal waveform to whole numbers.

Task description

From a real signal waveform, the measured value should be rounded up to the nearest integer value for each sample.

Solution

In the figure below the blue curve shows the original signal and the red curve shows the signal trend with rounded-up values.
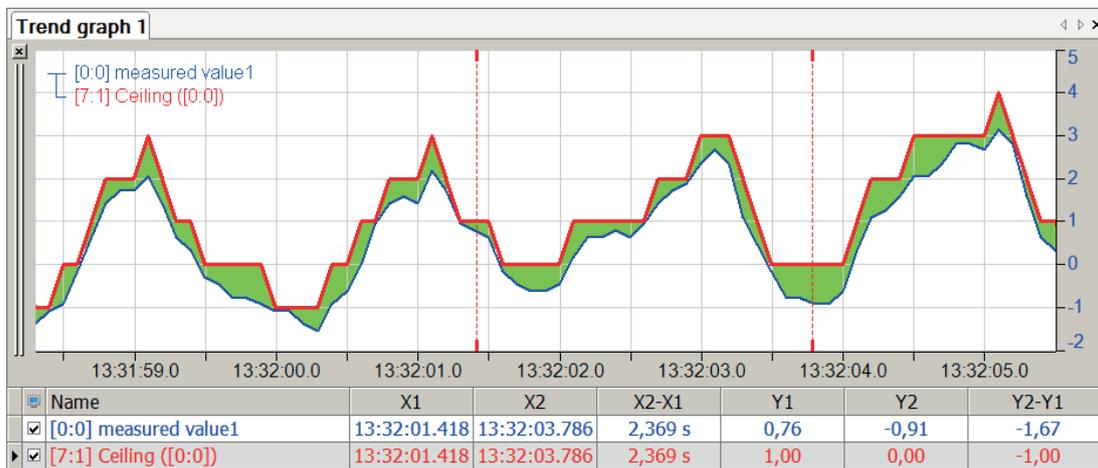
2.3.5   Diff

```
Diff('Expression')
```

Description

This function returns the differential dx/dt of 'Expression'.

Example

If 'Expression' is a length measuring signal, this diff-function can be used to determine a speed curve.

Task description

The speed curve should be determined from a length measuring signal and, from this, the acceleration curve.

Solution

Through repeated (iterative) performance of the diff-function using the values of the first as 'Expression' we obtain the curve of the acceleration.

In the figure below the blue curve shows the covered distance ("length"), the red curve shows the calculated course of speed and the green curve shows the calculated course of acceleration.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:0] measured value1	15:10:26.698	15:10:26.954	0,256 s	0,318	1,113	0,794
[7:3] Diff ([0:0])	15:10:26.698	15:10:26.954	0,256 s	0,202	3,125	2,923
[7:4] Diff ([7:3])	15:10:26.698	15:10:26.954	0,256 s	-104,48	0,00	104,48
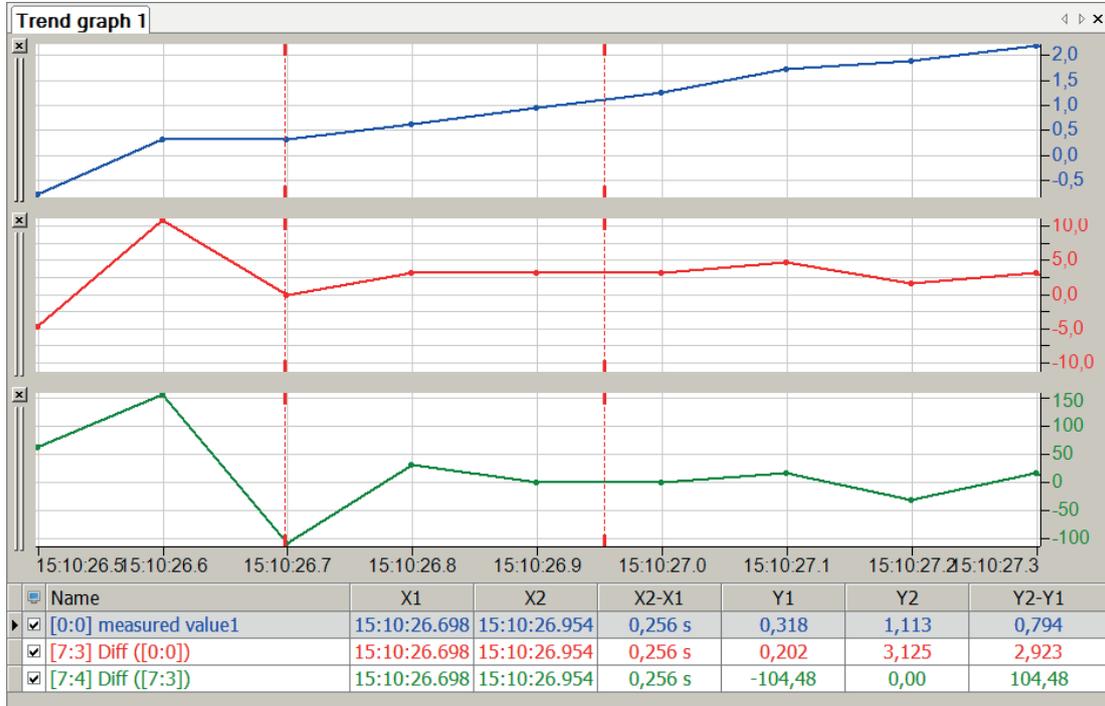
Tip

If only the course of acceleration is of interest and speed is not relevant, this can also be determined directly by recursively using the

Diff(Diff([0:0])) function.

2.3.6   Eff

```
Eff('Expression','Frequency')
```

Description

This function calculates the effective value of 'Expression' with a fundamental frequency of 'Frequency'.

The following formula is used to calculate the effective value:

$$E_{eff} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} e^2(n)}$$

e(n): Value of sample n of signal e ('Expression')
N: Number of samples per period

Note

| i | The result of the effective value function is always displayed in the subsequent cycle of the fundamental frequency. |

Example

For an alternating current course with a frequency of 0.1 kHz, which is overlaid by a second AC voltage with 0.5 kHz, the effective value of the voltage should be determined for both frequencies.

Solution

In the figure below the blue curve shows the AC current signal and the green curve shows the effective voltage with calculation frequency 0.5 kHz. The red straight line shows the effective voltage with calculation frequency 0.1 kHz.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:10] measured value11	11:12:48.19	11:12:50.19	2,00 s	2,000	-0,191	-2,190
[7:72] Eff([0:10], 0.1)	11:12:48.19	11:12:50.19	2,00 s	1,000	1,000	0,000
[7:73] Eff([0:10], 0.5)	11:12:48.19	11:12:50.19	2,00 s	0,335	1,651	1,316
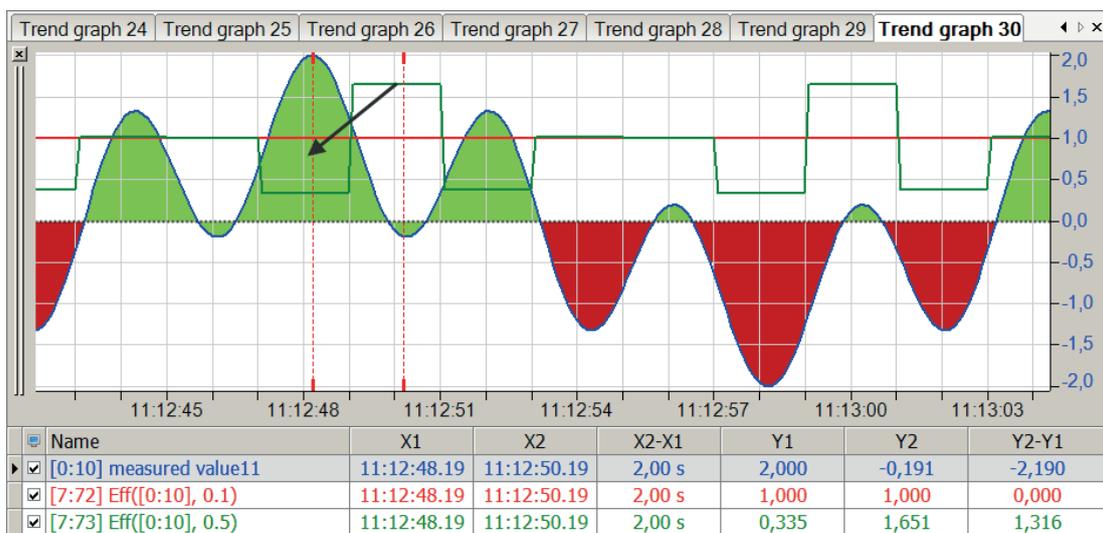
2.3.7   Exp

`Exp('Expression')`

Description

This function calculates the result of (e) $^{'Expression'}$

Example

Graphical presentation of the exp-function

Solution

In the figure below the blue curve shows the time function and the red curve shows the exponential function of time.
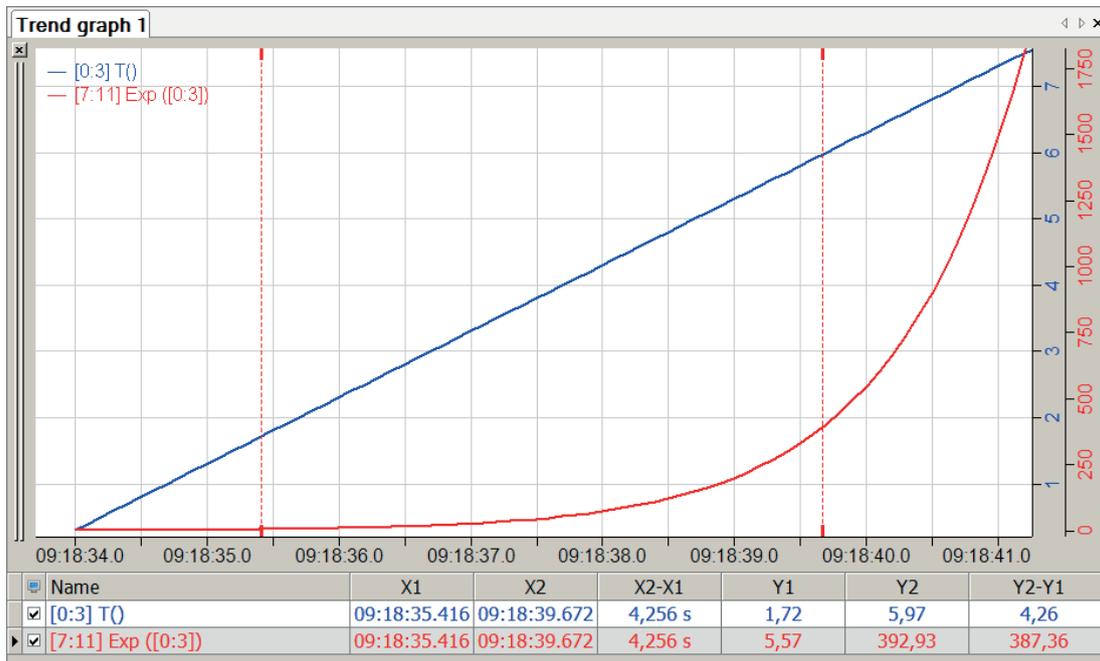
2.3.8   Floor

`Floor('Expression')`

Description

This function returns the largest integer value that is less than or equal to 'Expression'.

Example

Rounding down values of a signal waveform to whole numbers.

Task description

From a real signal waveform, the measured value should be rounded down to the nearest integer value for each sample.

Solution

In the figure below the blue curve shows the original signal and the red curve shows the signal trend with rounded-down values.
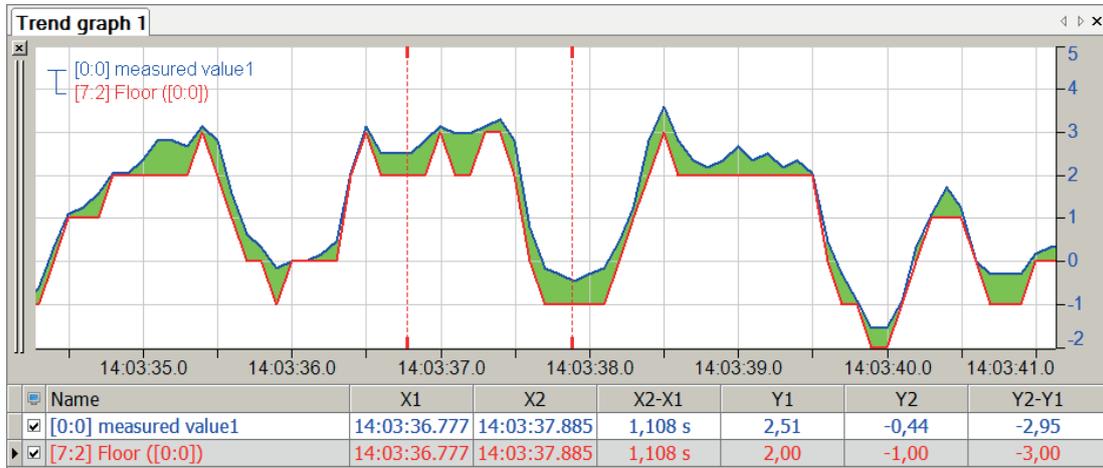
2.3.9 Int

```
Int('Expression','Reset')
```

Arguments

'Expression'	Measured value	
'Reset'	Optional digital parameter, which can be used to reset the integral or suppress the integration process. 'Reset' can be an expression as well.	
	'Reset' > 0	Integral is reset.
	'Reset' = 0	Integration released (default)

Description

This function returns the integral (y * dt) of 'Expression' as its result. The 'Reset' parameter can be used for resetting the integral to zero or suppressing the integration process, e.g. to integrate the same signal for periodical occurrences or reversing processes a number of times. 'Reset' can be an expression as well.

Examples:

Int([0:0])	No reset happens ('Reset' omitted)
Int([0:0],If(Mod(T(),20)=0,TRUE(),FALSE()))	The integral is reset every 20 seconds.
Int([0:0], [3.1])	e.g. with [3.1] = If([0:0]>10, 1, 0)
	The integral is reset as soon as the expression [3.1] returns TRUE, i.e. if the expression [0:0] exceeds the limit value 10.

Example 1

Tip

This function can be used in a virtual retentive module. Its result values can thus be obtained via stopping and restarting the measurement.

If 'Expression' is an acceleration signal, the distance covered can be determined by iteratively performing the Int-function.

Task description

The speed and distance covered should be determined with the help of an acceleration sensor.

Solution

In the figure below the blue curve shows the measured acceleration, the red curve shows the calculated course of speed and the green curve shows the calculated distance covered.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:6] measured value9	10:06:39.158	10:06:39.630	0,472 s	0,880	-1,065	-1,945
[7:8] Int ([0:6])	10:06:39.158	10:06:39.630	0,472 s	0,444	0,445	0,001
[7:9] Int ([7:8])	10:06:39.158	10:06:39.630	0,472 s	0,1795	0,4023	0,2228
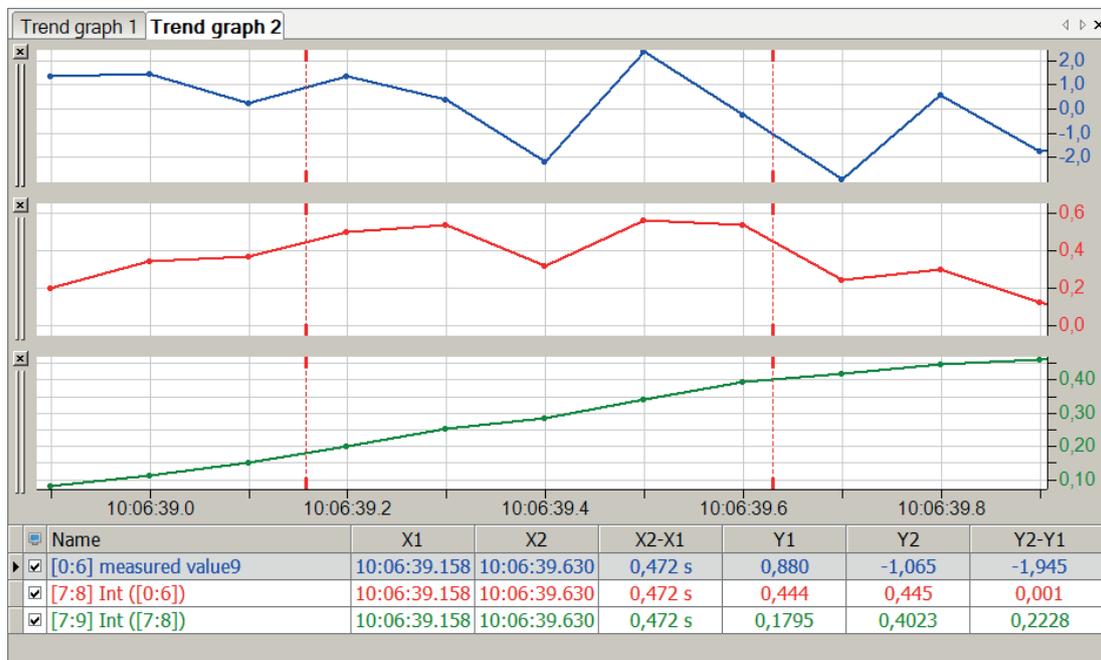
Tip

If only the distance covered is of interest, this can also be determined by recursively using the `Int(Int([0:6]))` function.

Example 2

The integral should be restarted at an interval of 20 seconds.

Task description

The modulo function can be used to reset the integral since every 20 seconds T() mod 20 = 0 applies.

Solution

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [0:3] T()	08:49:17.06	08:49:37.04	19,98 s	40,06	60,04	19,98
☑ [7:7] Mod ([0:3],20)	08:49:17.06	08:49:37.04	19,98 s	0,06	0,04	-0,02
▶ ☑ [7:6] Int ([0:0], If (Mod(T(),20)...	08:49:17.06	08:49:37.04	19,98 s	0,15	0,09	-0,06
☑ [0:0] measured value1	08:49:17.06	08:49:37.04	19,98 s	2,52	1,77	-0,75
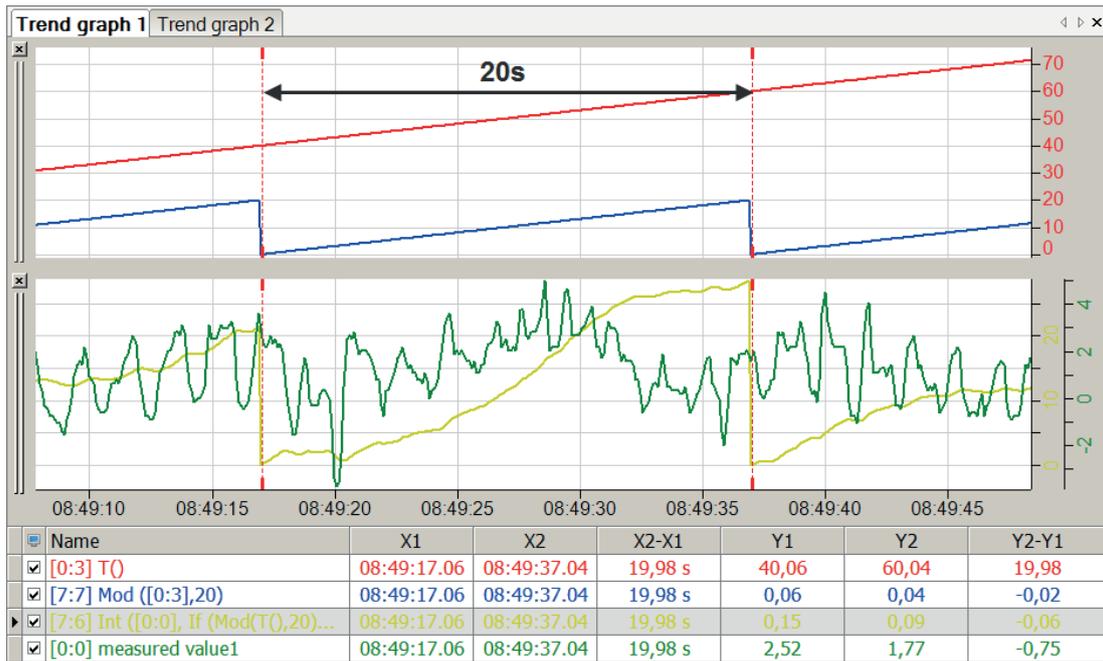
Red	Time function	Blue	Modulo20 of the time function
Green	Measured value	Yellow	Integral of the measured value with 20 seconds reset

2.3.10   Log

```
Log('Expression')
```

Description

This function returns the natural logarithm (ln x) of 'Expression' as its result.

Example

Calculating a known natural logarithm.

Solution

Logarithm of Euler's number e must constantly give 1: ln e = 1
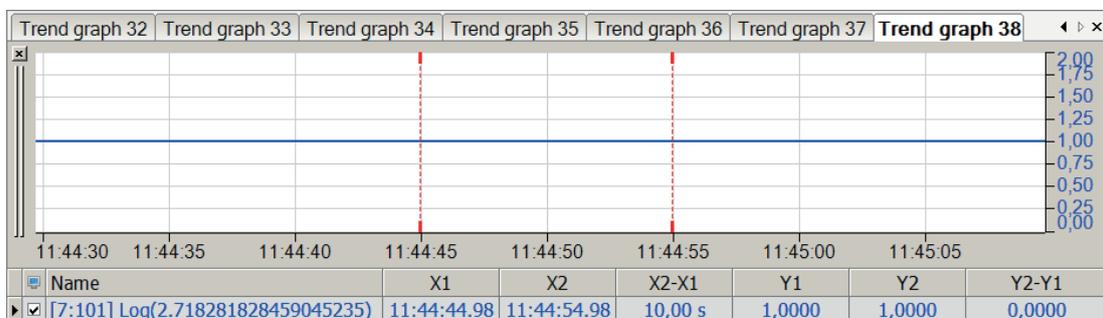
Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
▶ ☑ [7:101] Log(2.718281828459045235)	11:44:44.98	11:44:54.98	10,00 s	1,0000	1,0000	0,0000

Tip

Although negative values for 'Expression' do not produce an error message, they do not produce a result either.

2.3.11   Log10

```
Log10('Expression')
```

Description

This function returns the decadic logarithm (lg x) of 'Expression' as its result.

Example

Calculating known decadic logarithms.

Solution
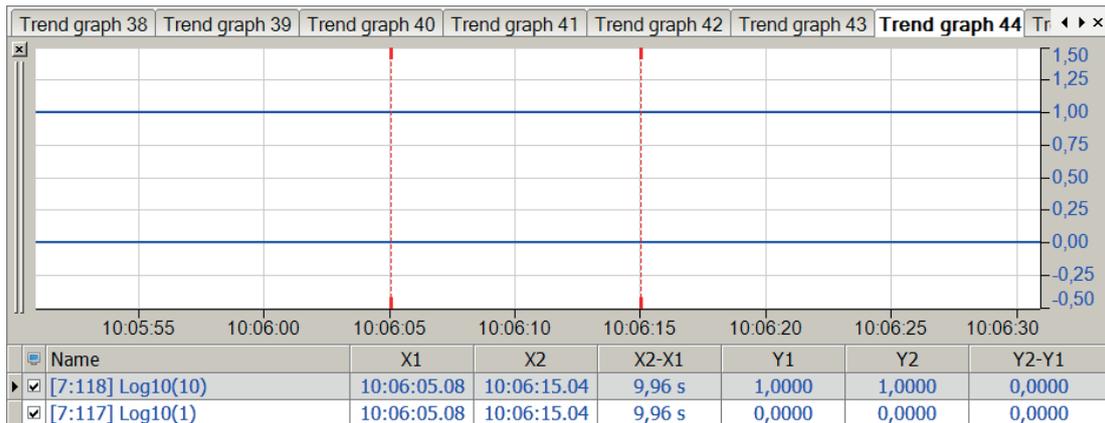
lg 1 = Log10(1) = 0

lg 10 = Log10(10) = 1

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [7:118] Log10(10)	10:06:05.08	10:06:15.04	9,96 s	1,0000	1,0000	0,0000
☑ [7:117] Log10(1)	10:06:05.08	10:06:15.04	9,96 s	0,0000	0,0000	0,0000

Tip

Although negative values for 'Expression' do not produce an error message, they do not produce a result either.

2.3.12   Mod

```
Mod('Expression1','Expression2')
```

Description

This function returns the modulo of 'Expression1' and 'Expression2' as its result. Internally, the function uses the fmod C-function, which permits the use of floating-point values for 'Expression1' and 'Expression2'.

Modulo r is the remainder of the division Expression1 / Expression2 so that the following relationship applies in the opposite direction:

Expression1 = Expression2 * i + r, where i is an integer number (integer).

Modulo r always has the same sign as 'Expression1' and the absolute value of r is always smaller than the absolute value of 'Expression2'.

If Expression1' < 'Expression2', the function returns the value of 'Expression1' as its result. Mathematically speaking, the modulo can also be described as "Expression1 modulo Expression2".

Examples: Mod(7, 3) = 1; seven divided by three equals two, remainder 1.

Mod(20.0, 10.5) = 9.5

Example

Only the seconds from the time function are relevant; hours and minutes should be cut off

Solution

With the help of Modulo60 of the time function, only the seconds are kept as the remainder.

In the figure below the blue curve shows the time function and the green curve shows Modulo60 of the time function.
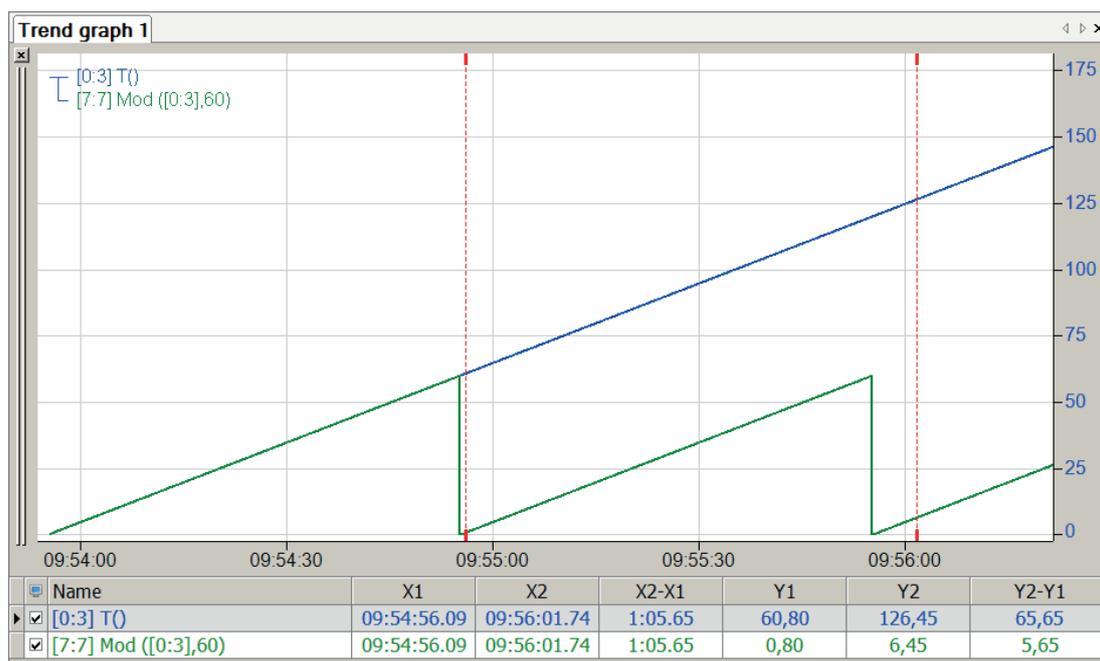
2.3.13   Pow

```
Pow('Expression1','Expression2')
```

Arguments

'Expression1'	Basis
'Expression2'	Exponent

Description

This function increases 'Expression1' (basis) to the power of 'Expression2' (exponent): ('Expression1')'Expression2'

Example

Calculating some important powers

Solution

(2)0 = Pow(2, 0) = 1

(2)-2 = Pow(2, -2) = 0.25

(-2)2 = Pow(-2, 2) = 4

(10)lg 2 = Pow(10, lg 2) = 2

(0)-1 = Pow(0, -1) = +∞ (infinity)

| Trend graph 39 | Trend graph 40 | Trend graph 41 | Trend graph 42 | Trend graph 43 | Trend graph 44 | **Trend graph 45** |

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[7:119] Pow(2, 0)	10:20:55.00	10:21:10.03	15,03 s	1,000	1,000	0,000
[7:120] Pow(2, -2)	10:20:55.00	10:21:10.03	15,03 s	0,250	0,250	0,000
[7:121] Pow(-2, 2)	10:20:55.00	10:21:10.03	15,03 s	4,000	4,000	0,000
[7:122] Pow(10, Log10(2))	10:20:55.00	10:21:10.03	15,03 s	2,000	2,000	0,000
[7:123] Pow(0, -1)	10:20:55.00	10:21:10.03	15,03 s	+unendlich	+unendlich	-

Tip

The exponentiation of 0 with -1, which is equivalent to a division by 0, does not return an error message but the limit value +infinity.

2.3.14   Round

```
Round('Expression')
```

Description

This function rounds 'Expression' up or down to the nearest whole number (integer).

Example

Rounding values of a signal waveform to whole numbers

Task description

From a real signal waveform, the measured value should be rounded to the nearest integer value for each sample.

Solution

In the figure below the blue curve shows the original signal and the red curve shows the signal trend with rounded values.
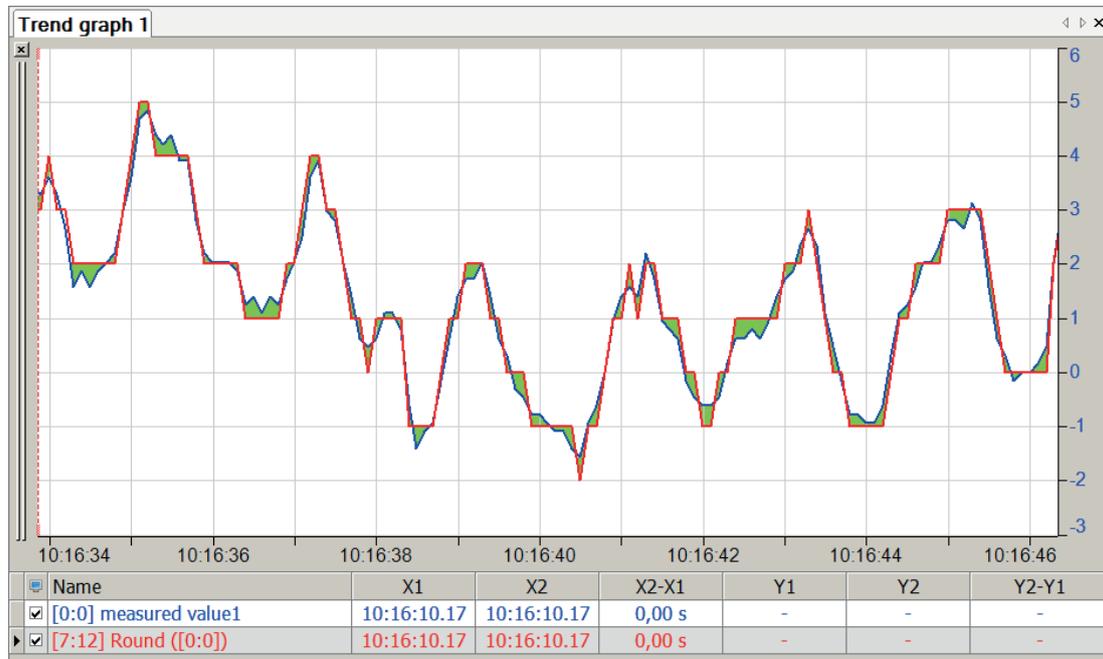
2.3.15   Sqrt

```
Sqrt('Expression')
```

Description

This function returns the square root of 'Expression' as its result.

Example

Calculating some known square roots

Solution

√(4) = Sqrt(4) = 2

√(2) = Sqrt(2) = 1.41421356…

√(-1) = Sqrt(-1) = i (complex calculation required)

Tip

Although negative values for 'Expression' do not produce an error message, they do not produce a result either.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [7:124] Sqrt(4)	10:40:40.04	10:40:54.97	14,94 s	2,000	2,000	0,000
☑ [7:125] Sqrt(2)	10:40:40.04	10:40:54.97	14,94 s	1,414	1,414	0,000
☑ [7:126] Sqrt(-1)	10:40:40.04	10:40:54.97	14,94 s	-	-	-
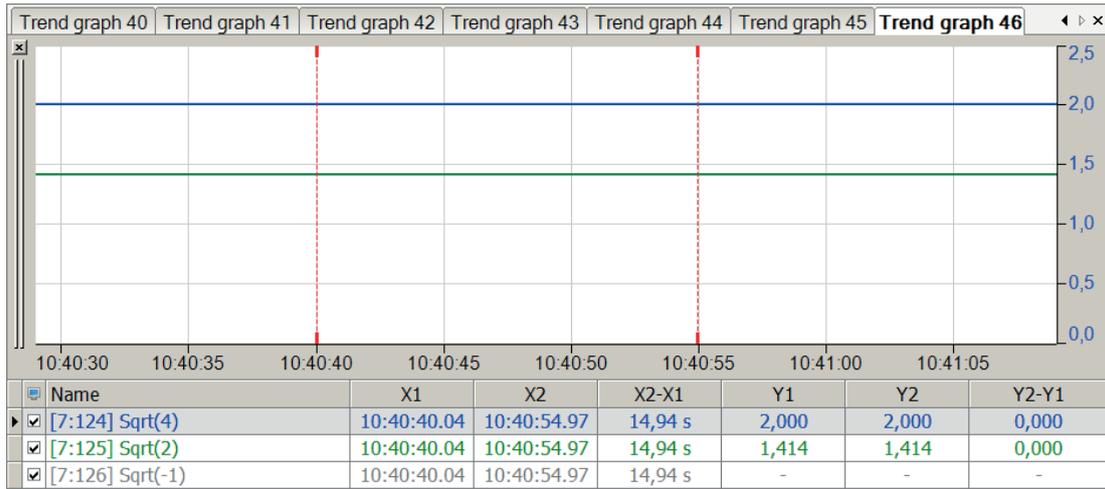
2.3.16   Truncate

```
Truncate('Expression')
```

Description

The truncate function truncates the decimal places of a floating point value. Negative numbers are thus rounded up to the nearest integer value, positive numbers rounded down.

Example

Truncating the decimal places of the values of a signal waveform

Task description

From a real signal waveform, the measured value should be displayed without decimal places for each sample.

Solution

In the figure below the blue curve shows the original signal and the red curve shows the signal trend with truncated decimal places.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [0:0] measured value1	08:45:12.241	08:45:12.955	0,715 s	-0,87	-3,36	-2,50
☑ [7:71] Truncate([0:0])	08:45:12.241	08:45:12.955	0,715 s	0,00	-3,00	-3,00
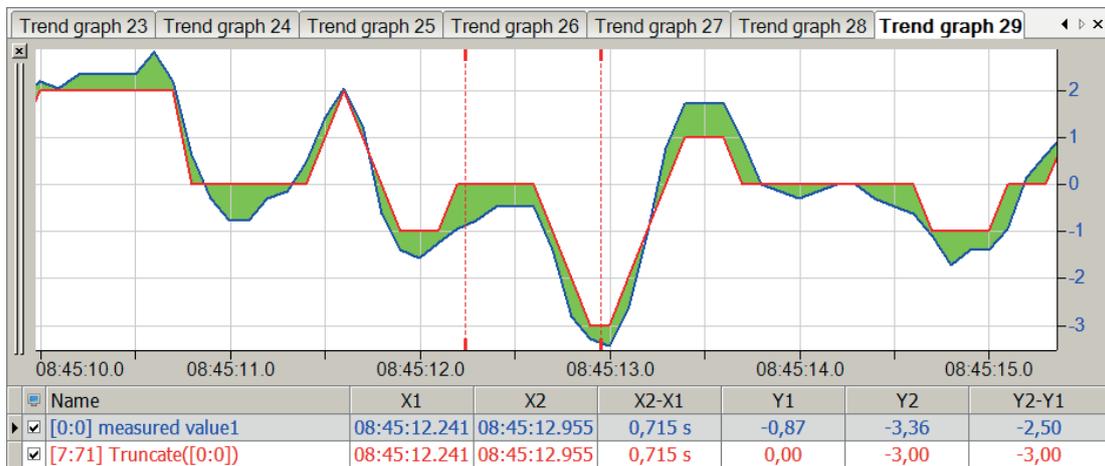
2.4   Trigonometric functions

```
Function('Expression')
```

Description

The standard functions and the corresponding inverse functions are available for the most varied kinds of calculations in which trigonometric functions are needed, for example, the calculation of power in AC-systems.

Function	Description
Sin('Expression')	This function returns the result as the sine of 'Expression' in rad.
Cos('Expression')	This function returns the result as the cosine of 'Expression' in rad.
Tan('Expression')	This function returns the result as the tangent of 'Expression' in rad.
Asin('Expression')	This function returns the result as the arcsine of 'Expression' in rad.
Acos('Expression')	This function returns the result as the arccosine of 'Expression' in rad.
Atan('Expression')	This function returns the result as the arctangent of 'Expression' in rad.

Tip

In order to generate signales, the GenerateSignal function can be used.

Example

Presentation of trigonometric functions

Solution
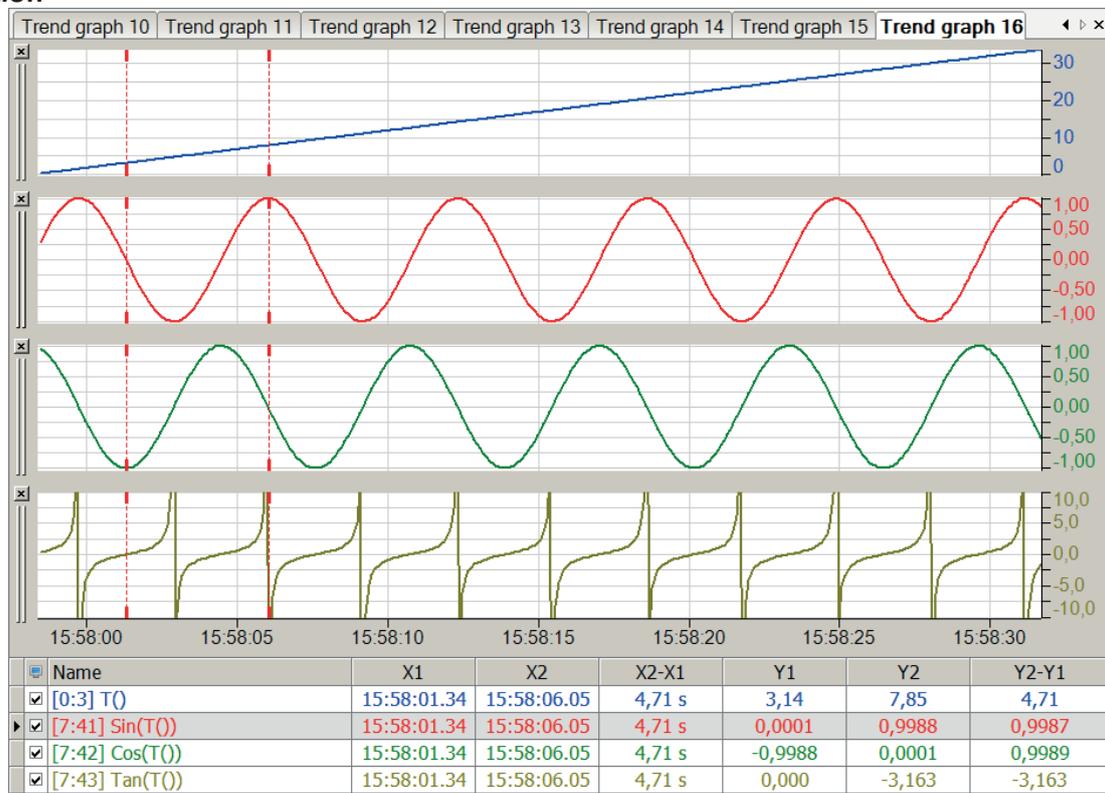
Blue	Time function T() as the basis for trigonometric functions	Red	Sine function of time
Green	Cosine function of time	Yellow	tangent function of time

`Pi()`

Description

The number Pi () is stored as a constant (p = 3.1415927…) in the system for various kinds of calculations. Use this function to insert the number pi into your calculation.

2.5   Statistical functions

2.5.1   Avg

`Avg('Expression', 'Reset=0')`

Arguments

'Expression'	Measured value for the average is formed	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation:	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to the instantaneous "Expression" value

Description

As a result, this function returns the average value of "Expression" since the measuring started or the last reset. The signal is written to the RAM of the computer. The arithmetic mean is continuously calculated. If 'Reset' = 1 (TRUE), then the result equals the actual value of "Expression."

Note

The result of the Avg-function is only displayed in the subsequent interval.

2.5.2   Avg2

```
Avg2('Expression1', 'Expression2', …)
```

Arguments

'Expression'	Measured value, for which the average is formed

Description

This function returns the average value of all arguments 'Expression1', 'Expression2', 'Expression3' etc. The arithmetic average will be formed from the actual values of all arguments in every acquisition cycle. Up to 1000 arguments are allowed.

Example

Evaluate the avarage value from two signals, a sine- and a sawtooth-shaped signal.

Solution
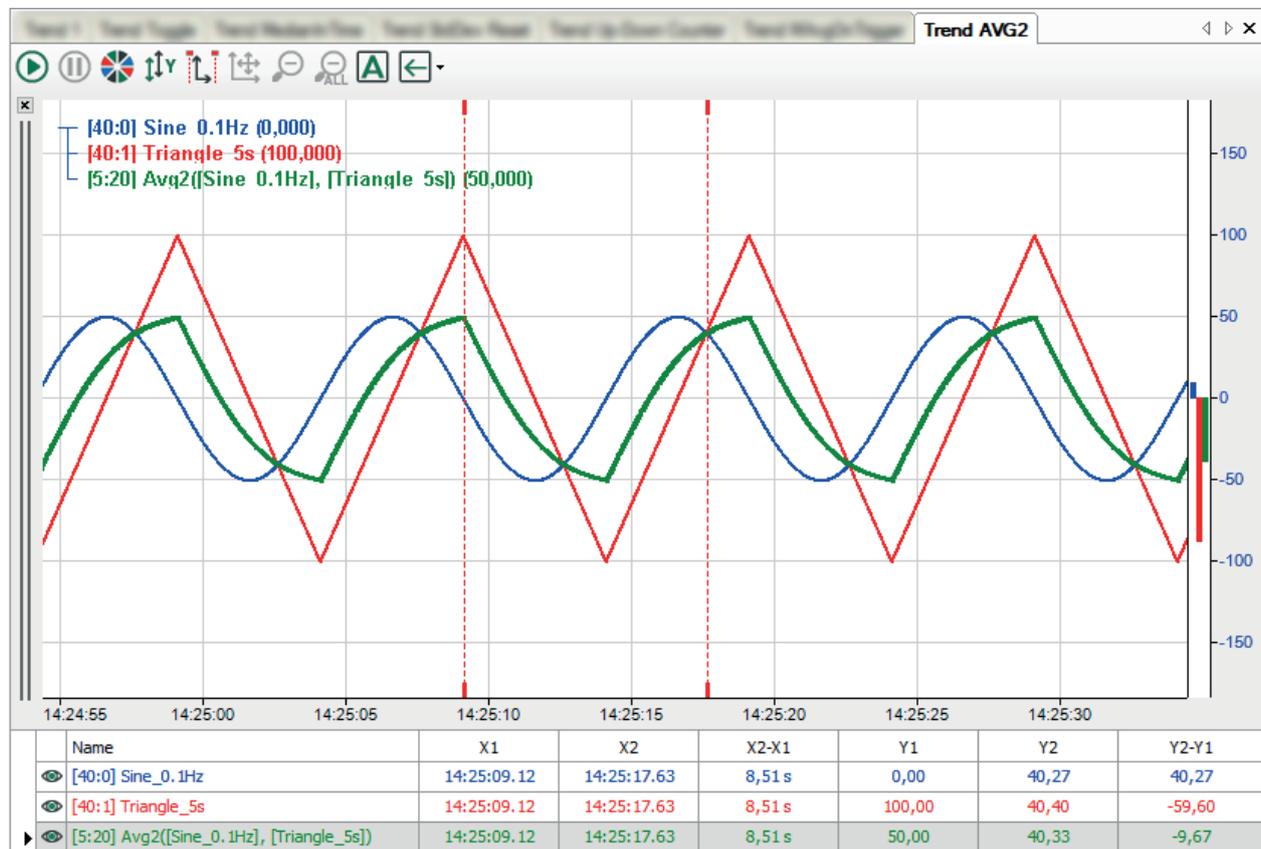
2.5.3   AvgInTime

```
AvgInTime('Expression','Interval',' Reset=0')
```

Arguments

'Expression'	Measurement value for which the average is formed	
'Interval'	Specification of the length of the interval in seconds	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation:	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to 0
	'Reset' = 2	Stop calculation and keep result
	'Reset' = 3	Calculate now and then stop calculation

Description

This function returns the average value per time segment of the 'interval' length of 'Expression' as its result. The signal is written to the RAM of the computer. After an interval has passed, the arithmetic average value over this interval is evaluated.

Tip

The result of the AvgInTime function is displayed only in the subsequent interval.

Example

Effects of the 'Reset' parameter

Task description

The 'Reset' parameter should switch from 0 to 1, 2, or 3 in 5-second intervals in order to make the effects of the parameter visible.

Solution

Three if-queries relay, via a modulo function of the time, the value 0 and the values 1, 2, 3 in intervals of five seconds on the 'Reset' parameter of the AvgInTime function.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [0:0] measured value1	13:08:23.6...	13:08:24.6...	1,000 s	2,19	3,62	1,42
☑ [7:13] AvgInTime ([0:0], 1)	13:08:23.6...	13:08:24.6...	1,000 s	1,54	2,62	1,08
☑ [7:14] AvgInTime ([0:0], 1, [7:17])	13:08:23.6...	13:08:24.6...	1,000 s	1,54	2,62	1,08
☑ [7:15] AvgInTime ([0:0], 1, [7:18])	13:08:23.6...	13:08:24.6...	1,000 s	1,54	1,54	0,00
▶ ☑ [7:16] AvgInTime ([0:0], 1, [7:19])	13:08:23.6...	13:08:24.6...	1,000 s	1,54	0,00	-1,54
☑ [7:17] If (Mod(T(),10)<5, 0, 3)	13:08:23.6...	13:08:24.6...	1,000 s	0,00	3,00	3,00
☑ [7:18] If (Mod(T(),10)<5, 0, 2)	13:08:23.6...	13:08:24.6...	1,000 s	0,00	2,00	2,00
☑ [7:19] If (Mod(T(),10)<5, 0, 1)	13:08:23.6...	13:08:24.6...	1,000 s	0,00	1,00	1,00

Red	Continuous calculation without 'Reset' indication	Green	('Reset' = 3) value is calculated before switching
Yellow	('Reset' = 2) value remains constant during switching	Purple	('Reset' = 1) value = 0 during switching

2.5.4   KurtosisInTime

```
KurtosisInTime('Expression','Interval',' Reset=0')
```

Arguments

'Expression'	Measured value, for which the kurtosis is formed	
'Interval'	Specification of the length of the interval in seconds, over which the kurtosis should be calculated.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation and set result to 0
	'Reset'=2	Stop calculation and keep result
	'Reset'=3	Calculate now and then stop calculation

Description

The calculation of the kurtosis is used, e.g. for the evaluation and analysis of oscillations. It serves to determine the number of outliers within an oscillation signal.

In mathematical terms, the kurtosis is a measure for the relative "flatness" of a distribution (compared to the normal distribution which has a kurtosis of zero). A positive kurtosis indicates

a tapering distribution (a leptokurtic distribution), whereas a negative kurtosis indicates a flat distribution (platykurtic distribution).

This statistical method is particularly suitable for analyzing random or stochastic signals, e.g. in terms of condition-based maintenance (Condition Monitoring) when analyzing oscillations. For characterizing the signal curve, methods of probability density or frequency are used. It is assumed that a noise signal with a Gaussian amplitude distribution can be measured in machines in good order after filtering out, e.g., rotational frequency oscillation components. In the event of damage, individual pulse signals interfere with this signal, altering the distribution function. An evaluation of the system's condition can be carried out through the formation of suitable statistical values, such as the crest factor or the kurtosis factor.

If regularly measured, these methods offer an overview of the machine status. However, the disadvantage is that after they have increased, the characteristic values decrease again. The reason for this is that the number of pulse signals increases with progressive damage. Whereas in turn this has influence on the effective value but barely no effect on the peak value.

Modifications of the time signal caused by shock pulses induce a change in the resulting distribution function. Distinctively discrete damage can cause an increase of the kurtosis factor. Its absolute value thus allows statements on a damage.

2.5.5   MAvg

```
MAvg('Expression','WindowInterval',' UpdateInterval=timebase',' Reset=0')
```

Arguments

'Expression'	Measured value, for which the average value is formed
'WindowInterval'	Specification in seconds of the length of the interval over which the average is formed; must be a multiple of 'UpdateInterval'.
'UpdateInterval'	Optional parameter (default = time base); specifies in which cycle the calculation is performed.
'Reset'	Optional parameter (default = 0) to stop and restart the calculation.
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation and set result to 0
	'Reset'=2	Stop calculation and keep result

Description

This function returns its result as the floating arithmetic mean value of 'Expression' calculated over a ' WindowInterval' in seconds. The calculation is performed in the *'UpdateInterval'* cycle. *'UpdateInterval'* is an optional parameter and is expressed in seconds. If *'UpdateInterval'* is not specified then it is set equal to the time base (default), i.e. as small as possible. The calculation is then carried out progressively by one sample in each case. The calculation can be carried out in larger intervals if a (multiple) value of the time base is entered for 'UpdateInterval'.

'WindowInterval' determines the period for which the mean is calculated each time. 'WindowInterval' must be a multiple of *'UpdateInterval'*. Otherwise, 'WindowInterval' is automatically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowInterval.'

Tip

Using these functions, signals and expressions that are not time-based, i.e. which have the basis length, frequency or 1/length, can also be processed. Instead of seconds, the X-axis range should then be entered in m, Hz or 1/m corresponding to the base.

Example

The mean values of the previous 2 and 20 seconds are relevant for a measured value.

Task description

The MAvg function is used once with an interval time of 2 seconds and once with an interval time of 20 seconds.

Solution

In the figure below the blue curve shows the measured value, the red curve shows the average over an interval of 2 seconds, and the green curve shows the average over an interval of 20 seconds
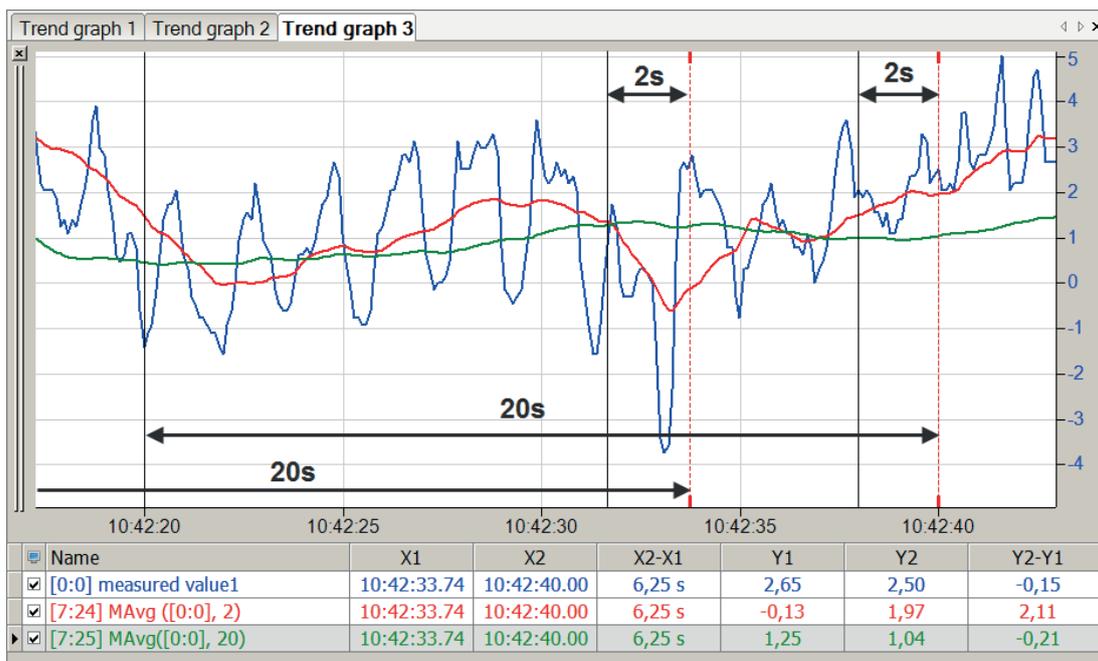
2.5.6   MAvgOnTrigger

`MAvgOnTrigger('Expression', 'Trigger', 'NumberOfValues*', 'Reset=0')`

Parameters ending with * are only evaluated once at the start of the acquisition..

Arguments

'Expression'	Measured value, for which the average is formed	
'Trigger'	Digital signal or expression as a trigger for the execution of the command	
'Number-OfVlaues*'	Number of samples which are used for the calculation of the average; Parameter will be evaluated at start of acquisition. Therefore, rather use a fix value instead of an expression.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation.	
	'Reset' =0	Perform calculation
	'Reset' =1	Stop calculation, delete all buffered data and set result to 0
	'Reset' =2	Stop calculation and keep result

Description

This function returns the moving arithmetic average of 'Expression', which is evaluated on every rising edge of 'Trigger'. Only as many values as specified by 'NumberOfValues' are taken into account for the calculation. After start of acquisition or after a reset with 'Reset' = 1, 'Trigger' has to be fired 'NumberOfValues' times before the first average value can be calculated. With each other trigger the average for the last 'NumberOfValues' triggered samples is calculated.

This makes it possible to calculate moving averages controlled by events. The calculation then can be performed e. g. per revolution of a machine or per produced part instead only linear over time.

Example

Apply MAvgOnTrigger on a triangular signal

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
▶ 👁 [40:1] Triangle_5s	17:06:05.72	17:06:31.16	25,44 s	-72,00	89,64	161,63
👁 [5:19] MAvgOnTrigger_A	17:06:05.72	17:06:31.16	25,44 s	1,60	0,00	-1,60
👁 [9.7] Trigger	17:06:05.72	17:06:31.16	25,44 s	1	0	-1
👁 [9:2] Resetwert	17:06:05.72	17:06:31.16	25,44 s	0,00	1,00	1,00

For explanatory reasons only 'NumberOfValues' = 5 has been specified in the example.

Not until the 5th trigger after start of acquisition a first average value (red 1.) is calculated from the 5 selected samples (blue dots). The distance between the trigger events does not matter. The small and even distance between the first 5 triggers is due to graphical reasons.

With each other trigger the new actual and the last 4 samples are taken into account for the calculation, returning the respective average values (green 2., 3., 4., 5. etc.)

2.5.7   Max

```
Max('Expression',' Reset=0')
```

Arguments

'Expression'	Signal, for which the maximum value should be determined	
'Reset'	Optional digital parameter that can be used to reset the maximum, e.g., in order to ignore the leveling-off processes of the measuring signal during the start-up phase. 'Reset' can be an expression as well.	
	'Reset' = TRUE	Stop calculation and set result to the instantaneous "Expression" value
	'Reset' = 0	Perform calculation; most recently detected maximum is displayed. (Default)

Description

This function returns the maximum value of 'Expression' as its result. It is displayed as a constant value (horizontal line) in the signal strip. Each value is compared to the previous one. If the new value is higher than the previous one, the higher value will be incorporated into the curve. If the new value is equal or lower than the previous one, the previous value will be included. With the digital 'Reset' signal, the maximum value calculation can be stopped and the result can be reset to the current value of the input signal. Without the reset signal there is no way to reset the display unless the measurement is stopped and restarted. 'Reset' can also be formulated as an expression.

Examples:

Max([0:0])	No reset takes place.
Max([0:0],If(Mod(T(),20)=0,TRUE(),FALSE()))	The maximum value is reset every 20 seconds.
Max([0:0],[3.1])	e.g. with [3.1] = If([0:0]<10, 1, 0) The maximum value is reset as soon as the expression [3.1] returns TRUE, i.e., if the expression [0:0] falls below the limit value of 10.

Example

Tip

> This function can be used in a virtual retentive module. Its result values can thus be obtained despite stopping and restarting the measurement.

The maximum should be determined for a signal. The start phase should be ignored in the calculation.

Task description

In order to remove fluctuations during the start phase, the maximum value is reset using the 'reset' function. This can be done using the TriggerConstant function, which waits for the leveling off of the signal. The setting of the 'reset' parameter is done via the edge detection ('One-Shot') of the trigger.

Note

> The 'Reset' parameter should not be permanently set to TRUE because the maximum value would then be reset permanently and thus correspond to the signal.

Solution
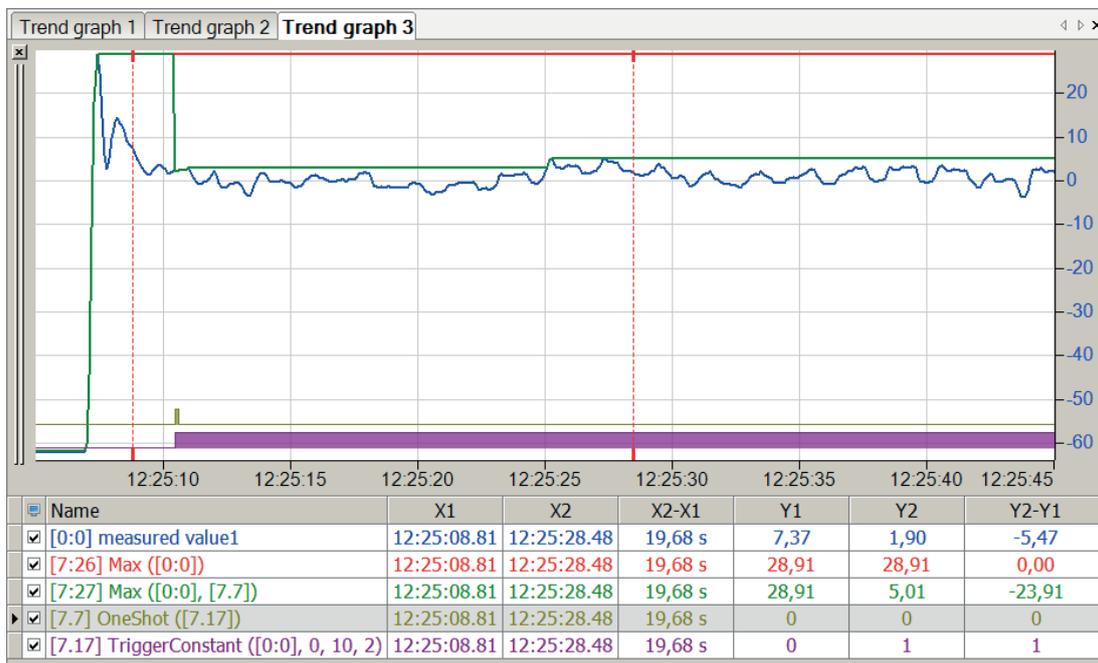
Blue	Measured value	Red	Falsified maximum value without 'Reset'
Green	Maximum value with reset after start phase	Yellow	One-shot function for triggering the 'Reset'
Purple	Trigger after the start phase		

2.5.8   Max2

`Max2('Expression1','Expression2', …)`

Description

This function returns the maximum of all expressions, 'Expression1', 'Expression2', etc. as its result. The expressions or signals are compared measured value by measured value, with the largest value in each case being presented as the result. Up to 1000 arguments are permitted.

Example

Only the larger value of the two measured values is relevant.

Solution

In the figure below the blue curve shows the measured value 1, the red curve shows the measured value 2, and the green cure shows the course of the maximum values.
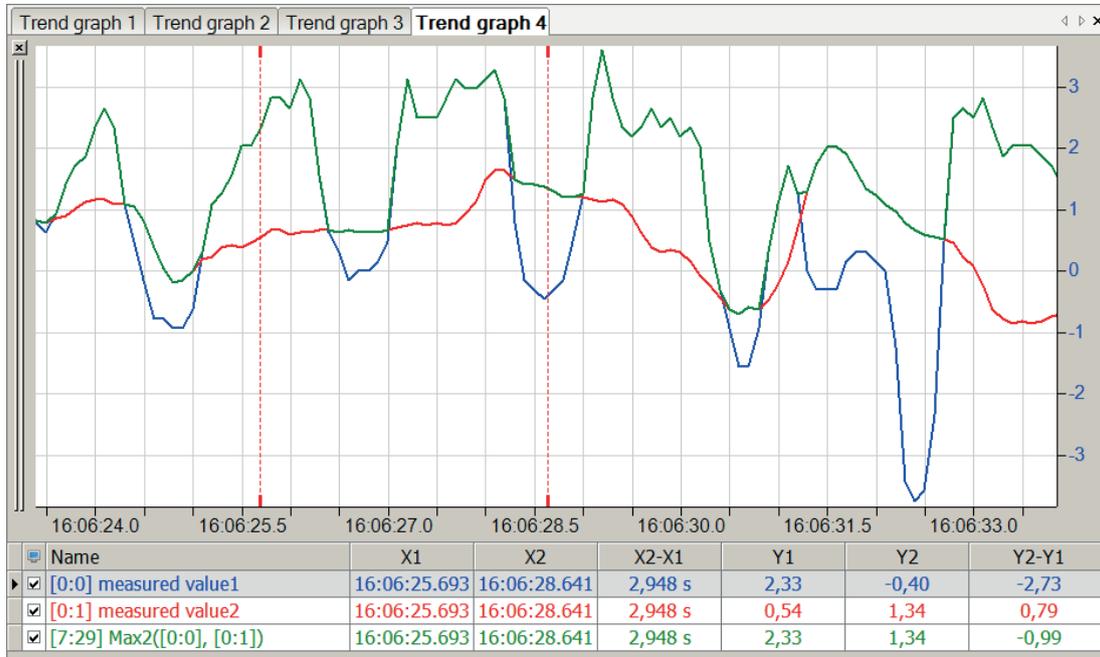
2.5.9   MaxInTime

```
MaxInTime('Expression','Interval',' Reset=0')
```

Arguments

'Expression'	Measured value, for which the maximum is formed	
'Interval'	Specification of the interval length in seconds, over which the maximum should be calculated.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset' =0	Perform calculation
	'Reset'=1	Stop calculation and set result to 0
	'Reset'=2	Stop calculation and keep result
	'Reset'=3	Calculate now and then stop calculation

Description

This function returns the maximum value of 'Expression' within each interval of the 'interval' length (in seconds) as its result.

Note

	The result of the MaxInTime function is displayed only in the subsequent interval.

Example

Effects of the 'Reset' parameter

Task description

The 'Reset' parameter should switch from 0 to 1, 2, or 3 in 5-second intervals in order to make the effects of the parameter visible.

Solution

Three If-queries relay, via a modulo function of the time, the value 0 and the values 1, 2, 3 in intervals of 5 seconds on the 'Reset' parameter of the MaxInTime function.

Blue	Measured value	Yellow	Continuous calculation without 'Reset' application
Red	('Reset' = 3) value is calculated before interruption, even within an interval (1)	Green	('Reset'= 2) value remains constant during interruption
Purple	('Reset' = 1) value=0 during interruption, even within an interval (1)		

2.5.10   Median2

```
Median2('Expression1', 'Expression2', …)
```

Arguments

'Expression'	Measured value(s), for which the median value is formed

Description

This function returns the median value of all arguments Every cycle, the actual values of all arguments are added and the sum is divided by two.

Example

The median of a constant and a periodic value should be formed.

Solution
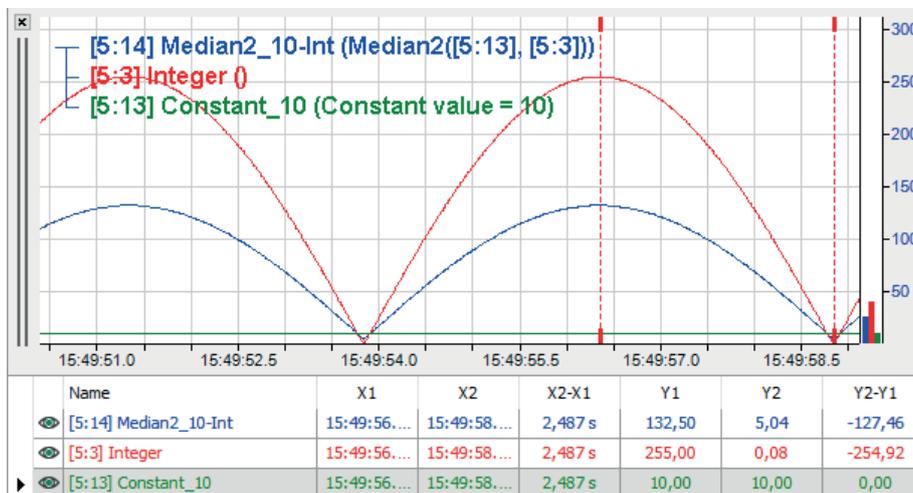
The example shows a median of 132.5 ((10 + 255)/2) at the peak of the curve.

2.5.11   MedianInTime

```
MedianInTime('Expression', 'Interval', ' Reset=0')
```

Arguments

'Expression'	Measured value, for which the median is formed	
'Interval'	Specification of the length of the interval in seconds	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation:	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to 0
	'Reset' = 2	Stop calculation, delete all buffered data and keep result
	'Reset' = 3	Calculate now and then stop calculation

Description

This function returns the median value per time segment of the 'interval' length of 'Expression' as its result. The signal is written to the RAM of the computer. After an interval has passed, the median value over this interval is evaluated.

Tip
The result of the MedianInTime function is displayed only in the subsequent interval.

Example

The following figure shows the calculation of the MedianInTime (blue) of a signal (red) and the impact of the parameter 'Reset' (green).

2.5.12   Min

```
Min('Expression',' Reset=0')
```

Arguments

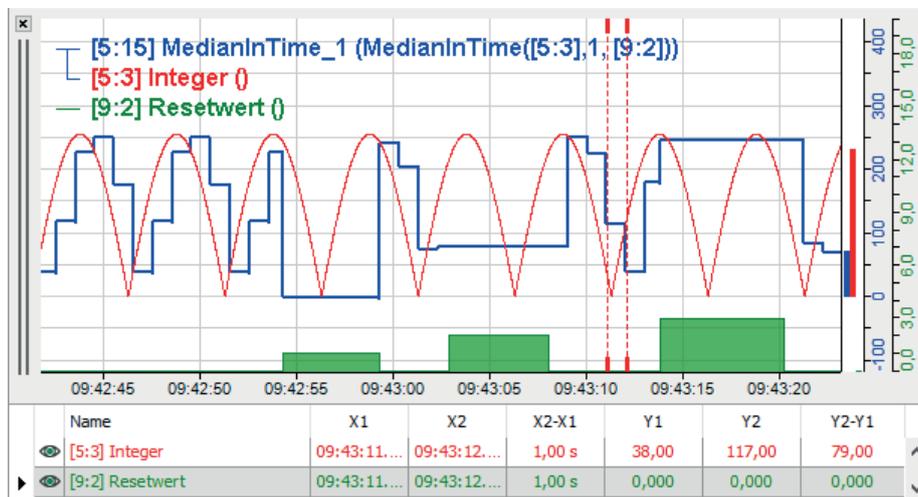| 'Expression'	Signal, for which the minimum value should be determined	
'Reset'	Optional digital parameter that can be used to reset the minimum, e.g., in order to ignore the leveling-off processes of the measuring signal during the start-up phase. 'Reset' can be an expression as well.	
	'Reset' = TRUE	Stop calculation and set result to the instantaneous "Expression" value
	'Reset' = 0	Perform calculation; most recently detected minimum is displayed. (Default)

Description

This function returns the minimum value of the 'Expression' signal as its result. It is displayed as a constant value (horizontal line) in the signal strip. Each value is compared to the previous one. If the new value is lower than the previous one, the lower value will be incorporated into the curve. If the new value is equal or higher than the previous one, the previous value will be included. With the digital 'Reset' signal, the minimum value calculation can be stopped and the result can be reset to the current value of the input signal. Without the Reset signal there is no way to reset the display unless the measurement is stopped and restarted. . 'Reset' can also be formulated as an expression.

Examples:

Min([0:0])	No reset takes place.
Min([0:0],If(Mod(T(),20)=0,TRUE(),FALSE()))	The minimum value is reset every 20 seconds.
Min([0:0],[3.1])	e.g. with [3.1] = If([0:0]<10, 1, 0)
	The minimum value is reset as soon as the expression [3.1] returns TRUE, i.e. if the expression [0:0] falls below the limit value 10.

Example

Tip

This function can be used in a virtual retentive module. Its result values can thus be obtained via stopping and restarting the measurement.

The minimum should be determined for a signal. The start phase should be ignored in the calculation.

Task description

In order to remove fluctuations during the start phase, the minimum value is reset using the 'reset' function. This can be done via the TriggerConstant function, which waits for the leveling off of the signal. The setting of the 'reset' parameter is done via the edge detection ('OneShot') of the trigger.

Note

> The 'reset' parameter should not be permanently set to TRUE because the minimum value would then be reset permanently and thus correspond to the signal.

Solution
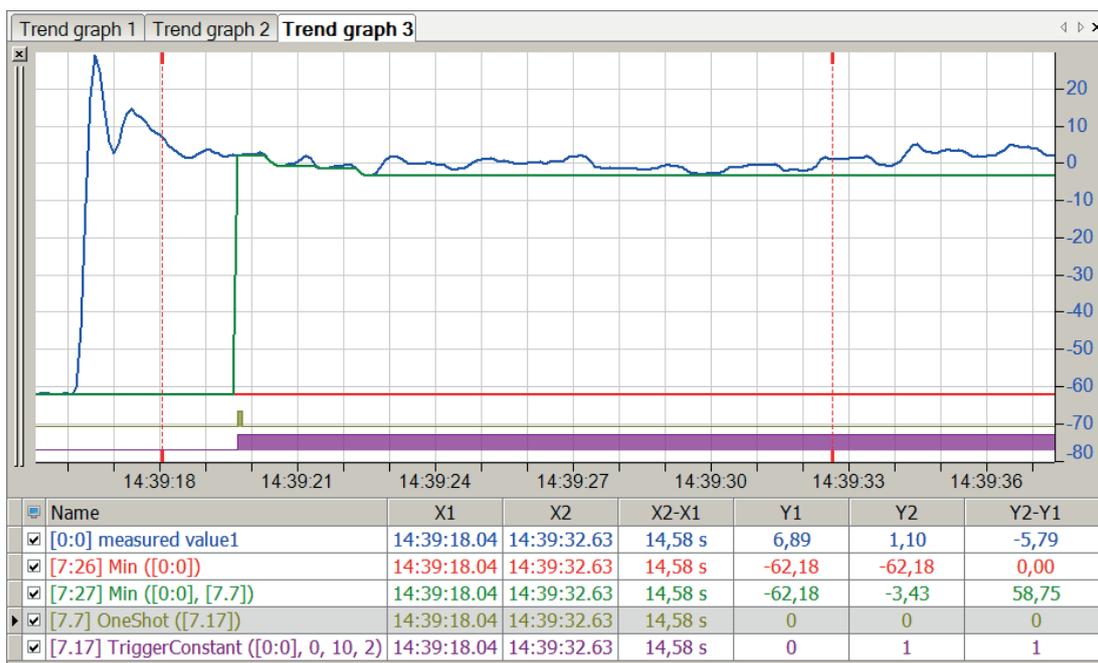
Blue	Measured value	Red	falsified minimum value without 'Reset'
Green	Minimum value with reset after start phase	Yellow	One-shot function for triggering the 'Reset'
Purple	Trigger after the start phase		

2.5.13   Min2

```
Min2('Expression1','Expression2')
```

Description

This function returns the minimum of all expressions, 'Expression1', 'Expression2', etc. as its result. The expressions or signals are compared measured value by measured value, with the largest value in each case being presented as the result. Up to 1000 arguments are permitted.

Example

Only the smaller value of the two measured values is relevant.

Solution

In the figure below the blue curve shows the measured value 1, the red curve shows the measured value 2, and the green cure shows the course of the minimum values.
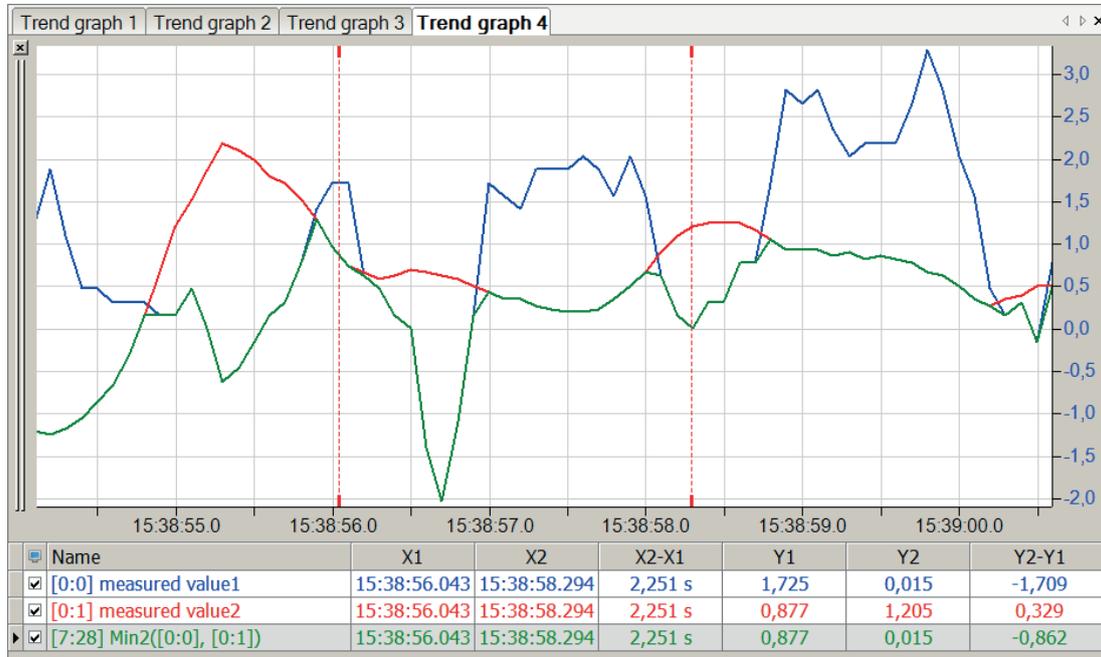
2.5.14   MinInTime

```
MinInTime('Expression','Interval',' Reset=0')
```

Arguments

'Expression'	Measured value, for which the minimum is formed	
'Interval'	Specification of the length of the interval in seconds, over which the minimum should be calculated.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation and set result to 0
	'Reset'=2	Stop calculation and keep result
	'Reset'=3	Calculate now and then stop calculation

Description

This function returns the minimum value of 'Expression' within each interval of the 'Interval' length (in seconds) as its result.

Note

The result of the MinInTime function is displayed only in the subsequent interval.

Example

Effects of the 'Reset' parameter

Task description

The 'Reset' parameter should switch from 0 to 1, 2, or 3 in 5-second intervals in order to make the effects of the parameter visible.

Solution

Three If-queries relay, via a modulo function of the time, the value 0 and the values 1, 2, 3 in intervals of 5 seconds on the 'Reset' parameter of the MinInTime function.

	Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑	[0:0] measured value1	16:45:07.500	16:45:09.500	2,000 s	4,695	3,602	-1,093
▶ ☑	[7:30] MinInTime([0:0],2)	16:45:07.500	16:45:09.500	2,000 s	-2,018	-0,770	1,248
☑	[7:31] MinInTime([0:0],2, [7:17])	16:45:07.500	16:45:09.500	2,000 s	-2,018	2,662	4,680
☑	[7:32] MinInTime([0:0],2, [7:18])	16:45:07.500	16:45:09.500	2,000 s	-2,018	-0,775	1,243
☑	[7:33] MinInTime([0:0],2, [7:19])	16:45:07.500	16:45:09.500	2,000 s	-2,018	0,000	2,018
☑	[7:17] If (Mod(T(),10)<5, 0, 3)	16:45:07.500	16:45:09.500	2,000 s	0,00	3,00	3,00
☑	[7:18] If (Mod(T(),10)<5, 0, 2)	16:45:07.500	16:45:09.500	2,000 s	0,00	2,00	2,00
☑	[7:19] If (Mod(T(),10)<5, 0, 1)	16:45:07.500	16:45:09.500	2,000 s	0,00	1,00	1,00

Blue	Measured value	Yellow	Continuous calculation without 'Reset' application
Red	('Reset' = 3) value is calculated before interruption, even within an interval (1)	Green	('Reset'= 2) value remains constant during interruption
Purple	('Reset' = 1) value=0 during interruption, even within an interval (1)		

2.5.15   MKurtosis

`MKurtosis('Expression','WindowInterval', 'UpdateInterval=timebase', 'Reset=0')`

Arguments

'Expression'	Measured value, for which the kurtosis is formed	
'WindowInter-val'	Specification in seconds of the length of the interval over which the kurtosis is formed; must be a multiple of 'UpdateInterval'.	
'UpdateInter-val'	Optional parameter (default = time base); specifies in which cycle the calculation is performed.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation.	
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation and set result to 0
	'Reset'=2	Stop calculation and keep result

Description

Returns the kurtosis value of 'Expression' every *'UpdateInterval'* seconds based on a floating window of 'WindowInterval' seconds.

The *'UpdateInterval'* parameter is optional. If not specified then it is set equal to the time base of the function (i.e. as small as possible).

'WindowInterval' must be a multiple of *'UpdateInterval'*. Otherwise, 'WindowInterval' is automatically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowInterval.'

For more information about Kurtosis, see Chapter ↗ *KurtosisInTime, page 46*

2.5.16   MMax

```
MMax('Expression','WindowInterval', 'UpdateInterval=timebase', 'Reset=0')
```

Arguments

'Expression'	Measured value, for which the maximum is formed	
'WindowInterval'	Specification in seconds of the interval length over which the maximum should be calculated; must be a multiple of 'UpdateInterval'.	
'UpdateInterval'	Optional parameter (default = time base); specifies in which cycle the calculation is performed.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation, reset and set result to 0
	'Reset'=2	Stop calculation and keep result

Description

This function returns the maximum of 'Expression' every 'UpdateInterval' seconds based on a floating window of 'WindowInterval' seconds.

The *'UpdateInterval'* parameter is optional. If not specified then it is set equal to the time base of the function (i.e. as small as possible).

'WindowInterval' must be a multiple of *'UpdateInterval'*. Otherwise, 'WindowInterval' is automatically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowInterval.'

2.5.17   MMaxDynamic

```
MMaxDynamic('Expression','WindowInterval', 'MaxWindowInterval*')
```

Arguments

'Expression'	Measured value, for which the maximum is formed
'WindowInterval'	Specification in seconds of the interval length over which the maximum should be calculated
'MaxWindowInter-val*'	Limit for 'WindowInterval'; the size of 'WindowInterval' will be limited to this value, given in seconds.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the maximum of 'Expression' over a floating window of 'WindowInterval' seconds. 'WindowInterval' can be a signal or expression and may change with every sample. The limit value of 'WindowInterval' is 'MaxWindowInterval' seconds.

2.5.18   MMedian

```
MMedian('Expression', 'WindowInterval', 'UpdateInterval=timebase', ' Reset=0')
```

Arguments

'Expression'	Measured value, for which the median is formed	
'WindowInterval'	Specification in seconds of the interval length over which the median should be calculated; must be a multiple of 'UpdateInterval'.	
'UpdateInterval'	Optional parameter (default = time base); specifies in which cycle the calculation is performed.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation:	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation, delete all buffered data and set result to 0
	'Reset' = 2	Stop calculation, delete all buffered data and keep result

Description

This function returns the median of 'Expression' every 'UpdateInterval' seconds based on a floating window of 'WindowInterval' seconds.

The 'UpdateInterval' parameter is optional. If not specified then it is set equal to the time base of the function (i.e. as small as possible).

'WindowInterval' must be a multiple of *'UpdateInterval'*. Otherwise, 'WindowInterval' is automatically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowInterval.'

'WindowInterval' and 'UpdateInterval' can be determined by expressions. Changes of these interval values are applied not until a reset.

Tip

The result of the MMedian function is displayed only in the subsequent interval.

2.5.19   MMin

```
MMin('Expression','WindowInterval', 'UpdateInterval=timebase', 'Reset=0')
```

Arguments

'Expression'	Measured value, for which the minimum is formed	
'WindowInterval'	Specification in seconds of the interval length over which the minimum should be calculated; must be a multiple of 'UpdateInterval'.	
'UpdateInterval'	Optional parameter (default = time base); specifies in which cycle the calculation is performed.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation, reset and set result to 0
	'Reset'=2	Stop calculation and keep result

Description

This function returns the minimum of 'Expression' every *'UpdateInterval'* seconds based on a floating window of 'WindowInterval' seconds.

The *'UpdateInterval'* parameter is optional. If not specified then it is set equal to the time base of the function (i.e. as small as possible).

'WindowInterval' must be a multiple of *'UpdateInterval'*. Otherwise, 'WindowInterval' is automatically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowInterval.'

2.5.20   MMinDynamic

`MMinDynamic('Expression','WindowInterval', 'MaxWindowInterval*')`

Arguments

'Expression'	Measured value, for which the minimum is formed
'WindowInterval'	Specification in seconds of the interval length over which the minimum should be calculated
'MaxWindowInterval'	Limit for 'WindowInterval'; the size of 'WindowInterval' will be limited to this value, given in seconds.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the minimum of 'Expression' over a floating window of 'WindowInterval' seconds. 'WindowInterval' can be a signal or expression and may change with every sample. The limit value of 'WindowInterval' is 'MaxWindowInterval' seconds.

2.5.21 MSkewness

```
MSkewness('Expression','WindowInterval', 'UpdateInterval=timebase', 'Reset=0')
```

Arguments

'Expression'	Measured value, for which the skewness is formed		
'WindowInter-val'	Specification in seconds of the interval length over which the skewness should be calculated; must be a multiple of 'UpdateInterval'.		
'UpdateInter-val'	Optional parameter (default = time base); specifies in which cycle the calcu-lation is performed.		
'Reset'	Optional parameter (default = 0) to stop and restart the calculation.		
	'Reset'=0	Perform calculation	
	'Reset'=1	Stop calculation and set result to 0	
	'Reset'=2	Stop calculation and keep result	

Description

Returns the skewness value of 'Expression' every *'UpdateInterval'* seconds based on a floating window of 'WindowInterval' seconds.

The *'UpdateInterval'* parameter is optional. If not specified then it is set equal to the time base of the function (i.e. as small as possible).

'WindowInterval' must be a multiple of *'UpdateInterval'*. Otherwise, 'WindowInterval' is auto-matically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowIn-terval.'

For more information about skewness, see Chapter ↗ *SkewnessInTime, page 67*

2.5.22 MStdDev

```
MStdDev('Expression', 'WindowInterval', 'UpdateInterval=timebase', ' Reset=0')
```

Arguments

'Expression'	Measured value, for which the standard deviation is formed		
'WindowInterval'	Specification in seconds of the interval length over which the median should be calculated; must be a multiple of 'UpdateInterval'.		
'UpdateInterval'	Optional parameter (default = time base); specifies in which cycle the cal-culation is performed.		
'Reset'	Optional parameter (default = 0) to stop and restart the calculation:		
	'Reset' = 0	Perform calculation	
	'Reset' = 1	Stop calculation, delete all buffered data and set result to 0	
	'Reset' = 2	Stop calculation, delete all buffered data and keep result	

Description

This function returns the the standard deviation 'Expression' every 'UpdateInterval' seconds based on a floating window of 'WindowInterval' seconds.

The 'UpdateInterval' parameter is optional. If not specified then it is set equal to the time base of the function (i.e. as small as possible).

'WindowInterval' must be a multiple of 'UpdateInterval'. Otherwise, 'WindowInterval' is automatically changed to the first multiple of *'UpdateInterval'* greater than or equal to 'WindowInterval.'

'WindowInterval' and 'UpdateInterval' can be determined by expressions. Changes of these interval values are applied not until a reset.

Tip

The result of the MStdDev function is displayed only in the subsequent interval.

2.5.23   SkewnessInTime

```
SkewnessInTime('Expression','Interval','Reset=0')
```

Arguments

'Expression'	Measured value, for which the skewness is formed	
'Interval'	Specification of the interval length in seconds, over which the skewness should be calculated.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to 0
	'Reset' = 2	Stop calculation and keep result
	'Reset' = 3	Calculate now and then stop calculation

Description

Like the kurtosis factor, the skewness factor can be used for evaluating and analyzing oscillations. The skewness factor can be used if the symmetrical properties of an oscillation signal are to be checked (e.g. acceleration signal).

With this function, the selected expression is divided into equal-duration intervals of size 'Interval'. For these intervals, the skewness is subsequently calculated.

In mathematical terms, this is the evaluation of the skewness of a distribution function. A distribution is called right-skewed (or positively-skewed) when the majority of the distribution is concentrated on the left side. A distribution is called left-skewed (or negatively-skewed) when the majority of the distribution is concentrated on the right. The skewness level is defined by the third order of the central moment of the distribution.

The procedure to calculate the skewness is similar to that of the KurtosisInTime function.

2.5.24   StdDev

StdDev('Expression', 'Reset=0')

Arguments

'Expression'	Signal, for which the standard deviation should be determined	
'Reset'	Optional parameter (default = 0) to stop and restart the	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to 0

Description

This function returns the standard deviation of 'Expression' as its result. If 'Reset' is TRUE the result is reset to 0.

Example
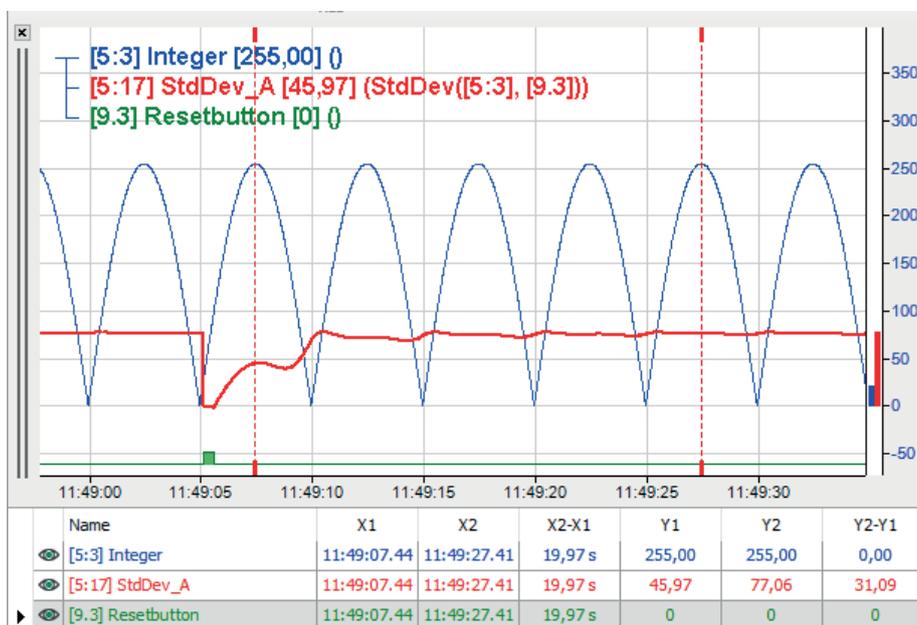
2.5.25   StdDev2

```
StdDev2('Expression1', 'Expression2', …)
```

Arguments

'Expression'	Measured value(s), for which the standard deviation is formed

Description

This function returns the standard deviation of all arguments

Example

Standard deviation of a constant and a variable value
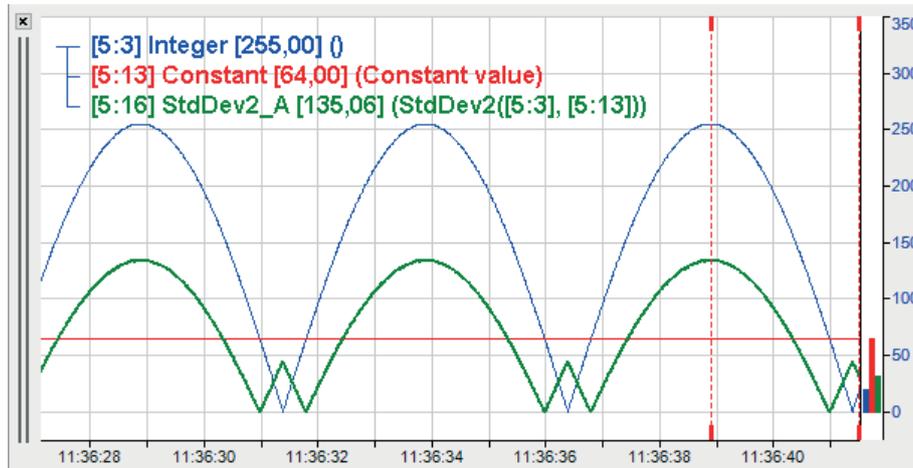
2.5.26   StddevInTime

```
StddevInTime('Expression','Interval',' Reset=0')
```

Arguments

'Expression'	Measured value, for which the standard deviation is formed	
'Interval'	Specification of the length of the interval in seconds, over which the standard deviation should be calculated.	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to 0
	'Reset' = 2	Stop calculation and keep result
	'Reset' = 3	Calculate now and then stop calculation

Description

This function returns the standard deviation of 'Expression' over each time interval of the length 'Interval' as its result. The calculation can be stopped using the optional parameter, 'Reset.'

The standard deviation is calculated by the following formula:

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n - 1}}$$

s_x = standard deviation
$\bar{x}$ = average value
n = number of samples

Example

For a signal waveform, the standard deviation should be determined in time intervals of one second. A reset is not necessary.

Solution

In the figure below the blue curve shows the measured values and the red curve shows the standard deviation in the intervals of length of one second.
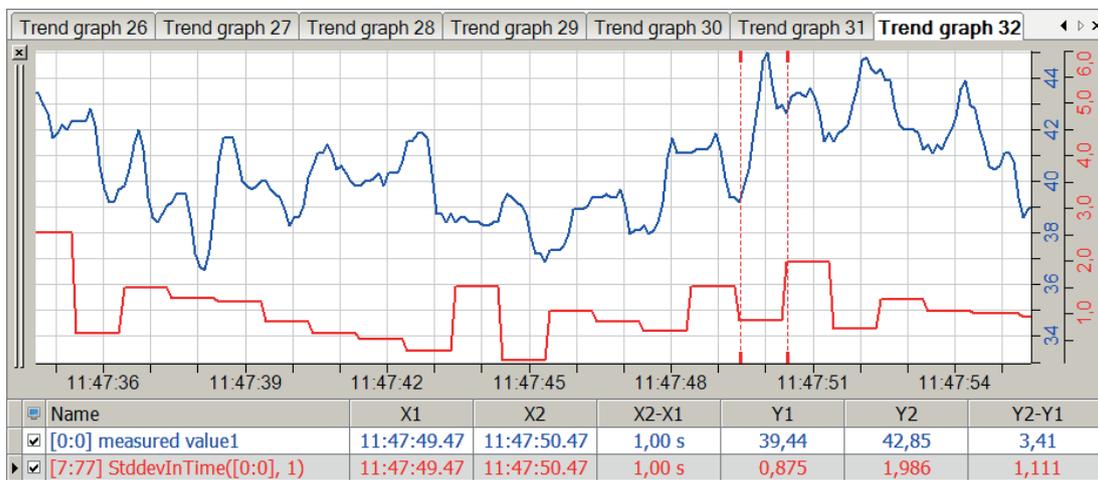

Note

The result of the StdDevInTime function is always displayed in the subsequent interval.

2.6 Trigger functions

2.6.1 Periodic trigger

`PeriodicTrigger('Interval*','StartTime*','UseSystemTime*')`

Arguments

'Interval*'	Interval in minutes, in which the value TRUE is returned	
'StartTime*'	Specification in minutes; produces the start time ('StartTime' modulo 'Interval') after using the modulo function	
'UseSystemTime*'	Determines whether the system time or internal high-resolution timer is used.	
	'UseSystemTime = 0	No system time but internal timer is used.
	'UseSystemTime' > 0	System time is used.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the value TRUE every 'Interval' minutes, starting at 'StartTime' modulo 'Interval' minutes. The 'UseSystemTime' flag determines whether the system time or the internal high-resolution timer is to be used.

Example

Displaying a trigger every 10 seconds

Solution

Display of the following functions:

- PeriodicTrigger(10/60, 10/60, 1)

- PeriodicTrigger(10/60, 11/60, 1)

- PeriodicTrigger(10/60, 15/60, 1)

with 10/60 min (corresponds to 10 s) for 'Interval,' again 10/60 min or 11/60 min (11 s) and 15/60 min (15 s) for 'StartTime' and 1 (system time) for 'UseSystemTime.'

As the following illustration shows, the triggers every 10 s differ in their start time. According to the calculation method, the trigger starts the interval of the red chart at time 0 s, the green chart at 1 s and the blue chart at 5 s after the acquisition.
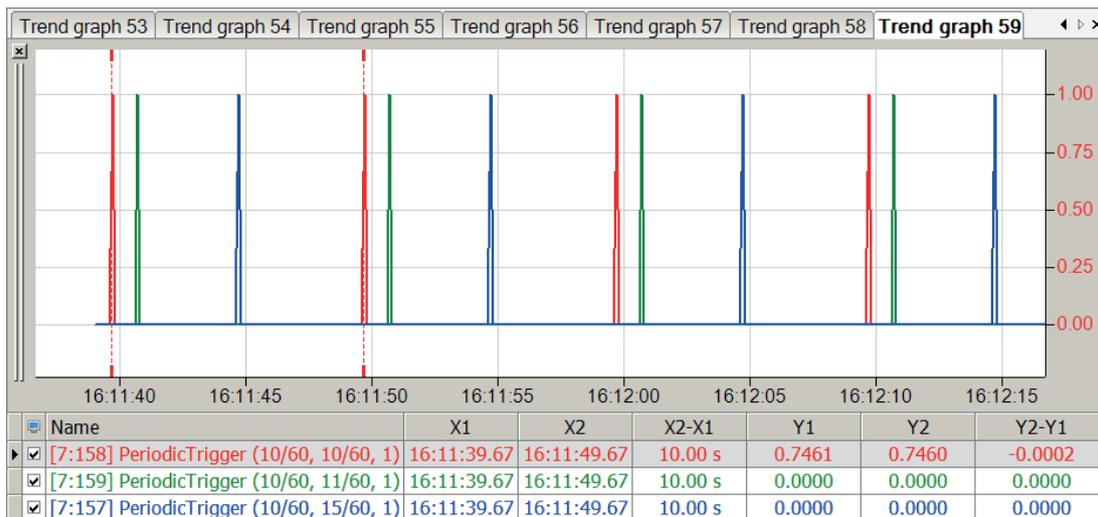
Note

Negative values for 'StartTime' are invalid and result in an error message.

2.6.2   TriggerChangeRate

`TriggerChangeRate('Expression','DeltaY*','DeltaT*','DeadTime*')`

Arguments

'Expression'	Measured value
'DeltaY*'	Required value distance for releasing the trigger
'DeltaT*'	Time interval considered; limit value = 1,000,000 × module timebase
'DeadTime*'	Time specification in seconds, for which the trigger condition must be fulfilled until release. If no delay is desired, the value 0 must be entered.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The function returns TRUE as long as the change of the measured value 'Expression' (dy) within the interval 'DeltaT' is greater than 'DeltaY'.

Note

The function compares the distance of the values with the time difference between DeltaT and DeltaY; the slope of the curve can also vary between the values. The time difference to be considered DeltaT is limited to 1-millionfold the module timebase.

Example

Differences in trigger initiation and impact of the 'DeadTime'

Solution
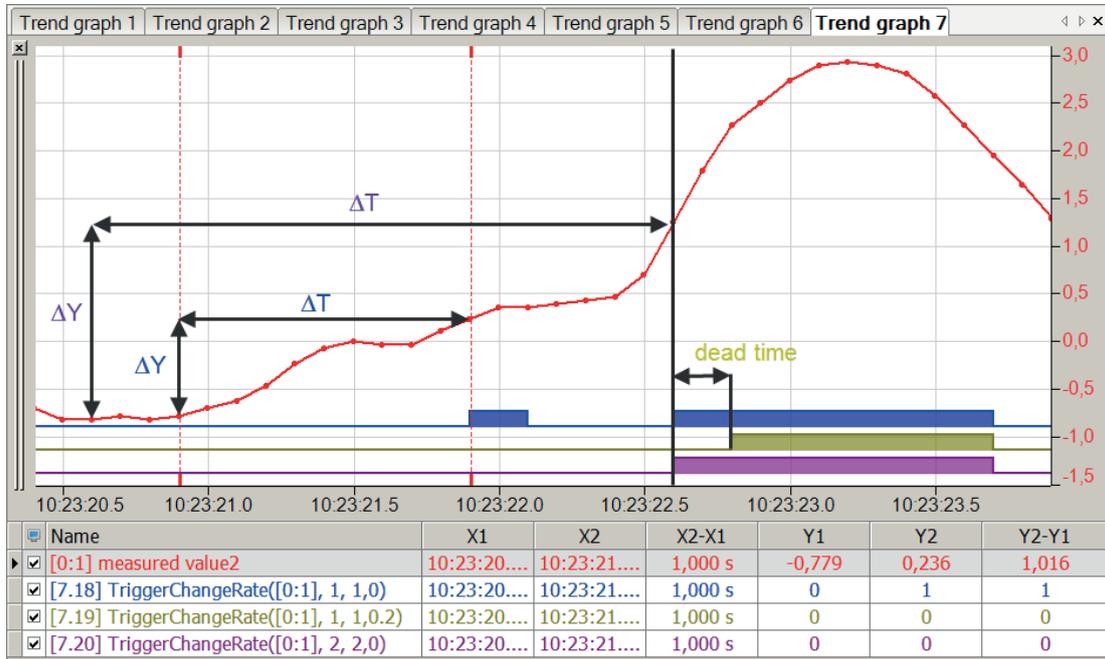
Red	Measured value	Blue	Trigger release with ΔY=1 and ΔT=1
Yellow	Trigger release with ΔY=1 and ΔT=1	Purple	Trigger release with ΔY=2 and ΔT=2

2.6.3   TriggerConstant

```
TriggerConstant('Expression','Level*','Epsilon*','DeadTime*')
```

Arguments

'Expression'	Measured value
'Level*'	Value specifying the center line of the area in which the trigger should release
'Epsilon*'	Value specifying the distance of both area boundaries for the center line
'DeadTime*'	Time specification in seconds, for which the trigger condition must be fulfilled until release. If no delay is desired, the value 0 must be entered.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The function returns TRUE as long as 'Expression' remains within the area ['Level' - 'Epsilon', 'Level' + 'Epsilon'] for at least the duration of the 'DeadTime'.

Note

The trigger signal is emitted (true) for the entire duration for which the level is exceeded or undershot.

Example

The trigger should be released if the measured value is longer than one second in a range between 0.5 and 1.5.

Solution

In the figure below the blue curve shows the original signal and the red bar shows the released trigger.
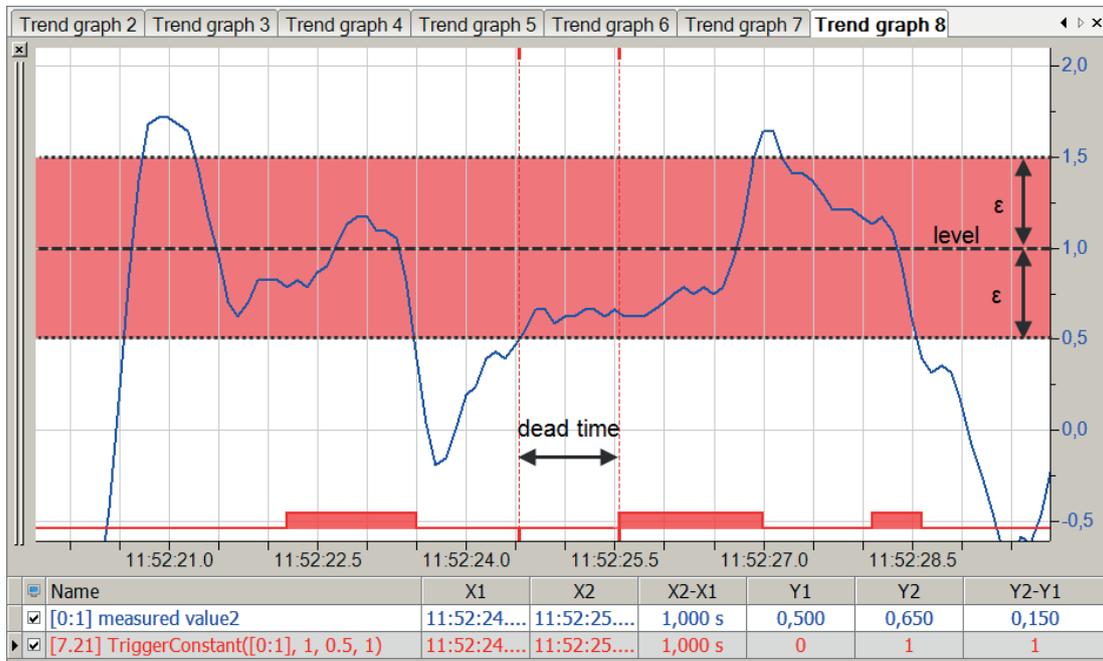
2.6.4   TriggerEdge

```
TriggerEdge('Expression','Level*','EdgeType*','DeadTime*')
```

Arguments

'Expression'	Measured value	
'Level*'	Specification of the level value	
'EdgeType*'	Specification of whether rising, falling or both edges, i.e. crossings of the level value are counted	
	'EdgeType' < 0	only falling edges (crossing in the negative direction)
	'EdgeType' > 0	only rising edges (crossing in the positive direction)
	'EdgeType' = 0	falling and rising edges
'DeadTime*'	Time specified in seconds for which the measured value must exceed or fall below the level value in order to release the trigger.	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

Triggers if 'Expression' exceeds or falls below 'Level' and stays on the same 'Level' side for at least DeadTime' seconds. If 'Expression' is a digital signal, 'Level' is fixed at 0.5. 'EdgeType' determines which edges or crossing are counted:

Note

Unlike the TriggerLevel and TriggerConstant functions, only a pulse is emitted when the level value is crossed.

Example

A trigger should be released in each case where the measured value is above 1.5 or below -0.5 for 0.5 seconds.

Solution

In the figure below the blue curve shows the measured value and the red bar shows the trigger if measured value is longer than 0.5 seconds above 1.5. The green bar shows the trigger if measured value is longer than 0.5 seconds below -0.5.
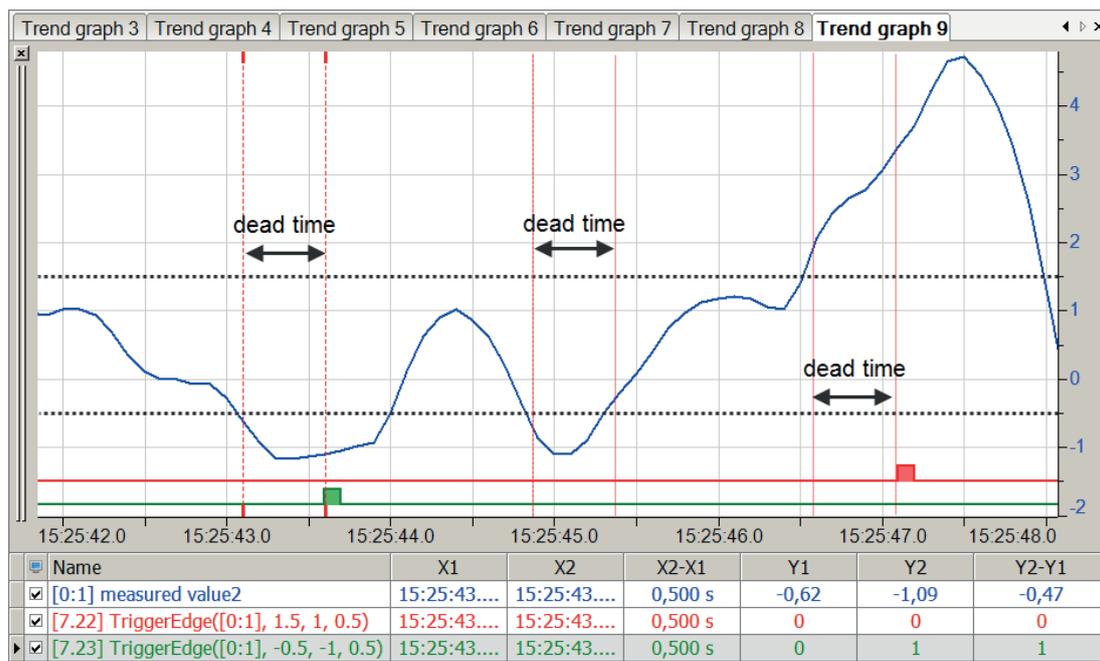
2.6.5   TriggerLevel

```
TriggerLevel('Expression','Level*','LevelType*','DeadTime*')
```

Arguments

'Expression'	Measured value	
'Level*'	Specification of the level value	
'LevelType*'	Specification of which side of 'Level' is considered	
	'LevelType' =0	below level
	'LevelType' = 1	above level
'DeadTime*'	Time specified in seconds for which the measured value must remain on the side considered of the level value in order to initiate the trigger.	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

Triggers when 'Expression' remains above or below 'Level' for at least 'DeadTime' seconds. 'LevelType' determines which 'level' side is monitored.

Note

Like for the TriggerConstant function, the trigger signal remains true until the value crosses 'Level' on the other side.

Example

A trigger should be released in each case where the measured value is above 0.8 and below -0.2 for 0.4 seconds. A trigger pulse should be emitted periodically for the entire duration of the exceeding or undershooting.

Solution

In the figure below the blue curve shows the measured value and the red bar shows the trigger. The trigger is true, beginning at 0.4 seconds after the measured value rose above 0.4 and it returns to false when the value dropped below 0.4. The green bar shows the trigger when the measured value falls below -0.2. The trigger is true 0.4 seconds after the value dropped below -0.2, and it is false agawn hen the value is >-0.2.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:1] measured value2	14:50:42.500	14:50:42.770	0,270 s	0,400	0,705	0,304
[7.24] TriggerLevel([0:1], 0.4, 1, 0.2)	14:50:42.500	14:50:42.770	0,270 s	0	0	0
[7.29] TriggerLevel([0:1], - 0.1, 0, 0.2)	14:50:42.500	14:50:42.770	0,270 s	0	0	0
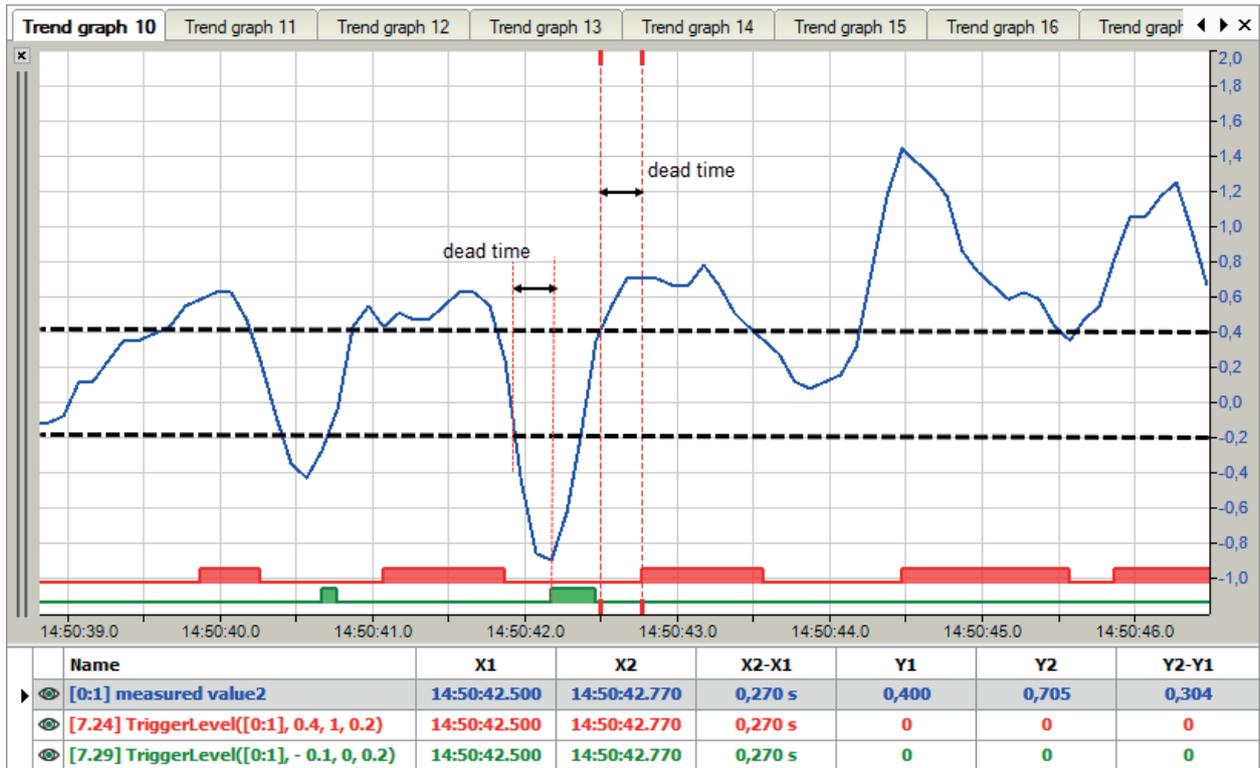
2.6.6 TriggerHarmonicLevel

```
TriggerHarmonicLevel('Expression',"'LimitProfile*'",'Harmonic*')
```

Arguments

'Expression'	Measuring signal
"'LimitProfile*'"	Name of the limit profile defined in the PQU
'Harmonic*'	Order (harmonic), the limit value of which is to be monitored, values 0 to 50

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

Trigger fires (result true) when the signal value is above the harmonic limit defined in the "LimitProfile" profile of PQU. The "Harmonic" parameter determines which limit value is used from the limit profile.

Example

When measured on medium voltage switchgear, the U12 conductor voltage should be monitored for its harmonic content. A trigger signal should be triggered if the limit value specified in EN50160 for the third harmonic in a 10 minute interval is exceeded.

The limit profile is configured in the I/O Manager in the spectrum modules. The profile name is "EN50160 – Medium voltage – Harmonics".
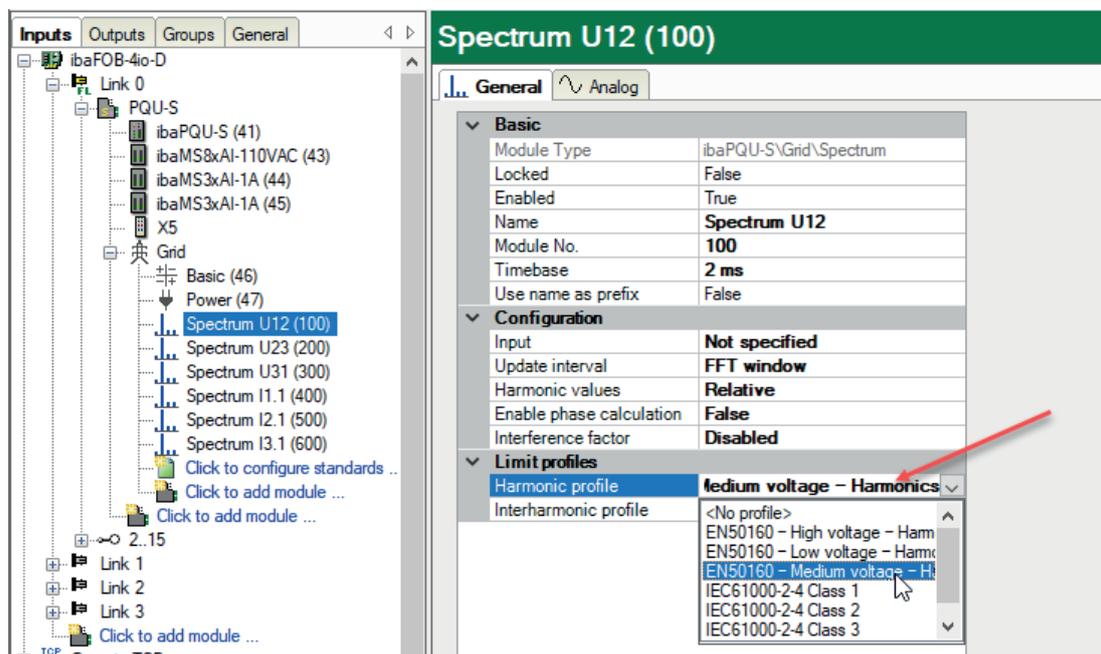
The limit values can be viewed with *Configure profiles*. In our example, the limit for the third harmonic is 5 %.
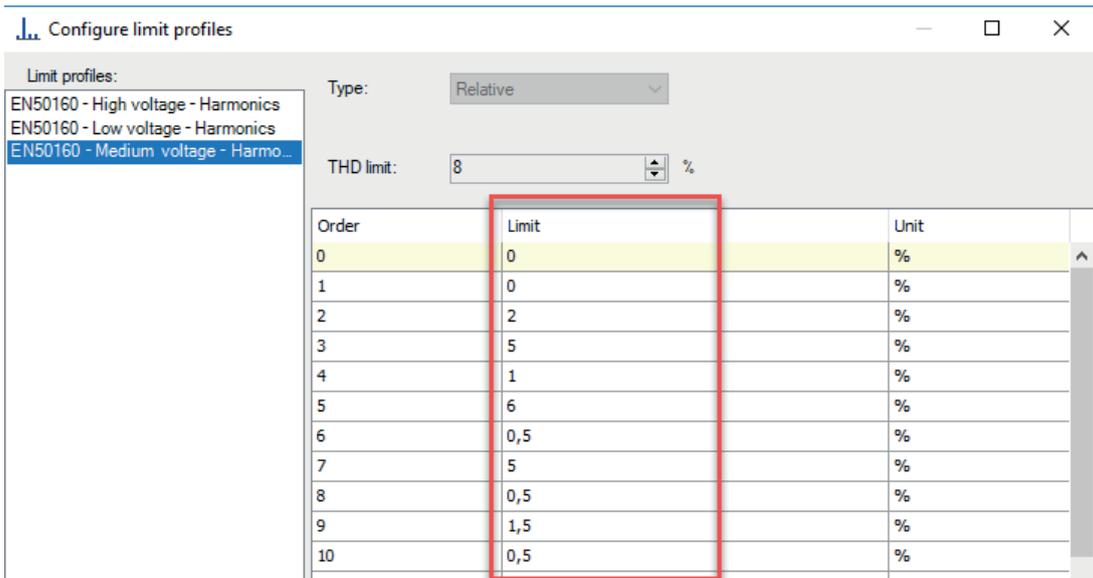
Solution
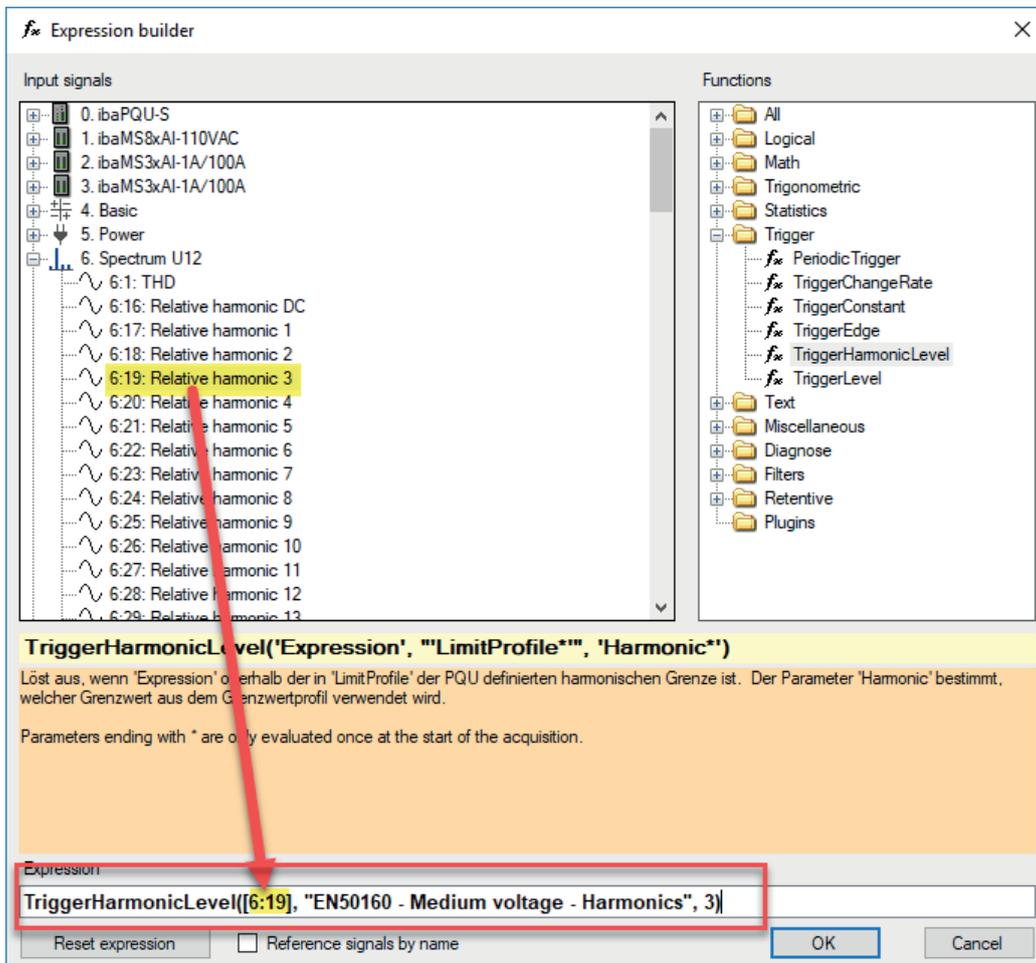
Fig. 1: Configuration of the TriggerHarmonicLevel expression

The trigger fires if the 10 min value of the third harmonic of U12 is above 5 %.

2.7 Text functions

2.7.1 CharValue

```
CharValue('Text', 'CharNumber=0')
```

Arguments

'Text'	Text signal	
'CharNumber'	Position of the character in the text, default = 0	

Description

This function returns the ASCII decimal value of the character at the 'CharNumber' position in a text. The default provides the first character (position = 0).

Example: If 'text' has the value "A_text for test", then the result of CharValue('Text', 0) is the ASCII value 65 for the uppercase A.

Example

Presenting the example referred to in the description

Solution

Query of the ASCII decimal value of the first character in the text "A_Text for test". A has the value 65 in the ASCII table.

Tip

  See ASCII table under http://www.ascii-code.com/.

2.7.2   CompareText

```
CompareText("'Text1'", "'Text2'", ' CaseSensitive=1*')
```

Comment:

The original name of this function was TextCompare. It was later renamed as CompareText. For compatibility reasons, the TextCompare function can still be used. The arguments are identical.

Arguments

'Text1'	Specification of the first comparison text	
'Text2'	Specification of the second comparison text	
'CaseSensitive'*	Optional parameter (default = 1) for determining whether a case sensitive comparison (consideration of upper and lower case) should be made	
	'CaseSensitive' ≠ 0	Take upper and lower case into account
	'CaseSensitive' = 0	Ignore upper and lower case

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function compares 2 texts alphabetically. This means that the characters of the 2 strings are compared individually starting with the left character. The function uses the current culture setting (Windows) to obtain culture-specific information such as spelling and alphabetical order. The comparison does not consider multi-digit values, word meaning or the length of the string. Blanks are taken into account in the comparison. With the optional parameter 'CaseSensitive=1', it can be determined whether upper and lower case should be considered (=1 or not specified), or not (<>1).

Dynamic text can be used by selecting a text signal. A text signal is set in square brackets []. It is also possible to enter constant text by just typing the text between the double quotes.

The function returns an analog value as the result.

Results

-1	if the characters of the first text appear in the alphabet before the characters of the second text.
0	if the two texts are equal
1	if the characters of the first text appear in the alphabet after the characters of the second text.

Example

The following table shows a couple of examples:

Text1	Text2	Result		Comment
		CompareText ("Text1","Text2",0)	CompareText ("Text1","-Text2",1)	
1234 abcd	1234 abcd	0	0	1 = 2
1234 abcd	1234 bcde	-1	-1	Text1 < Text2 "a" comes before "b"
1234 Abcd	1234 abcd	0	1	Text1 = Text2 (not case sensitive) Text1 > Text2 (case sensitive) "A" comes after "a"
12340 abcd	1234 ␣ abcd	1	1	Text1 > Text2 "0" comes after " ␣ "
1234 0abcd	1234 abcd	-1	-1	Text1 < Text2 "0" comes before "a"
12034 abcd	1234 abcd	-1	-1	Text1 < Text2 "0" comes before "3"
1234 abcd	1y34 abcd	-1	-1	Text1 < Text2 "2" comes before "y"
1z34 abcd	1Y34 abcd	1	1	Text1 > Text2 "z" comes after "Y"

2.7.3   ConcatText

```
ConcatText('Text1', Text2', …)
```

Arguments

'Text*n*'	Text signal

Description

This function returns the concatenation of 'Text1', 'Text2' etc. as its result.

If you want to use double quotes in static text, then write two double quotes after each other.

Example

The values of tree text signals should be concatenated.

Text signals are [37:0], [37:1] und [37:2].

Solution

```
ConcatText([37:0],[37:1],[37:2])
```

2.7.4   ConvertFromText

```
ConvertFromText('Expression', 'DecimalPoint*=0', 'Begin=0', 'End'=-1 (end of
text)', 'Base*=10', 'Bitsize*=32')
```

Arguments

'Expression'	Name of the text signal	
'DecimalSeparator*'	Decimal point	
	DecimalSeparator = 0	Period
	DecimalSeparator = 1	Comma
'Begin'	Index of the first character of the desired text, default = 0; if <0, it will be internally converted to 0.	
'End'	Index of the first character of the desired text, default = -1 (end of the text)	
'Base*'	Expected numerical base of the number in the text	
	'Base=2'	binary
	'Base=8'	octal
	'Base=10' (default)	decimal
	'Base=16'	hexadecimal
'Bitsize*'	Number of bits, the parsed number should have	Typ. values: 32 (default), 64, 16, 8

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The function parses a floating-point number from the text and delivers the numeric value as an decimal number. 0 is output if there is no number in the first position of the range defined by the 'Begin' and 'End' arguments. Exception: if there are only spaces present up to the first numeric character. Otherwise the text is read up to the first non-numerical position or maximum to the 'end'. Leading zeros before a number must not to be interrupted by spaces or non-numerical characters.

By the argument 'DecimalSeparator' you determine the decimal point. This is only relevant, if the number in the text is already a floating-point number ('Base=10'). 'DecimalSeparator' is ignored, if the argument 'Base' is not 10.

The 'Base' argument determines the numerical base expected in the text input. Possible values are 2 (binary), 8 (octal), 10 (decimal) and 16 (hexadecimal). While parsing the text, the function interprets the number in the text as a number with the numerical base 'Base' and converts it to the corresponding decimal number. If the number in the text is a decimal number you can omit the 'Base' argument. If, for instance, the number in the text is a hexadecimal number, then enter 16 for 'Base'. The result for the number FF in the text will be 255.

The "Bitsize" argument is only relevant if the 'Base' argument is set to something different than 10. It determines how many bits the parsed number should have. This is necessary for correctly parsing numbers that are represented in two's complement. For example, parsing the number

0xFFFF results in -1 when using 'Bitsize=16', but it results in 65535 when a larger Bitsize (e.g. 32) is used.

ConvertFromText with a 'Base' value different than 10 internally always works on 64-bit signed integers, so signal values larger than $2^{63} - 1$ are not correctly converted. Also note, that 'Bitsize' is limited to 64 and consequently only 64 bit signed integers can be parsed.

Example 1: Converting a decimal number

Reading in a defined text

Solution

The function is: ConvertFrom Text('Text', 0,22,-1)

The content of the text signal is: Voltage target value: 6.9 V

The voltage value starts with position 22. -1 is used as the end index so that values with more preceding and/or following commas can also be acquired.
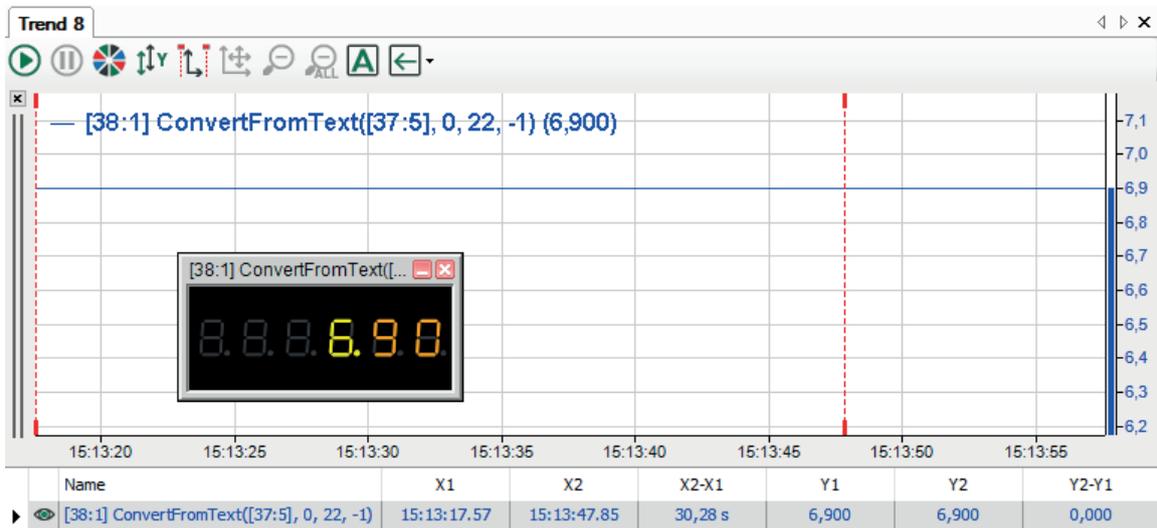
Example 2: Converting with numerical base

Parsing and converting a number from a text, taking into consideration the numerical base

The text line in the following examples contains a number beginning at position 17, which is converted by the function *ConvertFromText,* considering different numerical bases. The right column shows the syntax of the function. The resulting values (highlighted in green) are always decimal numbers.

ConvertFromText		
Text with number 255		
binary	0,00	ConvertFromText([30:5], 0, 17, - 1,2)
octal	173,00	ConvertFromText([30:5], 0, 17, - 1,8)
decimal	255,00	ConvertFromText([30:5], 0, 17)
hexadecimal	597,00	ConvertFromText([30:5], 0, 17, - 1,16)

The binary conversion has no result, because the numeric characters 2 and 5 do not exist in the binary system.

ConvertFromText		
Text with number 1011		
binary	11,00	ConvertFromText([30:5], 0, 17, - 1,2)
octal	521,00	ConvertFromText([30:5], 0, 17, - 1,8)
decimal	1011,00	ConvertFromText([30:5], 0, 17)
hexadecimal	4113,00	ConvertFromText([30:5], 0, 17, - 1,16)

The binary conversion shows a result, because the numerical characters 0 and 1 do exist in the binary system.

ConvertFromText		
Text with number FF		
binary	0,00	ConvertFromText([30:5], 0, 17, - 1,2)
octal	0,00	ConvertFromText([30:5], 0, 17, - 1,8)
decimal	0,00	ConvertFromText([30:5], 0, 17)
hexadecimal	255,00	ConvertFromText([30:5], 0, 17, - 1,16)

The conversion for binary, octal and decimal systems shows no result because the character F does not exist in these numerical systems.

2.7.5   ConvertToText

```
ConvertToText('Expression','IntegerDigits*=1','FractionalDigits*='',Plus-
Sign*=2','DecimalSeparator*=0','Base*=10')
```

Arguments

'Expression'	Expression (floating point value), which should be converted into text	
'IntegerDigits*'	Minimum number of integer digits	
'FractionalDig- its*'	Number of decimal digits	
'PlusSign*'	Representation of positive values (plus sign)	
	'PlusSign' = 0	Space
	'PlusSign' = 1	"+"
	'PlusSign' = 2	Nothing
	'PlusSign' = 3	Prefix for numbers with other bases than 10: 0x for hexadecimal numbers 0b for binary numbers 0o for octal numbers
'DecimalSepara- tor*'	Decimal point	
	'DecimalSepara- tor*'= 0 'DecimalSepara- tor*'= 1	Dot Comma
'Base*'	Numerical base of the resulting number	
	'Base=2'	binary
	'Base=8'	octal
	'Base=10' (default)	decimal
	'Base=16'	hexadecimal

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns a floating-point value as text. You can specify the minimum number of integer digits with the argument 'IntegerDigits'. By means of the argument 'FractionalDigits' you specify the number of decimal digits. If 'FractionalDigits' is smaller than zero, then only non-zero digits will be printed. If 'FractionalDigits' is larger than zero, then zeros are printed. The parameter 'PlusSign' determines the printing of the plus sign.

The 'Base' argument determines the numerical base which should be used for the representation of the converted number in the text. You can choose between the values 2 (binary), 8 (octal), 10 (decimal, default) or 16 (hexadecimal).

If the 'Base' argument has a different value than 10, it has an effect on other arguments:

- The 'IntegerDigits' argument determines the exact number of digits instead of the minimal number of digits like for decimal values.

- The 'FractionalDigits' and 'DecimalSeparator' arguments are ignored entirely. The behavior here is to round the number and therefore remove the fractional part.

You may use the 'PlusSign' argument to add a prefix to the resulting number, corresponding to its numerical base, e.g. "0x" for hexadecimal values.

Note

When converting negative numbers to text, two's complement representation is used. This uses 64 bits of precision, so the resulting strings become very large. To limit the length of the string, the 'IntegerDigits' argument is used. It is important to note, that excess digits are simply cut off, which can lead to negative numbers "looping", therefore becoming positive again, and vice-versa.

Example 1: Conversion of a floating-point value to text

Examples for the floating-point value [FloatValue] = 42.471130 with different argument settings

The 'Base' argument is not mentioned because the base is 10, which is the default value.

'Expression'	'Expression'	'IntegerDigits'	'FractionalDigits'	'PlusSign'	'DecimalPoint'
	[FloatValue] Value: 42.471130	1	4	2	1
Formula	ConvertToText([FloatValue],1,4,2,1)				
Result	42.4711				

'Expression'	'Expression'	'IntegerDigits'	'FractionalDigits'	'PlusSign'	'DecimalPoint'
	[FloatValue] Value: 42.471130	1	6	1	1
Formula	ConvertToText([FloatValue],1,6,1,1)				
Result	+42.471130				

'Expression'	'Expression'	'IntegerDig-its'	'Fraction-alDigits'	'PlusSign'	'DecimalPoint'
	[FloatValue] Value: 42.471130	1	-6	0	1
Formula	ConvertToText([FloatValue],1,-6,0,1)				
Result	42.47113				

Example 2 Conversion of decimal values to text with consideration of the numerical base

The 'PlusSign' argument is set to 3 in order to indicate the numerical base of the resulting number by a prefix.

Base 2 (binary)

Formula	ConvertToText([DecimalValue],8,-2,3,0,2)					
[DecimalVal-ue]	'IntegerDigits'	'FractionalDig-its'	'PlusSign'	'DecimalSep-arator'	'Base'	Result
9.00	8	-2	3	1	2	**0b00001001**
16.00	8	-2	3	1	2	**0b00010000**
255	8	-2	3	1	2	**0b11111111**

Base 8 (octal)

Formula	ConvertToText([DecimalValue],6,-2,3,0,8)					
[DecimalVal-ue]	'IntegerDigits'	'FractionalDig-its'	'PlusSign'	'DecimalSep-arator'	'Base'	Result
9.00	6	-2	3	1	8	**0o000011**
16.00	6	-2	3	1	8	**0o000020**
255	6	-2	3	1	8	**0o000377**

Base 16 (hexadecimal)

Formula	ConvertToText([DecimalValue],4,-2,3,0,16)					
[DecimalVal-ue]	'IntegerDigits'	'FractionalDig-its'	'PlusSign'	'DecimalSep-arator'	'Base'	Result
9.00	4	-2	3	1	16	**0x0009**
16.00	4	-2	3	1	16	**0x0010**
25.00	4	-2	3	1	16	**0x00FF**

2.7.6   CountText

`CounText('Text', 'CountOnlyDifferent*=0', 'Reset=0')`

Arguments

'Text'	Name of the text signal	
'CountOnlyDifferent*'	Optional parameter (default = 0)	
	'CountOnlyDifferent' <> 0	New text is only counted if it is different from the preceding text.
	'CountOnlyDifferent' = 0	Each new text is counted.
'Reset'	Optional parameter that can be used to reset the counter reading. 'Reset' can be an expression as well.	
	'Reset' > 0	Counter is reset.
	'Reset' = 0	Counting released (default)

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function counts the reception or the change of a text signal and returns the counter reading.

2.7.7   DeleteText

```
DeleteText('Text','StartPos','Lenght=100000')
```

Arguments

'Text'	Text signal or "text" where characters should be deleted
'StartPos'	Analog signal or number; specifies the start position for the deletion in the text. Counting starts at zero (0 = 1st character).
'Length'	Analog signal or number; specifies the number of characters to be deleted. Default=100000

Description

This function deletes 'Length' characters from a text beginning with the character at position 'StartPos'. If 'Length' is not specified, all characters from 'StartPos' until the end will be deleted.

Example

Starting at position 4, four characters should be deleted from a text.

Arguments	'Text'	'StartPos'	'Length'
	[TextSignal] Value: ABCDE1234XYZ	4	4
Formula	DeleteText([TextSignal],4,4)		
Result	ABCD4XYZ		

2.7.8   FindText

```
FindText("'Text1'", "'Text2'", ' CaseSensitive=1*')
```

Arguments

'Text1'	Specification of the first comparison text	
'Text2'	Specification of the second comparison text	
'CaseSensitive'*	Optional parameter (default = 1) for determining whether a case sensitive comparison (consideration of upper and lower case) should be made	
	'CaseSensitive' <> 0	Take upper and lower case into account
	'CaseSensitive' = 0	Ignore upper and lower case

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function checks whether 'Text2' is contained in 'Text1' and returns the index as a result (position in 'Text1, first character = index 0) where 'Text2' was found for the first time. Blanks are considered in the search. With the optional parameter 'CaseSensitive=1', it can be determined whether upper and lower case should be considered (=1 or not specified), or not (<>1).

Dynamic text can be used by selecting a text signal. A text signal is set in square brackets []. It is also possible to enter constant text by just typing the text between the double quotes.

The function returns an analog value as the result.

Results

-1	if the second text is not included in the first text
0....n	Position where 'Text2' is in 'Text1' the first time

Example

The following table shows a couple of examples:

Text1	Text2	Result		Comment
		FindText ("Text1","Text2",0)	FindText ("Text1","Text2",1)	
The Sun is shining.	-1	4	4	
The moon is shining	-1	-1	-1	
The sun is shining	-1	4	-1	

2.7.9   InsertText

```
InsertText('Text1','Text2','Pos')
```

Arguments

'Text1'	Text signal or "text" where another text should be inserted	
'Text2'	Text signal or "text" whose text should be inserted into 'Text1'	
'Pos'	Analog signal or number; specifies the position where the text should be inserted.	
	'Pos' ≤ 0	'Text2' is prepended to 'Text1'.
	'Pos' ≥ length of 'Text1'	'Text2' is appended to 'Text1'.

Description

This function inserts a 'Text2' into a 'Text1' at position 'Pos'.

Example

Arguments	'Text1'	'Text2'	'Pos'
	[TextSignal_1] Value: 123_XYZ	[TextSignal2] Value: abcd_	4
Formula	InsertText([TextSignal1],[TextSignal2],4)		
Result	123_abcd_XYZ		

2.7.10   MidText

```
MidText('Text','StartPos','Lenght=100000')
```

Arguments

'Text'	Text signal or "text" from where characters should be returned
'StartPos'	Analog signal or number; specifies the start position for the reading in the text. Counting starts at zero (0 = 1st character).
'Length'	Analog signal or number; specifies the number of characters to be read, Default=100000

Description

This function reads and returns 'Length' characters from a text beginning with the character at position 'StartPos'. If 'Length' is not specified, all characters from 'StartPos' until the end will be read.

Example

A standardized order number (text) contains always at the 6th position a four-digit customer code. This customer code should be read out.

Arguments	'Text'	'StartPos'	'Length'
	[TextSignal_Order] Value: 4711_AC-ME_1234XYZ	5	4
Formula	MidText([TextSignal_Order],5,4)		
Result	ACME		

2.7.11   ReplaceText

```
ReplaceText('Text','SearchText','ReplaceText')
```

Arguments

'Text'	Text signal or "text" with content to be replaced
'SearchText'	Text signal or "text" with the text to be replaced
'ReplaceText'	Text signal or "text" as replacement

Description

This function replaces all occurrences of 'SearchText' within 'Text' with 'ReplaceText'. All arguments can be text signals or static text. If you use static text in the formula, put it in double quotes. By using text signals, you can vary dynamically both the text to be replaced ('Search-Text') and the replacement text ('ReplaceText'). Even the basic text ('Text'), which contains parts to be replaced, can be a text signal.

If you want to use double quotes in static text, then write two double quotes after each other.

Examples

The following table shows several simplyfied applications.

Arguments	'Text'	'SearchText'	'ReplaceText'
	[TextSignal] Value: AAA BBB CCC	"BBB"	[ReplaceTextSignal] Wert: XXX
Formula	`ReplaceText([TextSignal],"BBB",[ReplaceTextSig-nal])`		
Result	AAA XXX CCC		

Arguments	'Text'	'SearchText'	'ReplaceText'
	[TextSignal] Value: AAA BBB CCC	[SearchTextSignal] Value: BB CC	"XXX"
Formula	`ReplaceText([TextSignal],[SearchTextSig-nal],"XXX")`		
Result	AAA BXXXC		

Arguments	'Text'	'SearchText'	'ReplaceText'
	"Hello Name"	"Name"	[ReplaceTextSignal] Value: John
Formula	`ReplaceText("Hello Name","Name",[Replace-TextSignal])`		
Result	Hello John		

2.7.12   TextLength

```
TextLength('Text')
```

Arguments

'Text'	Text signal or "text" whose length should be returned in number of charac-ters

Description

This function returns the total number of characters of a text.

Example

Arguments	'Text'
	"How many characters are in this text?"
Formula	TextLength("How many characters are in this text?")
Result	37

2.7.13   TrimText

```
TrimText('Text', 'TrimOption=0')
```

Arguments

'Text'	Text signal to be processed	
'TrimOption'	Optional argument, specifying which space characters should be removed	
	'TrimOption' = 0	Default; remove leading and trailing space characters.
	'TrimOption' = 1	Remove only leading space characters.
	'TrimOption' = 2	Remove only trailing space characters.
	'TrimOption' = 3	Remove all space characters, including the internal space characters.

Description

This function removes space characters from a text. By using the argument 'TrimOption' you can control, which space characters are concerned.

Example

In different steps, the space characters should be removed from the following text of a text signal 'TextToTrim'. The pipe characters mark begin and end of the text including space, but they are not part of the text.

| Lorem ipsum dolor sit amet |

Solution

`TrimText('TextToTrim')`	|Lorem ipsum dolor sit amet|
`TrimText('TextToTrim',1)`	|Lorem ipsum dolor sit amet |
`TrimText('TextToTrim',2)`	| Lorem ipsum dolor sit amet|
`TrimText('TextToTrim',3)`	|Loremipsumdolorsitamet|

2.8   Miscellaneous functions

2.8.1   ClientInfo

`ClientInfo('"ClientAddress*"','InfoType*')`

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns information about the client, which is specified by 'ClientAddress'. The parameter 'ClientAddress' can be the IP address or the name of the client computer (case insensitive). This information may be used for controlling other functions or for display and diagnostic purposes.

Specify the information type ('InfoType') you want to receive.

The following information types are supported:

Information types	Possible results
0	0 = No connection to server 1 = Client and server are connected
1	Number of signals requested by the client
2	0 = Client does not use a client license 1 = Client uses a client license
3	0 = Client does not use a QPanel license 1 = Client uses a QPanel license

Table 2: Information types and possible results of the ClientInfo function

2.8.2   ClientInfoText

`ClientInfoText('"ClientAddress*"','InfoType*')`

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns textual information about the client, which is specified by 'ClientAddress'. The parameter 'ClientAddress' can be the IP address or the name of the client computer (case insensitive). This information may be used for controlling other functions or for display and diagnostic purposes.

Specify the information type ('InfoType') you want to receive.

The following information types are supported:

Information types	Possible results
0	Name of the client computer
1	IP address of the client computer
2	Windows user name of the client
3	Name of the logged in ibaPDA user
4	Connection starttime of the client
5	ibaPDA Version of the client

Table 3: Information types and possible results of the ClientInfoText function

2.8.3 Count

`Count('Expression','Level*','Hysteresis*','EdgeType*', 'Reset=0')`

Arguments

'Expression'	Measured value	
'Level*'	Specification of the level value	
'Hysteresis'	Specification of a hysteresis band	
'EdgeType*'	Indication of whether rising, falling or rising and falling edges should be counted	
	'EdgeType' <0	only falling edges (leaving out hysteresis band in the negative direction)
	'EdgeType' >0	only rising edges (leaving out hysteresis band in the positive direction)
	'EdgeType' = 0	falling and rising edges
'Reset'	Optional digital parameter that can be used to reset the counter. 'Reset' can be an expression as well. An analog signal signal can be used for reset too, but it's value must be exactly = 1 (integer).	
	'Reset' = 1 or True	Counter is reset; analog reset signal: value has to be integer
	'Reset' ≠ 1	Counter value is retained / continues to count (default)

Parameters ending with * are only evaluated once at the start of the acquisition.

Note

The 'Reset' condition must not be related to the count function itself.

Description

The function counts the crossings of 'Expression' through 'Level'.

The 'Hysteresis' parameter can be used to define a tolerance band which is above and below 'Level' by equal amounts. Only complete crossings through the tolerance band are counted.

The 'EdgeType' parameter determines which kind of edges are counted.

The 'Reset' parameter is used to reset the counter value to 0. 'Reset' can also be formulated as an expression.

Examples:

Count([0:0],10,1,1)	No reset happens ('Reset' omitted)
Count([0:0],10,1,1,If(Mod(T(),20)=0,TRUE() m,FALSE()))	The counter is reset in a time interval of 20 seconds.
Count([0:0], [3.1])	e.g. with [3.1] = If([0:0]<1, 1, 0) The counter is reset as soon as the expression [3.1] returns TRUE, i.e. if the expression [0:0] falls below the limit value 1.

Tip

The COUNT function can also be used for binary signals. For this purpose, choose 0.5 for Level and, for example, 0.1 for Hysteresis. This then means that all changes from FALSE to TRUE and vice versa are detected and counted.

Example

Tip

This function can be used in a virtual retentive module. Its result values can thus be obtained despite stopping and restarting the measurement.

The number of deviations of the measuring signal from the value of 10 in a 20 second interval in both directions should be counted, but only after complete crossings through the hysteresis band of width 2. The maximum value of each interval without the start phase should be saved.

Task description

The 'Reset' parameter of the count function consists of an If query of the modulo function of time. The maximum value is reset once via the OneShot function as soon as the measured value has leveled off at the beginning of the measurement.

Solution
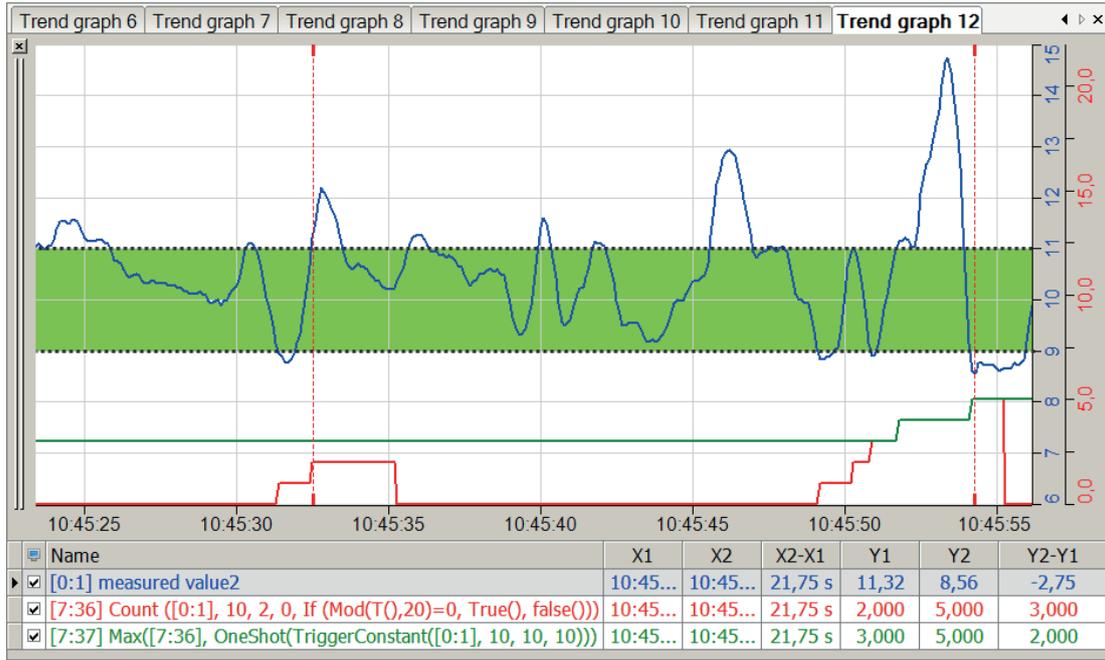
| Blue | Measured value | Green area | Hysteresis band |
| Red | Count function | Green signal | Maximum value of the intervals of the count function |

Tip

When specifying a hysteresis band of 2 around 'Level' 10, the crossings in the increasing direction are first counted for 'Expression'>11 and in the descending direction for 'Expression'<9.

2.8.4   CountUpDown

```
CountUpDown('Up','Down','Reset','UpperLimit=none','LowerLimit=none','ResetOnLim-
it=0')
```

Arguments

'Up'	Digital signal whose rising edge (FALSE --> TRUE) increases the counter by 1	
'Down'	Digital signal whose falling edge (TRUE --> FALSE) decreases the counter by 1	
'Reset'	Optional parameter (default = 0) to stop, reset and restart the calculation	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation and set result to 0
'UpperLimit'	Optional parameter (default = 0); upper limit for counter value	
'LowerLimit'	Optional parameter (default = 0); lower limit for counter value	
'ResetOnLimit'	Optional parameter (default = 0) for resetting the counter value when reaching a limit	
	'ResetOnLim-it'= 0	Counter value stays on limit value after reaching a limit
	'ResetOnLim-it' = 1	Counter value is reset to 0 after reaching a limit

Description

This function generates a counter value which is increased by 1 with every rising edge on 'Up' and decreased by 1 with every rising edge on 'Down'. Counting is not limited but can be reset to zero with 'Reset' = TRUE or 1.

'UpperLimit' and 'LowerLimit' are optional and can be specified either by a fix value or an expression. If limit arguments are specified and the counter reaches a limit, it does not continue counting but stays on the limit value until either a counter pulse in the other direction occurs or 'Reset' is set to TRUE.

If the optional parameter 'ResetOnLimit' is TRUE or 1, then the counter will be reset to zero as soon as it reaches or exceeds a limit.

Resetting to zero only works if the lower limit is <0 and the upper limit is >0.

Example
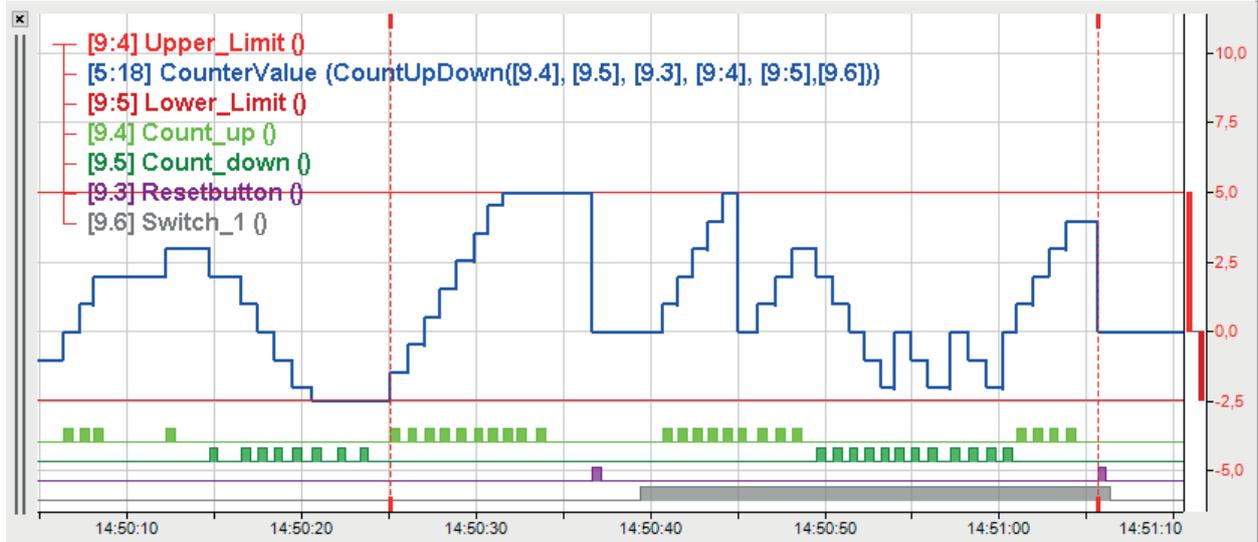
2.8.5   Delay

`Delay('Expression','NumberSamples*')`

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns a delayed copy of the 'Expression' signal. The delay is specified in number of measurements ('NumberSamples'). The result is a signal curve with the values of the original signal for 'NumberSamples' before the current time.

To avoid a memory overflow, 'NumberSamples' is limited to 10,000.

Tip

The time base of the function is relevant for the number of measurements for different time bases of the measured value and function.

Example

A measured value should be displayed delayed by 2 measured values and by 10 measured values.

Task description

For the chosen time base of 100 ms, a delay of two measured values corresponds to a time delay of 0.2 seconds and a delay of 10 measured values corresponds to 1 second.

Solution

In the figure below the blue curve shows the original signal and the red curve shows the delayed signal by 2 measured values (= 0.2 seconds). The green curve shows the delayed signal by 10 measured values (= 1 second).
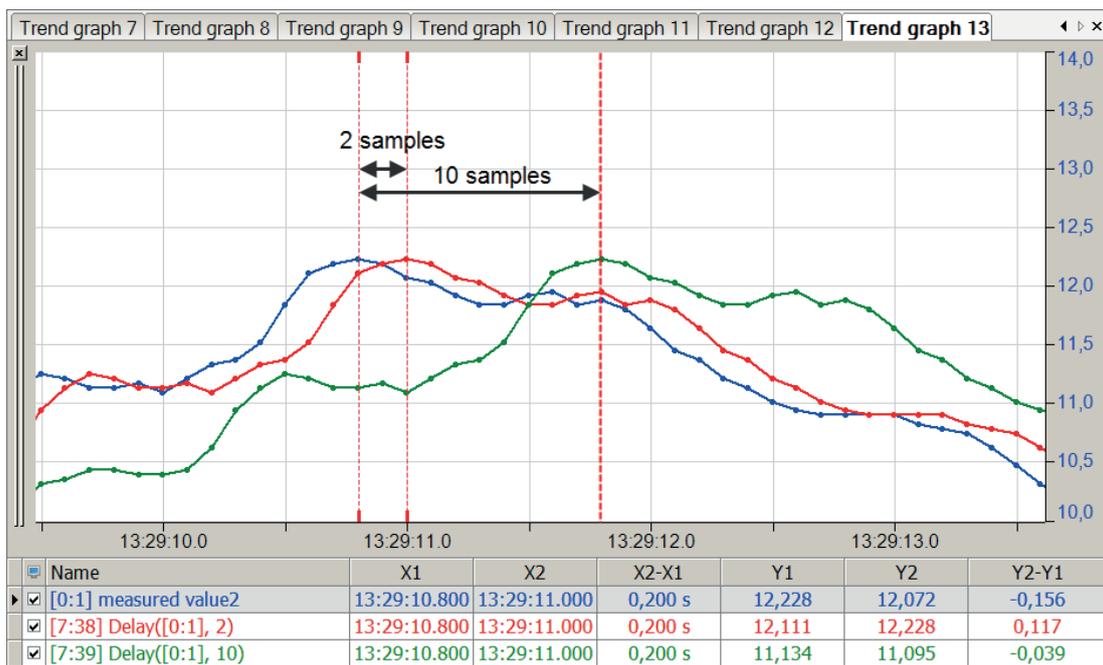
2.8.6   DelayLengthL

```
DelayLengthL('Expression','Length','MaxLengthDelta','DelayInMeter','Resolution*',
'Filter=0*')
```

Arguments

'Expression'	input signal	
'Length'	Length signal (in m)	
'MaxLengthDel-ta'	Upper limit to take into account the changes of the length signal	
'DelayInMeter'	Delay (in m)	
'Resolution*'	Resolution: Length basis of the result (in m)	
'Filter'*	Optional parameter (default = 0) to set the filter for time-length conversion	
	'Filter' = 1	Minimum filter
	'Filter' = 2	Maximum filter
	'Filter' = 0 and others	No filter

Parameters ending with * are only evaluated once at the start of the acquisition..

Description

This function uses the 'Length' length signal (in m) to create a length-based version of 'Expression' that is delayed over 'DelayInMeter' meters. Changes in the length signal that exceed the 'MaxLengthDelta' are ignored. The resolution is the length base of the result (in m).

'Filter' determines what filter is used during the time-to-length conversion.

Mainly for quality data monitoring in conjunction with ibaQDR, measured signals from distant sources can be aligned with regard to the product length. The distance between the signal sources determines the 'DelayInMeter' value.

2.8.7   DelayLengthV

```
DelayLengthV('Expression','Speed','DelayInMeter','Resolution*', 'Filter=0*')
```

Arguments

'Expression'	input signal	
'Speed'	Speed signal (in m/s)	
'DelayInMeter'	Delay (in m)	
'Resolution*'	Resolution: Length basis of the result (in m)	
'Filter'*	Optional parameter (default = 0) to set the filter for time-length conversion	
	'Filter' = 1	Minimum filter
	'Filter' = 2	Maximum filter
	'Filter' = 0 and others	No filter

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function uses the 'Speed' speed signal (in m/s) to create a length-based version of 'Expression' that is delayed over 'DelayInMeter' meters. The resolution is the length base of the result (in m).

'Filter' determines what filter is used during the time-to-length conversion.

2.8.8   DWORD

```
DWORD('Low','High')
```

Arguments

'Low'	16 bit integer: Word	
'High'	16 bit integer: Word	

Description

This function returns the 32 bit integer DWORD consisting of the int16 WORDS 'Low' and 'High.'

Example

Calculating two DWORDS from a simple 0 and 1 combination of 'Low' and 'High'.

	from int16 WORDS compound DWORD			
DWORD('Low','High')	int16 'High'	int16 'Low'	Conv.	Dec.
DWORD(0, 1)	0000 0000 0000 0001	0000 0000 0000 0000	216	65536
DWORD(1, 0)	0000 0000 0000 0000	0000 0000 0000 0001	20	1

Solution

The figure below shows the calculation of two simple DWORDS.
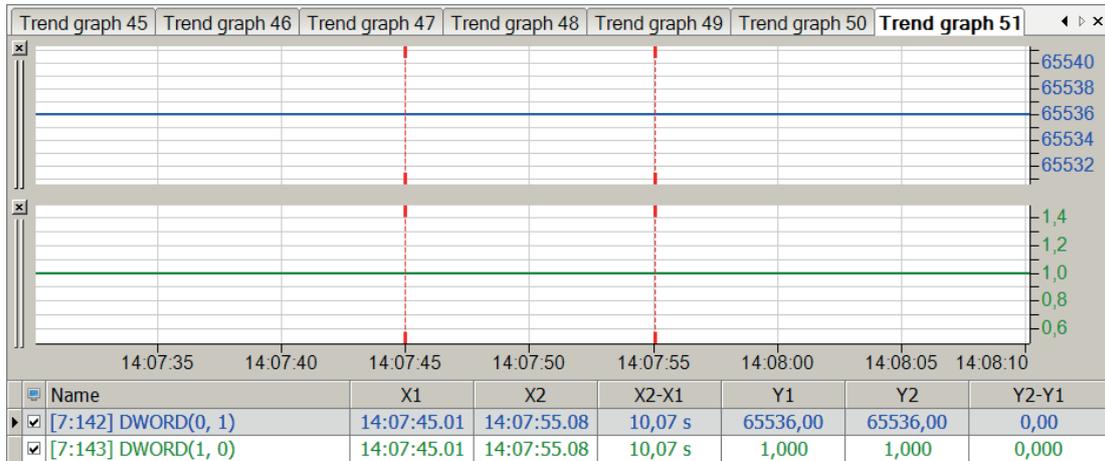
2.8.9   ElapsedTime

```
ElapsedTime('Start','Stop')
```

Description

This function returns the time since the last rising edge for 'Start'. The time is stopped at the rising edge for 'Stop'.

Example

Measurement of the time gap between the last rising edge of a measured value up to the next rising edge of a different measured value.

Solution

In the figure below the blue bar shows the 'Stop' signal and the red bar shows the 'Start' signal. The green curve shows the time that elapsed between the rising edges of 'Start' to 'Stop'.
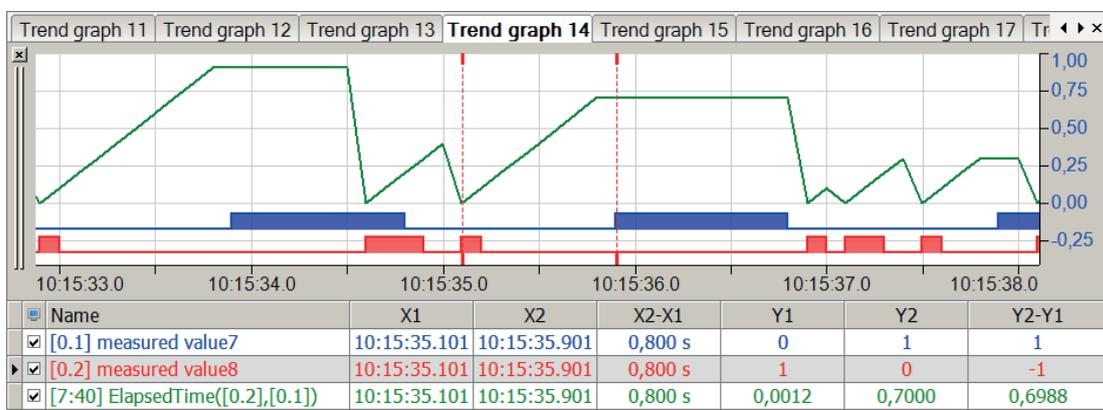
Note

By switching the edges in the sampling grid, the graphic presentation and calculation of the elapsed time may differ by one sample.

2.8.10   ExecuteCommand

```
ExecuteCommand('Trigger',"'Command'","'Arguments'","'UserName'","'Password'",'Tim-
eout=0')
```

Arguments

'Trigger'	Binary signal or expression as a trigger for the execution of the command
"'Command'"	Full and absolute path to the command (.exe, .bat etc.) that should be executed
"'Arguments'"	Arguments/parameters required to execute the command
"'UserName'"	Username of an account which has the permission to execute "'Command'".
"'Password'"	The password, related to the user name
'Timeout'	Optional parameter; if specified and >0, the execution will be aborted as soon as 'Timeout' [s] has expired.

Description

This function executes the "'Command'" "'Arguments'" command line for a rising edge of 'Trigger'. The return value is 1 (TRUE) while the invoked process is running. Thus, this function is to used as expression for digital signals. Other rising edges are ignored as long as the process is running.

The full, absolute path for the executable file must be specified for "'Command'", relative to the *ibaPDA* server.

If 'Timeout' is not specified, the command execution is not aborted and can run endlessly. If a value larger than zero is specified for 'Timeout', the execution of the command is aborted if it is not finished by 'Timeout' seconds.

The specification of username ("'UserName'") and password ("'Password'") is required for using this function. For security reasons the executables and programs called by this function must run under a dedicated user account. The account specified by "'UserName'" and "'Password'" must have the permission to execute the specified command.

When entering argument, username and password you can use the text encryption feature to obfuscate this information. The text encryption feature is offered as soon as you start to enter text between quotation marks ("…").

Note

i	If you use this function to call programs which generally have a user interface (GUI), then the GUI is not visible. Because the started program runs in the services session without an associated desktop, it is listed in the task manager, but it's GUI is missing.

Example 1

Computer shutdown

Task description

Depending on a signal from the system control due to switching to an uninterrupted power supply (USP) during a power outage, the *ibaPDA* computer should be shut down.

Solution

The "shutdown" command from the Windows Command Shell is used to shutdown the computer. A batch file is used because multiple commands, such as stopping the *ibaPDA* service, are required.

`ExecuteCommand([4.14], "D:\Schulung\ibaPDA_ExecuteCommand\Shutdown_PC.bat","","MyUsername","MyPassword")`

- `[4.17]`: Digital trigger signal to execute the command

- `D:\training\ibaPDA_ExecuteCommand\Shutdown_PC.bat`: Program path for the batch file

- `MyUsername` and `MyPassword`: Specifications for user account

Contents of the `Shutdown_PC.bat` batch file:

1	SETLOCAL
2	sc stop ibaPDAService
3	shutdown /f /s /t 10
4	ENDLOCAL

Example 2

Starting a batch file, which creates a text file with up-to-date content.

Task description

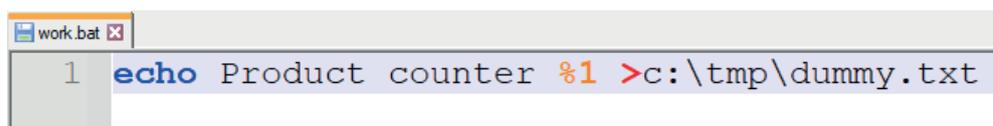On a rising edge of a digital signal an actual counter value should be written

Solution

`ExecuteCommand([3.17],"c:\tmp\work.bat", GenerateText("Items_%1"), "MyUserName","MyPassword",0)`

- `[3.17]`: Digital trigger signal to execute the command

- `c:\tmp\work.bat`: Program path for the batch file to be executed `work.bat`; specifiying a batch file is like using a command.
  Contents of the batch file (example):

- `GenerateText("Items_%1")`: Argument for the batch file; the result of this expression (generating a string with counter value) is put into the text file `dummy.txt` as parameter %1 of the batch file.
  Contents of the text file `dummy.txt` (example, dynamic content highlighted in yellow):

  ```
  dummy.txt
  1   Product counter Items_120212
  ```
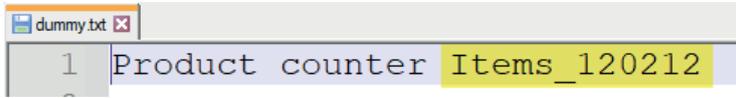
- `MyUserName, MyPassword`: Login credentials of the user account

- `0`: Value for timeout, i.e. the execution is not aborted.

2.8.11   GenerateSignal

```
GenerateSignal('Type', 'Amplitude=10','T1=1','T2=1')
```

Arguments

'Type'	Specification of the signal type	
	'Type' = 0	Sine function, with A = amplitude and T1 = period
	'Type' = 1	Cosine function, with A = amplitude and T1 = period
	'Type' = 2	Triangle function, with A = amplitude, T1 = rising edge time, T2 = falling edge time
	'Type' = 3	Block function, with A = amplitude, T1 = duration upper level, T2 = duration lower level
	'Type' = 4	Random signal with A = maximum amplitude
'Amplitude'	Optional parameter, specifying the amplitude; default = 10	
'T1'	Optional parameter, time specification 1 (except for type 4) given in s; default = 1	
'T2'	Optional parameter, time specification 2 (only for type 2 and type 3) given in s; default = 1	

Description

This function generates a test signal with the 'Amplitude' amplitude and periods 'T1' and 'T2.' The following signal types can be created:

Example

Several examples of generated signals

Solution

	Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
✓	[7:52] GenerateSignal(0, 2, 10)	10:17:00.24	10:17:34.38	34,14 s	1,831	-1,156	-2,987
✓	[7:53] GenerateSignal(1, 3, 5)	10:17:00.24	10:17:34.38	34,14 s	-2,03	0,99	3,02
✓	[7:54] GenerateSignal(2, 3.2, 1.5, 2.8)	10:17:00.24	10:17:34.38	34,14 s	1,05	1,64	0,60
✓	[7:55] GenerateSignal(3, 1.4, 2.2, 0.8)	10:17:00.24	10:17:34.38	34,14 s	1,400	1,400	0,000
✓	[7:56] GenerateSignal(4, 5)	10:17:00.24	10:17:34.38	34,14 s	0,54	2,45	1,91
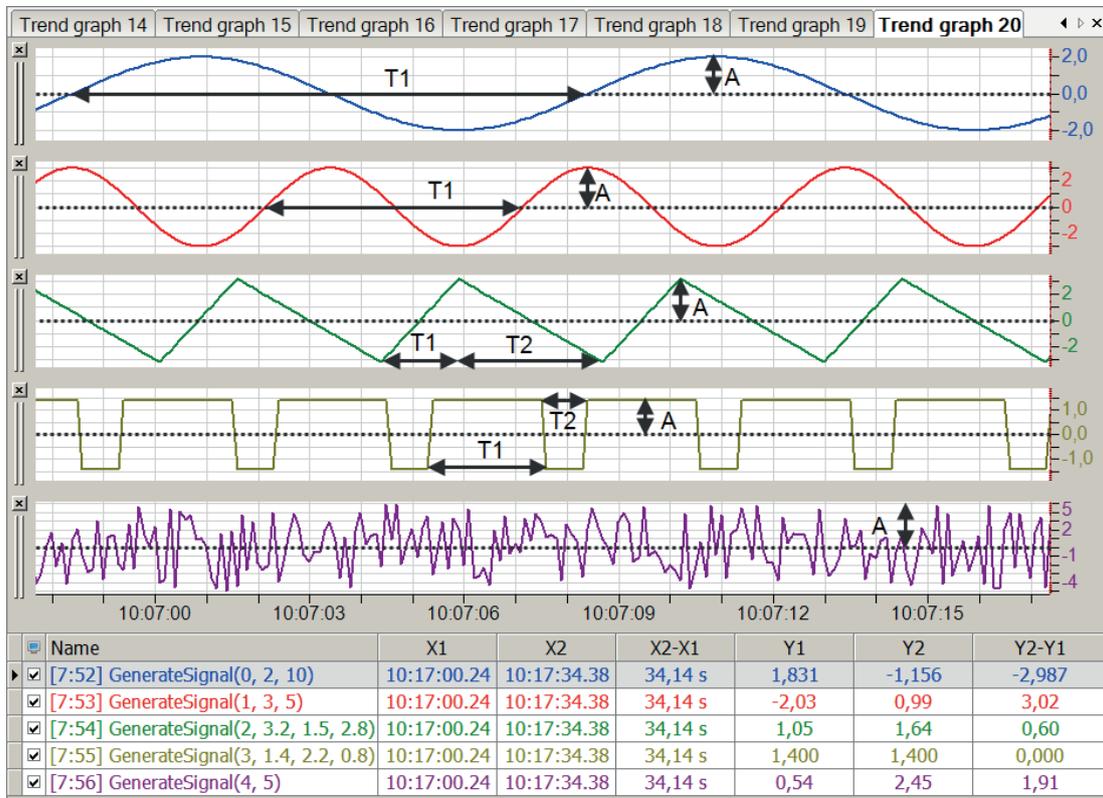
Blue	Sine function type 0	Red	Cosine function type 1
Green	Triangle function type 2	Yellow	Block function type 3
Purple	Random function type 4		

2.8.12   GenerateText

```
GenerateText('Text*')
```

Parameters ending with * are only evaluated once at the start of the acquisition..

Arguments

'Text'	Text which should be generated as content of the text signal. The text has to be put between quotation marks ("..").	
	The text will be evaluated only once at start of acquisition.	
	%n	Parameter, which can be inserted in the text and will be replaced in the generated text by an automatic counter value. The counter counts up by 1 with every nth cycle, based on the timebase of the *ibaPDA* system, e.g.
		%1 - with each cycle
		%2 - with each second cycle
		%10 - with each tenth cycle
		%100 - with each 100th cycle. Cycle

Description

This function generates a text signal. It can only be used in the signal table *Analog* of a suitable module (e.g. virtual module or output module)

The generated text signal can be used like other text signals, e.g. in a text creator module, a text digital display or a QPanel label.

You may use placeholders %1, %2,... in the text. The placeholder %n will be replaced by the counter value which increased every n samples.

Example
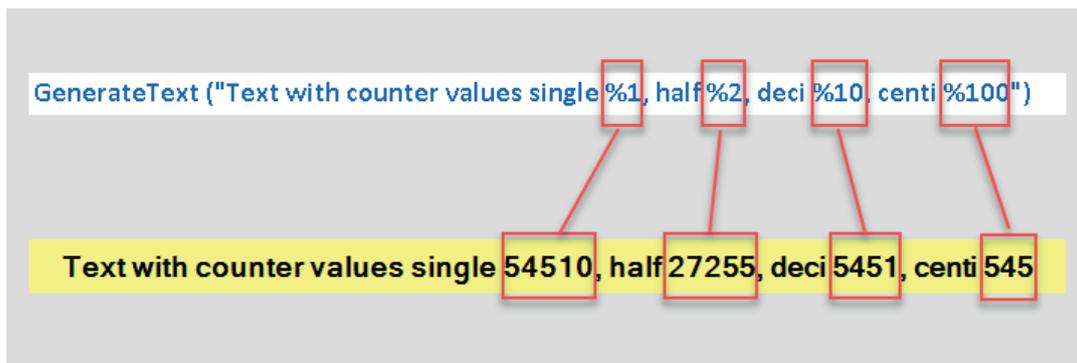
The upper line shows the complete function call. The lower line shows the result in a QPanel label.

2.8.13   GetFloatBit

```
GetFloatBit('Expression','BitNo')
```

Arguments

'Expression'	Any number
'BitNo'	This number (0…15 or 0…31) specifies the position of the desired digital signal in a 16-bit or 32-bit block in the data stream with regard to the related offset address. Increase of bit no. by 1 up to 15 (31), then increase of address by 2 (4).

Description

This function interprets 'Expression' as the bitmask of a float value and returns as its result the Boolean value of the 'BitNumber' bit of 'Expression'. Valid bit number sequence: 0 (LSB) to 31 (MSB).

This function was created especially for the case that 32 bits are packed in a float data for transmission or recording. The GetBitFloat function only takes the value of the specified 'BitNo' bit regardless of whether it is part of the mantissa or exponent. Unlike the GetBit function, there is no rounding of values.

Example

A value entered in 'Expression' is converted to the floating-point format IEEE 754 and the digits bits which are set to TRUE accordingly, are retrieved.

Solution

Floating-point numbers according to the IEEE 754 format can generally be written as follows: (sign)*mantissa*10^Exponent

The number 0.15625 reads as a floating-point number according to IEEE 754 as follows:

Sign (1 bit)	Exponent (8 bit)	Mantisse (23 bit)
0	0111 1100	0100 0000 0000 0000 0000 000

In this case, the LSB (least significant bit) is at the extreme right point.

Tip

Floating-point numbers according to the IEEE 754 format are written in the manner shown. The following points for the individual components should be noted:

Sign: 0 stands for positive; 1 for negative

Exponent: Since with the conversion to a floating-point number, the exponent value is shifted by 127, 127 must be subtracted from the displayed exponent in the conversion into a decimal number.

Mantissa: Because the mantissa has a 1 before the decimal point by default, this digit is no longer represented. All mantissa digits are therefore decimal digits and must take into account the conversion with negative exponents to base 2.

The figure below shows the retrieval of the valence of selected bits of the float value of 0.15625.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [7:98] GetFloatBit(0.15625, 31)	10:49:00.13	10:50:30.20	1:30.06	0,000	0,000	0,000
☑ [7:99] GetFloatBit(0.15625, 30)	10:49:00.13	10:50:30.20	1:30.06	0,000	0,000	0,000
☑ [7:92] GetFloatBit(0.15625, 29)	10:49:00.13	10:50:30.20	1:30.06	1,000	1,000	0,000
☑ [7:93] GetFloatBit(0.15625, 28)	10:49:00.13	10:50:30.20	1:30.06	1,000	1,000	0,000
☑ [7:94] GetFloatBit(0.15625, 27)	10:49:00.13	10:50:30.20	1:30.06	1,000	1,000	0,000
☑ [7:95] GetFloatBit(0.15625, 26)	10:49:00.13	10:50:30.20	1:30.06	1,000	1,000	0,000
☑ [7:96] GetFloatBit(0.15625, 25)	10:49:00.13	10:50:30.20	1:30.06	1,000	1,000	0,000
☑ [7:97] GetFloatBit(0.15625, 24)	10:49:00.13	10:50:30.20	1:30.06	0,000	0,000	0,000
☑ [7:91] GetFloatBit(0.15625, 21)	10:49:00.13	10:50:30.20	1:30.06	1,000	1,000	0,000
☑ [7:100] GetFloatBit(0.15625, 20)	10:49:00.13	10:50:30.20	1:30.06	0,000	0,000	0,000

2.8.14   GetIntBit

```
GetIntBit('Expression','BitNo')
```

Description

This function returns the Boolean value of the 'BitNo' bit of 'Expression' as its result after round-ing the 'Expression' to the nearest integer value. The rounding limit is in each case the next 0.5 increment. (2.49 → 2; 2.50 → 3).
Valid bit number sequence: 0 (LSB) to 31 (MSB)

Example

Representation of time in Boolean values

Solution

The figure below shows the GetIntBit function of the first six bits of the time function, T().

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:3] T()	14:46:47.06	14:47:16.92	29,86 s	1,36	31,22	29,86
[7.41] GetIntBit(T(), 0)	14:46:47.06	14:47:16.92	29,86 s	1	1	0
[7.42] GetIntBit(T(), 1)	14:46:47.06	14:47:16.92	29,86 s	0	1	1
[7.43] GetIntBit(T(), 2)	14:46:47.06	14:47:16.92	29,86 s	0	1	1
[7.44] GetIntBit(T(), 3)	14:46:47.06	14:47:16.92	29,86 s	0	1	1
[7.45] GetIntBit(T(), 4)	14:46:47.06	14:47:16.92	29,86 s	0	1	1
[7.46] GetIntBit(T(), 5)	14:46:47.06	14:47:16.92	29,86 s	0	0	0
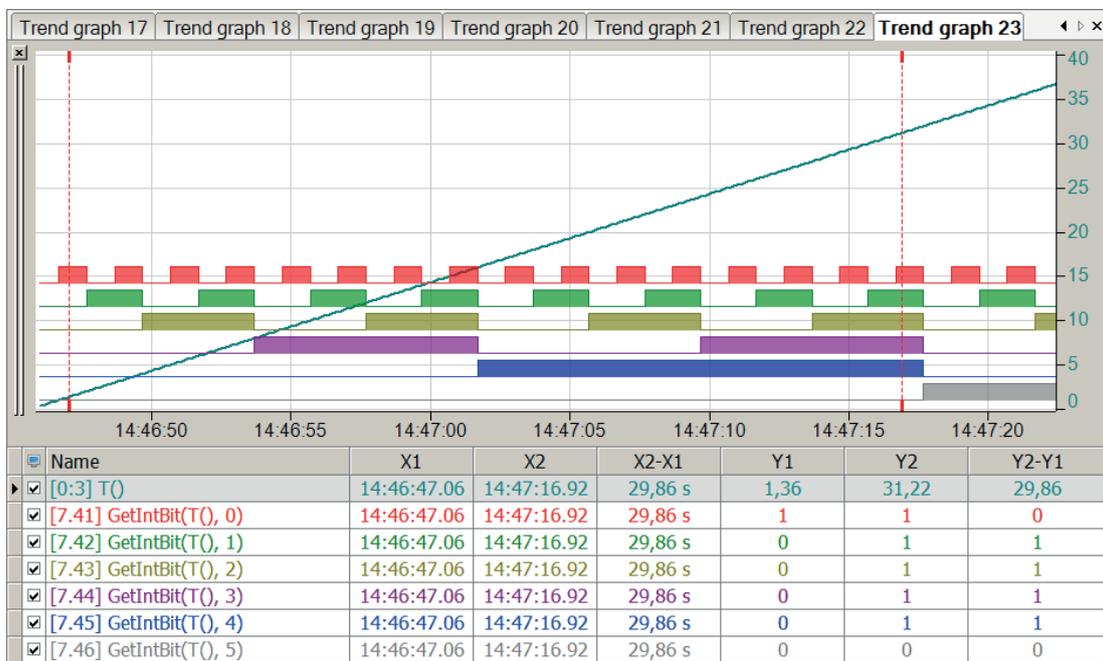
2.8.15   GetSignalMetaData

`GetSignalMetaData('Signal*"','InfoType*')`

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns metadata information of a signal. The parameter 'Signal' can be supplied as signal number ([m:n] resp. [m.n]) or as signal name ([SignalName]).

Specify the information type ('InfoType') you want to receive.

The following information types are supported:

Information types	Possible results
0	Signal name
1	Signal unit
2	Comment 1
3	Comment 2

Table 4: Information types and possible results of the GetSignalMetaData function

`GetSignalMetaData('Signal*"','InfoType*')`

2.8.16   GetSystemTime

```
GetSystemTime('Part*')
```

Arguments

'Part*'	Part to be displayed of the system time specification	
	'Part' = 0	milliseconds
	'Part' = 1	seconds
	'Part' = 2	minutes
	'Part' = 3	hours
	'Part' = 4	day of the month
	'Part' = 5	month
	'Part' = 6	year
	'Part' = 7	day of the year
	'Part' = 8	day of week (Monday=1, Sunday=7)
	'Part' = 9	system time in Unix time format Requires the "High precision" setting of the module to be enabled
	'Part' = 10	UTC offset in seconds

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the part specified under 'Part' of the system time.
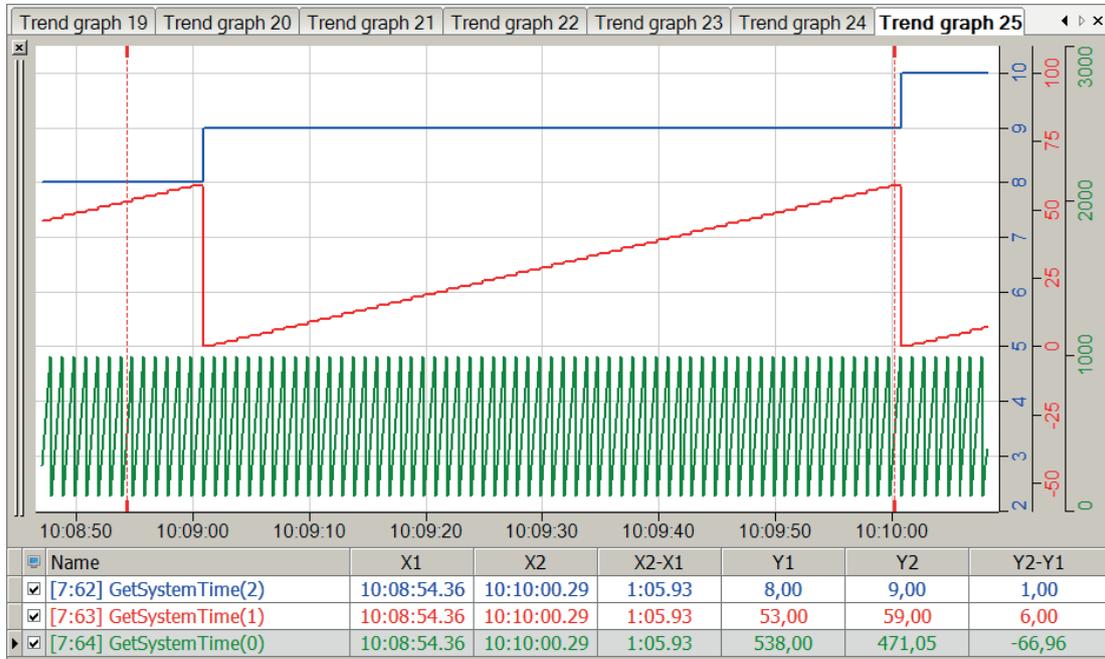

Tip

In *ibaQPanel*, the day of the week (Part 8) can easily be displayed with its name by using a multi-state display. For the other parts, the normal label works fine. Use a gauge to display the time on an analog-like clock in *ibaQPanel*.

Example

Output of the minute part, second part and millisecond part of the system time.

Solution

In the figure below the blue curve shows the minutes of the system time ('part' = 2) and the red curve shows the seconds of the system time ('part' = 1). The green curve shows the milliseconds of the system time ('part' = 0).

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [7:62] GetSystemTime(2)	10:08:54.36	10:10:00.29	1:05.93	8,00	9,00	1,00
☑ [7:63] GetSystemTime(1)	10:08:54.36	10:10:00.29	1:05.93	53,00	59,00	6,00
▶ ☑ [7:64] GetSystemTime(0)	10:08:54.36	10:10:00.29	1:05.93	538,00	471,05	-66,96

2.8.17   GetSystemTimeAsText

```
GetSystemTimeAsText('DateTimeFormat*="yyyy-MM-dd HH:mm:ss"', 'UTC*=0', 'Enable=1')
```

Arguments

'Argument'	Description	
'DateTimeFormat'	yy, yyyy	Year 2 or 4 digits
	M, MM	Month M: 1, 2, 3, 12; MM: 01, 02, 03, ...12
	MMM, MMMM	Name of month
	d, dd	Day d: 1, 2, 3, ...31; dd: 01, 02, 03, ... 31
	ddd, dddd	Day of week
	h, hh,	Hour 12-hour notation
	H, HH	Hour 24-hour notation
	m, mm	Minutes
	s,ss	Seconds
	f, ff, fff	Fractions of a second up to 0.1, 0.01, 0.001 (millisecond), ... precision
	tt	AM/PM indicator; only available if ibaPDA service runs in a culture that supports AM/PM (e.g. US-EN)
'UTC'	0 (default)	Date/time local
	1	UTC date and time
'Enable'	1	Optional argument; if 'Enable' is provided, then date/time will only be emitted when 'Enable' = 1 or true.
		If the argument is omitted, date/time will always be emitted.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the current date and time. Depending on the parameter 'UTC', date and time will be returned as local or UTC date/time.

The calculation can be controlled using the optional parameter, 'Enable.' If you want to take advantage of higher precision by using fractions of a second, you should provide the 'Enable' parameter and set it on 1 or true.

Examples

Examples for Tuesday, August 07, 2023, 17:05 (CEST)

Formula	Result
GetSystemTimeAsText("yyyy-MM-dd HH:mm:ss",0)	2023-08-07 17:05:42
GetSystemTimeAsText("yyyy-MM-dd HH:mm:ss",1)	2023-08-07 15:05:42
GetSystemTimeAsText("yy-MMM-dd hh:mm:ss",1)	23-Aug-07 03:05:42
GetSystemTimeAsText("yyyy-MM-dd HH:mm:ss fff",0)	2023-08-07 17:05:42 478

2.8.18   GetWeekOfYear

`GetWeekOfYear('Rule*=0', 'FirstDayOfWeek*=0')`

Arguments

'Rule*'	Definition rule for first week of the year	
	'Rule' = 0	The first week of the year is the one that includes at least 4 days of that year. (ISO 8601)
	'Rule' = 1	The first week of the year starts on January 1st.
	'Rule' = 2	The first week of the year starts on the first day of the week on or after January 1st.
'FirstDayOf-Week*'	Definition of first day of the week	
	'FirstDayOf-Week'= 0	Monday (ISO 8601)
	'FirstDayOf-Week'= 1	Tuesday
	'FirstDayOf-Week'=2	Wednesday
	'FirstDayOf-Week'=3	Thursday
	'FirstDayOf-Week'= 4	Friday
	'FirstDayOf-Week'= 5	Saturday
	'FirstDayOf-Week'= 6	Sunday

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the number of the current week in the current year. The week numbering is done according to 'Rule'.
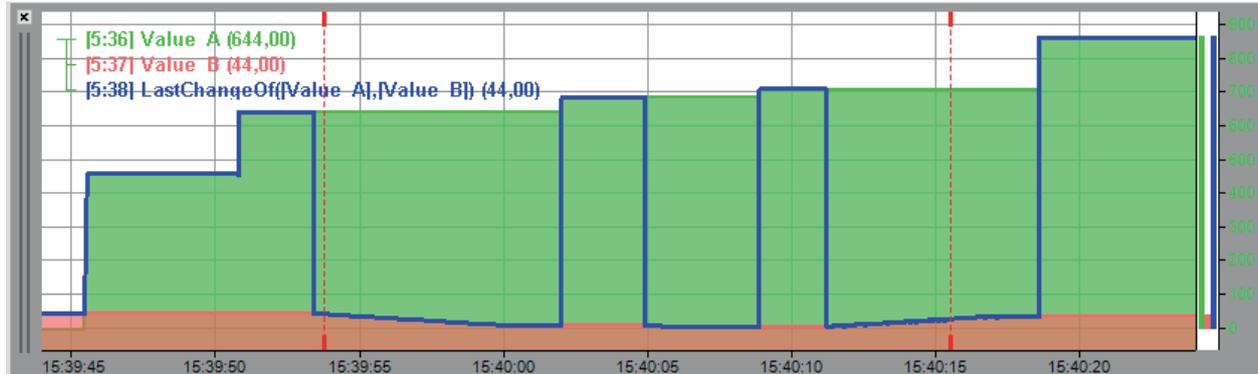
2.8.19   LastChangeOf

```
LastChangeOf('Expression1','Expresion2',...)
```

Description

This function returns the most recently changed value of a set of expressions ('Expression1','Expression2',...). If multiple source expressions change their value simultaneously, the result of this function will be the value of the first expression in the sequence of arguments whose value has changed.

Example

The expressions *Value_A* (green) and *Value_B* (red) are changing independently from each other. The result of *LastChangeOf* with these values follows always the value which changed at last.

2.8.20   LimitAlarm

```
LimitAlarm('Expression','Limit','DeadBand','Time',' Reset=0')
```

Arguments

'Expression'	Measured value	
'Limit'	Limit value, from which the function returns TRUE	
'DeadBand'	Specification of a dead zone below the limit value, within which the function does not reset to FALSE	
'Time'	Specification of the time, for which the measured value must be above the limit value until the function returns TRUE	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset' = 0	Perform calculation
	'Reset' = 1	Stop calculation, reset and set result to 0
	'Reset' = 2	Stop calculation and keep result

Description

This function monitors the measured value ('Expression') and sets the result to TRUE if the measured value is longer than the specified time ('Time') above the ('Limit') limit value. The result of the function becomes FALSE again if the measured value falls below the limit value by the value specified under the ('DeadBand') deadzone.

Tip

The LimitAlarm function can also be used for a lower limit For this purpose, only the measured value and the limit value must be flipped, i.e. multiplied by (-1).

e.g: LimitAlarm([0:1] **(-1)**, 9 **(-1)**, 0.5, 0.4)

Example

The function should return TRUE if the measured value is longer than 0.4 seconds above the value 11. The function should then return FALSE again if the measured value has fallen below 10.5.

Solution

In the figure below the blue curve shows the measured value and the red bar shows when the limit is exceeded. The green band shows the dead zone that prevents a direct FALSE setting of the function.
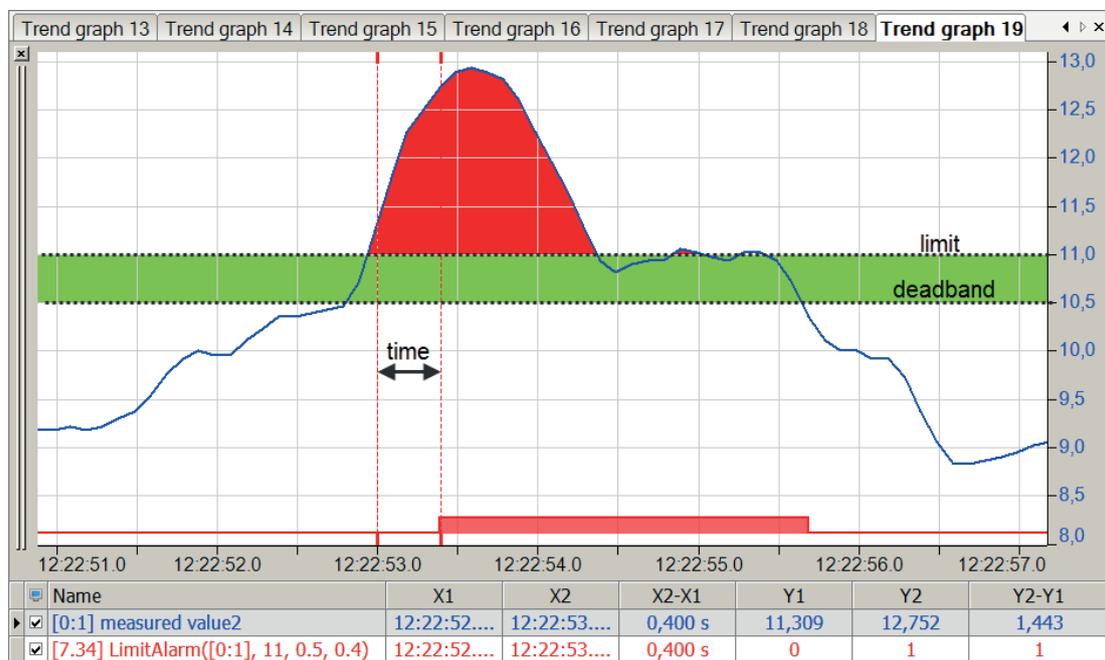
2.8.21   ModuleSignalCount

`ModuleSignalCount('ModuleNo*', 'SignalType*=0', 'Direction*=0')`

Arguments

'SignalType*'	Signal type	
	'SignalType' = 0	Analog and digital signals
	'SignalType' = 1	Analog signals only
	'SignalType' = 2	Digital signals only
'Direction*'	Input or output direction or both	
	'Direction'= 0	Inputs and Outputs
	'Direction'= 1	Inputs only
	'Direction'=2	Outputs only

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the number of active signals in a module with module number 'Module-No'. With the parameter 'SignalType' you can determine, which signal types are to be considered. Signals, which are configured in the module but not checked as *Active*, are not taken into account.

2.8.22   PulseFreq

```
PulseFreq('Expression','Omega=0*','EdgeType=2*' )
```

Arguments

'Expression'	Pulse counter signal	
'Omega*'	Filter frequency	
'EdgeType*'	Edge type to be counted	
	'EdgeType' = -1	Falling edges only
	'EdgeType' = 0	Rising and falling edges
	'EdgeType' = 1	Rising edges only
	'EdgeType' = 2	'Expression' is a pulse counter

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function computes the frequency of 'Expression' pulse counters or pulses. The unit of the result is pulses/sec or Hz.

A low-pass filter with an 'Omega' limit angular speed is applied to the result. If 'Omega' is 0 then the low-pass filter is deactivated. 'EdgeType' determines which edges of pulses should be counted.

Zero is returned as the calculated frequency if no pulse occurs in 1000 samples.

This function was especially created for using the WAGO incremental encoder 750-631. You may use the function to calculate the speed based on the pulse counter signal from the encoder. The pulse counter value is differentiated taking into consideration possible counter overflows. As the result of the differentiation may include interfering frequencies or noise, a low-pass filter can then be used. The filter frequency to be set should be slightly above the maximum pulse frequency.

2.8.23   RestartAcquisition

```
RestartAcquisition('Trigger')
```

Description

The acquisition is restarted when a rising edge on 'Trigger' occurs. The function returns the value 1 when the acquisition is restarted.

2.8.24   SampleAndHold

`SampleAndHold('Expression','Sample','Initial=0)`

Arguments

'Expression'	Measured value
'Sample'	Parameter that determines whether the function follows the measured value (1) or holds the last measured value (0). 'Sample' can be a condition itself or be determined by a different function.
'Initial'	Optional parameter (default = 0), which determines the initial value of the function when 'Sample' is inactive at the start of the measurement.

Description

This function is a sample-hold function. The output follows 'Expression' when 'Sample' = TRUE. It remains unchanged when 'Sample' = FALSE. With the optional 'Initial' parameter, the initial value of the output can be specified if the function is on "Hold" when called.

Example

The function should follow a measured value if it is positive and be held as soon as the measured value becomes negative.

Task description

An If query as the 'Sample' parameter returns the value 1 for positive values and the value 0 for negative values.

Solution

In the figure below the blue curve shows the measured value and the red curve shows the output value with held measured values.
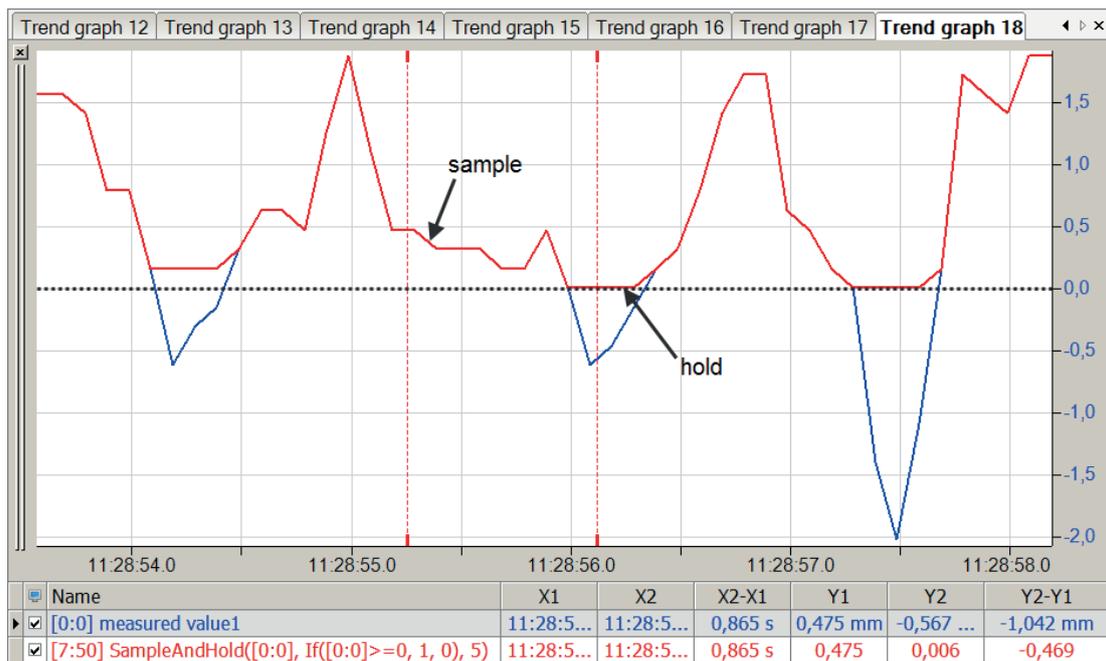
2.8.25   SampleOnce

```
SampleOnce('Expression','Sample')
```

Arguments

'Expression'	Measured value
'Sample'	Parameter which determines when the function should follow the measured value 'Expression' for exactly one sample. 'Sample' can be a condition itself or be determined by a different function.

Description

This function is a sample-once-function. The result of the function follows 'Expression' for the duration of one sample when a rising edge occurs on 'Sample'. For all other timestamps the result is NaN if 'Expression' is a numeric signal and an empty string if 'Expression' is a text signal.

2.8.26   Sign

```
Sign('Expression')
```

Description

This function returns the sign of 'Expression' as its result.

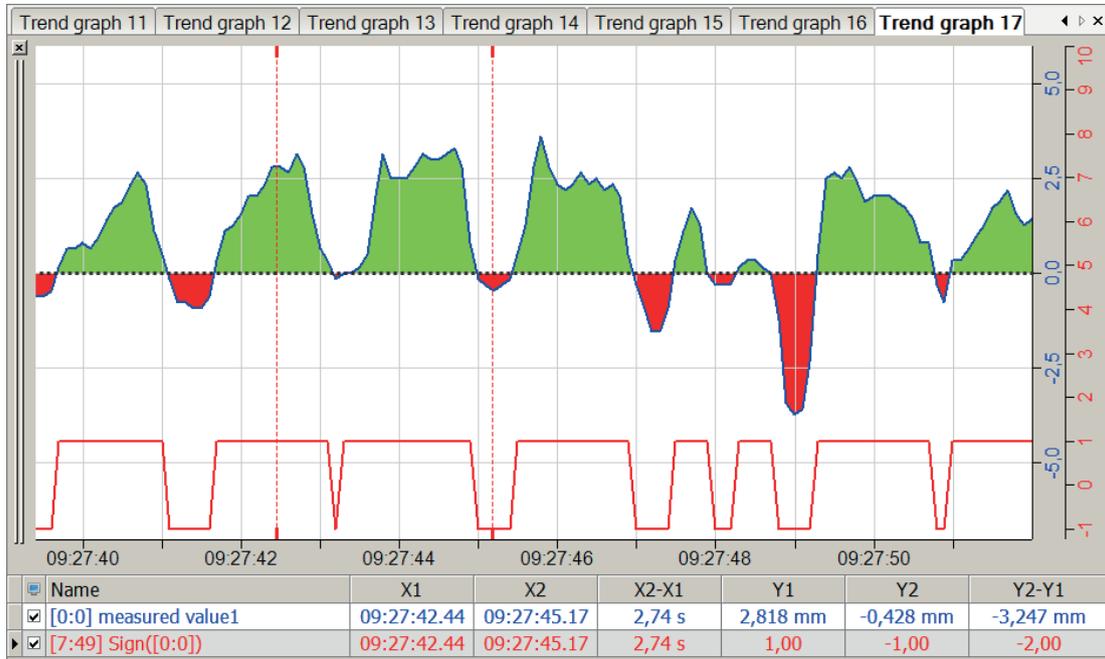'Expression' > 0 --> +1

'Expression' = 0 --> 0

'Expression' < 0 --> -1

Example

Only the sign from a measured value is relevant.

Solution

In the figure below the blue curve shows the measured value and the red curve shows the sign of the measured value.

2.8.27   T

`T()`

Description

This function returns the time elapsed since the start of measuring (in seconds).

Using the time function, time-dependent virtual variables, e.g. sine waves, can be calculated.

Example of a sine wave of 0.5 Hz: sin(2*PI()*0.5*T())

Tip

The time function can be entered In the expression editor simply by double clicking in other expressions, as is the case with a signal. The two brackets behind T are required for syntax reasons, as in the case of the number Pi.

Example

Presenting the time function and a time-dependent, virtual variable

Solution

Calculation of the time function and a time-depending sine curve
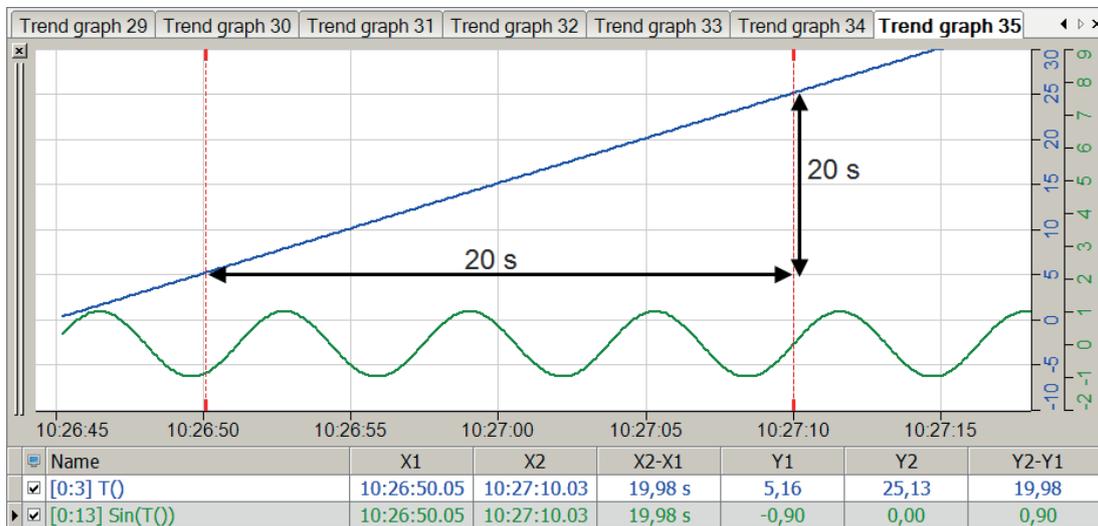
2.8.28   VarDelay

```
VarDelay('Expression','Delay', 'MaxDelay=30*')
```

Arguments

'Expression'	input signal
'Delay'	Time delay in seconds
'MaxDelay'*	Optional parameter (default = 30) to determine the maximum delay permitted in seconds

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function delays the 'Expression' signal by the 'Delay' time. In contrast to the Delay function, the delay time may change over time. *'MaxDelay'* specifies the maximum delay permitted and is preset to 30 s by default.
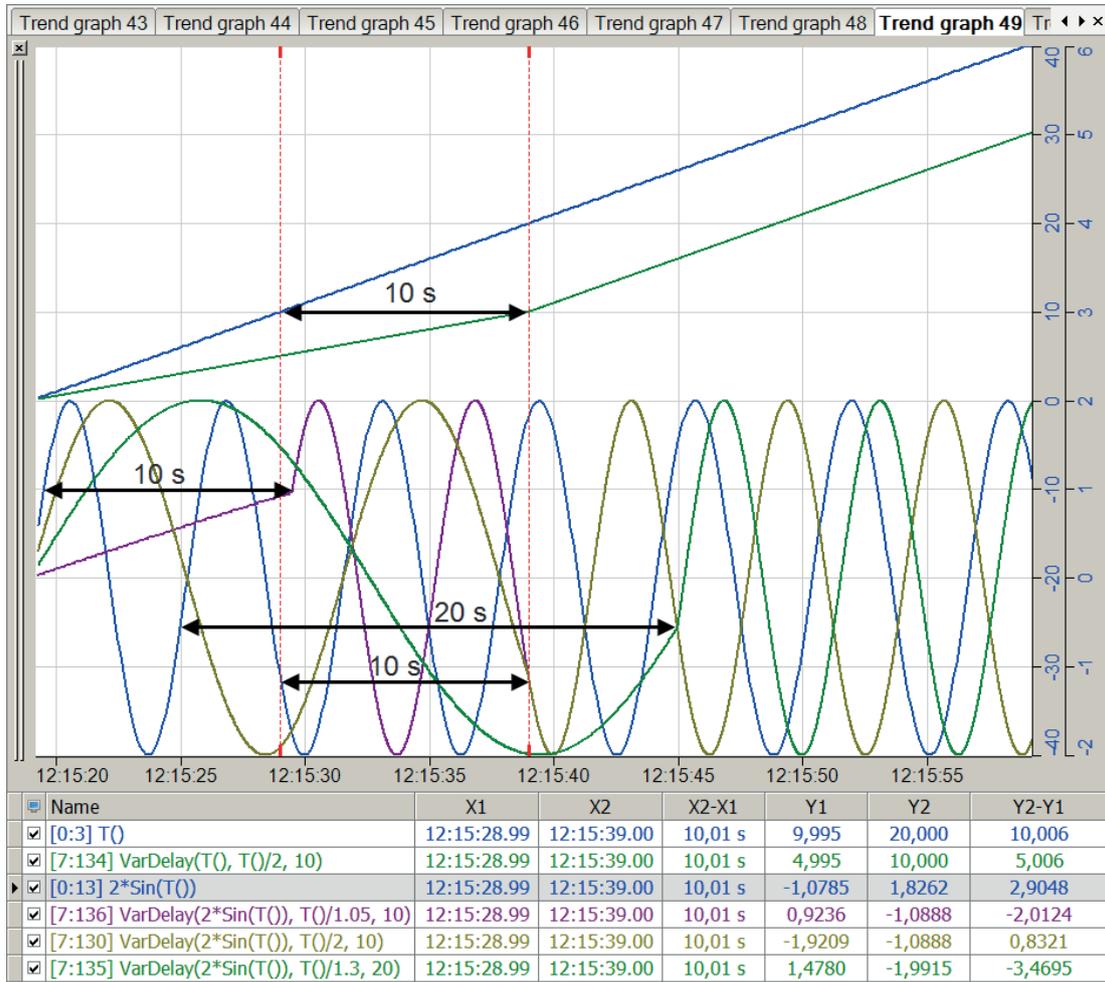
Example

Temporally variable delay of a signal reaching the maximum permitted delay

Solution

The time function T() is used as the variable delay. This and a sine signal are set to this variable delay and the resulting curves are recorded.

As soon as the maximum delay is reached, the delay remains constant whereas the resulting curves only represent the shifted output signals.

In the figure below the top part shows the delay of the time signal by a variable shift with a maximum value of 10 seconds. The bottom part shows the delay of a sinusoidal function with three different delay and maximum delay values.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
☑ [0:3] T()	12:15:28.99	12:15:39.00	10,01 s	9,995	20,000	10,006
☑ [7:134] VarDelay(T(), T()/2, 10)	12:15:28.99	12:15:39.00	10,01 s	4,995	10,000	5,006
☑ [0:13] 2*Sin(T())	12:15:28.99	12:15:39.00	10,01 s	-1,0785	1,8262	2,9048
☑ [7:136] VarDelay(2*Sin(T()), T()/1.05, 10)	12:15:28.99	12:15:39.00	10,01 s	0,9236	-1,0888	-2,0124
☑ [7:130] VarDelay(2*Sin(T()), T()/2, 10)	12:15:28.99	12:15:39.00	10,01 s	-1,9209	-1,0888	0,8321
☑ [7:135] VarDelay(2*Sin(T()), T()/1.3, 20)	12:15:28.99	12:15:39.00	10,01 s	1,4780	-1,9915	-3,4695

Tip

Negative values for 'Delay' do not produce an error message but are treated as 0, i.e. there is no delay.

2.8.29   WindowAlarm

```
WindowAlarm('Expression','Limit1','DeadBand1','Limit2','DeadBand2','Time',' Re-
set=0')
```

Arguments

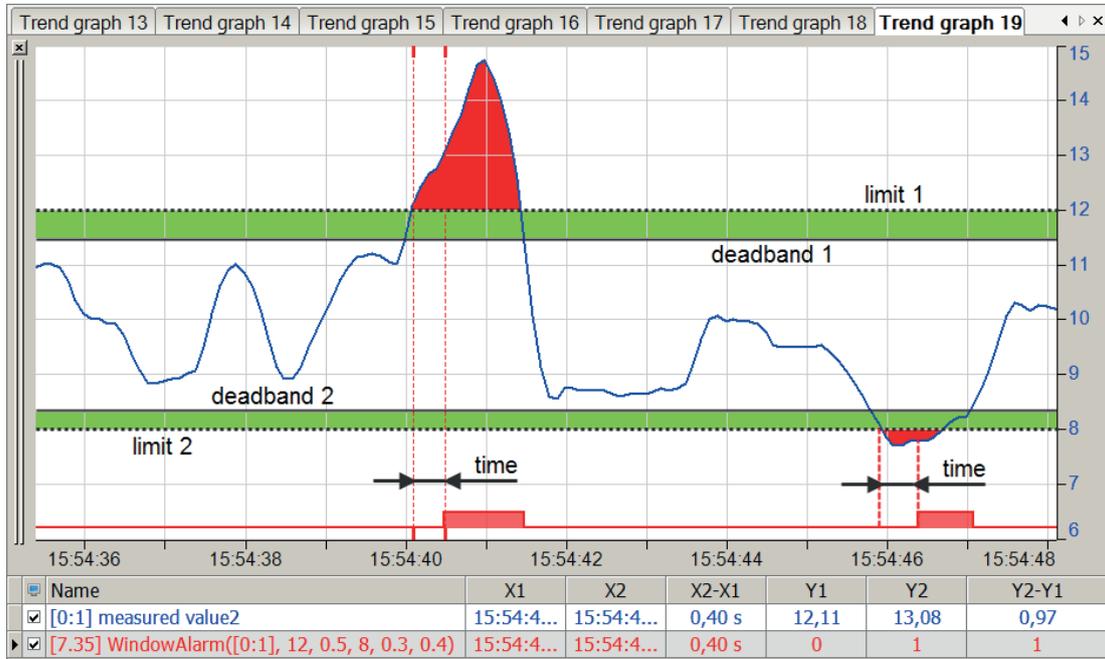| 'Expression'	Measured value	
'Limit1'	Upper limit value, from which the function returns TRUE	
'DeadBand1'	Specification of the dead zone below the upper limit value ('Limit1'), within which the function does not reset to FALSE	
'Limit2'	Lower limit value, from which the function returns TRUE	
'DeadBand2'	Specification of the dead zone above the lower limit value ('Limit2'), within which the function does not reset to FALSE	
'Time'	Specification of the time, for which the measured value must be greater than the upper limit or smaller than the lower limit until the function returns TRUE	
'Reset'	Optional parameter (default = 0) to stop and restart the calculation	
	'Reset'=0	Perform calculation
	'Reset'=1	Stop calculation, reset and set result to 0
	'Reset'=2	Stop calculation and keep result

Description

This function monitors the measured value ('Expression') and sets the result to TRUE if the measured value is longer than the specified time ('Time') outside the range between the upper limit value ('Limit1') and the lower limit value ('Limit2'). The result of the function becomes FALSE again if the measured value falls below the upper limit by the value specified under 'Dead-Band1', or exceeds the lower limit by the value specified under 'DeadBand2'.

Example

The function is to be triggered if the measured value is outside the value range between 8 and 12 for longer than 0.4 s. The upper dead zone should be 0.5, the lower dead zone 0.3.

Solution

In the figure below the blue curve shows the measured value and the red bar shows the triggers the WindowAlarm function. The green areas show the dead zones of the respective limits.

Name	X1	X2	X2-X1	Y1	Y2	Y2-Y1
[0:1] measured value2	15:54:4...	15:54:4...	0,40 s	12,11	13,08	0,97
[7.35] WindowAlarm([0:1], 12, 0.5, 8, 0.3, 0.4)	15:54:4...	15:54:4...	0,40 s	0	1	1

2.9   Diagnosis functions

2.9.1   CameraStatus

`CameraStatus('ModuleNo*','SignalNo*','Timeout=2*')`

Arguments

'ModuleNo*'	Module number of the ibaCapture server in the signal tree	
'SignalNo*'	Signal number (camera)	
'Timeout*'	Time in seconds in which the sync signal must change; default = 2	

Parameters ending with * are only evaluated once at the start of the acquisition..

Description

This function returns the status of an ibaCapture camera.

Results

0	Camera not ok
1	Camera OK

A camera is not ok when the value of the sync signal of the camera does not change within 'Timeout' seconds. Both the recording of this camera on the *ibaCapture* server as well as the synchronization between *ibaCapture* server and *ibaPDA* server must be running in order to facilitate the sync signal to change.

'ModuleNumber' is the module number of the ibaCapture server in the signal tree.

Example:

Status of "Camera1" with Timeout = 2 s: CameraStatus(18, 0, 2)

2.9.2   DataStoreInfo

`DataStoreInfo('DatastoreIndex*','InfoType*')`

Parameters ending with * are only evaluated once at the start of the acquisition..

Description

This function provides information about the selected data storage. This information may be used for controlling other functions or for display and diagnostic purposes.

For normal (PDA) data storage, use 'DatastoreIndex' >= 0.

For *ibaQDR* data storage, use 'DatastoreIndex' < 0.

The index can easily be obtained by looking at the tree structure in the configuration dialog of the data record. Index increases top-down.

Specify the information type ('InfoType') you want to receive.

The following information types are supported:

Information types	Possible results
0: recording status	0 = stopped
	1 = waiting for trigger
	2 = recording
	3 = posttrigger recording
1: Saving in the backup directory:	0 = base directory is used
	1 = backup directory is used
2: recorded time in the current file expressed in seconds	Value is updated every second.
3: the free space on the current hard disk expressed in MB	Value is updated every minute.
4: is ibaQDR synchronized?	0 = ibaQDR is NOT synchronized
	1 = ibaQDR is synchronized
5: Image triggers storing to backup directory	-1=No active or configured image triggers
	0=All image triggers are using the base directory
	1=All image triggers are using the backup directory
	2=Some image triggers are using the base directory and some are using the backup directory
6: Number of overlapping data files	Actual value

Table 5: Information types and possible results of the DataStoreInfo function

2.9.3   DataStoreInfoDB, ...Influx, ...Kafka, ...MindSphere, ...MQTT

```
DataStoreInfoDB('DatastoreIndex*','InfoType*')
```

```
DataStoreInfoInflux('DatastoreIndex*','InfoType*')
```

```
DataStoreInfoKafka('DatastoreIndex*','InfoType*')
```

```
DataStoreInfoMindSphere('DatastoreIndex*','InfoType*')
```

```
DataStoreInfoMQTT('DatastoreIndex*','InfoType*')
```

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function provides information about the selected database or cloud data stores. This information may be used for controlling other functions or for display and diagnostic purposes.

Identify the desired DB/Cloud data storage with the 'DatastoreIndex*' > = 0.

The index can easily be obtained by looking at the tree structure in the configuration dialog of the Data Storage. Index increases top-down.

Specify the information type ('InfoType') you want to receive.

The following information types are supported:

Information types	Possible results
0: Recording status	0 = stopped
	1 = waiting for trigger
	2 = recording
	3 = posttrigger recording
1: Data throughput	Value in kB/s
2: Is server connected?	0/False = no
	1/True = yes
3: Recorded time since last start trigger	Value in seconds; this will be constant 0 for continuous recording.
5: Current buffer usage	Value in %
6: Current file buffer usage	Value in %
7: Unprocessed bytes in file buffer	Value in %

Table 6: Information types and possible results of the DataStoreInfoDB function

2.9.4   DataStoreInfoHD

```
DataStoreInfoHD('DatastoreIndex*','InfoType*')
```

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

Note

| i | With ibaPDA v8.4 his function has been subject to a breaking change. In order to support all recording states of the triggered HD recording, the DataStoreInfoHD diagnostic function now uses the same values for the status (info type 0) as the other DataStoreInfo functions. If you have already used this function, please check your evaluation of the status values. |

This function provides information about the selected HD records. This information may be used for controlling other functions or for display and diagnostic purposes.

Identify the desired HD data records with the 'DatastoreIndex*' > = 0.

The index can easily be obtained by looking at the tree structure in the configuration dialog of the data record. Index increases top-down.

Specify the information type ('InfoType') you want to receive.

The following information types are supported:

Information types	Possible results
0: Recording status	0 = Stopped
	1 = Waiting for trigger
	2 = Recording
	3 = Post trigger recording
1: Data throughput	Value in KB/s
2: Is server connected?	0/False = no
	1/True = yes
5: Current buffer usage	Value in %
6: Current file buffer usage	Value in %
7: Unprocessed bytes in file buffer	Value in %

Table 7: Information types and possible results of the DataStoreInfoHD function

2.9.5   DongleInfo

`DongleInfo('InfoType*')`

Parameters ending with * are only evaluated once at the start of the acquisition.

Note

This function is still supported for backward compatibility purposes. Please use instead the *LicenseInfo* function, which serves the same purposes while supporting soft licenses as well.

Description

This function provides information about various properties related to the dongle. This information can be used for display and diagnostic purposes.

Specify the information type ('InfoType') you want to receive. This type of information is determined and output at the start of the measurement. If you want to receive more information from the dongle then you need to configure the function several times, with a different type of information each time.

The following information types are supported:

Information types	Possible results
0: Dongle available	TRUE = dongle available
	FALSE = no dongle or dongle defective
1: Dongle time limit in days	Value of the remaining lifetime of the dongle
2: Demo time limit in days	Value of the remaining lifetime of the dongle for trial periods / demo versions
3: ibaQDR acquisition time limit in seconds	Value of the remaining time which continues to run the ibaQDR system after the dongle has been removed.
10: Dongle inserted - counter	Number of times the dongle was inserted
11: Dongle removed - counter	Number of times the dongle was removed
12: Dongle changed - counter	Number of times the dongles were changed

Table 8: Information types and possible results of the DongleInfo function

---

Note

By evaluating information type 0, you can monitor whether a license container is present or available. In practice, you can use this function in *ibaQDR* to generate an email when it is detected that the license is missing (e.g., dongle removed). With *ibaQDR*, data collection continues for up to 4 days without a license to give you the opportunity to procure a new license container without jeopardizing production. This grace period does not exist with *ibaPDA*.

2.9.6   FobDLinkStatus

`FobDLinkStatus('BoardNo*','LinkNo*')`

Arguments

'BoardNo*'	Board number (0 to 7)	
'LinkNo*'	Link number (refer to the signal tree in the I/O Manager)	

Parameters ending with * are only evaluated once at the start of the acquisition..

Description

This function returns the status of an ibaFOB-D board link.

The card number (from 0 to 7) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The link number can be obtained from the signal tree in the I/O Manager.

Results

0	Link not active
1	Link OK
2	Link interrupted
3	Link RX ok, but flex ring interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

When using a bus monitor (such as ibaBM-DPM-S), at least one active slave must be configured in the I/O Manager.

2.9.7   FobFastLinkStatus

`FobFastLinkStatus('BoardNo*','LinkNo*','Filtered*')`

Arguments

'BoardNo*'	Board number (0 to 3)	
'LinkNo*'	Link number (refer to the signal tree in the I/O Manager)	
'Filtered*'	Filter settings (TRUE or FALSE)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the status of a link of an ibaFOB-X card in 32-bit mode (32 Mbit/s).

The card number (from 0 to 3) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The link number can be obtained from the signal tree in the I/O Manager. 'Filtered' can be TRUE or FALSE. The filtered link status ignores any change in link status that is faster than 40 ms.

Results

0	Link not active
1	Link OK
2	Link interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

When using a bus monitor (such as ibaBM-DPM-S), at least one active slave must be configured in the I/O Manager.

2.9.8   FobFlexDeviceStatus

`FobFlexDeviceStatus('BoardNo*','LinkNo*','Address*')`

Arguments

'BoardNo*'	Board number (0 to 7)	
'LinkNo*'	Link number (refer to the signal tree in the I/O Manager)	
'Address*'	Address of the device in the Flex ring (1 to 15)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the status of the Flex device with the 'Address' address at the 'LinkNo' link of a 'BoardNo' ibaFOB-D card.

The card number (from 0 to 7) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The link number can be obtained from the signal tree in the I/O Manager.

The device address can be set on the rotary switch of the device. Address values 1 to 15 are possible.

Results

0	Device not configured
1	Device OK
2	Device not connected

2.9.9   FobFLinkStatus

`FobFLinkStatus('BoardNo*','LinkNo*')`

Arguments

'BoardNo*'	Board number (0 to 3)	
'LinkNo*'	Link number (refer to the signal tree in the I/O Manager)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the status of a link of an FOB-S or an FOB-X card in 3-Mbit mode (3.3 Mbit/s).

The card number (from 0 to 3) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The link number can be obtained from the signal tree in the I/O Manager.

Results

0	Link not active
1	Link OK
2	Link interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

When using a bus monitor (such as ibaBM-DPM-S), at least one active slave must be configured in the I/O Manager.

2.9.10   FobMLinkStatus

`FobMLinkStatus('BoardNo*','LinkNo*')`

Arguments

'BoardNo*'	Board number (0 to 3)	
'LinkNo*'	Link number (refer to the signal tree in the I/O Manager)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the status of a link of an FOB-S card running in FOB-M mode (5 Mbit/s).

The card number (from 0 to 3) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The link number can be obtained from the signal tree in the I/O Manager.

Results

0	Link not active
1	Link OK
2	Link interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

2.9.11   FobPlusControlLinkStatus

```
FobPlusControlLinkStatus('BoardNo*')
```

Arguments

'BoardNo*'	Board number (0 to 3)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function provides the link status of an ibaFOB-PlusControl board.

The card number (from 0 to 3) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The board only has one link.

Results

0	Link not active
1	Link OK
2	Link interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

2.9.12   FobSDLinkStatus, FobSDexpLinkStatus

```
FobSDLinkStatus('BoardNo*')
```
```
FobSDexpLinkStatus('BoardNo*')
```

Arguments

'BoardNo*'	Board number (0 to 3)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the link status of an ibaFOB-SD board (PCI) or ibaFOB-SDexp board (PCIe).

The card number (from 0 to 3) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The board only has one link.

Results

0	Link not active
1	Link OK
2	Link interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

2.9.13   FobTDCLinkStatus, FobTDCexpLinkStatus

```
FobTDCLinkStatus('BoardNo*')
```

```
FobTDCexpLinkStatus('BoardNo*')
```

Arguments

'BoardNo*'	Board number (0 to 3)	

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the link status of an ibaFOB-TDC board (PCI) or ibaFOB-TDCexp board (PCIe).

The board number (from 0 to 3) can be obtained from the 7-digit display on the card or the graphical presentation in the I/O Manager. The board only has one link.

Results

0	Link not active
1	Link OK
2	Link interrupted

Prerequisites

This function only returns a value if active input modules are installed and configured in the I/O Manager.

2.9.14   ICPSensorStatus

`ICPSensorStatus('ModuleNo*','SensorNo*')`

Arguments

'Module-No*'	Module number of a Padu-8-ICP module
'SensorNo*'	Sensor number (between 0 and 7)

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function monitors the status of an ICP sensor. The first 'ModuleNo' parameter indicates the module number of an ibaPADU-8-ICP or ibaMS8xICP-/ IEPE module in the signal tree. The second parameter, 'SensorNo', specifies which sensor should be monitored. The sensor number goes from 0 to 7.

Results

0	Sensor OK
1	open loop detected

2.9.15   InterruptCycleTime

`InterruptCycleTime('Type=0*')`

Arguments

'Type*'	Optional parameter (default = 0), with which the type of the result value can be determined.	
	'Type' = 0	The actual value is returned.
	'Type' = 1	The minimum (smallest measured cycle period) is returned.
	'Type' = 2	The maximum (largest measured cycle period) is returned.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the Interrupt cycle time in microseconds. The value is not updated with each Interrupt, but with a lower cycle.

2.9.16   InterruptTime

```
InterruptTime('Type=0*')
```

Arguments

'Type*'	Optional parameter (default = 0), with which the type of the result value can be determined.	
	'Type' = 0	The actual value is returned.
	'Type' = 1	The minimum (smallest duration measured) is returned.
	'Type' = 2	The maximum (largest duration measured) is returned.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the Interrupt time (Interrupt duration) in microseconds. The value is not updated with each Interrupt, but with a lower cycle.

2.9.17   IsProjectModified

`IsProjectModified()`

Description

This function returns the result true, if there are unsaved modifications in the current project.

2.9.18   LicenseInfo

```
LicenseInfo('InfoType*')
```

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function provides information about various properties related to the license container (dongle or soft license). This information can be used for display and diagnostic purposes.

Specify the information type ('InfoType') you want to receive. This type of information is determined and output at the start of the acquisition. If you want to receive more information from the license container then you need to configure the function several times, with a different type of information each time.

The following information types are supported:

Information types	Possible results
0: License container available	TRUE = license container available
	FALSE = license container missing or defective
1: License time limit in days	Value of the remaining lifetime of the license
2: Demo time limit in days	Value of the remaining lifetime of the license for trial periods / demo versions
3: Time limit in seconds before the acquisition will be stopped because of a license removal	Value of the remaining time which continues to run the system after the license container has been removed.
10: License container inserted counter	Number of times the license container was inserted
11: License container removed counter	Number of times the license container was removed
12: License container changed counter	Number of times the license container were changed

Table 9: Information types and possible results of the LicenseInfo function

Note

By evaluating information type 0, you can monitor whether a license container is present or available. In practice, you can use this function in *ibaQDR* to generate an email when it is detected that the license is missing (e.g., dongle removed). With *ibaQDR*, data collection continues for up to 4 days without a license to give you the opportunity to procure a new license container without jeopardizing production. This grace period does not exist with *ibaPDA*.

2.9.19   MultiStationStatus

`MultiStationStatus()`

Description

Returns the current multistation mode.

Results

0	Standalone
1	Slave
2	Master
3	Unsynchronized station

2.9.20   PerformanceCounter

```
PerformanceCounter("'Category*'","'CounterName*'","'InstanceName*'")
```

Arguments

'Category*'	Text entry from the "Object" column of the Windows Performance Monitor
'Counter-Name*'	Text entry from the "Performance Indicator" column of the Windows Performance Monitor
'Instance-Name*'	Text entry from the "Instance" column of the Windows Performance Monitor

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The PerformanceCounter function can read, display and record the progress of certain performance characteristics of the computer. It returns the value of the performance counter or 0 if the performance counter does not exist. With the PerformanceCounter function, one performance indicator can be read in each case. If you want to display multiple performance indicators, then you have to configure the function repeatedly.

Example

Monitoring system performance

Solution

In order to select the performance characteristics that should be monitored under Windows 7, click on the Windows icon and search for Computer Management. Then on the left side under *System – Performance – Monitoring tools* you need to select the item *Performance monitoring*. You can add performance characteristics to be monitored with the green +- symbol in the top bar.
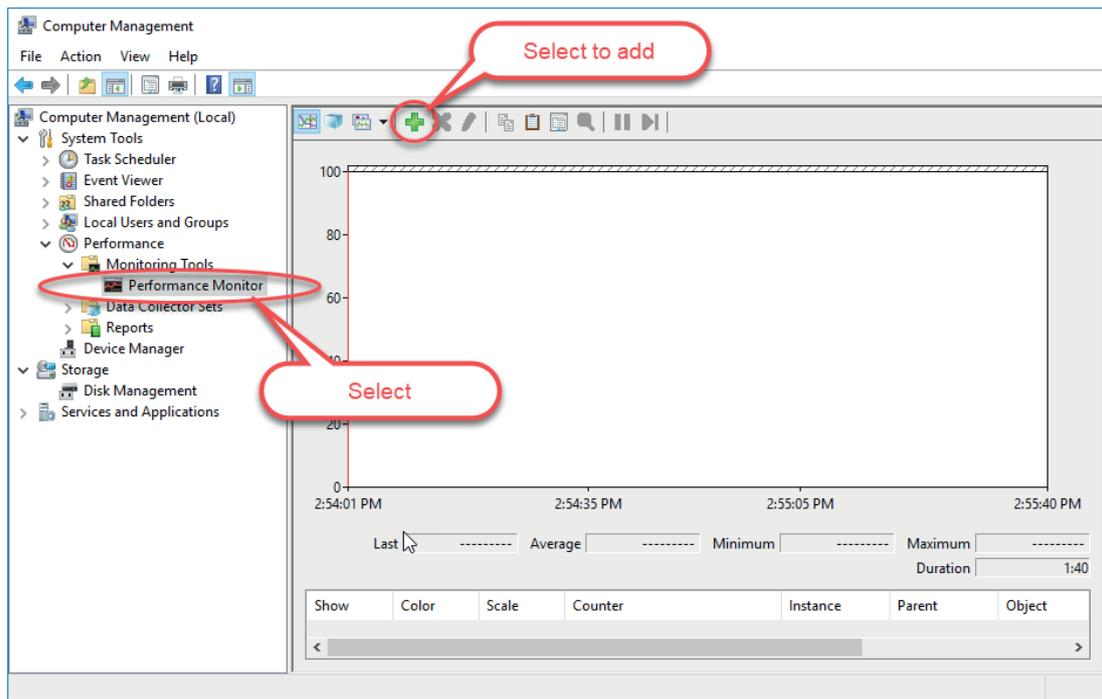
Fig. 2: Show new performance characteristics to be monitored

Select the performance indicators and then the instances. In this case, process, ibaPDA, and all instances are selected, then added and confirmed with <OK>. In the following illustration, the steps are marked and numbered.
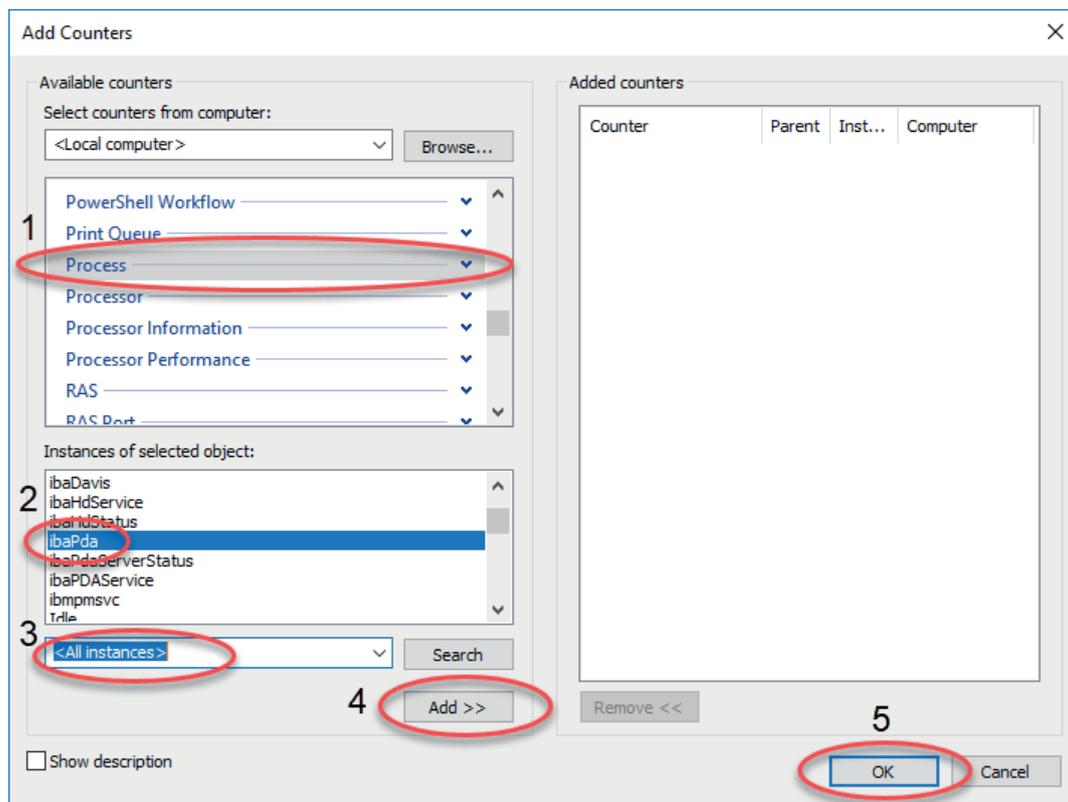

Fig. 3: Selecting the function arguments

The allocation to individual function arguments by PerformanceCounter is illustrated in the next illustration. You must accurately enter the names of the individual performance indicators, objects and instances for the entry in ibaPDA.
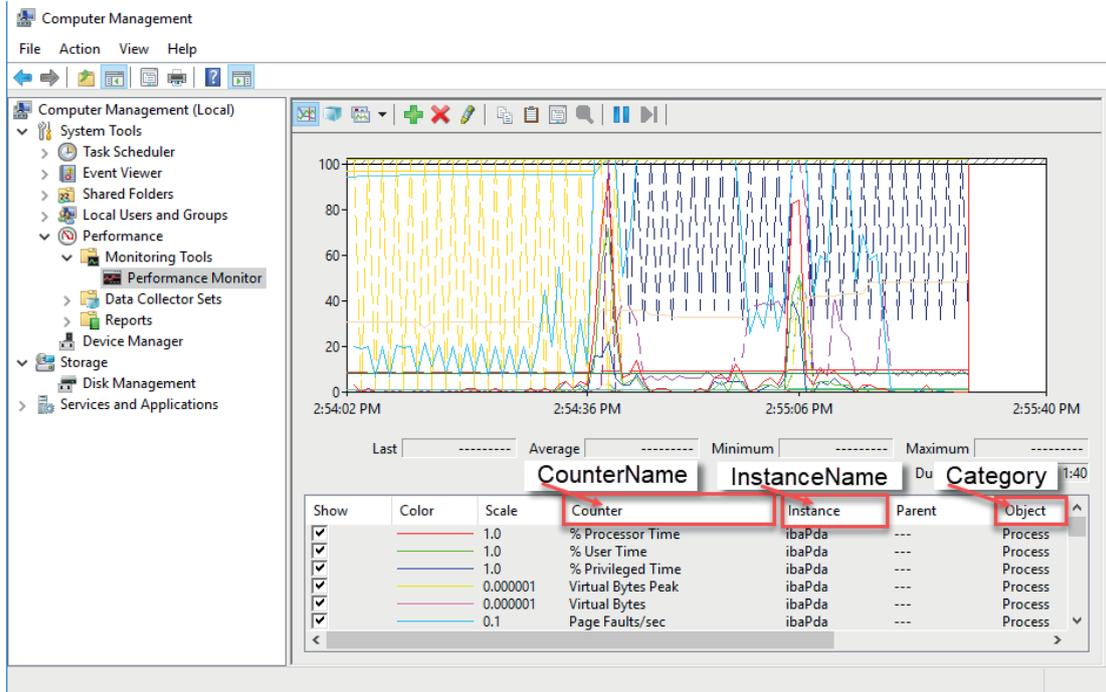

Fig. 4: Name of the function arguments

Example for the entry in ibaPDA for determining the processor time:

2.9.21   Ping

```
Ping("'Address'", 'Trigger', ' Timeout=5', ' Size=32')
```

Arguments

'Address'	IP address of the target computer
'Trigger'	Trigger signal (rising edge), with which ping is to be sent
'Timeout'	Optional parameter (default = 5) for setting the period to wait for a response before "no response" is shown as a result.
'Size'	Optional parameter: Size, in bytes, of the ping request (default = 32)

Description

This function sends a ping request to 'Address' upon a rising edge of 'Trigger'. The function returns the time, in milliseconds, needed before the ping response was received. If there is no response within 'Timeout' seconds then -1 is issued. The result is 0 as long as no ping request was sent. 'Size' determines the size of the ping request in bytes.

The function returns an analog value as the result.

Results

-1	no response to ping request within 'Timeout'
0	no ping request sent
n	Response time in ms

2.9.22   TimeSinceLastSync

```
TimeSinceLastSync()
```

Description

This function returns the time elapsed in seconds since the last time synchronization. If there was no time synchronization, the function returns -1.

Tip

Time synchronization is set in the configuration menu of ibaPDA then in the I/O Manager under the General node. Possible sources for the time synchronization are DCF77, IEC1131, PTP and HPCi via DGM200P or there is no time synchronization set.

2.9.23   TimeSyncStatus

```
TimeSyncStatus('Source*')
```

Arguments

'Source*'	Source for time synchronization	
	'Source' = -1	last source used for time synchronization
	'Source' = 0	DCF77 source 1
	'Source' = 1	DCF77 source 2
	'Source' = 2	IEC1131
	'Source' = 3	DGM200P
	'Source' = 4	PTP Slave
	'Source' = 5	ibaClock

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

This function returns the status of the selected time synchronization source.

The result can be as follows:

Results

0	Source is inactive
1	Source is active and valid
2	Source is active but not valid

Tip

Time synchronization is set in the configuration menu of *ibaPDA*, then in the I/O Manager under the General node.

2.10   Filter functions

2.10.1   BP

```
BP('Expression','Frenquency1*','Frequency2*')
```

Arguments

'Expression'	Measured value
'Frequency1*'	Specification of the lower limit of the frequency band for the filter
'Frequency2*'	Specification of the upper limit of the frequency band for the filter

Parameters ending with * are only evaluated once at the start of the acquisition..

Description

The function is a bandpass filter with a frequency band between the frequencies, 'Frequency1' and 'Frequency2'. The filter is a second-order Butterworth filter.

2.10.2   HP

```
HP('Expression','Frenquency*')
```

Arguments

'Expression'	Measured value
'Frequency*'	Specification of the limit frequency of the filter

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The function is a high-pass filter with the 'Frequency' limit frequency. The filter is a second-order Butterworth filter.

2.10.3   LP

```
LP('Expression','Omega*','Reset=0')
```

Arguments

'Expression'	Measured value	
'Omega*'	Specification of the angular frequency of the filter, which results in the frequency of the LP function to frequency = 'Omega'/(2*PI)	
'Reset'	Optional digital parameter that can be used to deactivate the function. 'Reset' can be an expression as well.	
	'Reset' > 0	Filter is deactivated, input signal is displayed unfiltered
	'Reset' = 0	Filter is applied

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The function is a low-pass filter with frequency = 'Omega'/(2*PI) applied to 'Expression'. The filter is a single-pole filter with a roll-off rate of 20 dB/decade. If the optional 'Reset' parameter is TRUE then the filter is deactivated and the result is the unfiltered 'Expression' input signal. 'Reset' can also be formulated as an expression.

Examples:

| LP([0:0],10)	Filter is active ('Reset' omitted).
LP([0:0],10,If([0:0]<0,1,0))	The filter function is only used for positive values.
LP([0:0],10,[3.1])	e.g. with [3.1] = If([0:0]>10, 1, 0)
	The filter is deactivated as soon as the expression [3.1] returns TRUE, i.e. if the expression [0:0] exceeds the limit value 10.

Example

Applying the filter function to a sine wave with 10 Hz, which is overlaid by a further wave with 50 Hz.

Task description

The filter function should be applied with the values 5, 10 and 40 for 'Omega'.

Solution
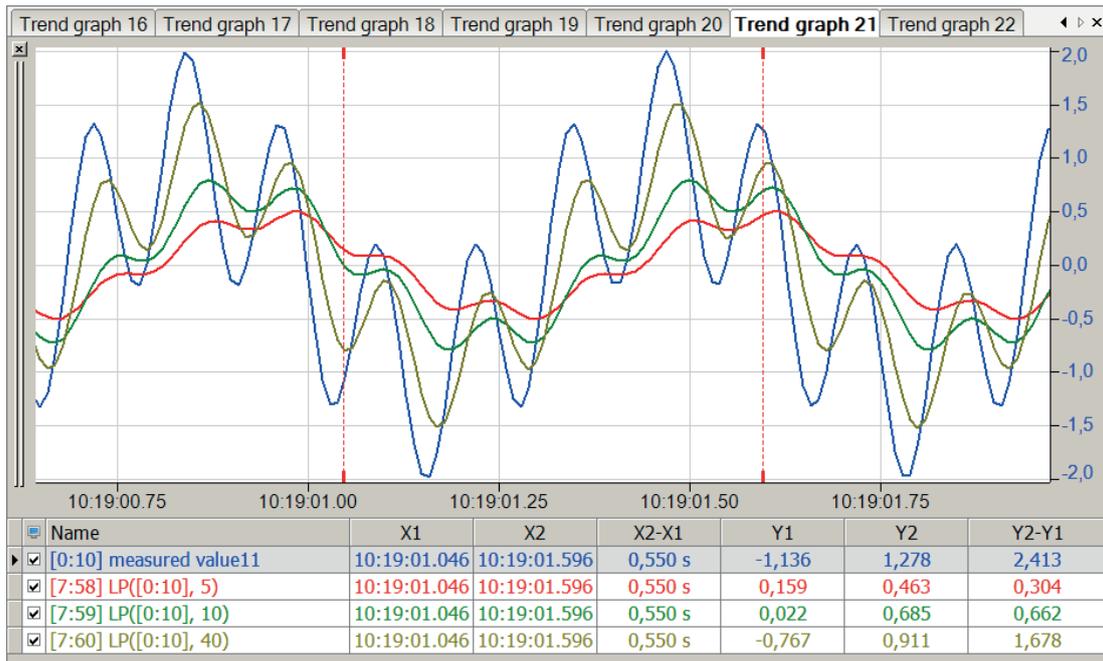
Blue	Original signal of a superimposed sine wave	Red	Filtered signal with 'Omega' = 5
Green	Filtered signal with 'Omega' = 10	Yellow	Filtered signal with 'Omega' = 40

The lower the value selected for 'Omega', the more the signal is attenuated.

2.10.4   EnvelopeSpectral

`EnvelopeSpectral('Expression','Frenquency1*','Frequency2*',' Cutoff frequency')`

Arguments

'Expression'	Measured value
'Frequency1*'	Specification of the lower limit of the frequency band for the envelope calculation Value may not be less than 10% of the Nyquist frequency.
'Frequency2*'	Specification of the upper limit of the frequency band for the envelope calculation Value may not be greater than 90% of the Nyquist frequency.
'Cutoff frequency'	Optional specification of a limit frequency for the configuration of a low-pass filter which will be applied to the envelope.

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The function calculates the spectral envelope of 'Expression' with a constant band pass between the 'Frequency1' and 'Frequency2' frequencies. The smallest bandwidth is 10% of the Nyquist frequency, i.e., 10% of half the sampling frequency.

The 'Expression' signal is rescanned at the time base of the virtual signal. No anti-aliasing filter is applied.

2.10.5   Preprocess

`Preprocess('Expression', "'Preprocess profile name*'")`

Arguments

'Expression'	Expression that should be preprocessed
'Preprocess profile name**'	Name of the preprocess profile or pre-processor

Parameters ending with * are only evaluated once at the start of the acquisition.

Description

The preprocess profile with the name, 'preprocess profile name,' is applied to 'Expression'.

Preprocess profiles can be configured in the pre-processor manager. You can access this manager via the expression editor of an Inspectra expert module.

2.11   Retentive functions

The functions in this group can be used in a virtual retentive module. Functions with a retentive character always keep their last result value until they are deliberately reset.

When using these functions in a virtual retentive module, the most recently calculated result value can be saved by stopping the measurement. When restarting the measurement, this value can be reloaded as the initial value.

Thus, long-term data, such as operating hours counter and media consumption, are not lost by stopping and starting the measurement.

Functions with this property are:

↗ *Count, page 99*

↗ *Int, page 34*

↗ *Max, page 51*

↗ *Min, page 57*

2.12   Plugins

The *ibaPDA* plugin system was designed to enable *ibaPDA* users to create their own features to perform custom calculations of measurement data with *ibaPDA* in real time. The functions can be used in expressions for virtual signals just like built-in functions. They will also appear in the expression builder (plugins).

To customize functions, you have to create a NET.dll file. This dll-file can be written in any .NET language (C#, C++, VB.NET, …).

Other documentation

For a detailed description of the plugin programming and configuration, please refer to the "ibaPDA Plugin" manual.

---

3   Support and contact

Support

Phone: +49 911 97282-14

Email: support@iba-ag.com

Note

	If you need support for software products, please state the number of the license container. For hardware products, please have the serial number of the device ready.

Contact

Headquarters

iba AG
Gebhardtstrasse 10-20
90762 Fuerth
Germany

Phone: +49 911 97282-0

Email: iba@iba-ag.com

Mailing address

iba AG
Postbox 1828
D-90708 Fuerth, Germany

Delivery address

iba AG
Gebhardtstrasse 10
90762 Fuerth, Germany

Regional and Worldwide

For contact data of your regional iba office or representative
please refer to our web site:

www.iba-ag.com