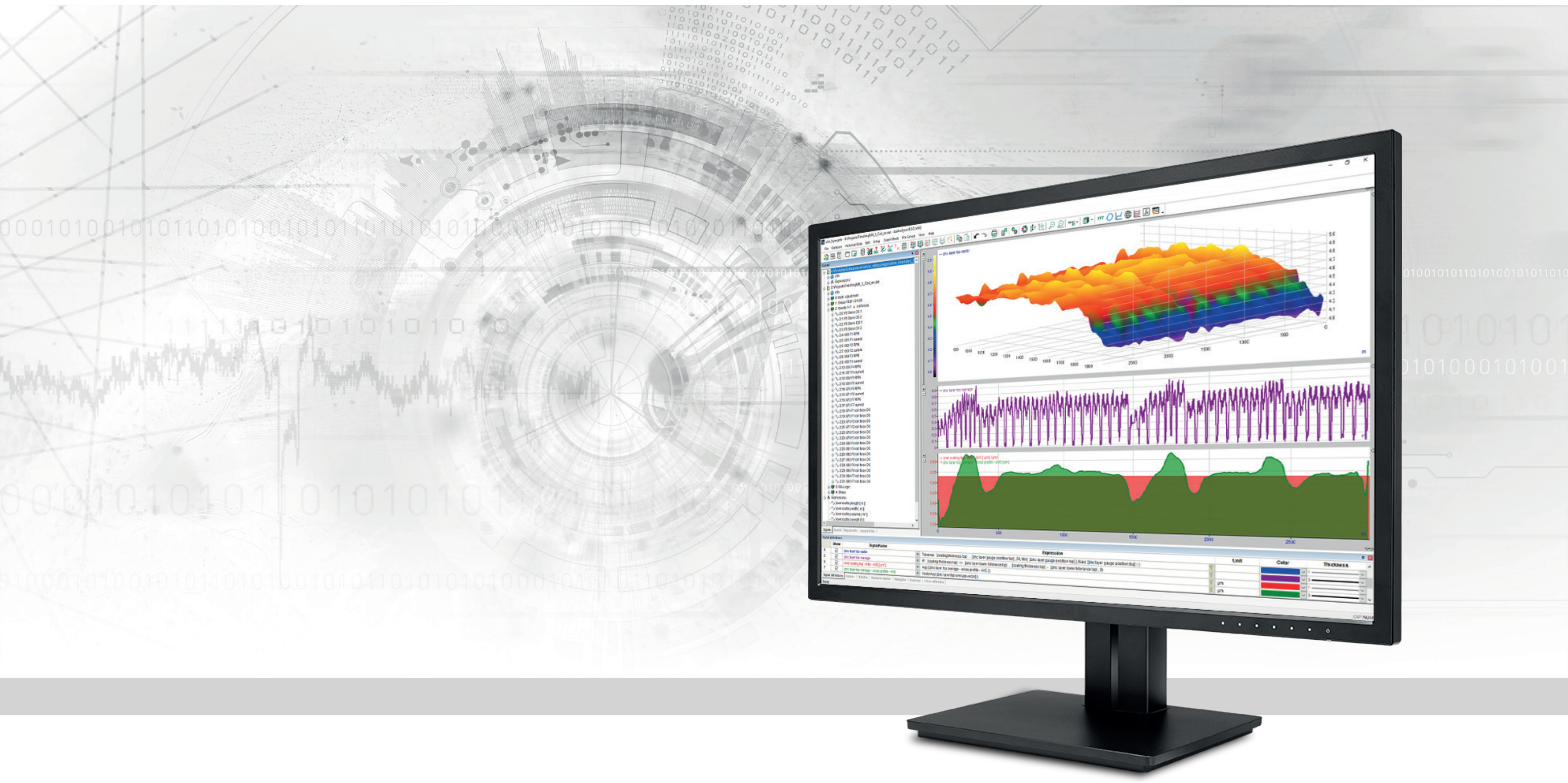


ibaAnalyzer

Expression builder

Manual part 3

Issue 8.3

Measurement Systems for Industry and Energy

www.iba-ag.com

Manufacturer

iba AG

Koenigswarterstrasse 44

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

Version	Date	Revision	Author	Version SW
8.3	06-2025	Functions updated acc. to v8.3.0	mm	8.3.0

Contents

1 About this documentation

This documentation describes the function and application of the software

ibaAnalyzer.

1.1 Target group

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 he/she is capable of assessing safety and recognizing possible consequences and risks on the basis of his/her specialist training, knowledge and experience and knowledge of the standard regulations.

This documentation addresses in particular professionals who are in charge of analyzing measured data and process data. Because the data is supplied by other iba products the following knowledge is required or at least helpful when working with *ibaAnalyzer*:

- Operating system Windows
- *ibaPDA* (creation and structure of the measuring data files)

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

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 describes the functionality of the *ibaAnalyzer* software in detail. It is designed both as a tutorial as well as a reference document.

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 *ibaAnalyzer* documentation (PDF version) is divided into six separate parts. Each part has its own section and page numbering beginning at 1 and is updated independently.

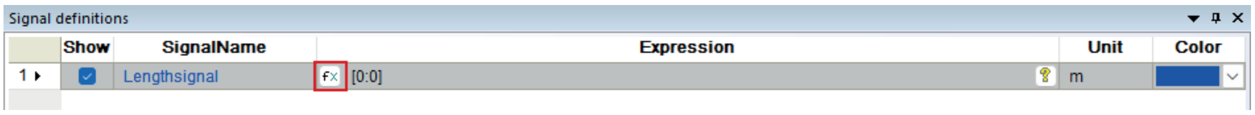
Part	Title	Content
Part 1	Introduction and Installation	General notes, licenses and add-ons Installation and program start User interface
Part 2	Working with *ibaAnalyzer*	Working with data file and analysis, presentation features, macro configuration, filter design, preferences, printing, export, interfaces to *ibaHD-Server*, *ibaCapture* and report generator
Part 3	Expression editor	Directory of all calculation functions in the expression builder, including explanation
Part 4	Database interface	Working with data from databases, connecting to the database, writing iba measurement data to databases, extracting the data from the database and analyzing the data.
Part 5	Interface for file extraction	Functions and settings for extracting data from iba data files to external file formats
Part 6	Application examples	*In preparation*

2 Functionality of the Expression builder

In *ibaAnalyzer*, you can use formulas to analyze the measured values. You can enter these formulas manually, e.g. in the signal table in the *Signal definitions* tab, or you can use the *Expression builder*.

The Expression builder is a tool for entering (mathematical) formulas or expressions. It simplifies the input of formulas and also provides a detailed list of possible operations and their syntax.

Opening the Expression builder
You can open the Expression builder by clicking the <fx> button in any table row in which you can enter a signal.

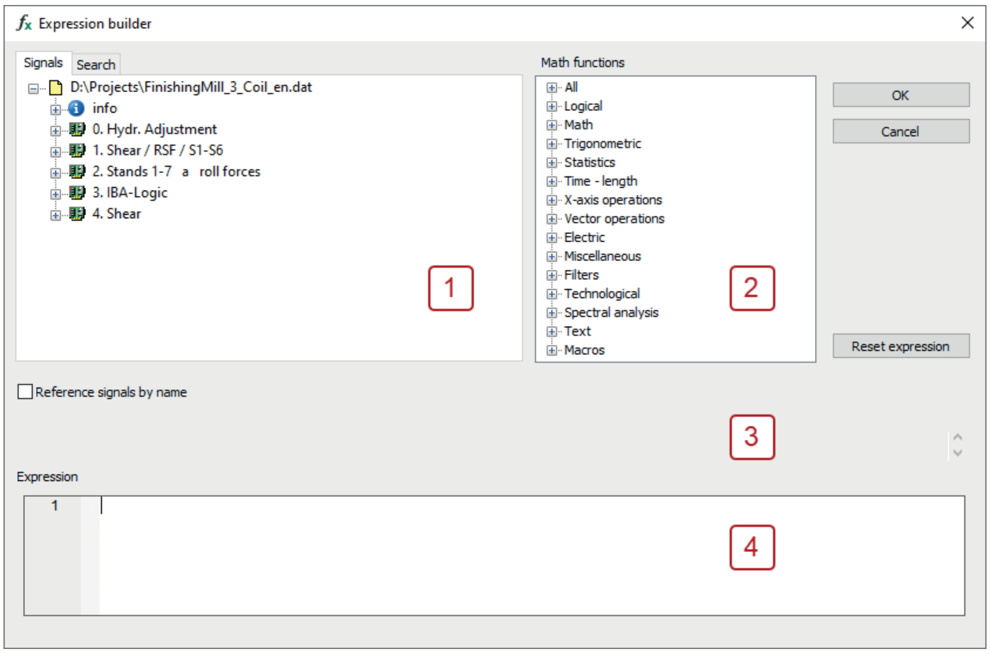


Note

The button *fx* in the toolbar does not open the Expression builder, but rather opens the dialog for the logical expressions. See *Logical expressions* in part 2 of the manual.

2.1 User interface of the Expression builder

The Expression builder consists of 4 areas.



1	Signal tree	3	Notes on the function
2	Function tree	4	Input field

In the signal tree on the left (1), you can find the original signals from the data file and additionally all the expressions created in the Expression builder. Select the signals or expressions to calculate with here.

The function tree on the right (2) shows all available mathematical operations and other functions sorted by topic.

If you select a function in the function tree, notes on the function (3) are displayed below, e.g. the syntax.

Enter the desired expression in the input field at the bottom (4). A tooltip for the function appears if the function is selected in the input field.

The <Reset expression> button deletes all entries from the command line.

If you activate the option *Reference signals by name*, the signal names are used in the printout instead of the usual signal identification [module number:signal number].

Note

When using the signal names as a signal reference, make sure that the signal names are unique.

2.2 Signals and wildcards in expressions

ibaAnalyzer marks signals in expressions using square brackets "[ ]". Each available signal has an identifier consisting of a module number and a signal number. Depending on the signal type, the module number and signal number are separated by different delimiters.

- Analog signals and text signals: separated by a colon ":"
- Digital signals: separated by a period "."

In some cases, a sub-signal number is also present, which is again separated using the same delimiter based on the signal type.

For files extracted by *ibaAnalyzer*, you can reference the results of calculations by their name.

Examples

[0:0]	Analog signal or text signal from module 0 and signal 0
[1.2]	Digital signal from module 1 and signal 2
[3:2:1]	Sub-signal of an analog signal from module 3, signal 2 and sub-signal 1
[Expression]	Expression with the name "Expression"

File indexes with multiple files

When multiple files are opened simultaneously, the file index also becomes part of the identifier. The first file has index 0, which can be omitted.

Examples

[0_0:0]	= [0:0], analog signal from the first file, module 0, signal 0
[2_1.0]	Digital signal from the file with index 2 (third file), module 1, signal 0

Wildcards in signal references

Wildcards allow you to reference multiple signals at once. The result is a vector. You can replace any part of the identifier (file index, module number, signal number, sub-signal number) using the following methods:

- Range with a hyphen, e.g., 3-6
- Asterisk "*" to refer to all available numbers

Examples

[*_0:0]	returns the signal [0:0] from all files opened in parallel
[0:3-5]	returns the signals [0:3], [0:4], and [0:5]
[0-2_3:4]	returns the signal [3:4] from the first 3 files opened in parallel
[*.0]	returns the first digital signal (with the number 0) from all available modules
[*_*:*]	returns all analog signals and text signals from all opened files

2.3 Usage of the Expression builder

You have several options for inserting operations, signals, and expressions into the input field, such as via drag & drop or double-click. This helps you avoid typing errors and work more efficiently.

Input using double-click
Position the cursor where you want to insert the operand or operation. Then double-click the desired item in the function tree or signal tree.

Note

Inserting signals and expressions via double-click only works in the Expression builder, but not in the signal grid on the *Signal Definitions* tab.

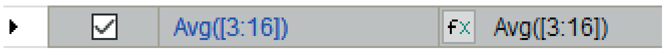
Intellisense input assistance
The Intellisense feature simplifies the manual entry of expressions in the signal table and the command line of the Expression builder. As you type, suggestions appear for functions, parameters, signals, or virtual expressions from the data file.

You can select suggestions by clicking or using the arrow keys, and confirm with <Enter>. As you continue typing, the list updates continuously with matching auto-complete suggestions until the command is complete.

The Expression builder automatically adds all necessary parentheses.

Navigating complex expressions
To keep track of complex expressions, use the keyboard shortcut <Ctrl> + <B> to jump back and forth between matching pairs of parentheses.

Closing the Expression builder and applying the expression
When you close the Expression builder by clicking <OK>, the created expression appears in the corresponding row of the table.



☑	Avg([3:16])	fx	Avg([3:16])

The expression is automatically used as the signal name, but you can manually replace it with plain text. For complex, cascaded expressions, it is recommended to use short and meaningful names to maintain clarity.

For information on how to handle invalid expressions, see ↗ *Diagnostics and syntax error detection*, page 14.

2.4 Diagnostics and syntax error detection

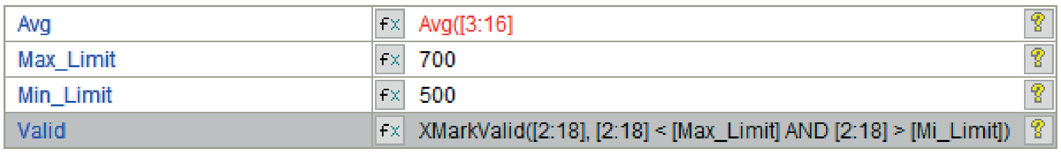
ibaAnalyzer offers different options for checking the correctness of the created expressions, e.g. when exiting the Expression builder or via the diagnostic function in the signal table.

Validation when exiting the Expression builder
If you have created an incorrect expression in the Expression builder and want to apply the expression with <OK>, you will receive a message with the option of correcting the expression. If you click on <Yes>, the cursor directly indicates the point where the error in the expression is suspected.

Tip

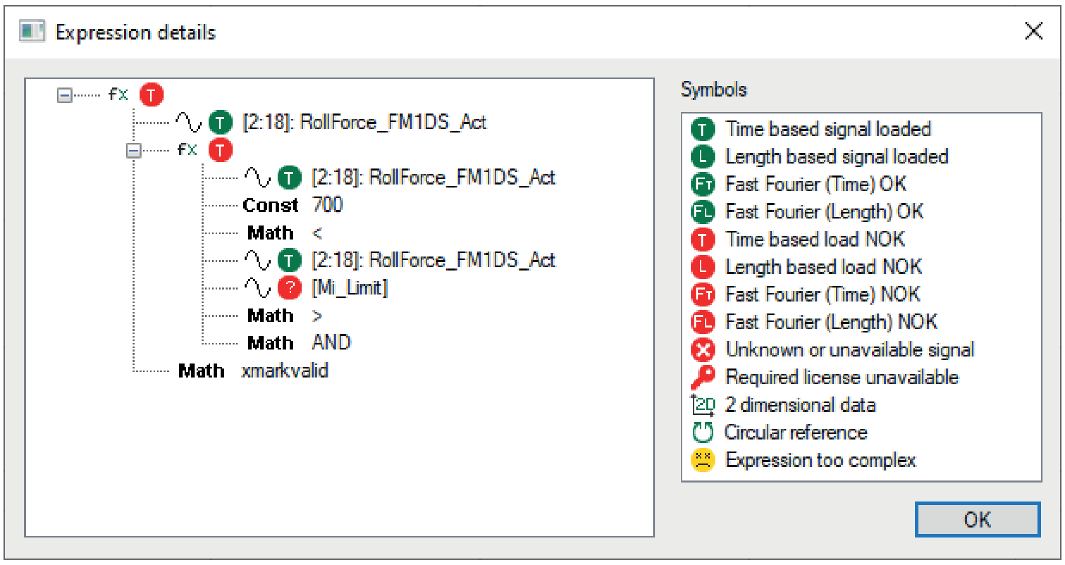
You can also manually check expressions in the Expression builder using the key combination <Ctrl> + <E>.

Diagnosis in the signal table
If you have manually entered a faulty expression in the signal table or ignored the message when closing the Expression builder, the faulty expression is displayed in red in the signal table. This helps you identify formal or syntactical errors that prevent the calculation from being executed.



Avg	fx	Avg([3:16]	?
Max_Limit	fx	700	?
Min_Limit	fx	500	?
Valid	fx	XMarkValid([2:18], [2:18] < [Max_Limit] AND [2:18] > [Mi_Limit])	?

If you have referenced a signal incorrectly, the expression is shown normally. However, you can open the diagnostics using the <?> button in the corresponding row to get more detailed information about the error. The faulty signal is marked with a red question mark.



Expression details

```
fx T
    ∿ T [2:18]: RollForce_FM1DS_Act
    fx T
        ∿ T [2:18]: RollForce_FM1DS_Act
        Const  700
        Math   <
        ∿ T [2:18]: RollForce_FM1DS_Act
        ∿ ? [Mi_Limit]
        Math   >
        Math   AND
    Math   xmarkvalid
```

Symbols

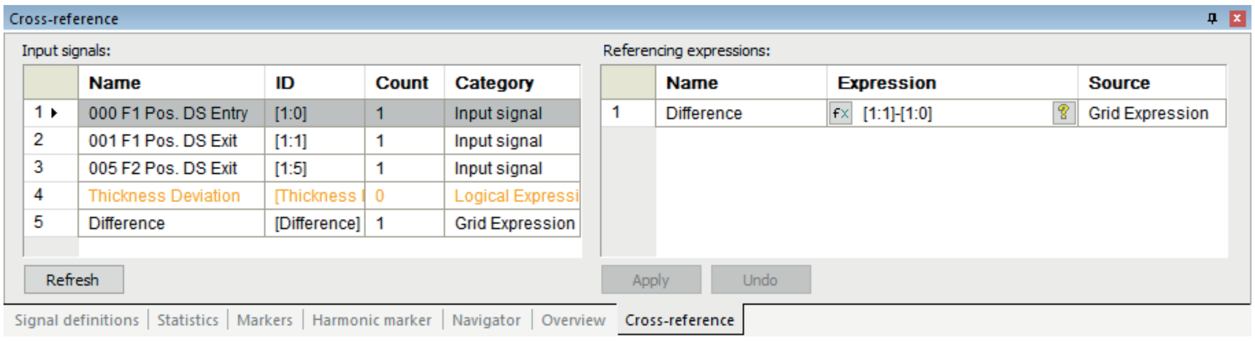
T	Time based signal loaded
L	Length based signal loaded
FT	Fast Fourier (Time) OK
FL	Fast Fourier (Length) OK
T	Time based load NOK
L	Length based load NOK
FT	Fast Fourier (Time) NOK
FL	Fast Fourier (Length) NOK
✗	Unknown or unavailable signal
🔑	Required license unavailable
2D	2 dimensional data
↻	Circular reference
☹	Expression too complex

OK

The faulty signal is marked with a red question mark.

2.5 References of logical expressions

On the *Cross-reference* tab of the signal table, you can review the usage and referencing of input signals and logical expressions. The tab shows a list of all input signals and expressions, and their usage in expressions and calculations. You can edit the expressions directly in the *Referencing expressions* table.



Cross-reference

Input signals:

	Name	ID	Count	Category
1 ▸	000 F1 Pos. DS Entry	[1:0]	1	Input signal
2	001 F1 Pos. DS Exit	[1:1]	1	Input signal
3	005 F2 Pos. DS Exit	[1:5]	1	Input signal
4	Thickness Deviation	[Thickness I	0	Logical Expressi
5	Difference	[Difference]	1	Grid Expression

Referencing expressions:

	Name	Expression	Source
1	Difference	fx [1:1]-[1:0]	Grid Expression

For more information see *ibaAnalyzer* manual part 1, chapter *Cross-reference tab*.

3 Logical functions

3.1 Comparative operations

```
e.g. 'Expression1' < 'Expression2'
```

>	greater than
>=	greater than or equal
<	less than
<=	less than or equal
<>	unequal
=	equal

Description

You can use the comparative operations >, >=, <, <=, <> and = to compare the values of two expressions (operands) with each other. As operands, you can enter original signals, calculated expressions or constant values.

The result of such an operation is the Boolean value TRUE or FALSE. You can present and evaluate the result as a new expression like a signal. This way, you can easily generate binary signals and use them then as conditions for other functions.

Note

If the crossing point of two curves 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. This means that any change from TRUE to FALSE (or vice versa) is always located at a measuring point.

The line that connects two measuring points in the presentation of analog values is just an visual approximation.

3.2 Boolean functions

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

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

Description
You can link binary expressions, such as digital signals, with each other using the Boolean functions AND, OR, NOT and XOR. As parameters, you can enter digital signals, calculated (binary) expressions or the numerical values 0 or 1.

According to the rules of Boolean logic, the functions return the value TRUE or FALSE. You can present and evaluate the result as a new expression like a signal. This way, you can easily generate binary signals and use them then as conditions for other functions.

Logical functions, truth table:

A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	
1	1	1	1	0	

3.3 Bitwise Boolean functions

```
e.g. bw_NOT ('Expression')
```

bw_AND	Bitwise AND
bw_OR	Bitwise OR
bw_XOR	Bitwise exclusive OR
bw_NOT	Bitwise NOT

Description
These functions are used for the bitwise linking of two analog values based on Boolean algebra. The functions return a 32-bit integer. 32-bit integers are expected as arguments.

If the arguments are not integers, the decimal part will be dropped before the operation is executed. If the arguments are too big so that their absolute value does not fit in a 32-bit integer, the operation is executed only on the 32 low-order bits.

When linking two analog values with a bw function, the individual bits of both values are logically linked. The result then is an analog value of the same type with a bit pattern in accordance with the logical link.

Example

For 2 analog values V1 = 15 and V2 = 2, the results are as follows:

	Dec. value	Bits	Hex	Result value
Output value V1	15	...1111	0x0000000F	
Output value V2	2	...0010	0x00000002	
V1 bw_AND V2		...0010	0x00000002	2
V1 bw_OR V2		...1111	0x0000000F	15
V1 bw_XOR V2		...1101	0x0000000D	13
bw_NOT (V1)		...0000	0xFFFFFFF0	-16

3.4 Branching

3.4.1 If

```
If('Condition','IF-True','IF-False')
```

Arguments

'Condition'	Condition as an operation with the Boolean results TRUE or FALSE
'IF-True'	Operation is performed if 'Condition' is TRUE
'IF-False'	Operation is performed if 'Condition' is FALSE

Description

You can use the *If* function for a conditioned execution of further calculations. Depending on the Boolean result of a 'Condition', which can itself be an operation, the operation 'IF-True' will be executed if the result is TRUE and the operation 'IF-False' if the result is FALSE.

Hence, different calculations can be executed in a process-controlled manner. Of course, you can use this function in a nested matter and thus realize further branches.

Note

If you combine signals with different sampling rates, the signals are automatically resampled before the calculation. The sampling rate (XBase) of the 'Condition' signal is decisive.

To obtain a result with a specific sampling rate, you can adjust the 'Condition' signal using the *Resample* function.

Tip

If you enter an analog signal for 'Condition', as a condition it will be checked whether the value is greater than (TRUE) or less than (FALSE) 0.5.

3.4.2 Switch

```
Switch ('Selector_Expression','Case_1_Expression','Value_1_Expression',
                              'Case_2_Expression','Value_2_Expression',
                              …
                              'Case_n_Expression','Value_n_Expression',
                              'Default_Value_Expr')
```

Arguments

'Selector_Expression'	Expression that is checked for different conditions
'Case_n_Expression'	Expression that is compared with 'Selector_Expression'
'Value_n_Expression'	Result if 'Select_Expression' and 'Case_n_Expression' match
'Default_Value_Expr'	Result if none of the 'Case_n_Expressions' match with 'Selector_Expression'

Description

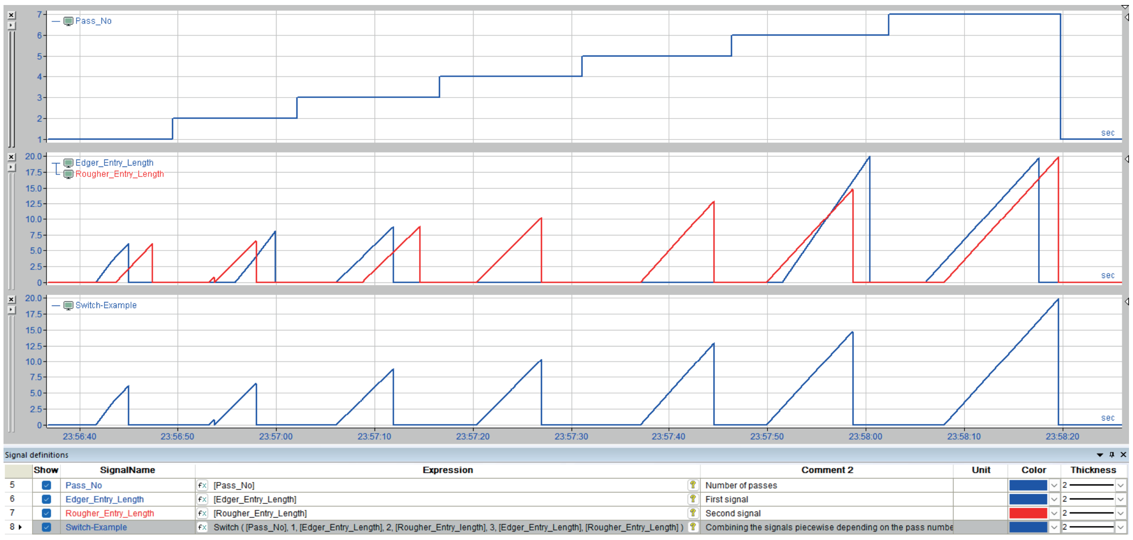
These instructions compare an incoming 'Selector_Expression' with any number of 'Case_n_Expressions' resembling the SQL statement CASE. At least 3 arguments are needed. With an even number of arguments, the last argument is automatically interpreted as 'Default_Value_Expr', which is used if none of the 'Case_n_Expressions' matches with the 'Selector_Expression'.

If 'Selector_Expression' and 'Case_n_Expression' match, the corresponding 'Value_n_Expression' is returned. If several 'Case_n_Expressions' match the input signal, the first is automatically selected.

The following signals are allowed as 'Selector_Expression':

- a numeric constant
- a text constant
- an equidistant and not equidistant sampled signal
- a text signal

In general, the types of comparison values must match, otherwise the corresponding case is not selected.



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	☑	Pass_No	[Pass_No]	Number of passes		🟦	2
6	☑	Edger_Entry_Length	[Edger_Entry_Length]	First signal		🟦	2
7	☑	Rougher_Entry_Length	[Rougher_Entry_Length]	Second signal		🟥	2
8	☑	Switch-Example	Switch ([Pass_No], 1, [Edger_Entry_Length], 2, [Rougher_Entry_length], 3, [Edger_Entry_Length], [Rougher_Entry_Length])	Combining the signals piecewise depending on the pass numbe		🟦	2

3.5 Edge Detection

3.5.1 OneShot

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

Description

This function returns the result TRUE, if the current measured value of 'Expression' is not equal to the previous one.

The function returns the result FALSE, if the current measured value does equal the previous one.

Tip

The function also works with non-equidistant measuring values.

3.5.2 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

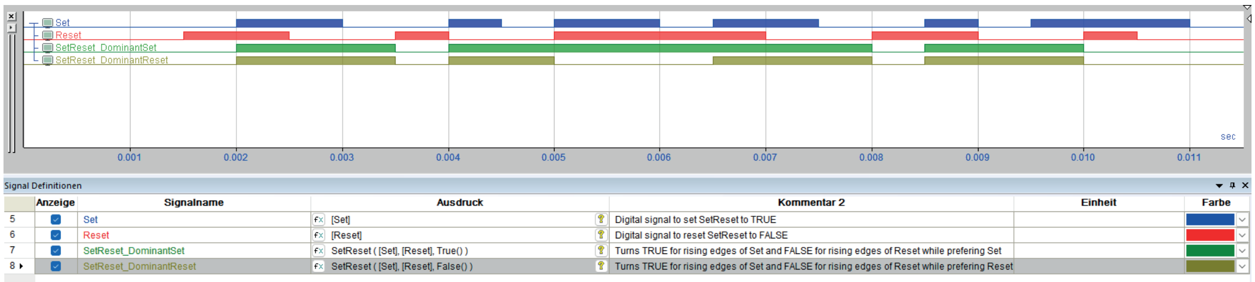
Description

Using this function, you can control a binary result (TRUE/FALSE) with the help of positive edges (transition from 0 to 1) of the arguments 'Set' and 'Reset'.

A rising edge of the 'Set' operand returns a static TRUE. A rising edge of the 'Reset' operand 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.



Signal Definitionen

	Anzeige	Signalname	Ausdruck	Kommentar 2	Einheit	Farbe
5	✓	Set	[Set]	Digital signal to set SetReset to TRUE		🟦
6	✓	Reset	[Reset]	Digital signal to reset SetReset to FALSE		🟥
7	✓	SetReset_DominantSet	SetReset ([Set], [Reset], True())	Turns TRUE for rising edges of Set and FALSE for rising edges of Reset while prefering Set		🟩
8	✓	SetReset_DominantReset	SetReset ([Set], [Reset], False())	Turns TRUE for rising edges of Set and FALSE for rising edges of Reset while prefering Reset		🟫

3.5.3 F_Trig and R_Trig

F_Trig

`F_Trig('Expression')`

Description

This function returns TRUE for a sample if there is a transition from TRUE to FALSE (falling edge) in 'Expression'.

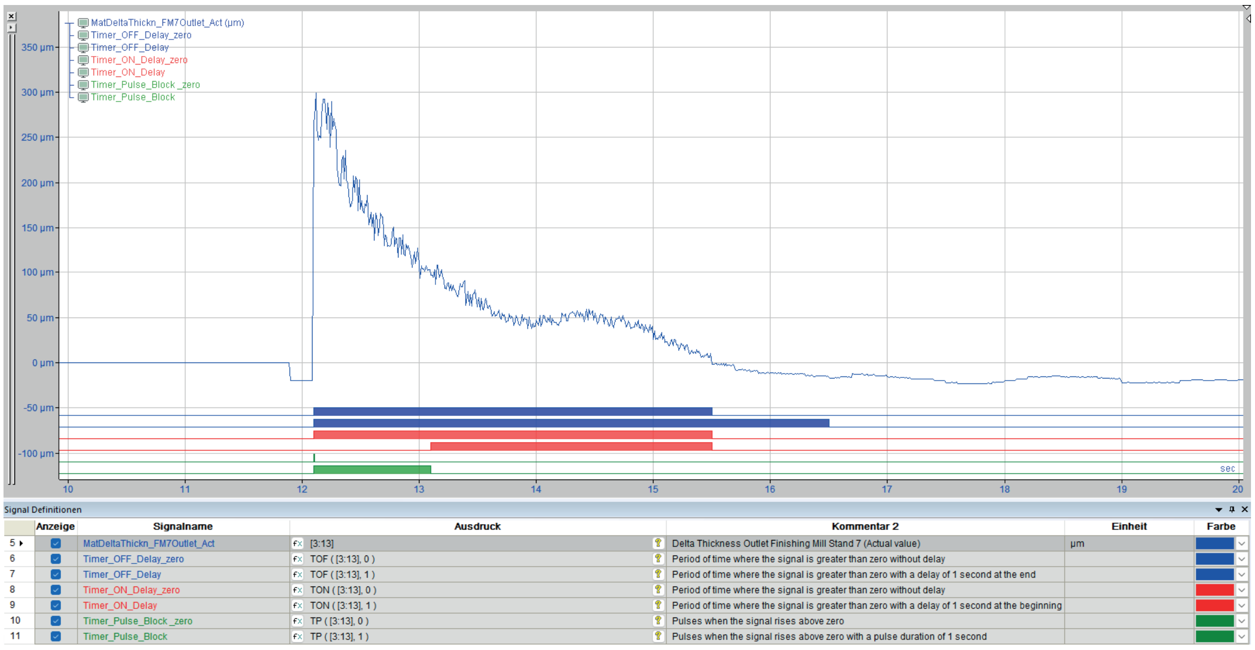
R_Trig

`R_Trig('Expression')`

Description

This function returns TRUE for a sample if there is a transition from FALSE to TRUE (rising edge) in 'Expression'.

3.6 Timer functions (IEC 61131-3)



Signal Definitionen

	Anzeige	Signalname	Ausdruck	Kommentar 2	Einheit	Farbe
5	✓	MatDeltaThickn_FM7Outlet_Act	[3:13]	Delta Thickness Outlet Finishing Mill Stand 7 (Actual value)	µm	🟦
6	✓	Timer_OFF_Delay_zero	TOF ([3:13], 0)	Period of time where the signal is greater than zero without delay		🟦
7	✓	Timer_OFF_Delay	TOF ([3:13], 1)	Period of time where the signal is greater than zero with a delay of 1 second at the end		🟦
8	✓	Timer_ON_Delay_zero	TON ([3:13], 0)	Period of time where the signal is greater than zero without delay		🟥
9	✓	Timer_ON_Delay	TON ([3:13], 1)	Period of time where the signal is greater than zero with a delay of 1 second at the beginning		🟥
10	✓	Timer_Pulse_Block _zero	TP ([3:13], 0)	Pulses when the signal rises above zero		🟩
11	✓	Timer_Pulse_Block	TP ([3:13], 1)	Pulses when the signal rises above zero with a pulse duration of 1 second		🟩

TOF

`TOF('in','pt')`

Description

Off Delay Timer. The output is switched off 'pt' seconds after switching off the 'in' input.

TON

`TON('in','pt')`

Description

On Delay Timer. The output is switched on 'pt' seconds after switching on the 'in' input.

TP

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

Description

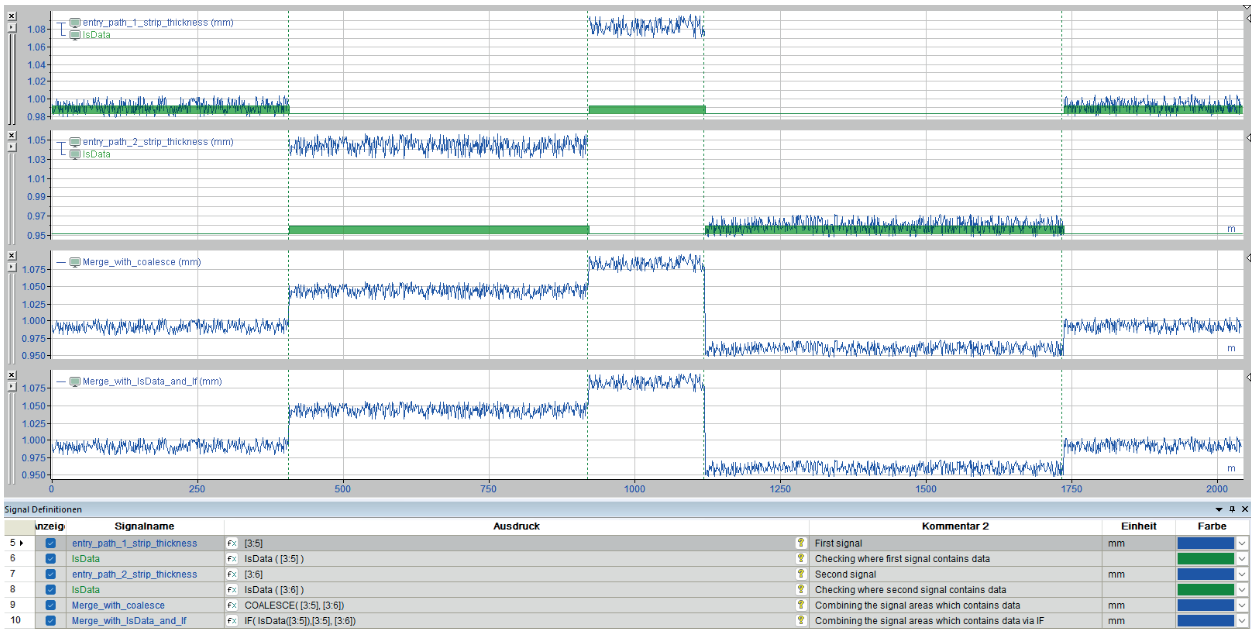
Pulse Timer. The output is switched on for 'pt' seconds after rising edge at the 'in' input.

Note

Due to the delayed switch-off, several pulses can merge if the delayed end overlaps with the start of another pulse.

A further rising edge during the output pulse does not extend the output pulse and does not restart the pulse.

3.7 IsData and Coalesce


entry_path_1_strip_thickness (mm)
IsData
entry_path_2_strip_thickness (mm)
IsData
Merge_with_coalesce (mm)
Merge_with_IsData_and_If (mm)
m
0
250
500
750
1000
1250
1500
1750
2000

Signal Definitionen

	nzeig	Signalname	Ausdruck	Kommentar 2	Einheit	Farbe
5	✓	entry_path_1_strip_thickness	[3:5]	First signal	mm	🟦
6	✓	IsData	IsData ([3:5])	Checking where first signal contains data		🟩
7	✓	entry_path_2_strip_thickness	[3:6]	Second signal	mm	🟦
8	✓	IsData	IsData ([3:6])	Checking where second signal contains data		🟩
9	✓	Merge_with_coalesce	COALESCE([3:5], [3:6])	Combining the signal areas which contains data	mm	🟦
10	✓	Merge_with_IsData_and_If	IF(IsData([3:5]),[3:5], [3:6])	Combining the signal areas which contains data via IF	mm	🟦

IsData

```
IsData('Expression','End')
```

Arguments

'Expression'	Input signal or expression
'End'	Length of the resulting signal

Description

The result of this operation is TRUE if measured values are available for 'Expression'. The result is FALSE if measured values are missing or signals are empty. You can use this function, for example, as condition for other calculations.

Optionally, you can enter the 'End' parameter. This parameter can reduce or extent the resulting signal of the function so that the resulting signal complies with other signals and can be used for further links. If you do not specify the 'End' parameter, the length of the result signal complies with that of the input signal (incl. invalid samples).

Coalesce

```
Coalesce ('Candidate1','Candidate2',…)
```

Description

Inspired by the corresponding SQL query, the function *Coalesce* returns the first of its arguments that contains data. As of *ibaAnalyzer* v8.3.0 or later, this function processes data sample by sample.

You can use this function, for example, to create a safeguard against missing signals in a data file.

3.8 IsNE

```
IsNE('Expression')
```

Description
The function indicates whether a signal contains non-equidistantly sampled data. It returns TRUE if the data loaded by evaluating 'Expression' is sampled non-equidistantly, otherwise FALSE.

This function is particularly useful when processing non-equidistantly sampled HD data, but it also works with other data sources, e.g., signals from data files or results of the *XY* function.

4 Mathematical functions

4.1 Basic arithmetic operations

4.1.1 Basic arithmetic operations +, -, *, /

```
e.g. 'Expression1' + 'Expression2'
```

Description

All signals and expressions can be processed by basic arithmetic operations (addition, subtraction, multiplication and division).

If you use digital signals or expressions as operands in basic arithmetic operations, *ibaAnalyzer* translates the TRUE values as 1.0 and FALSE as 0.0.

The result of a basic arithmetic operation is always an analog expression.

4.1.2 Abs

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

Description

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

Tip

Interpolated values at a sign change between two samples may differ in value.

4.1.3 Mod

```
Mod('Dividend','Divider')
```

Description

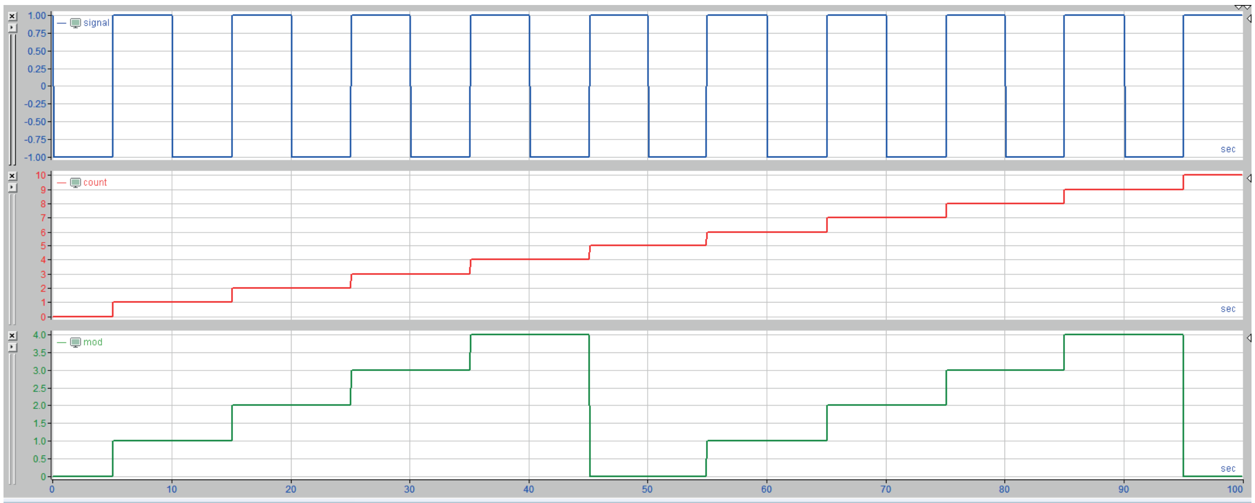
This function returns the modulo of 'Dividend' and 'Divider'. Internally, the function uses the fmod C-function, which permits the use of floating point values for 'Dividend' and 'Divider'.

The modulo r is the remainder of the division 'Dividend'/'Divider', thus the reverse applies:

Dividend = Divider * x + r, where x is an integer.

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

If 'Dividend' < 'Divider', then the function returns the value of 'Dividend'. Mathematically speaking, the remainder can also be described as "Dividend modulo Divider".



Signal definitions

	Show	SignalName	Expression	Unit	Color	Thickness
3	✓	signal	IF([sin] >0,-1, 1)		🟦	2
4	✓	count	Count([signal],0,0,1)		🟥	2
5	✓	mod	mod([count],5)		🟩	2

4.1.4 Ceiling, Floor and Round

Ceiling

`Ceiling('Expression')`

Description

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

Floor

`Floor('Expression')`

Description

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

Round

`Round('Expression')`

Description

This function rounds 'Expression' up or down to the nearest integer.

Tip

You can use the following expression to round to 2 decimal places:

```
ROUND(100 * [Signal]) / 100
```

4.2 Integral and differential calculation

4.2.1 Int

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

Arguments

'Expression'	Signal or expression	
'Reset'	Optional binary parameter 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 of 'Expression'. With the 'Reset' parameter, you can reset 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.

4.2.2 Diff (formerly Dif)

```
Diff('Expression','dy'=FALSE)
```

Description
This function returns the derivative (or the differential) of 'Expression'.

If you set the optional parameter 'dy' to TRUE(), only the difference between the measured values is calculated instead of the differential.

Example
If 'Expression' is a length based signal, you can determine a speed curve with the *Diff* function.

4.3 Powers and square roots

4.3.1 Pow

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

Arguments

'Expression1'	Base
'Expression2'	Exponent

Description
This function takes 'Expression1' (base) to the power of 'Expression2' (exponent).

Example
Calculating some important powers

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

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

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

(10)^(lg 2) = Pow(10, Log10(2)) = 2

(0)^-1 = Pow(0, -1) = +∞ (infinity)

Note

Raising 0 to the power of -1 does not produce an error message, but it also does not return a result.

4.3.2 Sqrt

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

Description
This function returns the square root of 'Expression'.

Note

Negative values for 'Expression' do not produce an error message, but they also do not return a result.

4.4 e functions and logarithms

4.4.1 Exp

```
Exp('Expression')
```

Description
This function calculates the expression $e^{'Expression'}$.

4.4.2 Log

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

Description
This function returns the natural logarithm of 'Expression'.

Note

Negative values for 'Expression' do not produce an error message, but they also do not return a result.

4.4.3 Log10

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

Description
This function returns the decadic logarithm of 'Expression'.

Note

Negative values for 'Expression' do not produce an error message, but they also do not return a result.

4.5 Pi

`Pi()`

Description

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

4.6 Sum

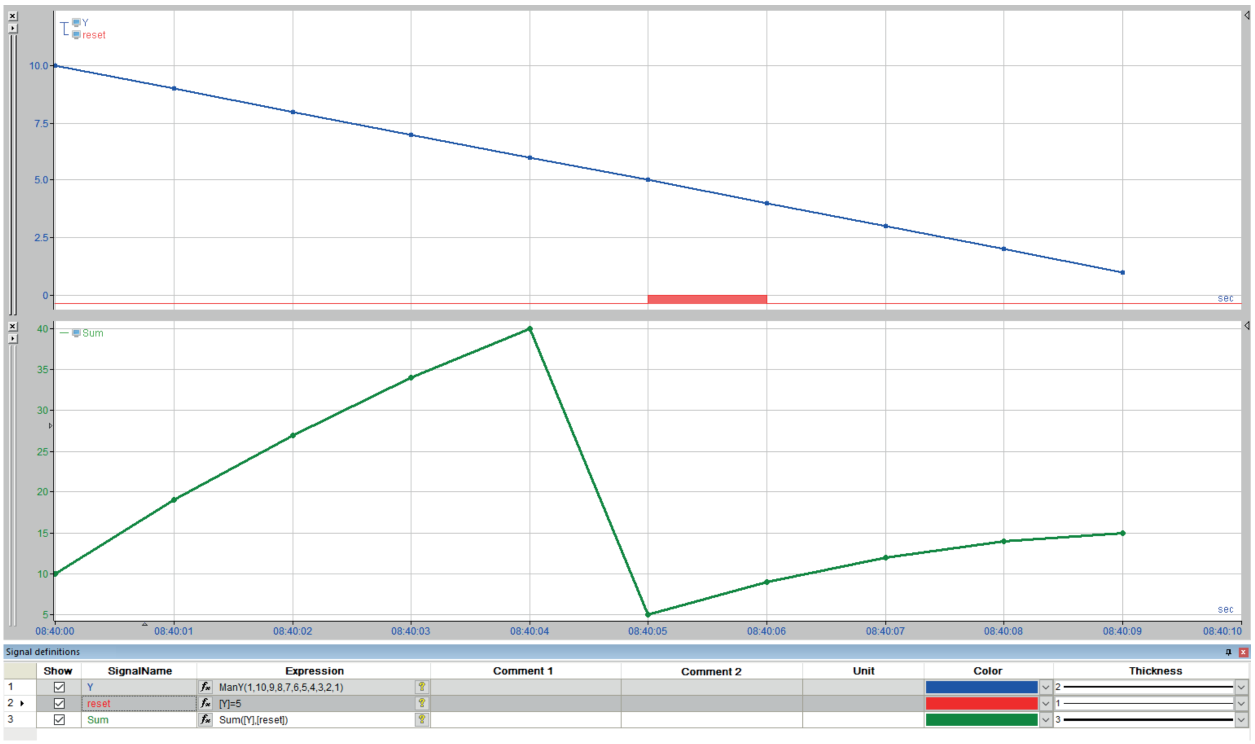
`Sum('Expression','Reset'=FALSE)`

Description

This operation summarizes all signal values of a function point by point. If the summation is interrupted by a reset value, then the summation starts again.

Example

The summation starts with the signal value 10 + 9 + 8 + ... 6. Here, 'Reset' = TRUE() causes an interruption, and the function is reset to zero. After that, the summation starts again.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Y	ManY(1,10,9,8,7,6,5,4,3,2,1)				🟦	2
2	✓	reset	[Y]=5				🟥	1
3	✓	Sum	Sum([Y],[reset])				🟩	3

5 Trigonometric functions

```
e.g. Sin ('Expression')
```

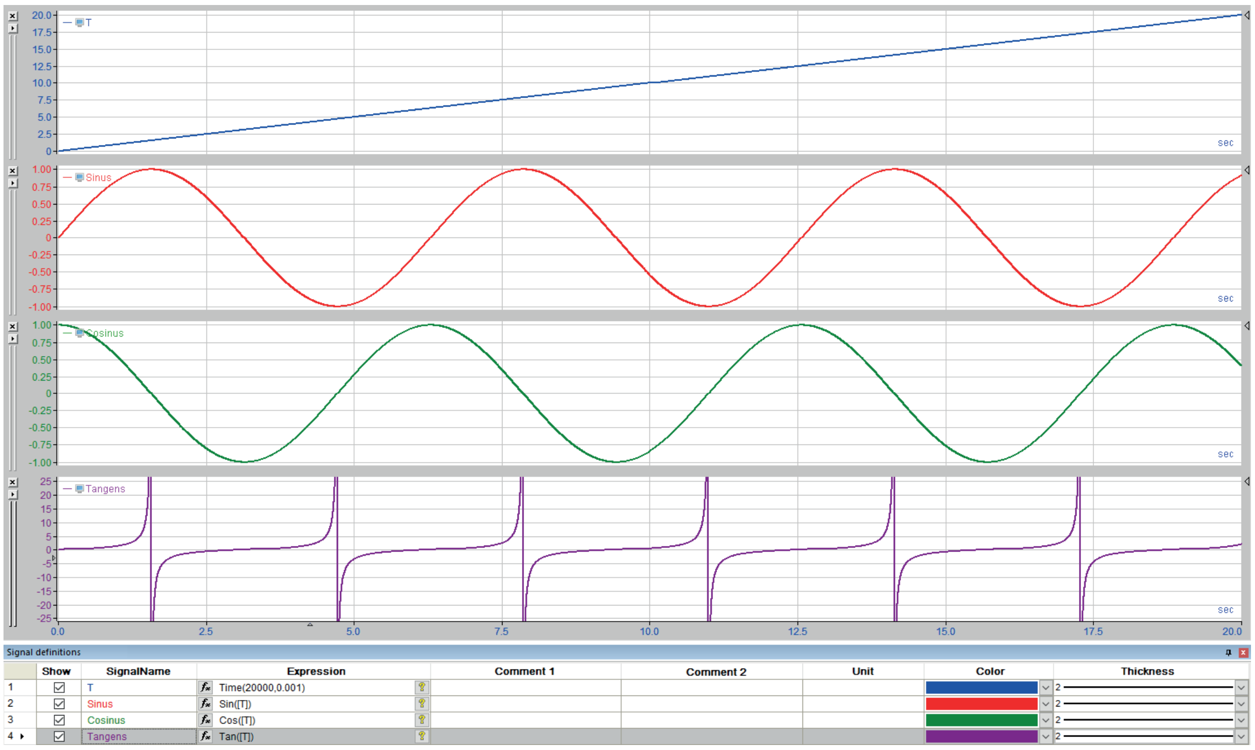
Sin('Expression')	This function returns the sine of 'Expression' in rad.
Cos('Expression')	This function returns the cosine of 'Expression' in rad.
Tan('Expression')	This function returns the tangent of 'Expression' in rad.
Asin('Expression')	This function returns the arcsine of 'Expression' in rad.
Acos('Expression')	This function returns the arccosine of 'Expression' in rad.
Atan('Expression')	This function returns the arctangent of 'Expression' in rad.
Atan2 ('X','Y')	This function returns the arctangent of 'Y'/'X'.

Description

The standard functions and the related inverse functions are available for various calculations in which trigonometric functions are needed, for example, the calculation of power in three-phase AC systems.

Example

Visualization of trigonometric functions:



Signal definitions

	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☑	T	Time(20000,0.001)				🟦	2
2	☑	Sinus	Sin([T])				🟥	2
3	☑	Cosinus	Cos([T])				🟩	2
4	☑	Tangens	Tan([T])				🟪	2

6 Statistical functions

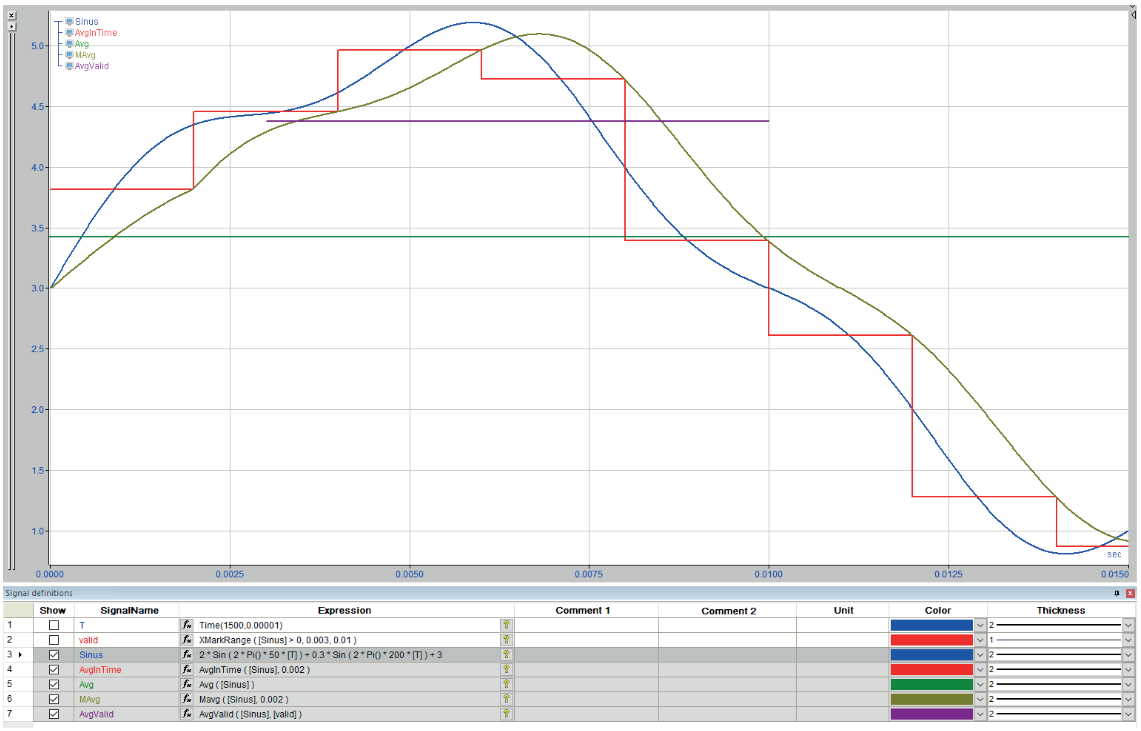
ibaAnalyzer supports the calculation of miscellaneous statistical functions. Four different versions are available for each function in a normal case:

- The standard function always calculates the corresponding size across the entire signal.
- The suffix *InTime* indicates that the corresponding size is formed over intervals of a specified length.
- The suffix *Valid* is used to calculate the corresponding size over intervals, which are marked with a binary signal.
- The prefix *M* indicates that the corresponding size is formed over moving intervals of a specified length.

You can find the vector statistics in the vector functions, see ↗ *Vector operations*, page 71.

You can find the RMS function in the electrical functions, see ↗ *RMS and Eff*, page 81.

6.1 Average (Avg)



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☐	T	Time(1500,0.00001)				🟦	2
2	☐	valid	XMarkRange ([Sinus] > 0, 0.003, 0.01)				🟥	1
3 ▸	☑	Sinus	2 * Sin (2 * Pi() * 50 * [T]) + 0.3 * Sin (2 * Pi() * 200 * [T]) + 3				🟦	2
4	☑	AvgInTime	AvgInTime ([Sinus], 0.002)				🟥	2
5	☑	Avg	Avg ([Sinus])				🟩	2
6	☑	MAvg	Mavg ([Sinus], 0.002)				🟫	2
7	☑	AvgValid	AvgValid ([Sinus], [valid])				🟪	2

Avg

`Avg('Expression')`

Description

This function returns the average of 'Expression'. The result is displayed as a constant value (horizontal line) in the graph.

AvgInTime

`AvgInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the average
'Interval'	Length of the interval over which to calculate the average

Description

This function returns the average value of 'Expression' per time segment of the length 'interval'.

AvgValid

`AvgValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which to form the average
'Valid'	Control signal

Description

This operation returns the average of 'Expression' for the interval (time or length) where a related control signal is TRUE.

MAvg

`MAvg('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the average
'Interval'	Length of the interval over which to calculate the average

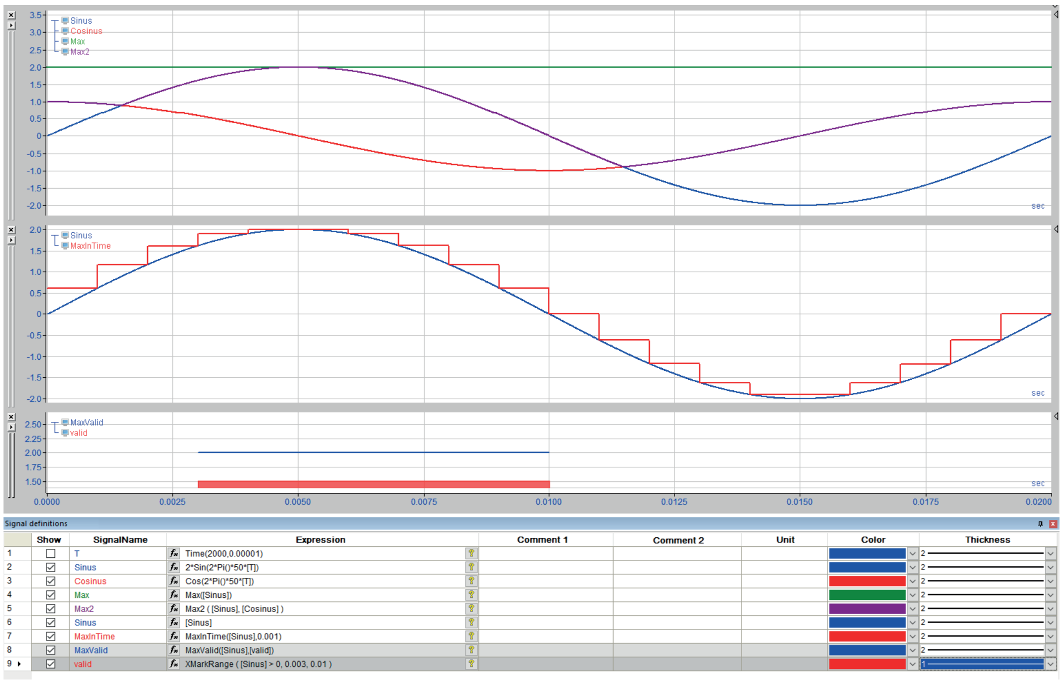
Description

This function returns the floating arithmetic average of 'Expression' calculated over a moving interval of length 'interval'.

Tip

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

6.2 Maxima (Max)



Signal definitions

	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☐	T	Time(2000,0.00001)				🟦	2
2	☑	Sinus	2*Sin(2*Pi()*50*[T])				🟦	2
3	☑	Cosinus	Cos(2*Pi()*50*[T])				🟥	2
4	☑	Max	Max([Sinus])				🟩	2
5	☑	Max2	Max2 ([Sinus], [Cosinus])				🟪	2
6	☑	Sinus	[Sinus]				🟦	2
7	☑	MaxInTime	MaxInTime([Sinus],0.001)				🟥	2
8	☑	MaxValid	MaxValid([Sinus],[valid])				🟦	2
9	☑	valid	XMarkRange ([Sinus] > 0, 0.003, 0.01)				🟥	1

Max

`Max('Expression')`

Description

This function returns the maximum value of 'Expression'. The result is displayed as a constant value (horizontal line) in the graph.

Max2

`Max2('Expression1','Expression2',…)`

Description

This function returns the maximum of all specified signals, 'Expression1', 'Expression2' etc. All signals are compared measured value by measured value, with the larger value in each case being presented as the result.

MaxInTime

`MaxInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which the maximum is formed
'Interval'	Length of the interval over which to calculate the maximum

Description

This function returns the maximum value of 'Expression' within each interval of the length 'Interval'. You can use this function for time based signals ('Interval' in seconds) as well as for length based signals ('Interval' in meters).

MaxValid

`MaxValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which the maximum is formed
'Valid'	Control signal

Description

This operation returns the maximum of 'Expression' for the interval (time or length) where a related control signal is TRUE.

MMax

`MMax('Expression','Interval')`

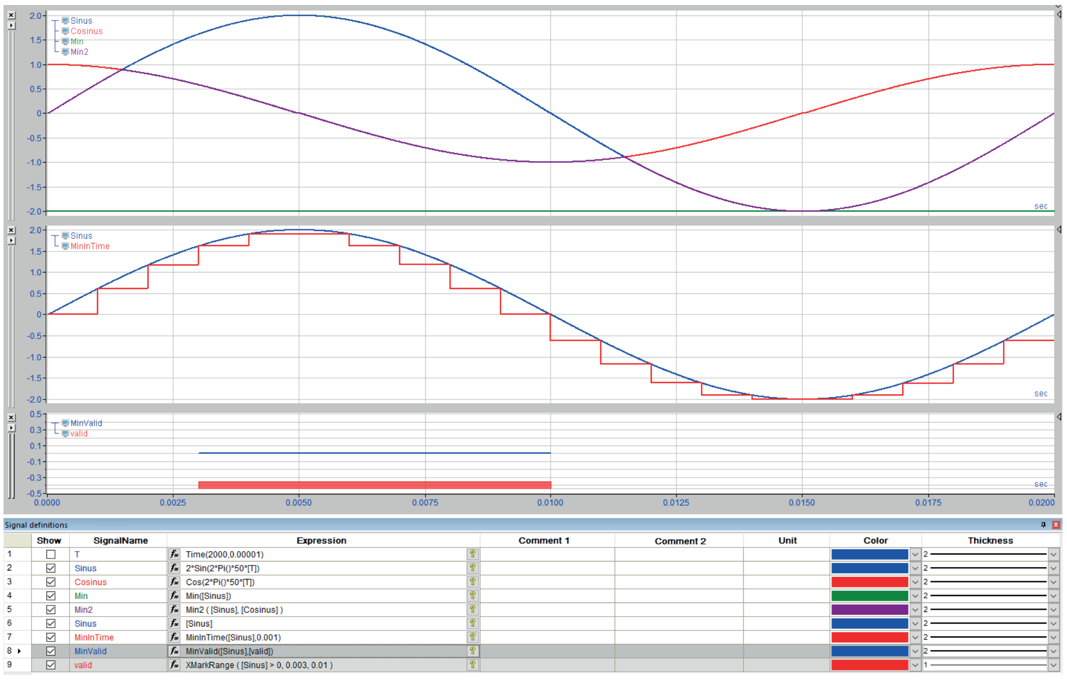
Arguments

'Expression'	Signal or expression for which the maximum is formed
'Interval'	Length of the interval over which to calculate the maximum

Description

This function returns the maximum of 'Expression' within a floating X-axis interval of the length 'Interval', advancing by one measuring point in each case.

6.3 Minima (Min)



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☐	T	Time(2000,0.00001)				🟦	2
2	✓	Sinus	2*Sin(2*Pi()*50*[T])				🟦	2
3	✓	Cosinus	Cos(2*Pi()*50*[T])				🟥	2
4	✓	Min	Min([Sinus])				🟩	2
5	✓	Min2	Min2 ([Sinus], [Cosinus])				🟪	2
6	✓	Sinus	[Sinus]				🟦	2
7	✓	MinInTime	MinInTime([Sinus],0.001)				🟥	2
8	✓	MinValid	MinValid([Sinus],[valid])				🟦	2
9	✓	valid	XMarkRange ([Sinus] > 0, 0.003, 0.01)				🟥	1

Min

`Min('Expression')`

Description

This function returns the minimum value of the 'Expression' signal. The result is displayed as a constant value (horizontal line) in the graph.

Min2

`Min2('Expression1','Expression2',…)`

Description

This function returns the minimum of all specified signals, 'Expression1', 'Expression2' etc. All signals are compared measured value by measured value, with the smaller value in each case being presented as the result.

MinInTime

`MinInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the minimum
'Interval'	Length of the interval over which to calculate the minimum

Description

This function returns the minimum value of 'Expression' within each interval of the length 'Interval'. You can use this function for time based signals ('Interval' in seconds) as well as for length based signals ('Interval' in meters).

MinValid

`MinValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which to form the minimum
'Valid'	Control signal

Description

This operation returns the minimum of 'Expression' for the interval (time or length) where a related control signal is TRUE.

MMin

`MMin('Expression','Interval')`

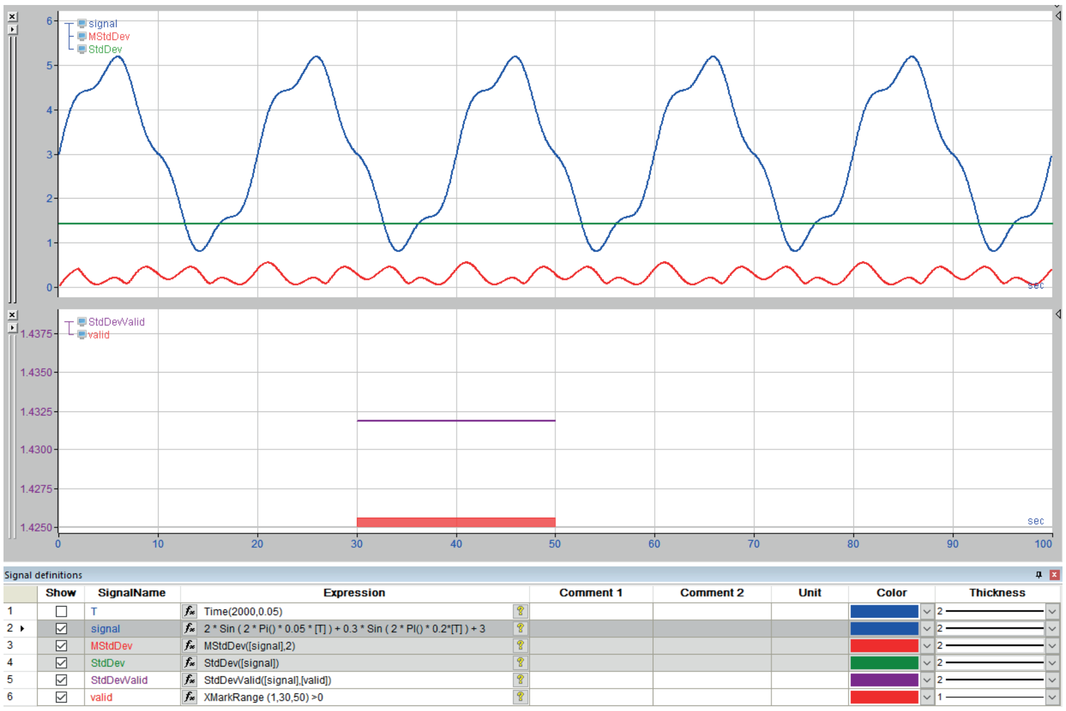
Arguments

'Expression'	Signal or expression for which to form the minimum
'Interval'	Length of the interval over which to calculate the minimum

Description

This function returns the minimum of 'Expression' within a floating X-axis interval of the length 'interval', advancing by one measuring point in each case.

6.4 Standard deviation (StdDev)



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☐	T	Time(2000,0.05)				🟦	2
2	✓	signal	2 * Sin (2 * Pi() * 0.05 * [T]) + 0.3 * Sin (2 * PI() * 0.2*[T]) + 3				🟦	2
3	✓	MStdDev	MStdDev([signal],2)				🟥	2
4	✓	StdDev	StdDev([signal])				🟩	2
5	✓	StdDevValid	StdDevValid([signal],[valid])				🟪	2
6	✓	valid	XMarkRange (1,30,50) >0				🟥	1

StdDev

`StdDev('Expression')`

Description

This function returns the standard deviation of 'Expression'.

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

n = number of samples

StdDevInTime

`StdDevInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the standard deviation
'Interval'	Length of the interval over which to calculate the standard deviation

Description

This function returns the standard deviation of 'Expression' over each time interval of the length 'Interval'.

Note

The result of the StdDevInTime function is always specified for the previous interval.

StdDevValid

`StdDevValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which to form the standard deviation
'Valid'	Control signal

Description

This function returns the standard deviation of 'Expression' for the interval (time or length) where a related control signal is TRUE.

MStdDev

`MStdDev('Expression','Interval')`

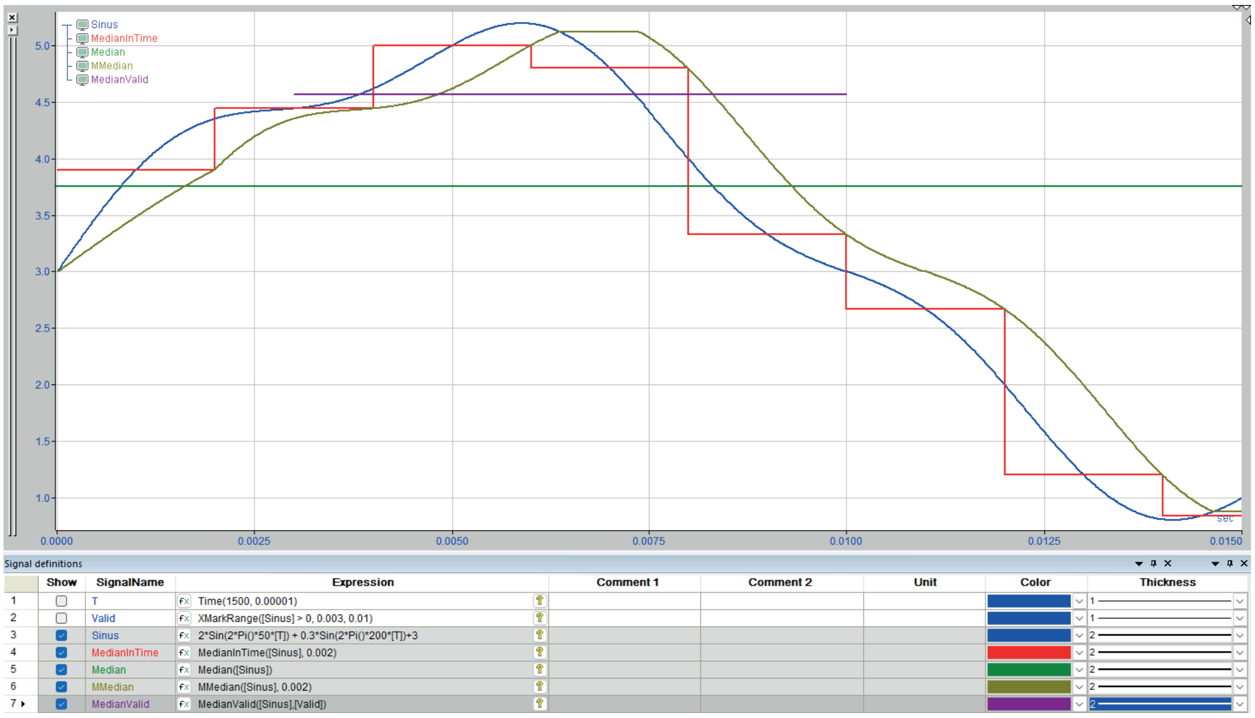
Arguments

'Expression'	Signal or expression for which to form the standard deviation
'Interval'	Length of the interval over which to calculate the standard deviation

Description

This function returns the floating standard deviation of 'Expression' over each time interval of the length 'Interval'. You can use this function for time based signals ('Interval' in seconds) as well as for length based signals ('Interval' in meters).

6.5 Median



Signal definitions

	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☐	T	Time(1500, 0.00001)				🟦	1
2	☐	Valid	XMarkRange([Sinus] > 0, 0.003, 0.01)				🟦	1
3	☑	Sinus	2*Sin(2*Pi()*50*[T]) + 0.3*Sin(2*Pi()*200*[T])+3				🟦	2
4	☑	MedianInTime	MedianInTime([Sinus], 0.002)				🟥	2
5	☑	Median	Median([Sinus])				🟩	2
6	☑	MMedian	MMedian([Sinus], 0.002)				🟫	2
7	☑	MedianValid	MedianValid([Sinus],[Valid])				🟪	2

Median

`Median('Expression')`

Description

This function returns the median of 'Expression'. The median is the measured value for which 50 % of all measured values are smaller and 50 % are larger. The result is displayed as a constant value (horizontal line) in the graph.

MedianInTime

`MedianInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the median
'Interval'	Length of the interval over which to calculate the median

Description

This function returns the median of 'Expression' within each interval of the length 'Interval'.

MedianValid

`MedianValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which to form the median
'Valid'	Control signal

Description

This operation returns the median of 'Expression' for the interval (time or length) where a related control signal is TRUE.

MMedian

`MMedian('Expression','Interval')`

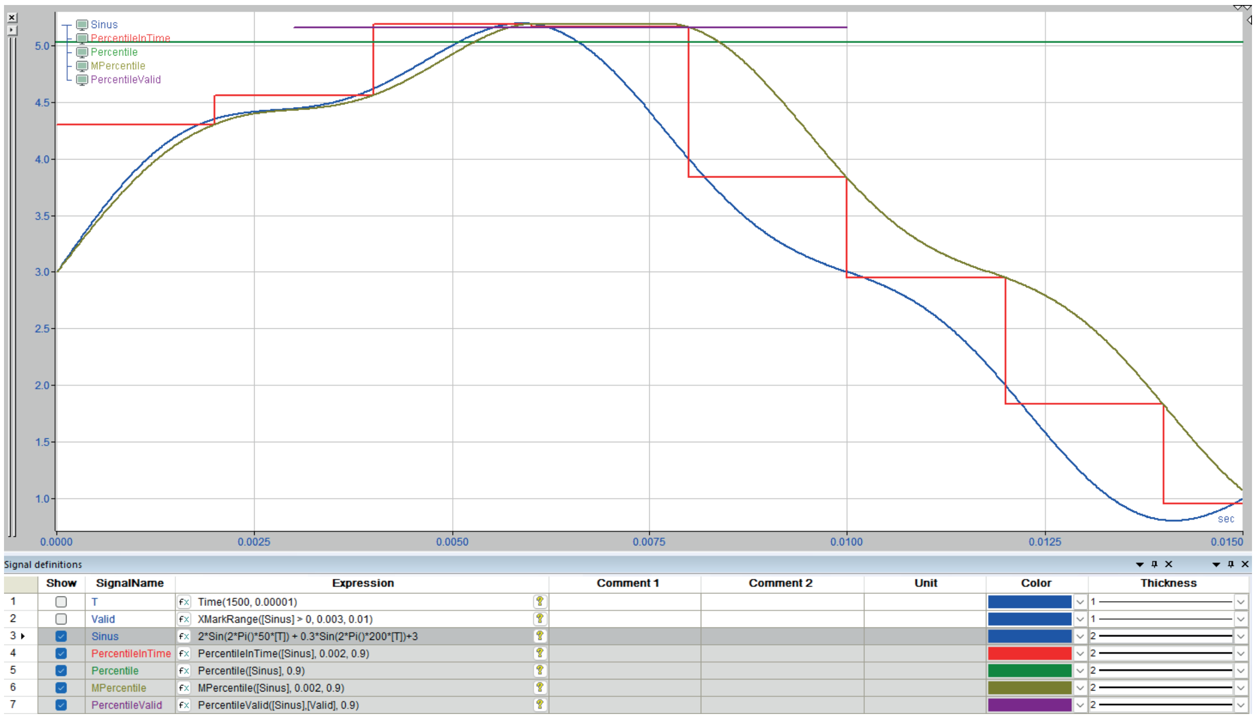
Arguments

'Expression'	Signal or expression for which to form the median is formed
'Interval'	Length of the interval over which to calculate the median

Description

This function returns the floating median of 'Expression' calculated over an 'Interval' in seconds.

6.6 Percentile



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☐	T	Time(1500, 0.00001)				🟦	1
2	☐	Valid	XMarkRange([Sinus] > 0, 0.003, 0.01)				🟦	1
3	☑	Sinus	2*Sin(2*Pi()*50*[T]) + 0.3*Sin(2*Pi()*200*[T])+3				🟦	2
4	☑	PercentileInTime	PercentileInTime([Sinus], 0.002, 0.9)				🟥	2
5	☑	Percentile	Percentile([Sinus], 0.9)				🟩	2
6	☑	MPercentile	MPercentile([Sinus], 0.002, 0.9)				🟫	2
7	☑	PercentileValid	PercentileValid([Sinus],[Valid], 0.9)				🟪	2

Percentiles

```
Percentile('Expression','Percentile'=0.5)
```

Arguments

'Expression'	Signal or expression for which to form the percentile
'Percentile'	Percentile

Description

This function returns the defined percentile of 'Expression'.

The p% percentile is the smallest value of a set of measured values which is greater than p% of the number of values measured.

A typical percentile is the 50 % percentile, the so-called median. The median divides the set of values measured into two equal halves: 50 % of all values measured are smaller than the median value, the remaining 50 % are greater than or equal to it. Further typical percentiles are 25 % and 75 % which, together with the median, enable the division of a set of values measured into four groups, the so-called quartiles. (< 25%, < 50%, < 75%, ≥ 75%).

The *Percentile* function determines the percentile value of the total number of measuring points of a signal. Enter the percentile as a decimal value, i.e.:

- 50 % --> 'Percentile' = 0.5 (default value)
- 75 % --> 'Percentile' = 0.75
- 95.9 % --> 'Percentile' = 0,959

This function is, for example, particularly useful when assessing the quality of a product where a particular property must comply with a defined classification.

PercentileInTime

```
PercentileInTime('Expression','Interval','Percentile'=0.5)
```

Arguments

'Expression'	Signal or expression for which to form the percentile
'Interval'	Length of the interval over which to form the percentile
'Percentile'	Percentile

Description

This function returns the percentile of 'Expression' over each time interval of the length 'Interval'.

PercentileValid

```
PercentileValid('Expression','Valid','Percentile'=0.5)
```

Arguments

'Expression'	Signal or expression for which to form the percentile
'Valid'	Length of the interval over which to form the percentile
'Percentile'	Percentile

Description

This function returns the percentile of 'Expression' for every interval (time or length) for which a related control signal 'Valid' is TRUE.

PercentileInTime

```
PercentileInTime('Expression','Interval','Percentile'=0.5)
```

Arguments

'Expression'	Signal or expression for which to form the percentile
'Interval'	Length of the interval over which to form the percentile
'Percentile'	Percentile

Description

This function returns the percentile of 'Expression' over each time interval of the length 'Interval'.

MPercentile

```
MPercentile('Expression','Interval','Percentile'=0.5)
```

Arguments

'Expression'	Signal or expression for which to form the percentile
'Interval'	Length of the moving interval over which to form the percentile
'Percentile'	Percentile

Description

This function returns the moving percentile of 'Expression' over each interval of the length 'Interval'. You can use this function for time based signals ('Interval' in seconds) as well as for length based signals ('Interval' in meters).

6.7 Correlation and covariance (Correl, CoVar)

Correl

```
Correl('Expression1','Expression2')
```

Arguments

'Expression1/2'	Signals or expressions for which to calculate the correlation coefficient

Description

This function calculates the correlation coefficient between 'Expression1' and 'Expression2'. The entire recording length is taken into account. The function returns a constant value.

Mcorrel

```
Mcorrel('Expression1','Expression2','Interval')
```

Arguments

'Expression1/2'	Signals or expressions for which to calculate the correlation coefficient
'Interval'	Length of the interval over which to calculate the correlation coefficient

Description

This function calculates the correlation coefficient between 'Expression1' and 'Expression2' over floating intervals of the length 'Interval' measured in s, m, Hz or 1/m.

CoVar

```
CoVar('Expression1','Expression2')
```

Arguments

'Expression1/2'	Signals or expressions for which to calculate the covariance

Description

This function calculates the covariance between 'Expression1' and 'Expression2'. The entire recording length is taken into account. The function returns a constant value.

MCoVar

```
MCoVar('Expression1','Expression2','Interval')
```

Arguments

'Expression1/2'	Signals or expressions for which to calculate the covariance
'Interval'	Length of the interval over which to calculate the covariance

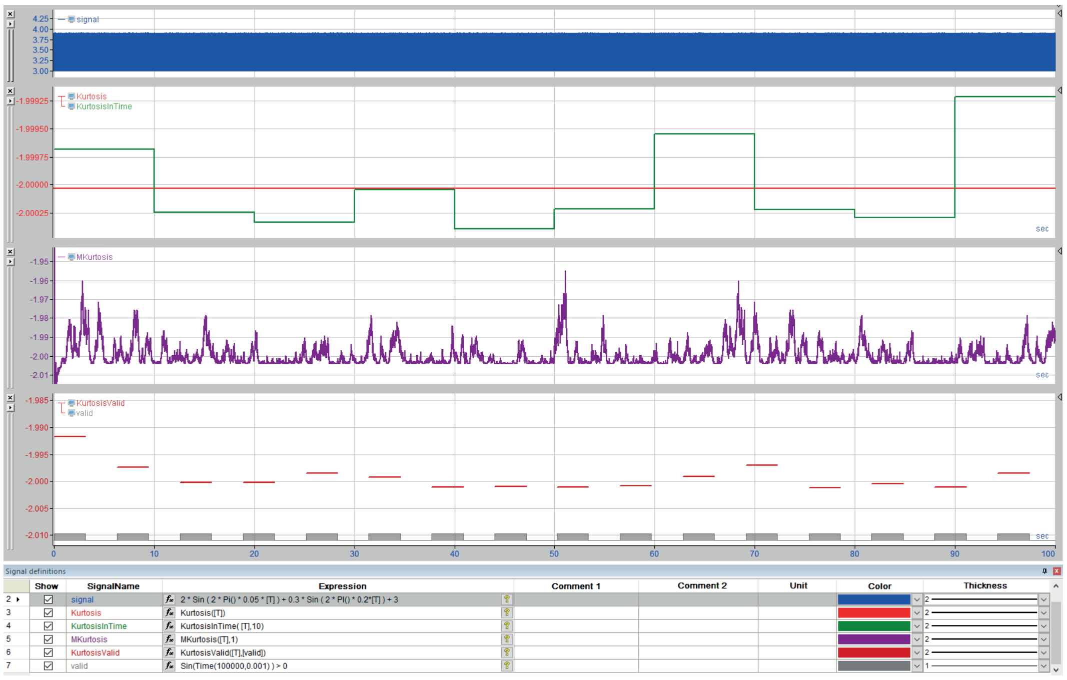
Description

This function calculates the covariance between 'Expression1' and 'Expression2' over floating intervals of the length 'Interval' measured in s, m, Hz or 1/m.

6.8 Kurtosis

The calculation of the *kurtosis* is used, for example, for the evaluation and analysis of vibrations. It serves to determine the number of outliers within an vibration 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).



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
2	✓	signal	2 * Sin (2 * Pi() * 0.05 * [T]) + 0.3 * Sin (2 * PI() * 0.2*[T]) + 3				🟦	2
3	✓	Kurtosis	Kurtosis([T])				🟥	2
4	✓	KurtosisInTime	KurtosisInTime([T],10)				🟩	2
5	✓	MKurtosis	MKurtosis([T],1)				🟪	2
6	✓	KurtosisValid	KurtosisValid([T],[valid])				🟥	2
7	✓	valid	Sin(Time(100000,0.001)) > 0				⬛	1

This statistical method is particularly suitable for analyzing random or stochastic signals, e.g., in terms of condition-based maintenance (condition monitoring) when analyzing vibrations.

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 vibration components. In the event of damage, individual pulse signals interfere with this signal, altering the distribution function. By choosing suitable characteristic values such as the *crest factor* or the *kurtosis factor,* the condition of the machine can be evaluated.

If regularly measured, these methods offer an overview of the machine status. However, the disadvantage is that after they increase, the characteristic values decrease again. The reason for this is that the number of pulse signals increases with progressive damage. This in turn influences the effective value but barely effects the peak value.

Modifications of the time signal caused by shock pulses induce a change in the resulting distribution function. Thus, damages with a distinctly discrete nature can cause the *kurtosis factor* to increase sharply. Its absolute value thus allows statements on a damage.

The calculation of the kurtosis is similar to the calculation of the standard deviation *StdDev*. *ibaAnalyzer* uses the following formula:

$$\frac{(n+1)n}{(n-1)(n-2)(n-3)} * \frac{\sum_{i=1}^{n}(x_i-\bar{x})^4}{s^4} - 3 * \frac{(n-1)^2}{(n-2)(n-3)}$$

n: Number of measured values

s: Standard deviation

Kurtosis

`Kurtosis('Expression')`

Description

This operation returns the kurtosis of the selected time signal.

KurtosisInTime

`KurtosisInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the kurtosis
'Interval'	Length of the interval over which to calculate the kurtosis

Description

With this operation, the selected expression is divided into equal-duration intervals of the length 'Interval'. For these intervals, the kurtosis is subsequently calculated.

KurtosisValid

`KurtosisValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which to form the kurtosis
'Valid'	Control signal

Description

This operation describes the kurtosis for those intervals in which a related control signal is TRUE.

MKurtosis

`MKurtosis('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the kurtosis
'Interval'	Length of the interval over which to form the kurtosis (in seconds)

Description

This operation calculates the kurtosis of 'Expression' over a floating X-axis interval of fixed length 'Interval'.

6.9 Skewness

Like the kurtosis factor, you can use the skewness factor for evaluating and analyzing vibrations. You can use the skewness factor to check the symmetrical properties of a vibration signal.

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

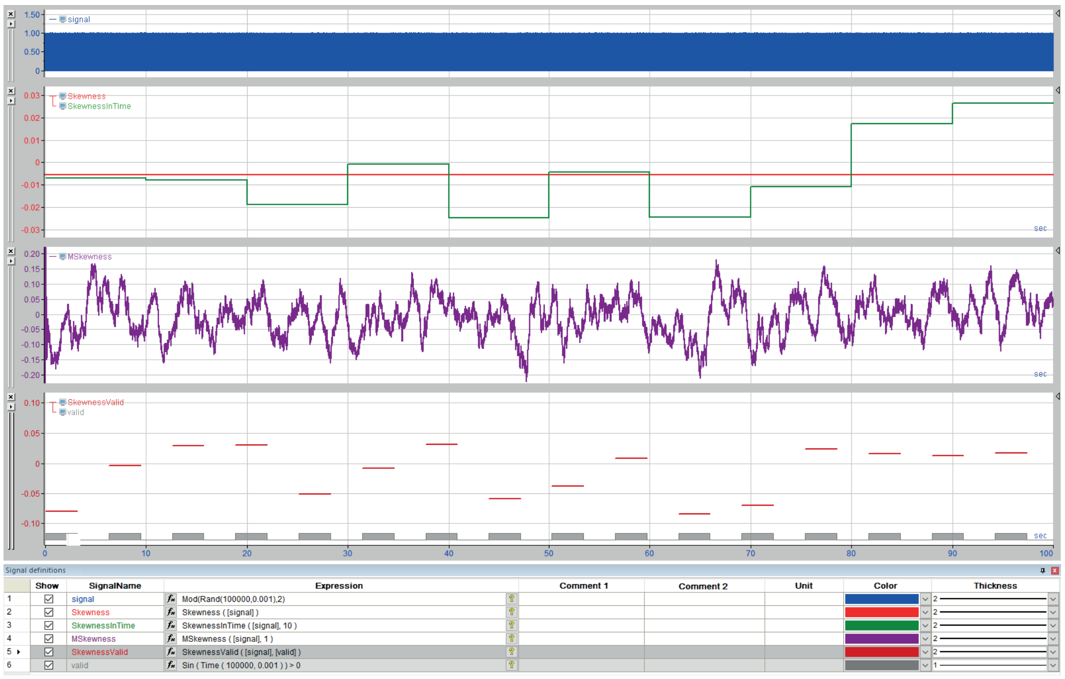
The calculation of the skewness is similar to the kurtosis and standard deviation functions. *ibaAnalyzer* uses the following formula:

$$\frac{n}{(n-1)(n-2)} * \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{s^3}$$

n: Number of measured values

s: Standard deviation

The different skewness calculations provide the following results:



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	signal	Mod(Rand(100000,0.001),2)				🟦	2
2	✓	Skewness	Skewness ([signal])				🟥	2
3	✓	SkewnessInTime	SkewnessInTime ([signal], 10)				🟩	2
4	✓	MSkewness	MSkewness ([signal], 1)				🟪	2
5	✓	SkewnessValid	SkewnessValid ([signal], [valid])				🟥	2
6	✓	valid	Sin (Time (100000, 0.001)) > 0				⬛	1

Skewness

```
Skewness('Expression')
```

Description

This operation returns the skewness of the selected time signal 'Expression'.

SkewnessInTime

`SkewnessInTime('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the skewness
'Interval'	Length of the interval over which to calculate the skewness

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

SkewnessValid

`SkewnessValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression for which to form the skewness
'Valid'	Control signal

Description
This operation calculates the skewness over those intervals in which a related control signal is TRUE.

MSkewness

`MSkewness('Expression','Interval')`

Arguments

'Expression'	Signal or expression for which to form the skewness
'Interval'	Length of the moving interval over which to calculate the skewness

Description
This operation calculates the skewness of 'Expression' over a floating X-axis interval of length 'Interval'.

6.10 CountSamples

```
CountSamples('Expression','Reset'=FALSE)
```

Arguments

'Expression'	Signal or expression for which to determine the number of signal points	
'Reset'	Optional binary parameter to reset or suppress the counting process. 'Reset' can be an expression as well.	
	'Reset'=TRUE()	Counting process is reset.
	'Reset'=FALSE()	counting process released. (default)

Description

This function determines the number of the individual signal points regardless of whether the signal points are equidistant or not. Invalid samples are not counted. If the input signal is invalid, the constant value 0 is supplied as the result.

Tip

You can also use this function, for example, in combination with *XMarkValid*, see XMark functions ↗ *XMarkRange and XMarkValid*, page 64.

6.11 Sort and XSort

Sort

`Sort('Expression','Descending'=FALSE)`

Arguments

'Expression'	Signal or expression for which to sort the samples
'Descending'	Optional binary parameter for reversing the sorting sequence

Description

This function sorts all samples of a curve ('Expression') by their values in ascending order from left to right.

Default: Sorting in ascending order ('Descending'=FALSE). If you want to sort the samples in descending order from left to right, set 'Descending' to TRUE.

XSort

`XSort('Expression','Descending'=FALSE)`

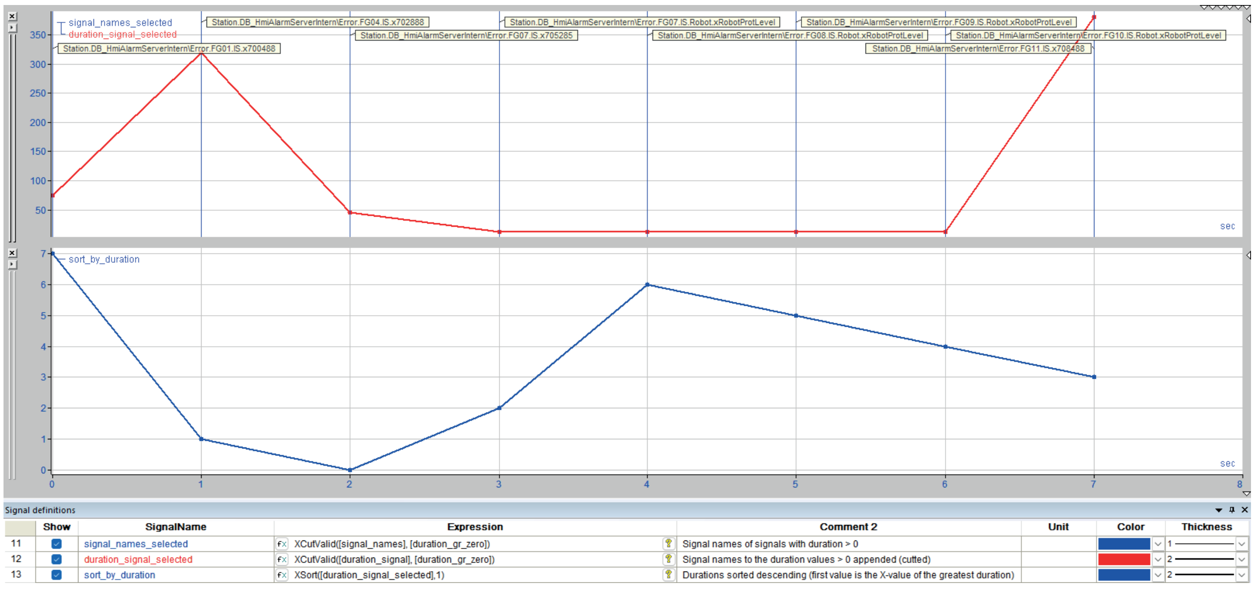
Arguments

'Expression'	Signal or expression for which to sort the samples
'Descending'	Optional binary parameter for reversing the sorting sequence

Description

This function sorts all (X,Y)-pairs of a curve ('Expression') by their Y-values in ascending order from left to right and returns X-values. This means that the last Y-value of the sorted signal corresponds to the X-value at which 'Expression' has its maximum. The X-axes and Y-axes of the sorted signal are the same length.

Default: Sorting in ascending order ('Descending'=FALSE). If you want to sort the pairs in descending order from left to right (the last value corresponds to the X-value at which 'Expression' reaches its minimum), set 'Descending' to TRUE.


signal_names_selected
duration_signal_selected
Station.DB_HmiAlarmServerIntern\Error.FG01.IS.x700488
Station.DB_HmiAlarmServerIntern\Error.FG04.IS.x702888
Station.DB_HmiAlarmServerIntern\Error.FG07.IS.x705285
Station.DB_HmiAlarmServerIntern\Error.FG07.IS.Robot.xRobotProtLevel
Station.DB_HmiAlarmServerIntern\Error.FG08.IS.Robot.xRobotProtLevel
Station.DB_HmiAlarmServerIntern\Error.FG09.IS.Robot.xRobotProtLevel
Station.DB_HmiAlarmServerIntern\Error.FG10.IS.Robot.xRobotProtLevel
Station.DB_HmiAlarmServerIntern\Error.FG11.IS.x708488
sort_by_duration
sec
Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
11	✓	signal_names_selected	XCutValid([signal_names], [duration_gr_zero])	Signal names of signals with duration > 0		🟦	1
12	✓	duration_signal_selected	XCutValid([duration_signal], [duration_gr_zero])	Signal names to the duration values > 0 appended (cutted)		🟥	2
13	✓	sort_by_duration	XSort([duration_signal_selected],1)	Durations sorted descending (first value is the X-value of the greatest duration)		🟦	2

7 Time length functions

7.1 ConvertBase

```
ConvertBase('Expression','From','To')
```

Arguments

'Expression'	Signal or expression for which to modify the base
'From' 'To'	Setup of the base to switch from or to 0 = time 1 = length 2 = frequency 3 = inverse length

Description
This operation converts an expression from one base into another base. No physical conversion or scaling is carried out.

You can use this function to change the reference value of a signal. This can be useful if length based reference values are used for additional calculations. The existing signal, however, is only time based.

7.2 Resample

```
Resample('Expression','Basis','interpolate'=TRUE)
```

Arguments

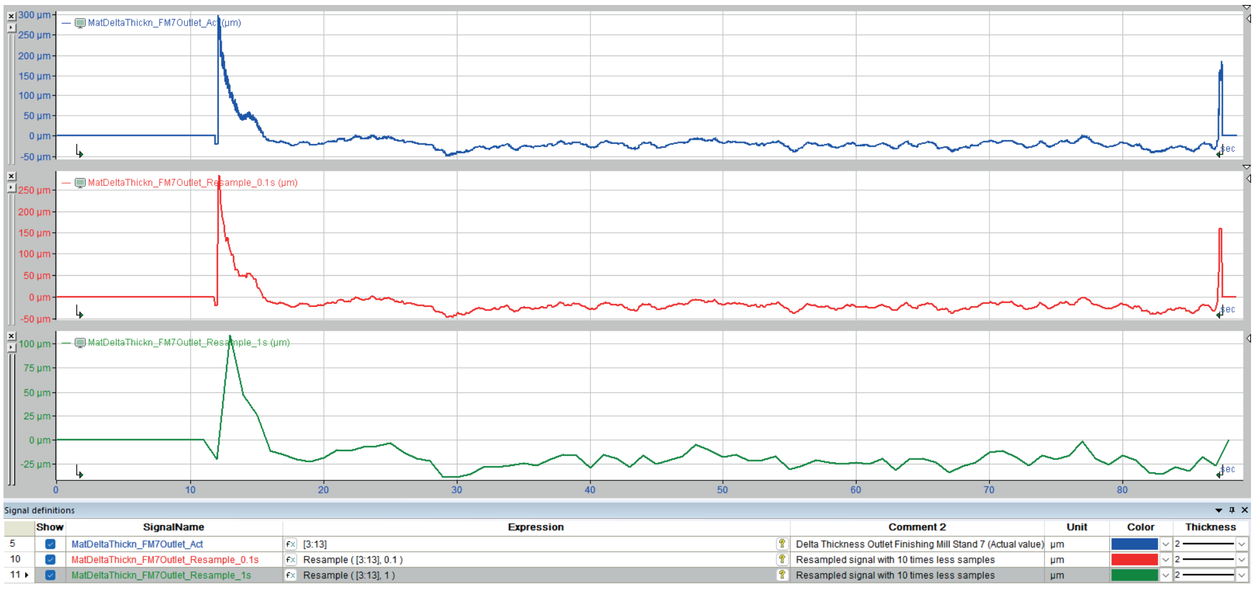
'Expression'	Signal or expression to resample
'Base'	New sampling rate of the result
'interpolate'	Optional parameter to prevent the automatic interpolation for the new measured values

Description

This operation returns the signal trend of 'Expression' with a new time basis. The momentary values are transferred from the original curve chronologically correct in line with the new time base, so that the length of the new curve remains practically the same. You can also use this function for length based signals. In this case, enter the value of a distance in m rather than a time value.

Tip

A curve can be graphically smoothed if a larger time basis is used in the *Resample* function because fewer points are connected to each other. The values are not averaged.



MatDeltaThickn_FM7Outlet_Act (µm)
MatDeltaThickn_FM7Outlet_Resample_0.1s (µm)
MatDeltaThickn_FM7Outlet_Resample_1s (µm)
sec
Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	MatDeltaThickn_FM7Outlet_Act	[3:13]	Delta Thickness Outlet Finishing Mill Stand 7 (Actual value)	µm	🟦	2
10	✓	MatDeltaThickn_FM7Outlet_Resample_0.1s	Resample ([3:13], 0.1)	Resampled signal with 10 times less samples	µm	🟥	2
11	✓	MatDeltaThickn_FM7Outlet_Resample_1s	Resample ([3:13], 1)	Resampled signal with 10 times less samples	µm	🟩	2

7.3 Time functions

7.3.1 Time

```
Time('Count','Basis')
```

Arguments

'Count'	Number of measuring points to be created
'Basis'	Sampling rate of the resulting signal

Description
This function returns a linear, time-proportional signal with a number of 'Count' values at a distance of 'base'. The time base is stated in seconds. The time values are entered both on the X-axis and on the Y-axis.

Note

You do not need to load a data file as a precondition for using the time function.

7.3.2 InfoFieldTime

```
InfoFieldTime('FileIndex',"'InfoField'",'Begin'=0,'End'=Text end)
```

Arguments

'FileIndex'	Index of the data file
'InfoField'	Info field to read; put in quotation marks.
'Begin'	Optional: First character of the field content to be read. If you do not specify a value, the entire content is read.
'End'	Optional: Last character of the field content to be read. If you do not specify a value, the content is read from 'Begin' to the last character.

Description

This function interprets the content of an info field as time, if possible, and returns the relative time since the start of *ibaAnalyzer* in seconds. The result is a constant value in the graph.

Note

The *InfoFieldTime* function expects that the specified info field can be interpreted as time (date, hours, minutes, seconds, ...). If this is not possible, an empty graph is generated.

If the info field is to be read out as a numerical value or as text, you must use the *InfoField* or *InfoFieldText* functions.

Note

If you double-click on the desired info field in the signal tree that describes a time (e.g. "starttime"), *ibaAnalyzer* automatically inserts the corresponding function as new signal into the signal table. If required, you then only have to adjust the signal name, beginning and end.

This method also works in the input box of the Expression builder. The function will then be inserted at the cursor position.

7.3.3 AbsoluteTime and RelativeTime

Absolute Time

```
AbsoluteTime('Time','DoSync'=FALSE)
```

Arguments

'Time'	Relative time to be converted
'DoSync'	Optional parameter that determines whether to align the absolute time with the starting time of the currently displayed time axis

Description

This function transforms the relative time information 'Time' (e.g. generated with *XFirst*, *XLast* or *XValues*) into absolute time. A vector with constant or varying entries is returned here depending on whether the input time is constant or not. The optional binary parameter 'DoSync' determines whether to align the result with the starting time of the currently displayed time axis.

The result is a vector with the following entries, which can be read out with *GetRows*:

- Index 0: milliseconds
- Index 1: seconds
- Index 2: minutes
- Index 3: hours
- Index 4: day of the month
- Index 5: month
- Index 6: year
- Index 7: day of the year
- Index 8: Weekday (1=Monday, 2=Tuesday, … , 7=Sunday)

Note

If you specify 'Time' 0, the function returns the start time of the data file, or the UNIX epoch start time (01/01/1970, 00:00:00) if no data file is loaded.

RelativeTime

```
RelativeTime('Year','Month','Day','Hour','Minute','Second','Millisecond'=0)
```

Arguments

'Year'	Year of the absolute time
'Month'	Month of the absolute time
'Day'	Day of the absolute time
'Hour'	Hour of the absolute time
'Minute'	Minute of the absolute time
'Second'	Second of the absolute time
'Millisecond'	Millisecond of the absolute time

Description

This function transforms an absolute time (e.g. generated with *AbsoluteTime*) into relative time, i.e. the time since the start of *ibaAnalyzer* in seconds. The optional parameter 'Millisecond' can increase the accuracy. The result is a constant value in the graph.

7.4 Conversion from time to length reference (TimeToLength)

TimeToLength

`TimeToLength('Expression','Speed','Precision')`

Arguments

'Expression'	Signal or expression to convert to length
'Speed'	Speed signal
'Precision'	Optional parameter to determine the sampling rate of the resulting signal in m

Description

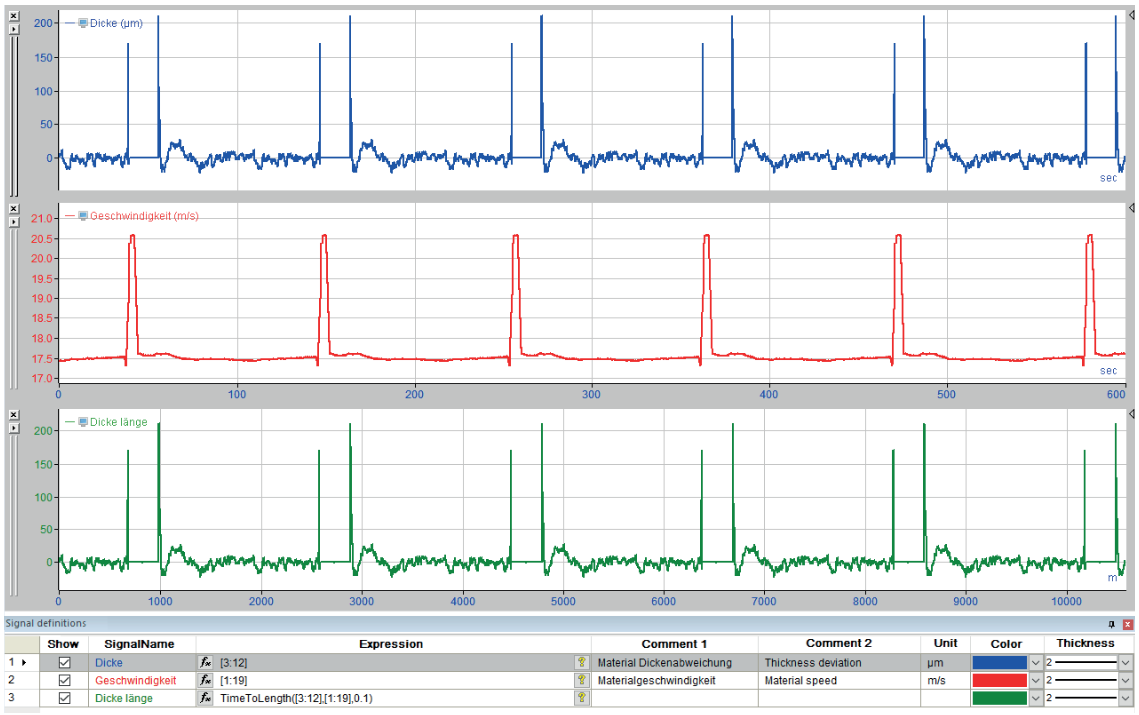
This function converts the time based signal 'Expression' into a length based signal, with the speed of the measuring object 'Speed' serving as the speed vector in m/s.

This function can convert any signal for which a matching speed signal is available into a length based presentation. This means that it is possible to present not just the relationship between measured value and time but also between measured value and distance traveled.

Taking the example of a steel strip in a rolling mill, this function can determine the distribution of measured values over the strip length. If the process was designed in such a manner that the beginning and end of measurement are in exact conformity with the head and tail ends of the strip, this function can then calculate the total strip length. The largest length value determined is entered as the scale end value of the X-axis (autoscale).

'Precision' is an optional parameter in m. If no precision value is defined, the points for the length based curve are calculated and entered in the graph on the basis of the number of measuring points of the original signal. If a precision value is defined, for example, 0.1, a new length-related value is calculated and entered as a point of the curve every 0.1 m.

The following illustration shows time-length function *TimeToLength*.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Dicke	[3:12]	Material Dickenabweichung	Thickness deviation	µm	🟦	2
2	✓	Geschwindigkeit	[1:19]	Materialgeschwindigkeit	Material speed	m/s	🟥	2
3	✓	Dicke länge	TimeToLength([3:12],[1:19],0.1)				🟩	2

TimeToLengthL

```
TimeToLengthL('Expression','Length','Precision')
```

Arguments

'Expression'	Signal or expression to convert to length
'Length'	Length signal
'Precision'	Optional parameter to determine the sampling rate of the resulting signal in m

Description

This function converts the time based signal 'Expression' into a length based signal, with a length signal 'Length' as the position in m.

The explanations given under *TimeToLength* apply accordingly, however, with the only difference that a suitable length or position signal is used instead of the speed.

8 X-axis operations

8.1 FillGaps

`FillGaps('Expression')`

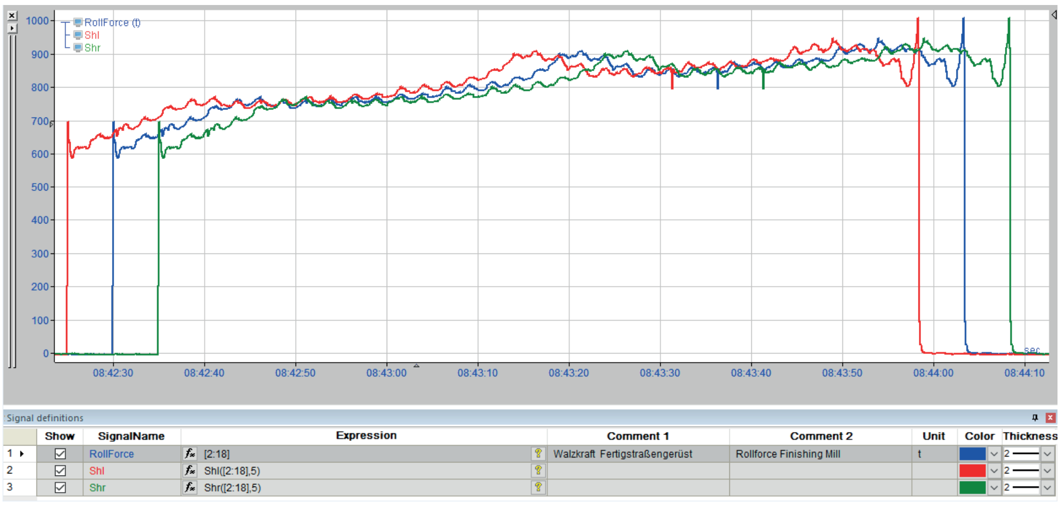
Description

You can use the function *FillGaps* to fill gaps in a signal 'Expression' through linearly interpolated entries.

This function is especially useful when gaps are created by NULL entries during trend queries from a database.

8.2 Shift along the X-axis

The following illustration shows the functions shift left (red) and shift right (green).



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	RollForce	[2:18]	Walzkraft Fertigstraßengerüst	Rollforce Finishing Mill	t	🟦	2
2	✓	Shl	Shl([2:18],5)				🟥	2
3	✓	Shr	Shr([2:18],5)				🟩	2

Arguments

'Expression'	Signal or expression to be moved
'Distance'	Distance in seconds or meters for length-related signals

You can use the functions for time based signals ('Distance' in seconds) as well as for length based signals ('Distance' in meters).

Shl

`Shl('Expression','Distance')`

This operation returns a signal curve which is shifted by 'Distance' to the left on the X-axis compared to the original signal. Otherwise, the signal remains unchanged.

Shr

`Shr('Expression','Distance')`

This operation returns a signal curve which is shifted by 'Distance' to the right on the X-axis compared to the original signal. Otherwise, the signal remains unchanged.

8.3 VarDelay

```
VarDelay('Expression','Delay')
```

Arguments

'Expression'	Signal or expression
'Delay'	Time delay in seconds

Description
This operation returns the 'Expression' delayed by a time constant 'Delay', which can also be negative.

8.4 XAlignFft

```
XAlignFft('FixedExpression','AlignExpression','Start','End','MinScale',
'MaxScale','ScaleStep','QualityParam')
```

Arguments

'FixedExpression'	Signal or expression that serves as a reference
'AlignExpression'	Signal or expression that is adjusted to the reference
'Start'	Beginning of the X-axis range relative to the zero point of 'FixedExpression'
'End'	End of the X-axis range
'MinScale'	Smallest x-scaling factor to be checked
'MaxScale'	Biggest x-scaling factor to be checked
'ScaleStep'	Step size for controlling the precision and speed of the algorithm
'QualityParam'	Quality criterion to determine the distance measurement between time series

Description
This functions align length based signals with the same physical significance that were measured at different points in the process.

Some parameters are described in more detail below.

'FixedExpression'
A thickness measurement being considered "fixed" in the course of an algorithm, i.e. not scalable or shiftable. This should be the thickness measurement containing the profile of the other measurement. (In the hot strip vs. cold strip comparison, this would be the hot strip.)

'AlignExpression'
The result of the alignment later refers to this thickness measurement. Thus, this measurement has to be scaled and shifted with the result values.

'Start'
The interval from 'Start' to 'End' indicates the X-intercept where the signal 'FixedExpression' can be moved. The zero here is the zero of the expression. Also negative values are permitted. If the measurement 'AlignExpression' is allowed to protrude 10 axis units on the left-hand side in the *ibaAnalyzer* compared to 'FixedExpression', then the following must apply: Start = -10 .

'End'

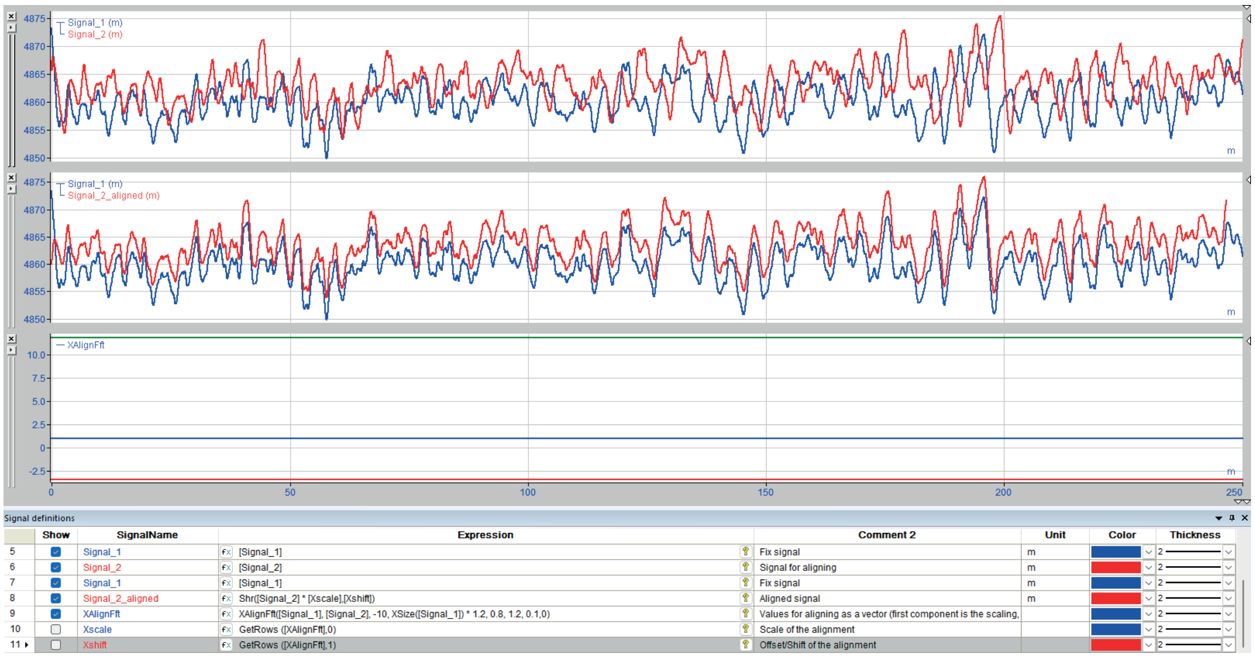
This parameter specifies the end of the interval just described. It is recommended to select this end dependent on the length of 'FixedExpression'. So, for example,
End = XSize([FixedExpression]) or End = 1.2 * Xsize([FixedExpression]) if an surplus of 20 % is allowed.

'ScaleStep'

This parameter controls the ratio between precision and speed. The smaller the value, the slower and more reliable the algorithm works. The higher the value, the more the algorithm is accelerated by a heuristic. In case of too high values for 'ScaleStep', this can lead to a wrong result. For an optimal result, it is recommended to transfer the resolution of the measuring data. If, for example, the samples have a distance of 10 cm, then 'ScaleStep' = 0.1. If the calculation of results takes too long, the value can be increased.

'QualityParam'

The parameter defines the quality criterion used by the function (distance measure between time series). The default setting 0 represents for the Mean Squared Error (MSE), and 1 refers to the Gleichläufigkeit coefficient (GLC).



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Signal_1	[Signal_1]	Fix signal	m	🟦	2
6	✓	Signal_2	[Signal_2]	Signal for aligning	m	🟥	2
7	✓	Signal_1	[Signal_1]	Fix signal	m	🟦	2
8	✓	Signal_2_aligned	Shr([Signal_2] * [Xscale],[Xshift])	Aligned signal	m	🟥	2
9	✓	XAlignFft	XAlignFft([Signal_1], [Signal_2], -10, XSize([Signal_1]) * 1.2, 0.8, 1.2, 0.1,0)	Values for aligning as a vector (first component is the scaling,		🟦	2
10		Xscale	GetRows ([XAlignFft],0)	Scale of the alignment		🟦	2
11		Xshift	GetRows ([XAlignFft],1)	Offset/Shift of the alignment		🟥	2

8.5 XBase and XOffset

XBase

`XBase('Expression')`

Description

This function determines the recording base (the interval between samples). The base can be time based, length based, or frequency based.

In case of an equidistantly sampled signal, the function provides the distance between two samples in X-axis units.

If the samples of a signal do not have the same distance, the distance will be displayed in X-axis units which would be determined when re-sampling on equidistant samples. By default, this is the smallest distance between two samples of the signal.

XOffset

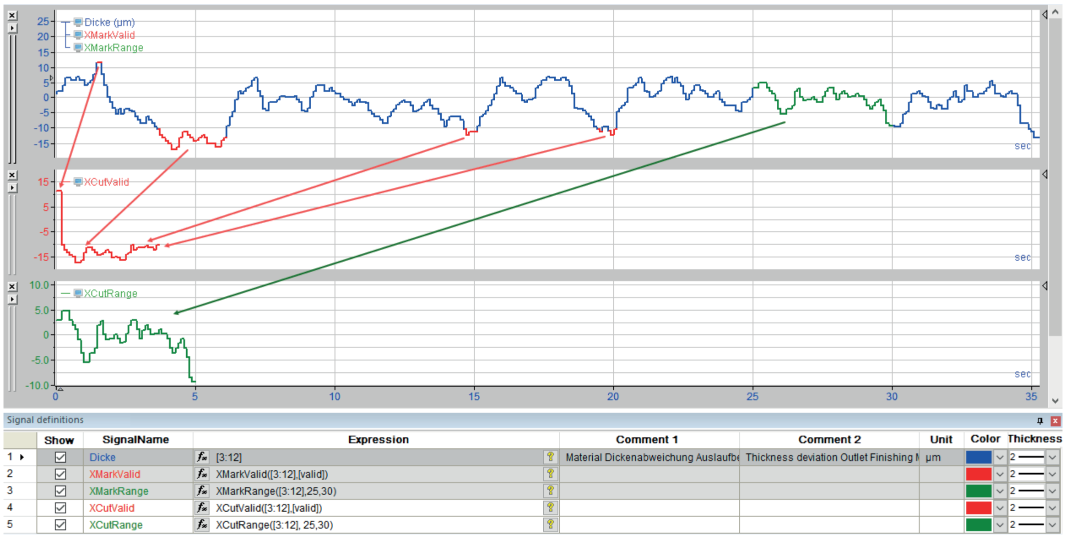
`XOffset('Expression')`

Description

This function provides the time distance of the first sample of a signal from the beginning of the data file in seconds. The result is negative if the signal starts earlier and positive if it starts later.

If several data files are opened at the same time and the *Synchronize files on recording time* option is enabled, the offset is not necessarily determined relative to the start of the selected signal's data file, but rather to the start of the data file with the earliest start time.

8.6 XCutRange and XCutValid



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Dicke	[3:12]	Material Dickenabweichung Auslaufbe	Thickness deviation Outlet Finishing M	µm	🟦	2
2	✓	XMarkValid	XMarkValid([3:12],[valid])				🟥	2
3	✓	XMarkRange	XMarkRange([3:12],25,30)				🟩	2
4	✓	XCutValid	XCutValid([3:12],[valid])				🟥	2
5	✓	XCutRange	XCutRange([3:12], 25,30)				🟩	2

XCutRange

`XCutRange('Expression','From','To')`

Arguments

'Expression'	Signal or expression from which to cut out a part
'From'	Start of the selected range in seconds or meters
'To'	End of the selected range in seconds or meters

Description

You can use this function to cut out a part of a curve. The 'From' and 'To' parameters define the beginning and end of the segment to be cut out.

The cut out part is moved to the beginning of a separate graph. However, since the X-axis (time or length) remains unchanged, the correct time or length reference of the signal is no longer given.

You can use the function for time based signals ('From' and 'To' in seconds) as well as for length based signals ('From' and 'To' in meters).

XCutValid

`XCutValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression from which to cut out a part
'Valid'	Binary signal that describes the selected range

Description

This function cuts out all the samples of a signal trend 'Expression' depending on a 'Valid' condition if this condition supplies the value TRUE. The 'Valid' parameter is a Boolean expression. This can be a digital signal, the result of a comparative operation, or any other binary expression. Samples for which the condition is FALSE are not used.

The cut out parts are placed, one after another, at the beginning of a new graph.

You can use the function for time based signals as well as for length based signals.

8.7 XFirst, XLast and XNow

XFirst

```
XFirst('Expression','Skip'=0,'SkipInitialEdge'=FALSE)
```

Arguments

'Expression'	(Boolean) input signal
'Skip'	Optional for skipping the first rising edges
'SkipInitialEdge'	Decides whether the first sample is counted as a rising edge if it is TRUE

Description

This function returns that value on the X-axis (time in s or position in m) for which the 'Expression' is TRUE for the first time. 'Expression' is a Boolean expression. This can be a digital signal, the result of a comparative operation, or any other binary expression.

The 'Skip' parameter allows you to skip edges. For example, to get the value of the third rising edge, set 'Skip'=2.

Tip

To return the value of a falling edge, use the following expression:

```
XFirst(NOT('Expression'))
```

XLast

```
XLast ('Expression','Skip'=0,'SkipFinalEdge'=FALSE)
```

Arguments

'Expression'	(Boolean) input signal
'Skip'	Optional for skipping the last falling edges
'SkipFinalEdge'	Decides whether the last sample is counted as a falling edge if it is TRUE

Description

This function returns that value on the X-axis (time [s] or position [m]) for which the 'Expression' is TRUE for the last time. This means that 'Expression' must be a Boolean quantity. This can be a digital input signal, the result of a comparative operation, or any other binary expression.

The 'Skip' parameter allows you to skip edges. For example, to get the value of the third-to-last rising edge, set 'Skip'=2.

Tip

To return the value of a falling edge, use the following expression:

```
XLast(NOT('Expression'))
```

XNow

`XNow()`

Description

This function returns the relative time since the last start of *ibaAnalyzer*. If no data file is loaded, the function returns the relative time since the UNIX start time (01.01.1970; 00:00:00).

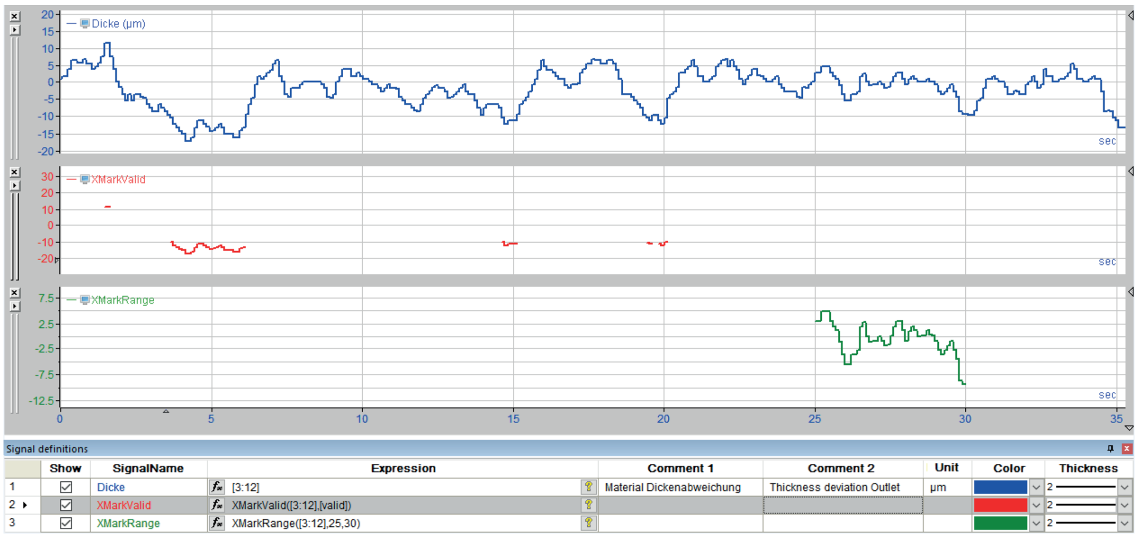
8.8 XMarker1 and XMarker2

`XMarker1 () and XMarker2 ()`

Description

This function returns the position of the marker X1 or X2 on the X-axis.

8.9 XMarkRange and XMarkValid



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Dicke	[3:12]	Material Dickenabweichung	Thickness deviation Outlet	µm	🟦	2
2	✓	XMarkValid	XMarkValid([3:12],[valid])				🟥	2
3	✓	XMarkRange	XMarkRange([3:12],25,30)				🟩	2

XMarkRange

`XMarkRange('Expression','From','To')`

Arguments

'Expression'	Signal or expression from which to select a part
'From'	Start of the selected range in seconds or meters
'To'	End of the selected range in seconds or meters

Description

You can use this function to cut out part of a curve – similar to the *XCutRange* function. The 'From' and 'To' parameters define the beginning and end of the segment to be cut out.

The part cut out is displayed in a separate graph, however, it stays in the original position on the time or position axis. The samples outside the specified range are discarded.

You can use the function for time based signals ('From' and 'To' in seconds) as well as for length based signals ('From' and 'To' in meters).

XMarkValid

`XMarkValid('Expression','Valid')`

Arguments

'Expression'	Signal or expression from which to select a part
'Valid'	Digital signal that describes the selected range

Description

Similar to the *XCutValid* function, this function cuts out all the samples of a signal trend 'Expression' depending on a 'Valid' condition if this condition supplies the value TRUE. The 'Valid' parameter is a Boolean expression. This can be a digital signal, the result of a comparative operation, or any other binary expression. Samples for which the condition is FALSE are not used.

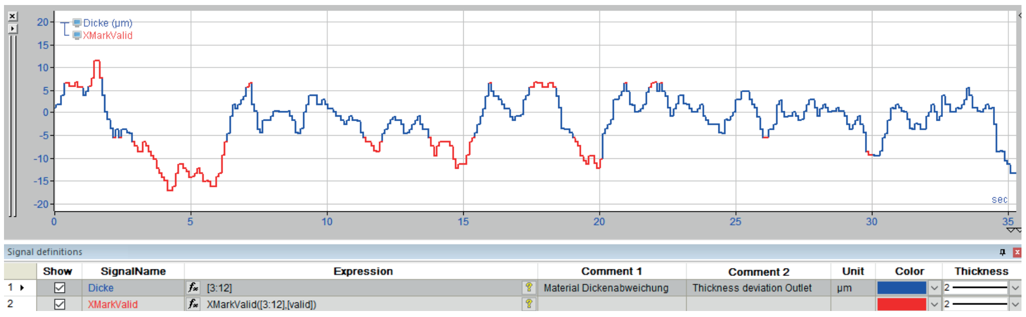
The cut out parts are displayed in a new graph, retaining their X positions.

You can use the function for time based signals as well as for length based signals.

Tip

The *XMarkValid* function is useful, for example, to highlight limit value violations by using different colors in a signal trend by showing the result signal in the same graph and on the same Y-axis as the original signal. By choosing different colors, you can easily identify ranges of the limit value violation.

Example: Values within the tolerance range = blue; values out of tolerance = red.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Dicke	[3:12]	Material Dickenabweichung	Thickness deviation Outlet	µm	🟦	2
2	✓	XMarkValid	XMarkValid([3:12],[valid])				🟥	2

8.10 XMirror, XStretch and XStretchScale

XMirror

`XMirror('Expression')`

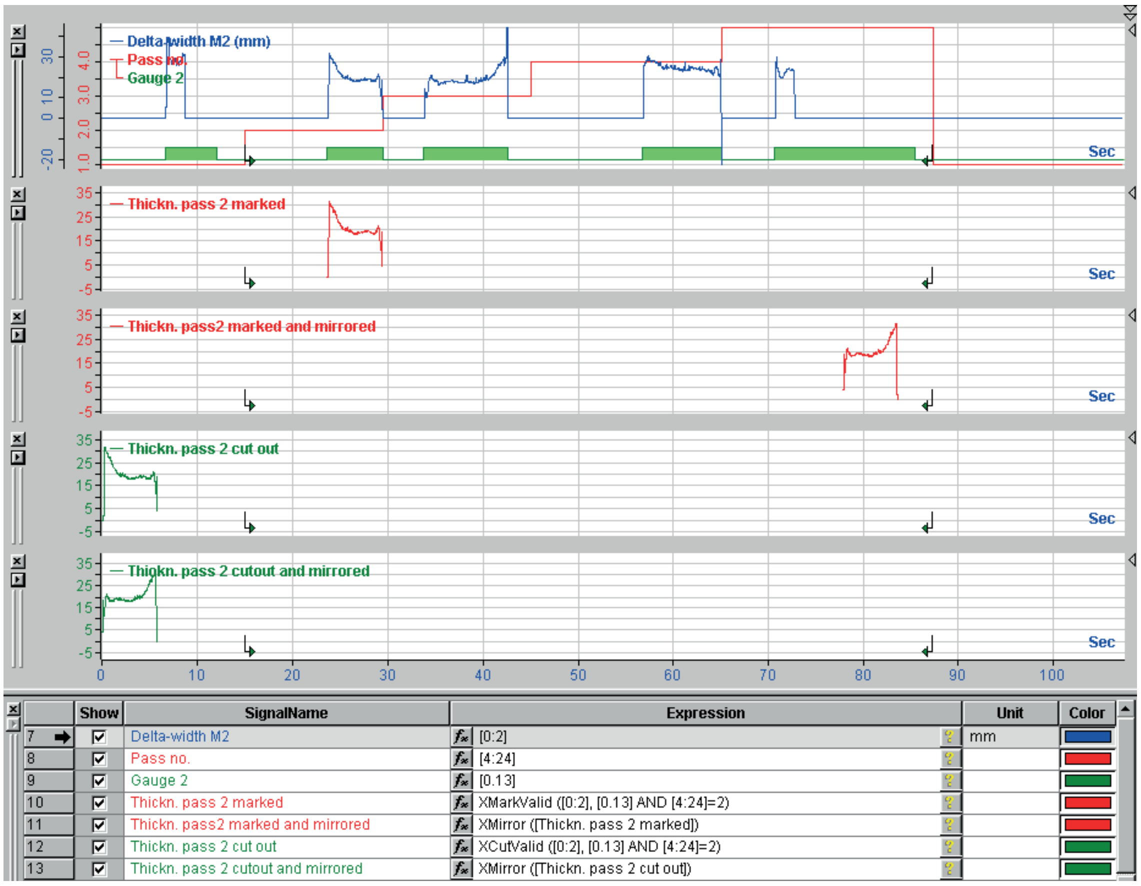
Description

You can use this function to mirror a complete graph (exchanging the beginning and end). The graph is mirrored around the vertical central axis of the entire signal graph.

You can use the function for time based signals as well as for length based signals.

In this way, you can compare measuring graphs of reversing processes (direction reversal) more easily. In rolling mills, for example, the head and tail end of the strip can be exchanged during (even) reversing passes in order to graphically neutralize the direction reversal. However, in order to compare several passes to each other, the corresponding measured values must first be cut out of the original signal using the *XCutValid* function, so that these values can be individually mirrored and subsequently placed on top of each other.

The figure shows the different results of the mirroring operation, depending on whether the segment to be mirrored was previously cut out using *XMarkValid* (red) or *XCutValid* (green).



	Show	SignalName	Expression	Unit	Color
7	☑	Delta-width M2	[0:2]	mm	🟦
8	☑	Pass no.	[4:24]		🟥
9	☑	Gauge 2	[0.13]		🟩
10	☑	Thickn. pass 2 marked	XMarkValid ([0:2], [0.13] AND [4:24]=2)		🟥
11	☑	Thickn. pass2 marked and mirrored	XMirror ([Thickn. pass 2 marked])		🟥
12	☑	Thickn. pass 2 cut out	XCutValid ([0:2], [0.13] AND [4:24]=2)		🟩
13	☑	Thickn. pass 2 cutout and mirrored	XMirror ([Thickn. pass 2 cut out])		🟩

XStretch

`XStretch('Expression','Reference expression')`

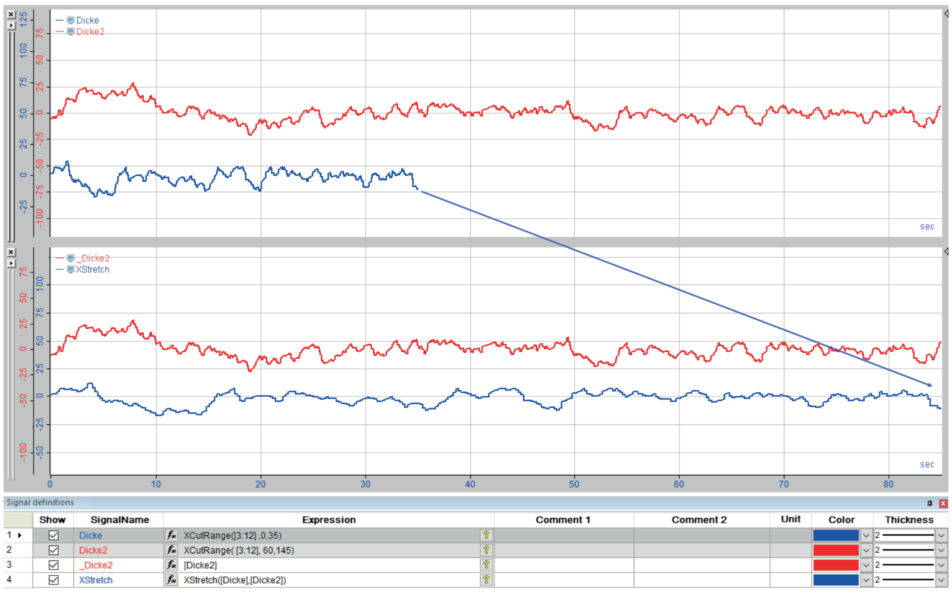
Description

You can use this function to graphically stretch the signal curve to the same (final) length of another signal.

You can use the function for time based signals as well as for length based signals.

In this way, it is, for example, possible to correlate measured values of a rolled strip from the roughing mill to those from the finishing mill or to compare the individual passes of a reversing mill to each other.

In the figure, the rolling force curve of the first pass (blue curve) is stretched to the final length corresponding to the ninth pass.



Signal definitions

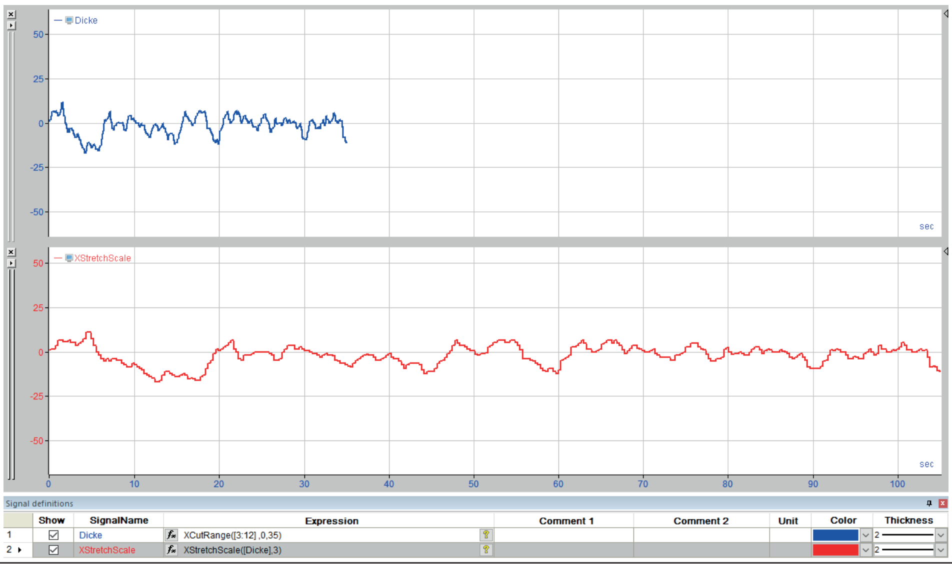
	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Dicke	XCutRange([3:12] ,0,35)				🟦	2
2	✓	Dicke2	XCutRange([3:12], 60,145)				🟥	2
3	✓	_Dicke2	[Dicke2]				🟥	2
4	✓	XStretch	XStretch([Dicke],[Dicke2])				🟦	2

XStretchScale

`XStretchScale('Expression','Scale')`

Description

You can use this function to stretch the signal curve by a specified factor. The scaling factor is also used if the curve is already provided with an offset.



Signal definitions

	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	Dicke	XCutRange([3:12] ,0,35)				🟦	2
2	✓	XStretchScale	XStretchScale([Dicke],3)				🟥	2

8.11 XSize and XSumValid

XSize

```
XSize ('Expression')
```

Description

This function returns the total length of 'Expression' in units of the X-axis (time in s or position in m). The result is at a constant value of 0 if the input signal is invalid.

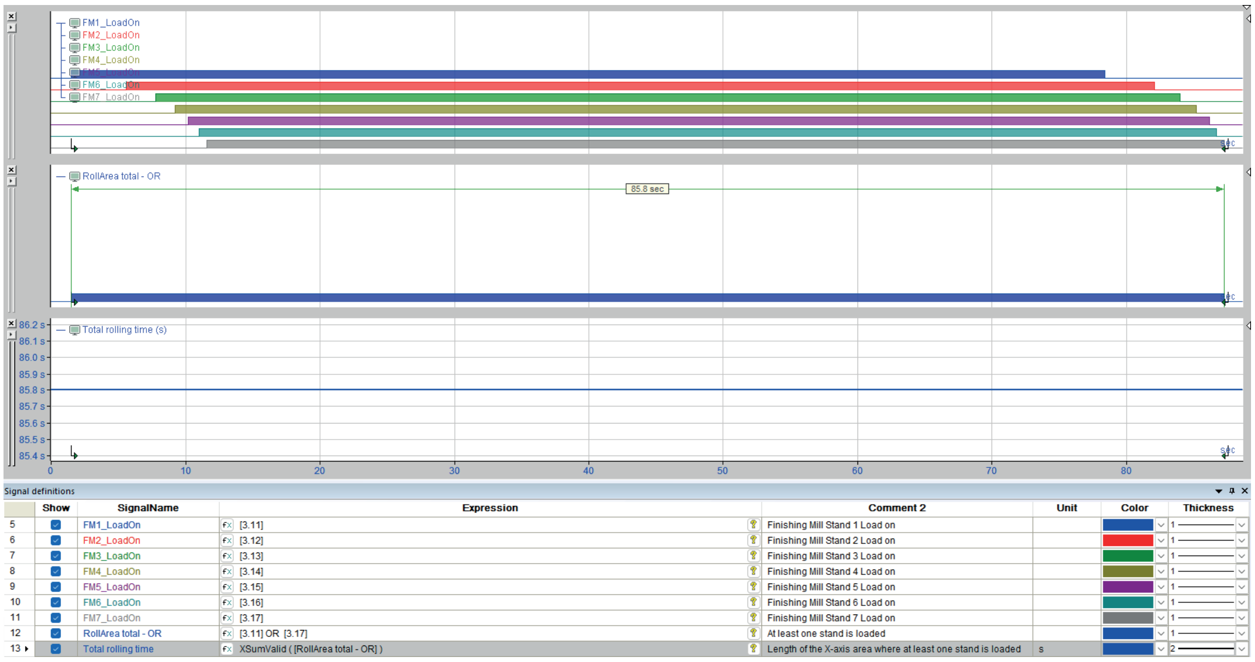
XSumValid

```
XSumValid ('Expression')
```

Description

This function determines the duration or length for which the condition 'Expression' is TRUE. Any samples for which the condition is not true (FALSE) are not used. 'Expression' is a Boolean quantity. This can be a digital signal, the result of a comparative operation, or any other binary expression.

If the input signal is invalid, the result is at a constant value of 0.



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	FM1_LoadOn	[3.11]	Finishing Mill Stand 1 Load on		🟦	1
6	✓	FM2_LoadOn	[3.12]	Finishing Mill Stand 2 Load on		🟥	1
7	✓	FM3_LoadOn	[3.13]	Finishing Mill Stand 3 Load on		🟩	1
8	✓	FM4_LoadOn	[3.14]	Finishing Mill Stand 4 Load on		🟫	1
9	✓	FM5_LoadOn	[3.15]	Finishing Mill Stand 5 Load on		🟪	1
10	✓	FM6_LoadOn	[3.16]	Finishing Mill Stand 6 Load on		🟩	1
11	✓	FM7_LoadOn	[3.17]	Finishing Mill Stand 7 Load on		⬜	1
12	✓	RollArea total - OR	[3.11] OR [3.17]	At least one stand is loaded		🟦	1
13	✓	Total rolling time	XSumValid ([RollArea total - OR])	Length of the X-axis area where at least one stand is loaded	s	🟦	2

8.12 XValues and YValues

XValues

`XValues('Expression')`

Description

This function returns the X-values for every sample of an expression. You can also use this function for signals or expressions which are not time based, i.e. length based (m), frequency-based (Hz) or inverse length based (1/m).

With a usual time continuous or length continuous signal, the function returns a rising straight line, writing the time or length values along the Y-axis in base units (s, m). The function also works with non-equidistant signals.

YValues

`YValues('Expression','TimeBase'=1)`

Arguments

'Expression'	Signal or expression
'TimeBase'	Time base of the resulting signal

Description

This function returns the Y-values for every sample of an expression. Regardless of whether the input signal is sampled equidistant or not, the result is an equidistant signal with the time base 'TimeBase'.

The specification of the time base is optional and 'TimeBase'=1 is used as the default value.

8.13 XY

```
XY('Expression1','Expression2','Precision')
```

Arguments

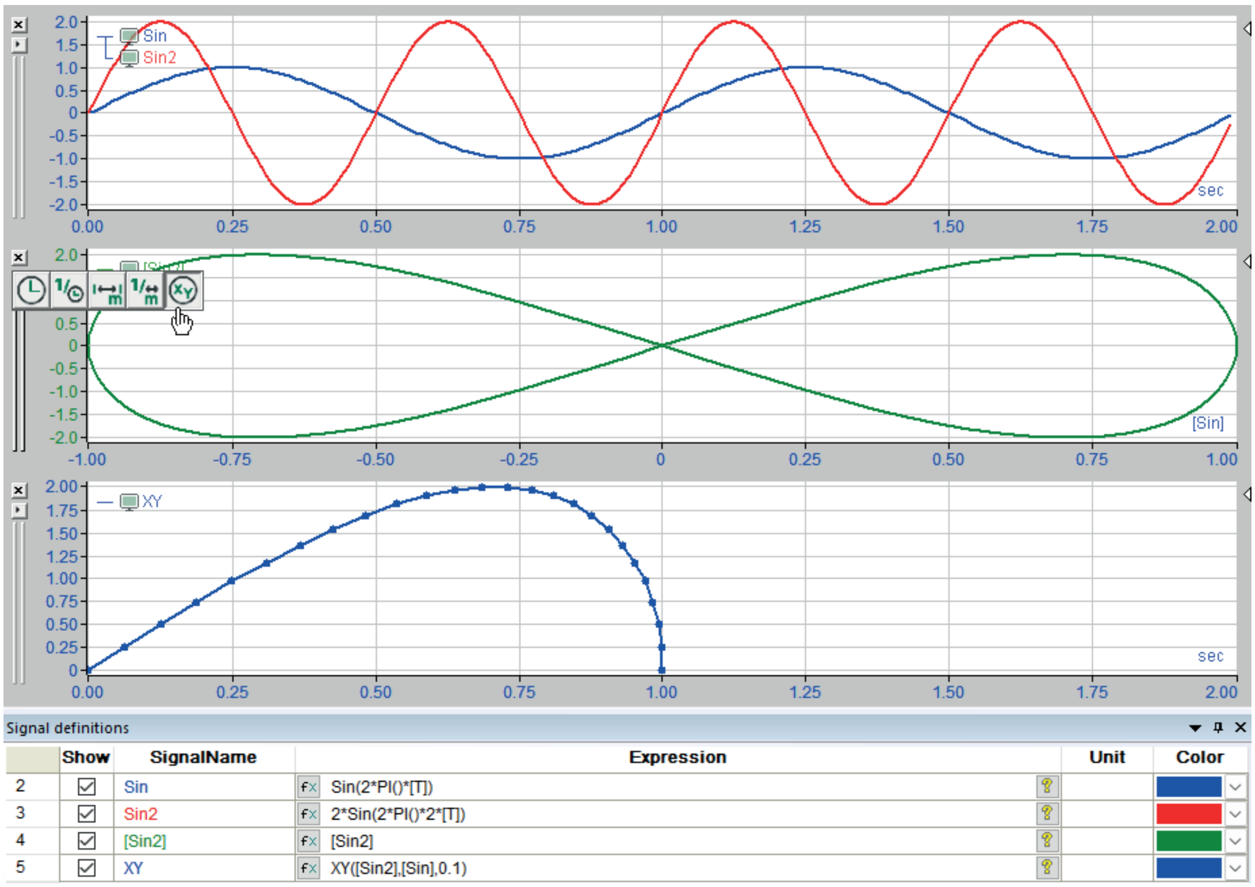
'Expression1'	Signal or expression containing the Y-values of the new signal
'Expression2'	Signal or expression containing the X-values of the new signal
'Precision'	Optional parameter to determine the sampling rate of the resulting signal

Description

You can use this function if the result from the X-Y presentation is to be used for additional operations. After the selection, the signals of the X-axis and Y-axis are assigned to the function.

Please note that in the resulting function, the distances between the signal points are not the same as the distances of the original signals. Also the distance between the signal points is different. With the 'Precision' parameter, you can determine a fixed distance between the signal points. If you do not enter a parameter, the shortest distance of the signal points is used as fixed value for all subsequent operations.

You can also determine the X-Y presentation using the X-axis mode. The illustration shows the results of the X-axis mode and the *XY* function in comparison.



Signal definitions

	Show	SignalName	Expression	Unit	Color
2	✓	Sin	Sin(2*PI()*[T])		🟦
3	✓	Sin2	2*Sin(2*PI()*2*[T])		🟥
4	✓	[Sin2]	[Sin2]		🟩
5	✓	XY	XY([Sin2],[Sin],0.1)		🟦

For more information, see part 2 of the *ibaAnalyzer* manual, chapter *X-axis mode X – Y*.

9 Vector operations

Vector operations extend the analysis options for 2-dimensional signals.

You can create vectors or arrays in different ways:

- Group several signals in *ibaPDA* and mark the group as vector.
- Arrange several signals in the logical expressions in *ibaAnalyzer*.
- Various calculation functions return vectors as results, e. g. FFT functions.

In *ibaAnalyzer*, you can display vectors in 2D top view and 3D view. Using the vector operations in the Expression builder, you can use the vector data for further calculations.

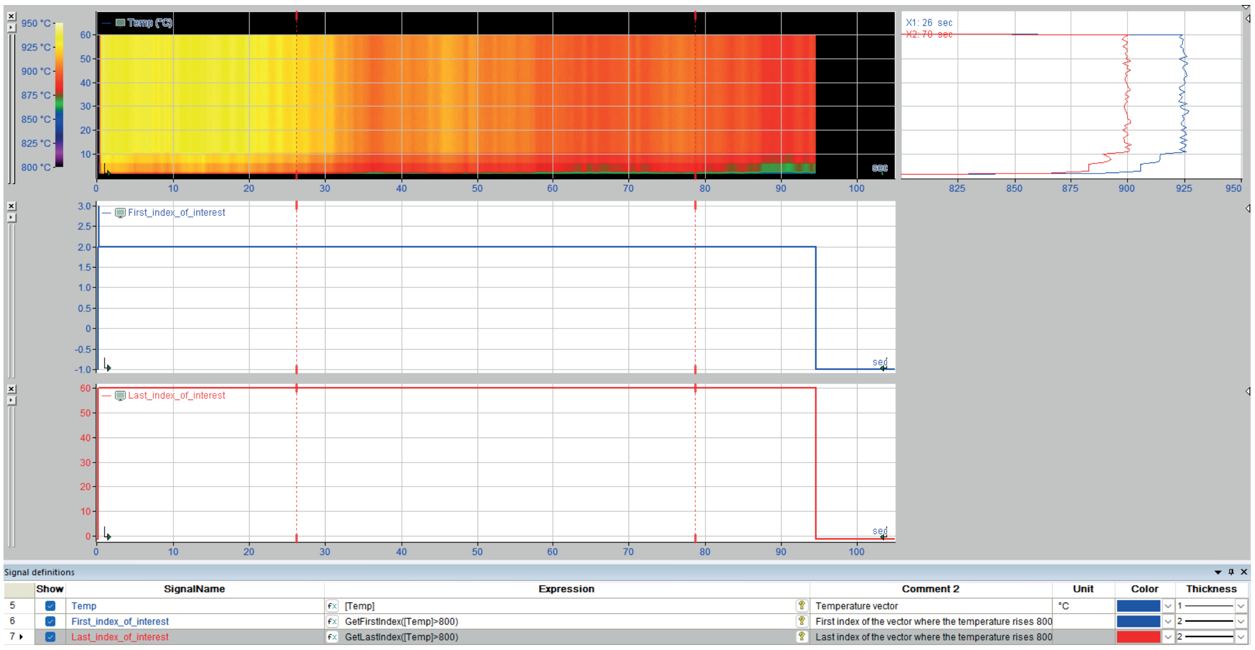
You can find extensive notes on these visualizations and their settings in part 2 of the *ibaAnalyzer* manual, chapter *Views*.

9.1 GetFirstIndex and GetLastIndex

`GetFirstIndex('Condition') and GetLastIndex('Condition')`

Description

These functions return the index of the first or last signal in the vector for which the condition 'Condition' is TRUE. The vector itself should be an operand of 'Condition'. If 'Condition' is FALSE for all signals of the vector, the function returns -1.



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Temp	[Temp]	Temperature vector	°C	🟦	1
6	✓	First_index_of_interest	GetFirstIndex([Temp]>800)	First index of the vector where the temperature rises 800		🟦	2
7	✓	Last_index_of_interest	GetLastIndex([Temp]>800)	Last index of the vector where the temperature rises 800		🟥	2

9.2 GetRows

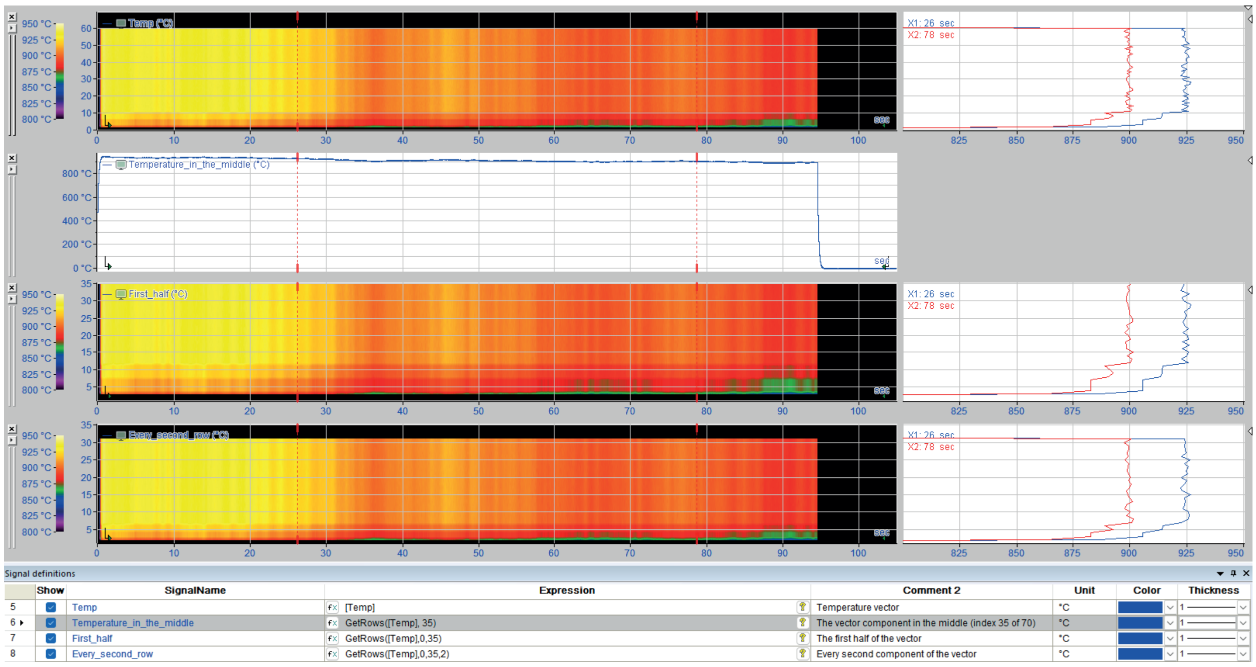
`GetRows('Vector','StartIndex','Counter','Step')`

Arguments

'Vector'	Vector from which to read the individual entries
'StartIndex'	First index from which to read the entries
'Counter'	Number of entries
'Step'	Step size

Description

This function extracts rows of values from a vector (array). A total number 'Counter' of entries is read out based on a 'StartIndex' (the smallest possible index is 0) in steps of the size 'Step'.


Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Temp	[Temp]	Temperature vector	°C	🟦	1
6	✓	Temperature_in_the_middle	GetRows([Temp], 35)	The vector component in the middle (index 35 of 70)	°C	🟦	1
7	✓	First_half	GetRows([Temp],0,35)	The first half of the vector	°C	🟦	1
8	✓	Every_second_row	GetRows([Temp],0,35,2)	Every second component of the vector	°C	🟦	1

9.3 GetZoneCenters

```
GetZoneCenters('Vector')
```

Description
This function determines the position of the center of the zone on the Y-axis for each zone of the vector. The only argument of the function is the vector. The result is a vector with a number of values in accordance with the number of zones.

Example
The *GetZoneCenters* function is useful if it is applied to the result of an *FftInTime* function. The *FftInTime* function returns a vector with *n* "zones" as its result which comply with the frequency bands (bins). With the *GetZoneCenters* function, the center frequencies of the individual bands of the spectrum and thus the frequency vector can be determined. This allows you to differentiate or integrate in the frequency domain by multiplying or dividing the results of the *FftInTime* and *GetZoneCenters* function accordingly.

9.4 GetZoneOffset

```
GetZoneOffset('Vector')
```

Description
This function determines the offset of the first zone, i.e. the position of the center of the first zone, based on the zero line of the Y-axis. The only argument of the function is the vector. The result is a constant value.

9.5 GetZoneWidths

```
GetZoneWidths('Vector')
```

Description
This function determines the width of each zone of the vector in units of the Y-axis. The only argument of the function is the vector. The result is a vector with a number of values in accordance with the number of zones.

9.6 MakeVector

```
MakeVector(r_0,r_1,…,r_n)
```

Description
This function creates a vector with the value series *r_0* to r_n. The arguments can be constant values or signals and expressions. This is comparable with the creation of a vector in the *Logical expressions* dialog.

Example
The *MakeVector* function mainly serves to enable macros to return multi-dimensional signals as their results. In the macro editor, the partial results of different calculations can be declared as interim values within the macro. As final macro result, a vector can be defined whose arguments are the interim values. The vector is basically used as container for macro results to simplify the macro interface.

9.7 SetZoneWidths

```
SetZoneWidths('Vector','Widths','Offset')
```

Arguments

'Vector'	Vector with (measured) values of the result vector
'Widths'	Vector containing zone widths as values
'Offset'	Distance of the zone center of the first zone from the zero line

Description
This function creates a vector with specified zone widths. In doing so, the values of the result vector are taken from a vector 'Vector' and the zone widths from a vector 'Widths'. Since the vector with the zone widths can use expressions as arguments, this function can be used to generate vectors with different zone widths depending on the loaded data. The expressions for defining the zone widths should be constant over time and not change for data loaded once. If this is not the case, the width values will be averaged over the overall period.

Example
The function *SetZoneWidths (MakeVector(1,2,3,2,1),MakeVector(2,4,10,4,2), -10)* creates the same vector as the one that was created with the logical expressions.

For more information, see part 2 of the *ibaAnalyzer* manual, chapter *Logical expressions*.

9.8 Traverse and TraverseW

Traverse

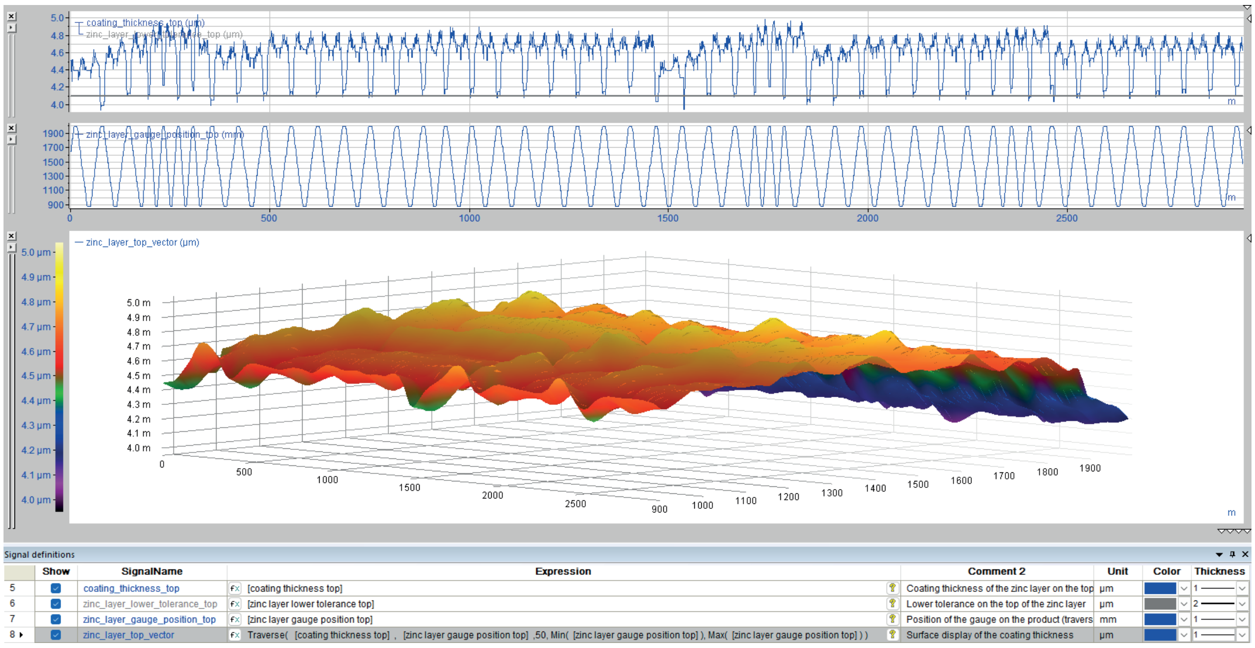
`Traverse('Signal','Position','N'=40,'Min','Max','Avg'=TRUE)`

Arguments

'Signal'	Signal measured by a traversing measuring device
'Position'	Position of the traversing measuring device along the traverse profile
'N'	Number of zones of the resulting vector
'Min'	Optional threshold for the minimum of the range to be mapped
'Max'	Optional threshold for the maximum of the range to be mapped
'Avg'	Optional binary parameter for averaging several samples within a zone pass; The average is formed by default

Description

This function converts signals originating from a traversing measuring device into a vector for 2-dimensional visualization.


coating_thickness_top (µm)
zinc_layer_lower_tolerance_top (µm)
zinc_layer_gauge_position_top (mm)
zinc_layer_top_vector (µm)

Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	coating_thickness_top	[coating thickness top]	Coating thickness of the zinc layer on the top	µm	🟦	1
6	✓	zinc_layer_lower_tolerance_top	[zinc layer lower tolerance top]	Lower tolerance on the top of the zinc layer	µm	⬜	2
7	✓	zinc_layer_gauge_position_top	[zinc layer gauge position top]	Position of the gauge on the product (travers	mm	🟦	1
8	✓	zinc_layer_top_vector	Traverse([coating thickness top] , [zinc layer gauge position top] ,50, Min([zinc layer gauge position top]), Max([zinc layer gauge position top]))	Surface display of the coating thickness	µm	🟦	1

TraverseW

```
TraverseW('Signal','Position','Widths','Offset'=0,'Avg'=TRUE)
```

Arguments

'Signal'	Signal measured by a traversing measuring device
'Position'	Position of the traversing measuring device along the traverse profile
'Widths'	Width of the resulting zones
'Offset'	Optional offset of the first zone
'Avg'	Optional binary parameter for averaging several samples within a zone pass; The average is formed by default

Description
This function works similarly to *Traverse*, with the difference being that the dimensions of the resulting vector are set directly via the zone widths 'Widths' and an optional offset parameter.

The 'Widths' parameter must be a vector that contains the width for each zone.

9.9 VectorAvg

```
VectorAvg('Vector')
```

Description
This function calculates the average of the cross profile for each sample, i.e. the average of all vector tracks per point in time or per X-axis position. The function returns a one-dimensional signal showing the curve of the cross profile average value over the time or length of the vector signal with the same number of samples.

9.10 VectorKurtosis

```
VectorKurtosis('Vector')
```

Description
This function calculates the kurtosis of the cross profile for each sample. The function returns a one-dimensional signal showing the curve of the cross-profile kurtosis over the time or length of the vector signal with the same number of samples.

Note

Please note that at least 4 signal points must be available, i.e. the vector must have at least 4 entries.

9.11 VectorMarkRange

```
VectorMarkRange('Vector','PositionFrom','PositionTo')
```

Arguments

'Vector'	Vector with input signals
'PositionFrom'	Start of the range to be selected
'PositionTo'	End of the range to be selected

Description
This function returns a partial vector of 'Vector' with a zone width from 'PositionFrom' (lower edge) to 'PositionTo' (upper edge).

The positions must be indicated in units of the Y-axis. The positions can be both fixed values and signals or expressions and thus be dependent on the data loaded.

The expressions for defining the positions should be constant over time and not change for data loaded once. If this is not the case, the position values will be averaged over the overall period.

9.12 VectorMin and VectorMax

VectorMax

```
VectorMax('Vector')
```

Description
This function calculates the maximum of the cross profile for each sample, i.e. the maximum value of all vector tracks per point in time or per X-axis position. The function returns a one-dimensional signal showing the curve of the cross profile maximum over the time or length of the vector signal with the same number of samples.

VectorMin

```
VectorMin('Vector')
```

Description
This function calculates the minimum of the cross profile for each sample, i.e. the minimum value of all vector tracks per point in time or per X-axis position. The function returns a one-dimensional signal showing the curve of the cross profile minimum over the time or length of the vector signal with the same number of samples.

9.13 VectorMedian

```
VectorMedian('Expression')
```

Description
This function calculates the median of the cross profile for each sample, i.e. the median of all vector traces at each point in time or each X-axis position. The function returns a one-dimensional signal showing the curve of the cross-profile median over time or length of the vector signal with the same number of samples.

9.14 VectorPercentile

```
VectorPercentile('Vector','Percentile'=0.5)
```

Arguments

'Vector'	Vector with input signals
'Percentile'	Percentile between 0 and 1

Description

This function calculates the percentile of the cross profile for each sample.

The second argument in addition to the 'Vector' is the specification of the 'Percentile' to be calculated. Default value is 0.5 (median). The function returns a one-dimensional signal showing the curve of the cross profile percentile over the time or length of the vector signal with the same number of samples.

9.15 VectorPolynomial and VectorLSQPolyCoef

VectorPolynomial

```
VectorPolynomial('Coefs','Vector','PolynomialType'=0)
```

Arguments

'Coefs'	Coefficient of the interpolation polynomial; calculated with *VectorLSQPolyCoef*
'Vector'	Sampling points for the analysis of the interpolation polynomial
'PolynomialType'	Defines the base polynomials to use 0 = Lagrange (default) 1 = Chebyshev I 2 = Chebyshev II 3 = Legendre

Description

You can use this function to show the interpolation polynomial that is described by the coefficient 'Coefs' as a result of the function *VectorLSQPolyCoef*.

The sampling points for analyzing the polynomial are determined by the sampling points of 'Vector'. If a zone offset or the zone width are specified, these are also used, otherwise the indices are used as Y-values.

You can use the optional parameter 'PolynomialType' to apply different base polynomials. The function supports the types Lagrange (default), Chebyshev I, Chebyshev II and Legendre.

Note

Note that the entries of 'Vector' are not relevant here.

VectorLSQPolyCoef

```
VectorLSQPolyCoef('Vector','Degree','PolynomialType'=0)
```

Arguments

'Vector'	Vector whose entries are used to calculate the least squares approximation polynomials
'Degree'	Degree of the interpolation polynomial
'PolynomialType'	Defines the base polynomials to use 0 = Lagrange (default) 1 = Chebyshev I 2 = Chebyshev II 3 = Legendre

Description
This function is the extension of the function *LSQPolyCoef* to vectors. The coefficients of an interpolation polynomial of degree 'Degree' are calculated using the least squares method for each cross section. The indices of the vector are used as a basis for this, unless a zone offset or the zone width were set when the vector was created. In this case, the corresponding values are used as the basis.

You can use the optional parameter 'PolynomialType' to apply different base polynomials. The function supports the types Lagrange (default), Chebyshev I, Chebyshev II and Legendre.

9.16 VectorSkewness

```
VectorSkewness('Vector')
```

Description
This function calculates the skewness of the cross profile for each sample. The function returns a one-dimensional signal showing the curve of the cross-profile skewness over time or length of the vector signal with the same number of samples.

Note

Please note that at least 4 signal points must be available, i.e. the vector must have at least 4 entries.

9.17 VectorStdDev

```
VectorStdDev('Vector')
```

Description
This function calculates the standard deviation of the cross profile for each sample. The function returns a one-dimensional signal showing the curve of the cross profile standard deviation over the time or length of the vector signal with the same number of samples.

9.18 VectorSum

```
VectorSum('Vector')
```

Description
This function calculates the sum of all values of the cross profile for each sample. The function returns a one-dimensional signal showing the curve of the value sum in the cross profile over the time or length of the vector signal with the same number of signals.

Example
If you divide the *VectorSum* expression by the number of vector tracks, the result is the same as with the *VectorAvg* function.

9.19 VectorToSignal and SignalToVector

VectorToSignal

```
VectorToSignal('Vector','XBase')
```

Arguments

'Vector'	Vector with (constant) input signals
'XBase'	Sampling rate of the resulting signal

Description
This function generates a one-dimensional signal from the elements of a vector along the cross profile. Every sample of the resulting signal corresponds to an element of the vector. The result complies with the cross profile. If the entries in the vector are not constant, a cross-profile of the average values of the individual signals is formed.

The 'XBase' parameter is optional. If 'XBase' is not indicated, the zone widths and the offset of the vector are used. The resulting signal can also receive non-equidistant samples.

Example
In connection with the *YatX* functions and the marker position, the *VectorToSignal* function can be used to display the cross profile at any position in the vector:

VectorToSignal (YatX([Vector],XMarker1()))

SignalToVector

```
SignalToVector ('Signal')
```

Description
Unlike the *VectorToSignal* function, the function *SignalToVector* creates a vector with constant entries from 'Signal'. The zone width and offset are determined by the sampling rate of the input signal. Note that, unlike *VectorToSignal*, this function has no optional argument for determining the zone widths. The *SetZoneWidth* can be used for this purpose.

Note

Ideally, the input signal has fewer than 1000 samples, as otherwise the signal will not be evaluated and will be marked as too complex.

10 Electrical function

10.1 RMS and Eff

`RMS('Expression','Frequency'=50) and Eff('Expression','Frequency'=50)`

Arguments

'Expression'	Signal or expression for which to determine the effective value
'Frequency'	Fundamental frequency

Description

This function calculates the so called "root mean square" value (or the effective value) of 'Expression' with a fundamental frequency of 'Frequency'.

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

e(n): Sample n of signal e ('Expression')

N: Number of samples in a period

Example

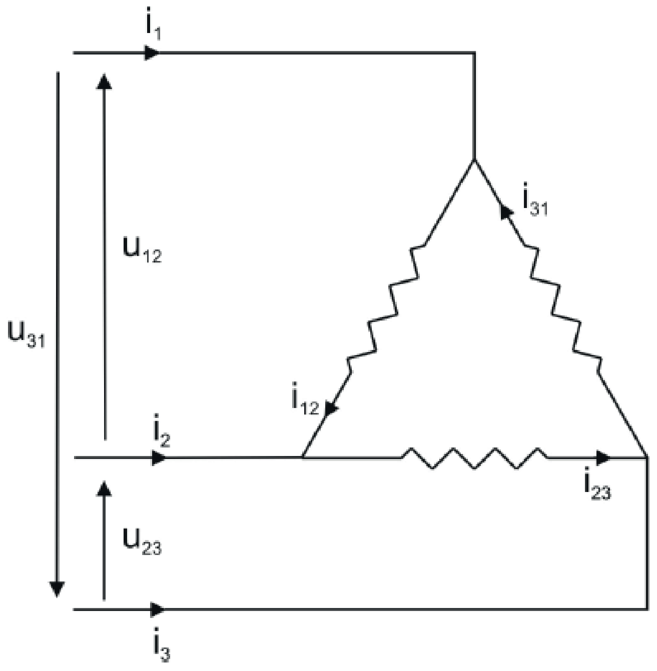
For an alternating voltage course with a frequency of 0.1 kHz, which is overlaid by a second AC voltage with 0.5 kHz, you can determine the effective value of the voltage for both frequencies by applying the function *Eff* with a second argument 0.1 or 0.5.

Note

There is no difference between the functions *RMS* and *Eff*. Both functions are supported by *ibaAnalyzer* for compatibility reasons.

10.2 Delta functions

The delta functions use the line voltages and line currents in a delta grid to calculate the different power values.



Arguments

'u12', 'u13', 'u23'	Connection voltage (same as the phase voltages)
'i1', 'i2', 'i3'	Connection currents
'Frequency'	Fundamental frequency

Note

These functions are typically applied to a delta grid, but they can be applied to any grid in which the line voltages and currents are measurable.

DeltaCollectiveUeff

```
DeltaCollectiveUeff('u12','u13','u23','Frequency'=50)
```

Calculates the collective effective voltage in a delta grid:

$$U_{eff} = \sqrt{\frac{1}{3}(U_{12,eff}^2 + U_{23,eff}^2 + U_{31,eff}^2)}$$

$U_{xy,eff}$: the effective value of the line voltage U_{xy}

DeltaCollectiveIeff

```
DeltaCollectiveIeff('i1','i2','i3','Frequency'=50)
```

Calculates the collective effective current in a delta grid:

$$I_{eff} = \sqrt{\sum_{x=1}^{3} I_{x,eff}^2}$$

$I_{x,eff}$: the effective value of the line current i_x

DeltaActiveP

```
DeltaActiveP('u13','u23','i1','i2','Frequency'=50)
```

Calculates the active power in a delta grid:

$$P = \frac{1}{N}\sum_{n=1}^{N}\left[u_{23}(n)i_2(n) + u_{13}(n)i_1(n)\right]$$

N : the number of samples in a period

u_{xy} : the voltage between the connection x and y ($u_{13} = -u_{31}$)

i_x : the current in line x

DeltaApparentP

```
DeltaApparentP('u12','u13','u23','i1','i2','i3','Frequency'=50)
```

Calculates the apparent power in a delta grid:

$$S = U_{eff} I_{eff}$$

U_{eff} : the collective effective voltage

I_{eff} : the collective effective current

DeltaReactiveP

```
DeltaReactiveP('u12','u13','u23','i1','i2','i3','Frequency'=50)
```

Calculates the reactive power in a delta grid:

$$Q = \sqrt{S^2 - P^2}$$

S : apparent power

P : active power

DeltaReactivePS

```
DeltaReactivePS('u13','u23','i1','i2','Frequency'=50)
```

Calculates the signed reactive power Q_S in the delta grid.

DeltaActivePFactor

```
DeltaActivePFactor('u12','u13','u23','i1','i2','i3','Frequency'=50)
```

Calculates the active power factor in a delta grid:

$$\cos\varphi = \frac{P}{S}$$

S : apparent power

P : active power

DeltaReactivePFactor

```
DeltaReactivePFactor('u12','u13','u23','i1','i2','i3','Frequency'=50)
```

Calculates the reactive power factor in a delta grid:

$$\tan\varphi = \frac{Q}{P}$$

Q : reactive power

P : active power

DeltaReactivePFactorS

```
DeltaReactivePFactorS('u13','u23','i1','i2','Frequency'=50)
```

Calculates the signed reactive power factor in a delta grid:

$$\tan\varphi = \frac{Q_S}{P}$$

Q_S : signed reactive power

P : active power

10.3 Harmonic functions

HarmEff

```
HarmEff('u','Nharm','Frequency'=50)
```

Calculates the effective value of 'Nharm' harmonic component of signal 'u'.

$$u_{Real,k} = \frac{2}{N}\sum_{n=0}^{N-1} u(n)\cos\frac{2\pi kn}{N}$$

$$u_{Imag,k} = \frac{2}{N}\sum_{n=0}^{N-1} u(n)\sin\frac{2\pi kn}{N}$$

$$U_k = \frac{\sqrt{u_{Real,k}^2 + u_{Imag,k}^2}}{\sqrt{2}}$$

u(n) : Sample n of signal u

$u_{Real,k}$: the real part of the kth harmonic component of u

$u_{Imag,k}$: the imaginary part of the kth harmonic component of u

U_k : the effective value of the kth harmonic component of u

HarmPhase

```
HarmPhase('u','Nharm','Frequency'=50)
```

Calculates the phase of 'Nharm' harmonic component of signal 'u':

$$\varphi_k = -a\tan\left(\frac{u_{Imag,k}}{u_{Real,k}}\right)$$

$u_{Real,k}$: the real part of the kth harmonic component of u

$u_{Imag,k}$: the imaginary part of the kth harmonic component of u

ϕ_k : the phase offset of the kth harmonic component of u

StarHarmUGeff

```
StarHarmUGeff('u1','u2','u3','Frequency'=50)
```

Calculates the effective negative sequence voltage U_{Geff}:

$$U_G = \frac{1}{3}\left[u_{1,1} + u_{2,1}(-\frac{2}{3}\pi) + u_{3,1}(-\frac{4}{3}\pi)\right]$$

$$U_{Geff} = \frac{\sqrt{U_{G,real}^2 + U_{G,imag}^2}}{\sqrt{2}}$$

$u_{x,1}$: Fundamental harmonic component (complex) of phase voltage u_x

StarHarmUMeff

```
StarHarmUMeff('u1','u2','u3','Frequency'=50)
```

Calculates the positive sequence system voltage U_{Meff}:

$$U_M = \frac{1}{3}\left[u_{1,1} + u_{2,1}(\frac{2}{3}\pi) + u_{3,1}(\frac{4}{3}\pi)\right]$$

$$U_{Meff} = \frac{\sqrt{U^2_{M,real} + U^2_{M,imag}}}{\sqrt{2}}$$

$u_{x,1}$: Fundamental harmonic component (complex) of phase voltage u_x

StarHarmUnSym

```
StarHarmUnSym('u1','u2', 'u3','Frequency'=50)
```

Calculates the voltage unbalance in a star grid:

$$SYM = \frac{U_{Geff}}{U_{Meff}} \times 100$$

The result is expressed in %.

WeightedDistortionFactor

```
WeightedDistortionFactor('u','Nharm'=50,'Frequency'=50)
```

Calculates the weighted distortion factor of 'u' (all phases) using 'Nharm' harmonics:

$$D_W = \frac{\sqrt{\sum_{n=2}^{Nharm} n^2 U_n^2}}{U_1}$$

U_n : Effective value of the nth harmonic component of signal u

UnweightedDistortionFactor

```
UnweightedDistortionFactor('u','Nharm'=50,'Frequency'=50)
```

Calculates the unweighted distortion factor of 'u' (all phases) using 'Nharm' harmonics:

$$D_{UW} = \frac{\sqrt{\sum_{n=2}^{Nharm} U_n^2}}{\sqrt{\sum_{n=1}^{Nharm} U_n^2}}$$

U_n : Effective value of the n_{th} harmonic component of signal u

10.3.1 TIF

```
TIF ('u','Nharm'=50,'Frequency'=50)
```

Calculates the Telephone Interference Factor of 'u', considering the first 'Nharm' harmonics.

$$TIF = \frac{1}{U_1}\sqrt{\sum_{n=2}^{Nharm}(K_n \times P_n \times U_n)^2}$$

K_n = 5*n*Fequency

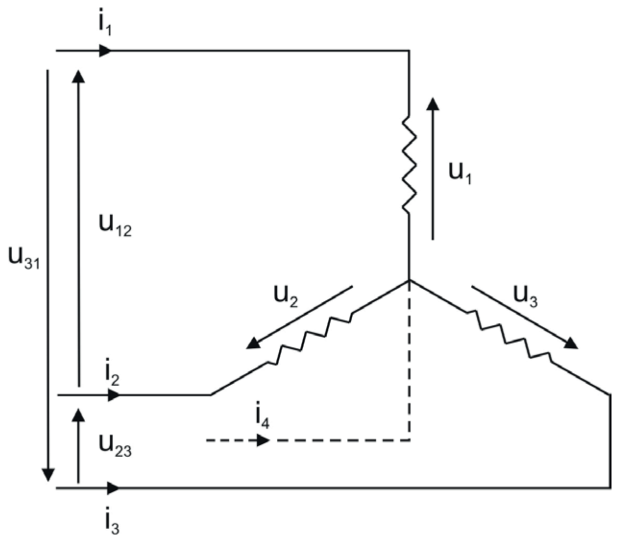
P_n = BTS coefficient (British Telephone System)

U_n : Effective value of the voltage the nth harmonic component of signal u

U_1 : Examples

10.4 Star functions

The star functions use the phase voltages and phase currents to calculate the different power values.



Arguments

'u1','u2','u3'	Phase voltages (Connection voltage = Sqrt(3) * phase voltage)
'i1','i2','i3'	Phase currents
'i4'	Neutral line
'Frequency'	Fundamental frequency

Note

These functions are typically applied to a star grid but they can be applied to any grid in which the phase voltages and currents are measurable.

StarCollectiveUeff

```
StarCollectiveUeff('u1','u2','u3','Frequency'=50)
```

Calculates the collective effective voltage in a star grid:

$$U_{eff} = \sqrt{\sum_{x=1}^{4} U^2_{x\_eff}}$$

U_{x_eff} : the effective value of phase voltage u_x

$u_4 = u_1 + u_2 + u_3$

StarCollectiveIeff

```
StarCollectiveIeff('i1','i2','i3','i4','Frequency'=50)
```

Calculates the collective effective current in a star grid:

$$I_{eff} = \sqrt{\sum_{x=1}^{4} I^2_{x,eff}}$$

$I_{x,eff}$: the effective value of the line current i_x

StarActiveP

```
StarActiveP('u1','u2','u3','i1','i2','i3','Frequency'=50)
```

Calculates the active power in a star grid:

$$P = \sum_{x=1}^{3} \left(\frac{1}{N} \sum_{n=1}^{N} u_x(n) i_x(n) \right)$$

N : Number of samples in a period

u_x : Voltage of phase x

i_x : Current of phase x

StarApparentP

```
StarApparentP('u1','u2','u3','i1','i2','i3','i4','Frequency'=50)
```

Calculates the apparent power in a star grid:

$$S = U_{eff} I_{eff}$$

U_{eff} : the collective effective voltage

I_{eff} : the collective effective current

StarReactiveP

```
StarReactiveP('u1','u2','u3','i1','i2','i3','i4','Frequency'=50)
```

Calculates the reactive power in a star grid:

$$Q = \sqrt{S^2 - P^2}$$

S : apparent power

P : active power

StarReactivePS

```
StarReactivePS('u1','u2','u3','i1','i2','i3','Frequency'=50)
```

Calculates the signed reactive power Q_S in the star grid.

StarActivePFactor

```
StarActivePFactor('u1','u2','u3','i1','i2','i3','i4','Frequency'=50)
```

Calculates the active power factor in a star grid:

$$\cos\varphi = \frac{P}{S}$$

S : apparent power

P : active power

StarReactivePFactor

```
StarReactivePFactor('u1','u2','u3','i1','i2','i3','i4','Frequency'=50)
```

Calculates the reactive power factor in a star grid:

$$\tan\varphi = \frac{Q}{P}$$

Q : reactive power

P : active power

StarReactivePFactorS

```
StarReactivePFactorS('u1','u2','u3','i1','i2','i3','Frequency'=50)
```

Calculates the signed reactive power factor in a star grid:

$$\tan\varphi = \frac{Q_S}{P}$$

Q_S : signed reactive power

P : active power

11 Miscellaneous functions

11.1 Count

`Count('Expression','Level'=0.5,'Hysteresis'=0,'EdgeType'=1,'Reset'=FALSE)`

Arguments

'Expression'	Signal or expression	
'Level'	Specification of the level value	
'Hysteresis'	Specification of a hysteresis band	
'EdgeType'	Edge type to 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 binary parameter to reset the counter. 'Reset' can be an expression as well.	
	'Reset'=TRUE	Counter is reset.
	'Reset'=FALSE	Counter value is retained/continues to count. (default)

Note

The 'Reset' condition must not refer to the *Count* function itself.

Description

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

You can use the 'Hysteresis' parameter to define a tolerance band that 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. You can also formulate 'Reset' as an expression.

Tip

You can also use the *Count* function for digital signals. For this purpose, choose 0.5 for 'Level' and, for example, 0.1 for 'Hysteresis'. By this all changes from FALSE to TRUE and vice versa will be detected and counted.

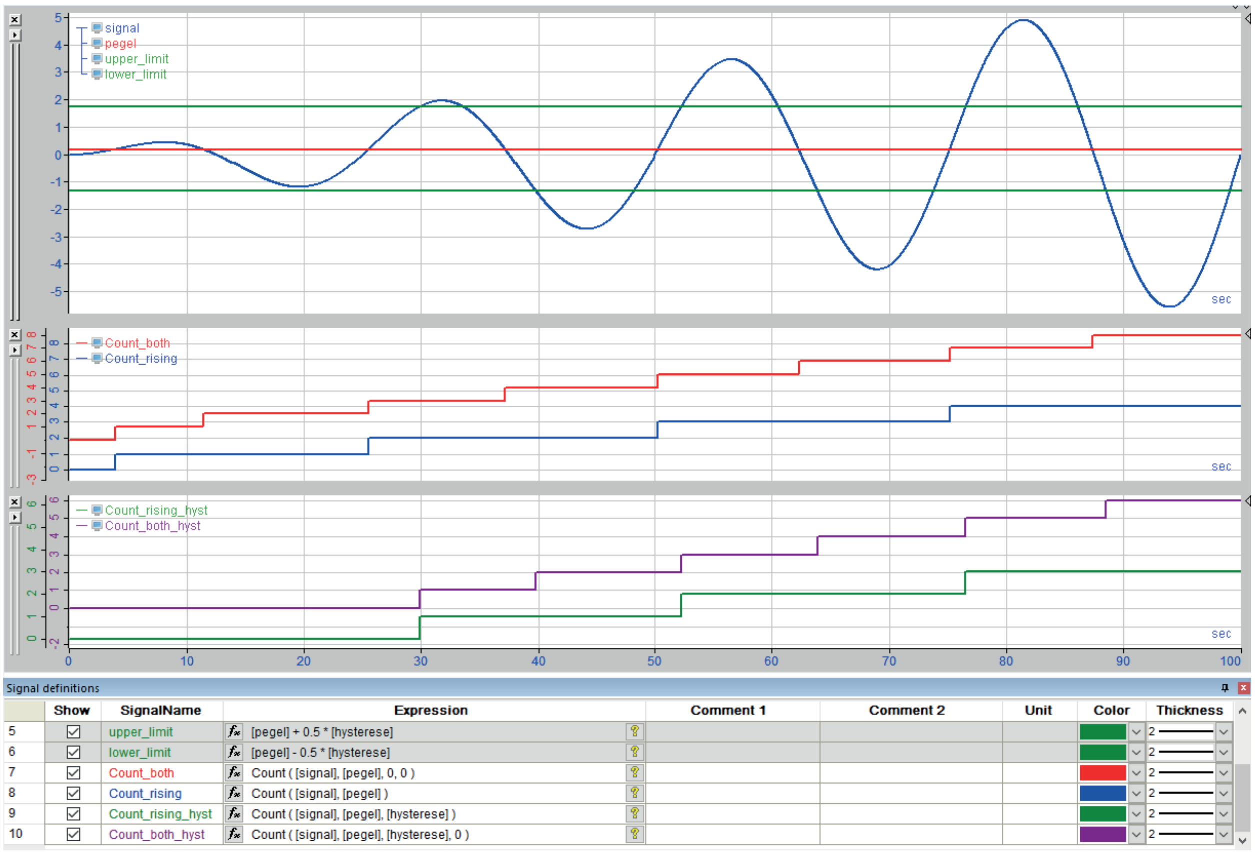
Example

Specification:

- 'Level': 2.5
- 'Hysteresis': 2.0

Result:

When using hysteresis, level crossings in the ascending direction are not counted until 'Expression' is > 3.5 and in the descending direction until 'Expression' is < 1.5.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
5	✓	upper_limit	[pegel] + 0.5 * [hysterese]				🟩	2
6	✓	lower_limit	[pegel] - 0.5 * [hysterese]				🟩	2
7	✓	Count_both	Count ([signal], [pegel], 0, 0)				🟥	2
8	✓	Count_rising	Count ([signal], [pegel])				🟦	2
9	✓	Count_rising_hyst	Count ([signal], [pegel], [hysterese])				🟩	2
10	✓	Count_both_hyst	Count ([signal], [pegel], [hysterese], 0)				🟪	2

11.2 Debounce

`Debounce ('Expression','Debounce interval')`

Arguments

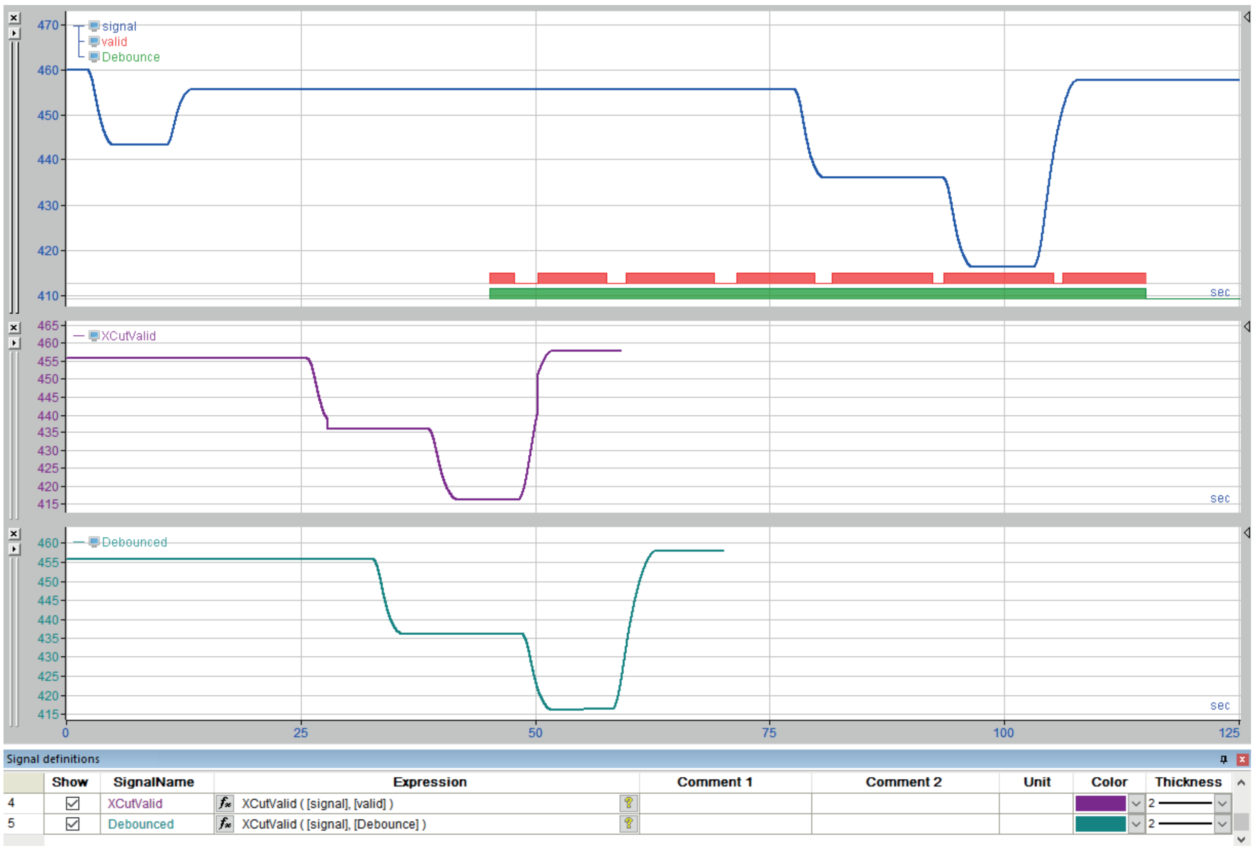
'Expression'	Signal to debounce
'Debounce interval'	Dead time (reaction time)

Description

This function delivers a debounced signal trend of 'Expression' with 'Debounce interval' as dead time in seconds. 'Debounce interval' is interpreted as position in m for length based signals.

The function works in a manner similar to an OFF-delay time relay, however, with the difference that the signal change from TRUE to FALSE (falling edge) is presented in real time, i.e. without delay, unless another change from FALSE to TRUE (rising edge) occurs during the time set.

In this way, it is possible to smooth unsteady signals, for example, from photocells or limit switches. This is particularly important if these signals are used as conditions in certain operations, such as *XMarkValid* or *XCutValid*, because every discontinuity would interrupt the calculation of the operation, so that result values would be lost. The following picture shows the difference.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
4	☑	XCutValid	XCutValid ([signal], [valid])				🟪	2
5	☑	Debounced	XCutValid ([signal], [Debounce])				🟦	2

11.3 DynSignal

`DynSignal('TextExpression')`

Description

This function returns the value of the signal or expression specified by 'TextExpression'.

'TextExpression' must result in a text string that *ibaAnalyzer* can interpret as a new, valid expression. As a result, *ibaAnalyzer* then provides the signal that is derived by evaluating this expression.

A typical use case is the generation of a valid signal ID in the loaded data file, e.g. a signal ID consisting of [module number:signal number] for analog signals or [module number.signal number] for digital signals.

The function is used in particular in connection with the data point concept from the field of iba energy measuring technology. Various metadata is read, which is stored for each measuring point in structured form in the info area of the data file. The content of this metadata is the module number under which the corresponding measured value is stored in the data file.

This makes it possible to standardize the evaluation of measurement point data, even when the I/O configuration varies from system to system.

Example

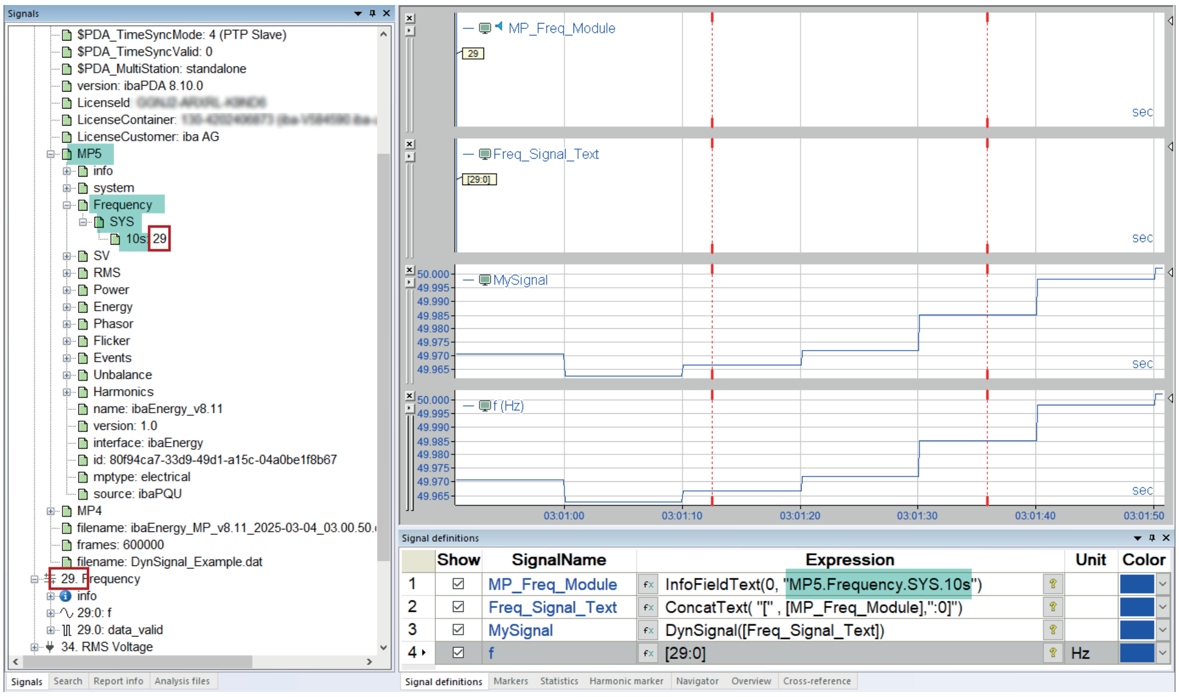
In the analysis, the goal is to determine the calculated 10-second value of the frequency at a specific measurement point.

The module number under which this value is stored as a signal can be found in the metadata field:

`MeasurementPoint.Frequency.SYS.10s`

In the following example, the 10-second frequency value at measurement point 5 (MP5) is retrieved.

The measured value is stored in module 29, signal 0 in the data file, which corresponds to the ID [29:0].



```
$PDA_TimeSyncMode: 4 (PTP Slave)
$PDA_TimeSyncValid: 0
$PDA_MultiStation: standalone
version: ibaPDA 8.10.0
LicenseId [redacted]
LicenseContainer: [redacted]
LicenseCustomer: iba AG
MP5
  info
  system
  Frequency
    SYS
      10s: 29
  SV
  RMS
  Power
  Energy
  Phasor
  Flicker
  Events
  Unbalance
  Harmonics
  name: ibaEnergy_v8.11
  version: 1.0
  interface: ibaEnergy
  id: 80f94ca7-33d9-49d1-a15c-04a0be1f8b67
  mptype: electrical
  source: ibaPQU
MP4
filename: ibaEnergy_MP_v8.11_2025-03-04_03.00.50.
frames: 600000
filename: DynSignal_Example.dat
29. Frequency
  info
  29:0: f
  29.0: data_valid
34. RMS Voltage
```

Signal definitions

	Show	SignalName	Expression	Unit	Color
1	☑	MP_Freq_Module	InfoFieldText(0, "MP5.Frequency.SYS.10s")		🟦
2	☑	Freq_Signal_Text	ConcatText("[" , [MP_Freq_Module],":0]")		🟦
3	☑	MySignal	DynSignal([Freq_Signal_Text])		🟦
4	☑	f	[29:0]	Hz	🟦

The retrieval occurs in three steps, as reflected in the three rows of the signal table:

1. Retrieval of the content of the info field *MP5.Frequency.SYS.10s* with the *InfoFieldText* function. The result "29" is temporarily stored as the text signal *MP_Freq_Module* (see top graph).
2. To obtain the correct form of the signal specification, the *ConCat* function is used to combine the character "[", the text signal *MP_Freq_Module* and the characters ":0]" to form a string. The combined text is temporarily stored as the text expression *Freq_Signal_Text* (see second graph from the top).
   When specifying the signal number (here: ":0"), you can also use wildcards or enter ranges.
   - ":*" – all signals of the module
   - ":n-m" – signals from-to (e.g. ":20-29")
3. Finally, the text expression *Freq_Signal_Text* is interpreted via the *DynSignal* function as a signal ID consisting of [module number:signal number] and its value is read out as the signal *MySignal* (see third graph from the top).
4. This line displays the signal [29:0]. The value is identical to that of *MySignal*.

11.4 Envelope

```
Envelope('Expression','Interval')
```

Arguments

'Expression'	Signal or expression around which to form the envelope curve
'Interval'	X-axis interval

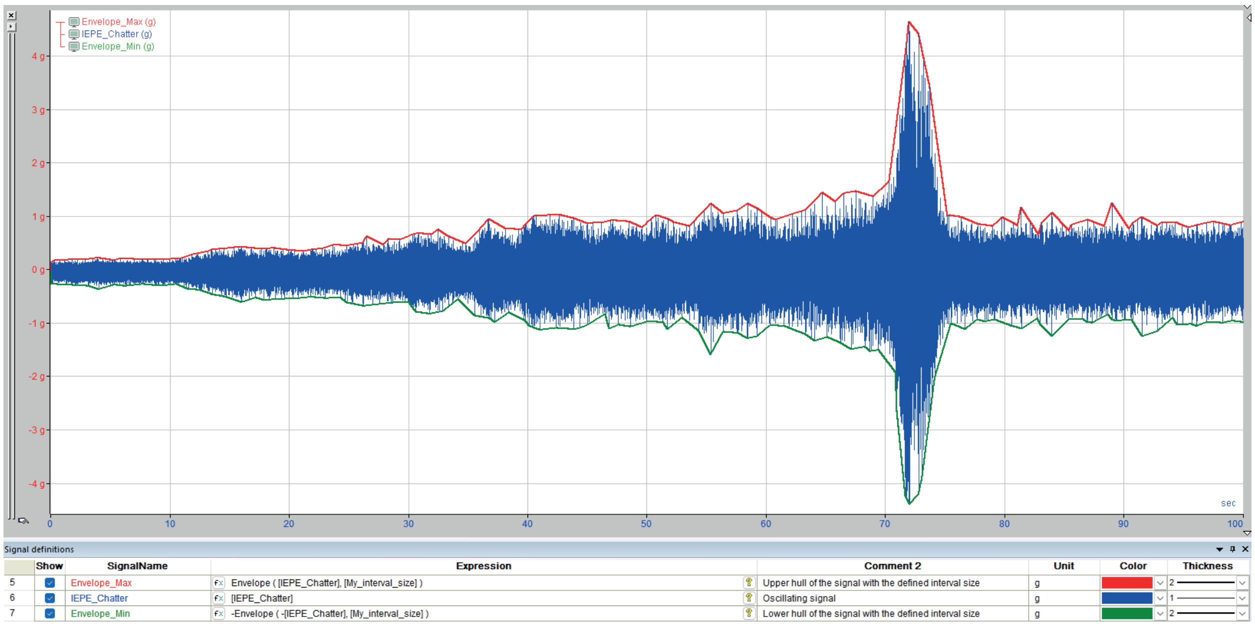
Description

This function calculates the upper envelope of a signal or expression. The envelope is formed by linking the high peaks of the signal curve. The parameter 'Interval' sets the quality of the envelope curve. Without this parameter, only the maximum peak is considered over the entire recording length of the signal. The parameter 'Interval' specifies the length of an interval in base units of X-axis (s, m, Hz, 1/m). By using this parameter, the peaks inside the interval are considered too and the envelope nestles against the signal curve.

Tip

To form an envelope along the lower side of the signal curve, you can enter the same function in the following form. In this case the low peaks (minimum values) will be linked.

```
-Envelope (-'Expression','Interval')
```



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Envelope_Max	Envelope ([IEPE_Chatter], [My_interval_size])	Upper hull of the signal with the defined interval size	g	🟥	2
6	✓	IEPE_Chatter	[IEPE_Chatter]	Oscillating signal	g	🟦	1
7	✓	Envelope_Min	-Envelope (-[IEPE_Chatter], [My_interval_size])	Lower hull of the signal with the defined interval size	g	🟩	2

11.5 False and True

```
False() and True()
```

Description

These operands have the constant value 0 or 1.

In Boolean operations (AND, OR etc.) the value is taken for logical 0 (FALSE) or logical 1 (TRUE). In arithmetic operations and in combination with analog values, the value is taken for 0.0 or 1.0 ("fixed zero" or "fixed one").

11.6 GetBit and GetBitMask

GetBit

```
GetBit('Expression','Bitnumber')
```

Description

This function returns the Boolean value of the 'Bitnumber' bit of 'Expression' after rounding to the nearest integer value. The rounding limit is in each case the next 0.5 increment. (2.48 --> 2; 2.50 -->3). Valid bit number sequence: 0 (LSB) to 15 (MSB).

Note

The function does not apply to integers with 64 bits because these data types are not supported by *ibaPDA* and thus cannot be included in a data file.

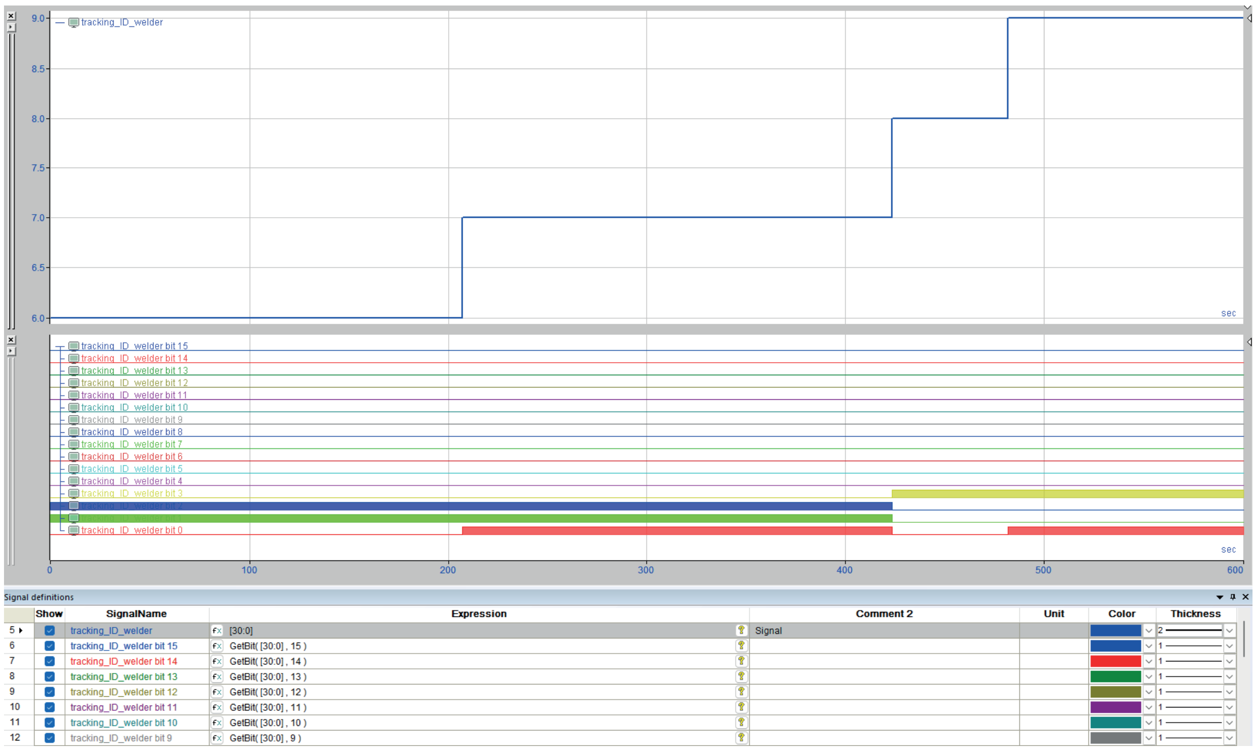
Example

In the table below, the least significant byte of an integer value with the bits 0...7 is shown as an example. In order to represent the values 0...8, the individual bits are highlighted as with "X" (X = TRUE). The left column shows the initial values, which represented by the summed powers of 2 (header), e.g., 1 = 2^0; 2 = 2^1; 3 = 2^0 + 2^1.

Bit no.	7	6	5	4	3	2	1	0
0								
1								X
2							X	
3							X	X
4						X		
5						X		X
6						X	X	
7						X	X	X
8					X			

Tip

If you want to reduce one or more 8-, 16- or 32-bit integers to single bits, open the context menu of the desired signal in the signal tree and select *Show bits* in the context menu. All bits are then shown as individual digital signals, without programming the *GetBit* function. The internal method of this function is the same as for *GetBit*.



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	tracking_ID_welder	[30:0]	Signal		🟦	2
6	✓	tracking_ID_welder bit 15	GetBit([30:0] , 15)			🟦	1
7	✓	tracking_ID_welder bit 14	GetBit([30:0] , 14)			🟥	1
8	✓	tracking_ID_welder bit 13	GetBit([30:0] , 13)			🟩	1
9	✓	tracking_ID_welder bit 12	GetBit([30:0] , 12)			🟫	1
10	✓	tracking_ID_welder bit 11	GetBit([30:0] , 11)			🟪	1
11	✓	tracking_ID_welder bit 10	GetBit([30:0] , 10)			🟦	1
12	✓	tracking_ID_welder bit 9	GetBit([30:0] , 9)			⬛	1

GetBitMask

```
GetBitMask('Expression','Bitnumber')
```

Description

This function interprets 'Expression' as a bit mask of a float value and returns the value of the bit 'Bitnumber'. Valid range: 0 (LSB) to 31 (MSB)

This function was specifically developed for working with data from SimadynD in one particular case where up to 32 digital values are recorded in packed format as a float variable. The *GetBitMask* function only evaluates the valence of the specified bit 'Bitnumber' irrespective of whether it is part of the mantissa or of the exponent. In contrast to the *GetBit* function, there is no rounding to the next integer.

Tip

If you want to reduce one or more 32 bit floating values to single bits, open the context menu of the desired signal in the signal tree and select *Show bits* in the context menu. All bits are then shown as individual digital signals. The internal method of this function is the same as for *GetBitMask*.

11.7 HighPrecision

```
HighPrecision('Expression')
```

Description
With this function, 'Expression' is marked as quantity with double precision. Calculations which are then performed with 'Expression' are implemented with double precision, even if the original expression only has single precision.

On the one hand, double precision has the advantage that calculations can be performed more precisely. On the other hand, however, it requires twice as much memory. Therefore, *ibaAnalyzer* automatically decides based on the input arguments which precision to use for calculation.

11.8 InfoField, ChannelInfoField and ModuleInfoField

These functions read out information from an info field of a data file, a signal or a module.

Note

The functions *InfoField, ChannelInfoField* and *ModuleInfoField* expect a numerical value. To read out text, use the functions *InfoFieldText*, *ChannelInfoFieldText* and *ModuleInfoFieldText*. See ↗ *InfoFieldText, ChannelInfoFieldText and ModuleInfoFieldText*, page 126.

Arguments

'Index'	Index of the data file, the signal or module
'InfoField'	Info field to read; put in quotation marks.
'Begin'	Optional: First character of the field content to be read. If you do not specify a value, the entire content is read.
'End'	Optional: Last character of the field content to be read. If you do not specify a value, the content is read from 'Begin' to the last character.

InfoField

```
InfoField('FileIndex',"'InfoField'",'Begin'=0,'End'=Text end)
```

Tip

If you double-click on the desired info field, *ibaAnalyzer* automatically inserts the corresponding function as new signal into the signal table. If required, you then only have to customize the signal name and beginning or end. This method also works in the input box of the Expression builder. The function will then be inserted at the cursor position.

ChannelInfoField

```
ChannelInfoField('ChannelIndex',"'InfoField'",'Begin'=0,'End'=Text end)
```

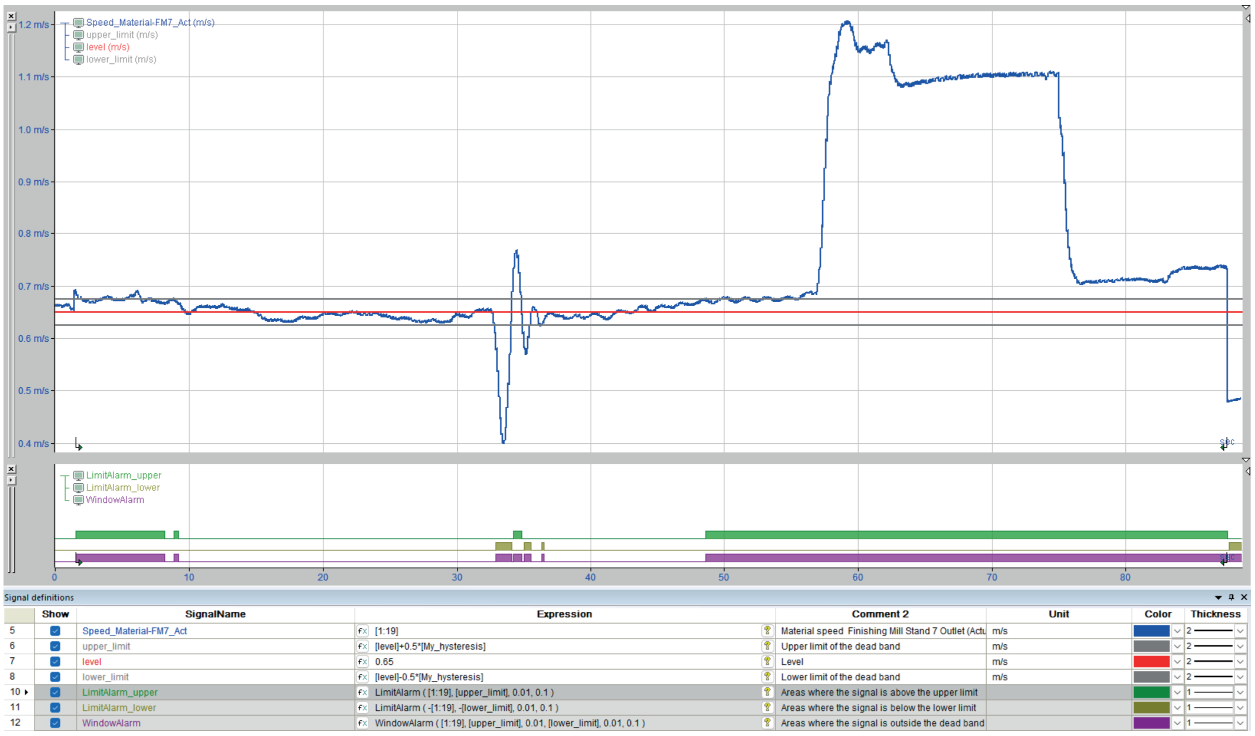
ModuleInfoField

```
ModuleInfoField('FileIndex','ModuleIndex',"'InfoField'",'Begin'=0,'End'=Text end)
```

Note

You have to specify 2 indices for this function: The index of the data file as the first argument and the index of the module as the second.

11.9 LimitAlarm and WindowAlarm



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Speed_Material-FM7_Act	[1:19]	Material speed Finishing Mill Stand 7 Outlet (Actu	m/s	🟦	2
6	✓	upper_limit	[level]+0.5*[My_hysteresis]	Upper limit of the dead band	m/s	⬜	2
7	✓	level	0.65	Level	m/s	🟥	2
8	✓	lower_limit	[level]-0.5*[My_hysteresis]	Lower limit of the dead band	m/s	⬜	2
10	✓	LimitAlarm_upper	LimitAlarm ([1:19], [upper_limit], 0.01, 0.1)	Areas where the signal is above the upper limit		🟩	1
11	✓	LimitAlarm_lower	LimitAlarm (-[1:19], -[lower_limit], 0.01, 0.1)	Areas where the signal is below the lower limit		🟫	1
12	✓	WindowAlarm	WindowAlarm ([1:19], [upper_limit], 0.01, [lower_limit], 0.01, 0.1)	Areas where the signal is outside the dead band		🟪	1

LimitAlarm

`LimitAlarm('Expression','Limit','DeadBand','Time')`

Arguments

'Expression'	Signal or expression
'Limit'	Limit from which the function returns TRUE
'DeadBand'	Specification of a dead zone below the limit within which the function does not reset to FALSE
'Time'	Specification of the time for which 'Expression' must be above the limit until the function returns TRUE

Description

This function monitors the signal ('Expression') and sets the result to TRUE if the signal is above the threshold ('Limit') longer than the specified time ('Time'). The result of the function returns to FALSE when the signal falls below the threshold by the value specified under the dead zone ('DeadBand').

Tip

You can also use the *LimitAlarm* function for a lower limit. For this purpose, flip the signal and the limit, i.e. multiply by (-1).

For example: LimitAlarm([0:1] *(-1), 9 *(-1), 0.5, 0.4)

WindowAlarm

```
WindowAlarm('Expression','Limit1','DeadBand1','Limit2','DeadBand2','Time')
```

Arguments

'Expression'	Signal or expression
'Limit1'	Upper limit from which the function returns TRUE
'DeadBand1'	Specification of the dead zone below the upper limit ('Limit1') within which the function does not reset to FALSE
'Limit2'	Lower limit from which the function returns TRUE
'DeadBand2'	Specification of the dead zone above the lower limit ('Limit2') within which the function does not reset to FALSE
'Time'	Specification of the time for which 'Expression' must be greater than the upper limit or smaller than the lower limit until the function returns TRUE

Description

This function monitors the signal ('Expression') and sets the result to TRUE if the signal is longer than the specified time ('Time') outside the range between the upper threshold ('Limit1') and the lower threshold ('Limit2'). The result of the function returns to FALSE when the signal falls below the upper limit by the value specified under 'DeadBand1', or exceeds the lower limit by the value specified under 'DeadBand2'.

11.10 ManY

`ManY('Xbase','y0','y1',…)`

Arguments

'XBase'	Sampling rate of the resulting signal
'y0', 'y1',…	Y-values of the resulting signal

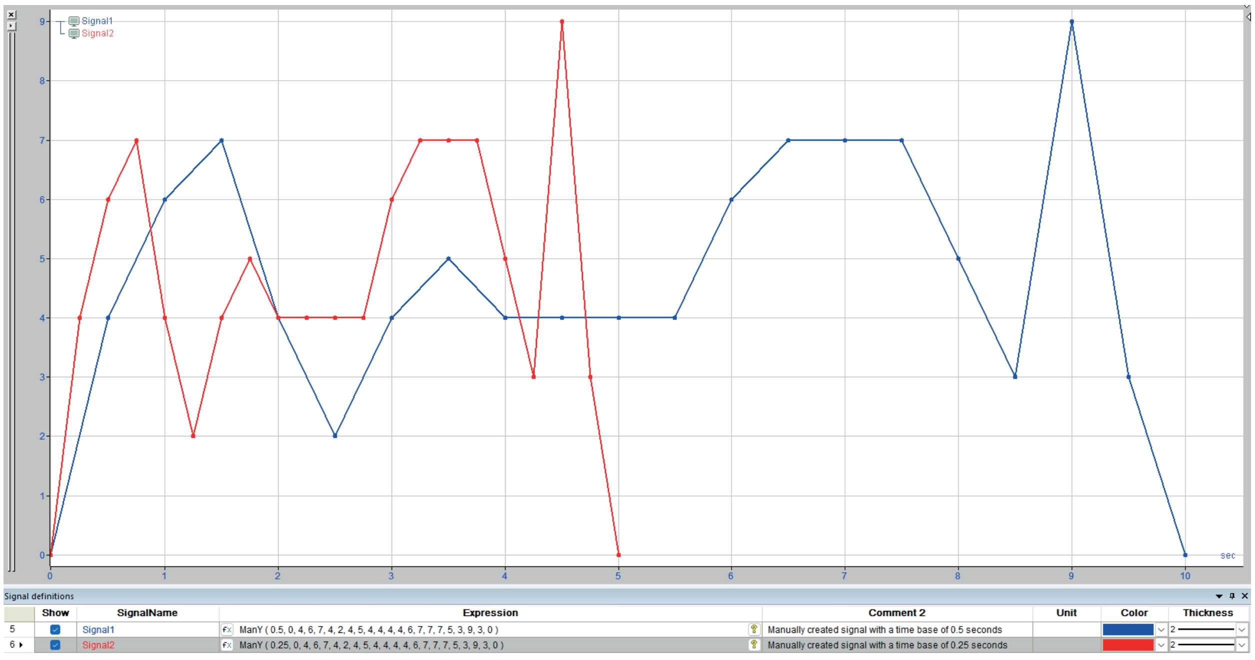
Description

You can use this function to manually create a signal trend with the "measured values" of 'y0'…'y99', each at a time or position distance of 'Xbase' apart. The 'Xbase' value is expressed in seconds for time-related values and in meter for length-related values. The number of points is limited to 100.

In this way, you can, for example, enter reference curves to which the signals measured in the field are then compared. Furthermore, you can add data which is not available as a measuring value to an analysis. Using this function, you can also manually generate text signals and enter different values.

Tip

If you put at least one of the parameters y0 to y99 (max.) in quotation marks, the entered characters are not used as numerical values but as ASCII characters.



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Signal1	ManY (0.5, 0, 4, 6, 7, 4, 2, 4, 5, 4, 4, 4, 4, 6, 7, 7, 7, 5, 3, 9, 3, 0)	Manually created signal with a time base of 0.5 seconds		🟦	2
6	✓	Signal2	ManY (0.25, 0, 4, 6, 7, 4, 2, 4, 5, 4, 4, 4, 4, 6, 7, 7, 7, 5, 3, 9, 3, 0)	Manually created signal with a time base of 0.25 seconds		🟥	2

11.11 PulseFreq

```
PulseFreq('Expression','Omega'=0,'EdgeType'=2,'MinFreq'=0.05)
```

Arguments

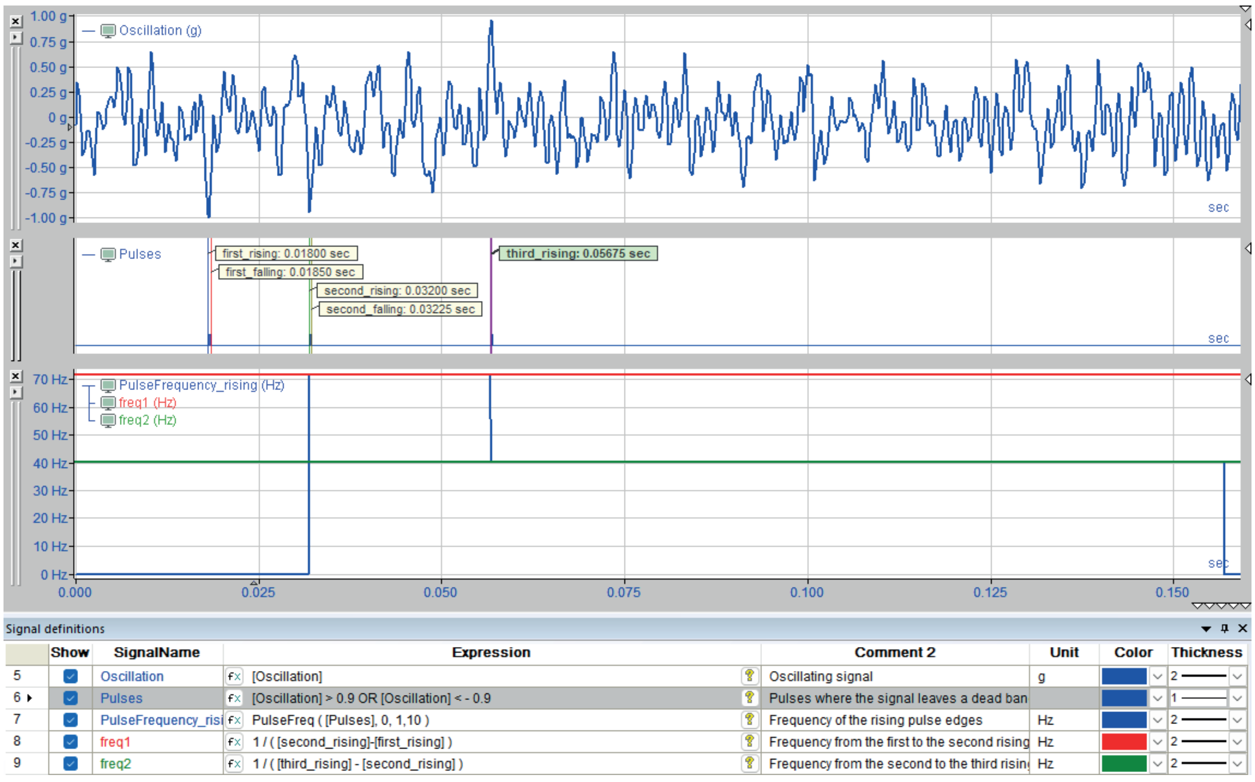
'Expression'	Pulse counter signal	
'Omega'	Filter frequency	
'EdgeType'	Edge type to be counted	
	'EdgeType' = -1	Falling edges only
	'EdgeType' = 0	Rising and falling edges
	'EdgeType' = 1	Rising edges only
	'EdgeType' = 2	'Expression' is a pulse counter
'MinFreq'	Smallest frequency that is shown	

Description

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

A low-pass filter with filter frequency 'Omega' is applied to the result. If 'Omega' is 0, then the low-pass filter is deactivated. 'EdgeType' determines which edges of pulses to count. If no pulse occurs in 1000 samples, the function returns 0 as the calculated frequency.

The frequency is only output as soon as the second edge occurs. The value is based on the distance between the first and second edge, with the calculation only taking place at the time of the second edge.



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Oscillation	[Oscillation]	Oscillating signal	g	🟦	2
6	✓	Pulses	[Oscillation] > 0.9 OR [Oscillation] < - 0.9	Pulses where the signal leaves a dead ban		🟦	1
7	✓	PulseFrequency_risi	PulseFreq ([Pulses], 0, 1,10)	Frequency of the rising pulse edges	Hz	🟦	2
8	✓	freq1	1 / ([second_rising]-[first_rising])	Frequency from the first to the second rising	Hz	🟥	2
9	✓	freq2	1 / ([third_rising] - [second_rising])	Frequency from the second to the third risin	Hz	🟩	2

This function was especially designed for using the WAGO incremental encoder 750-631. You can use the function to calculate the speed based on the pulse counter signal from the encoder.

The pulse counter value is differentiated while accounting for possible overflow. Because the result of the differentiation may include interfering frequencies or noise, a low-pass filter is subsequently applied. The filter frequency is ideally set slightly above the maximum pulse frequency.

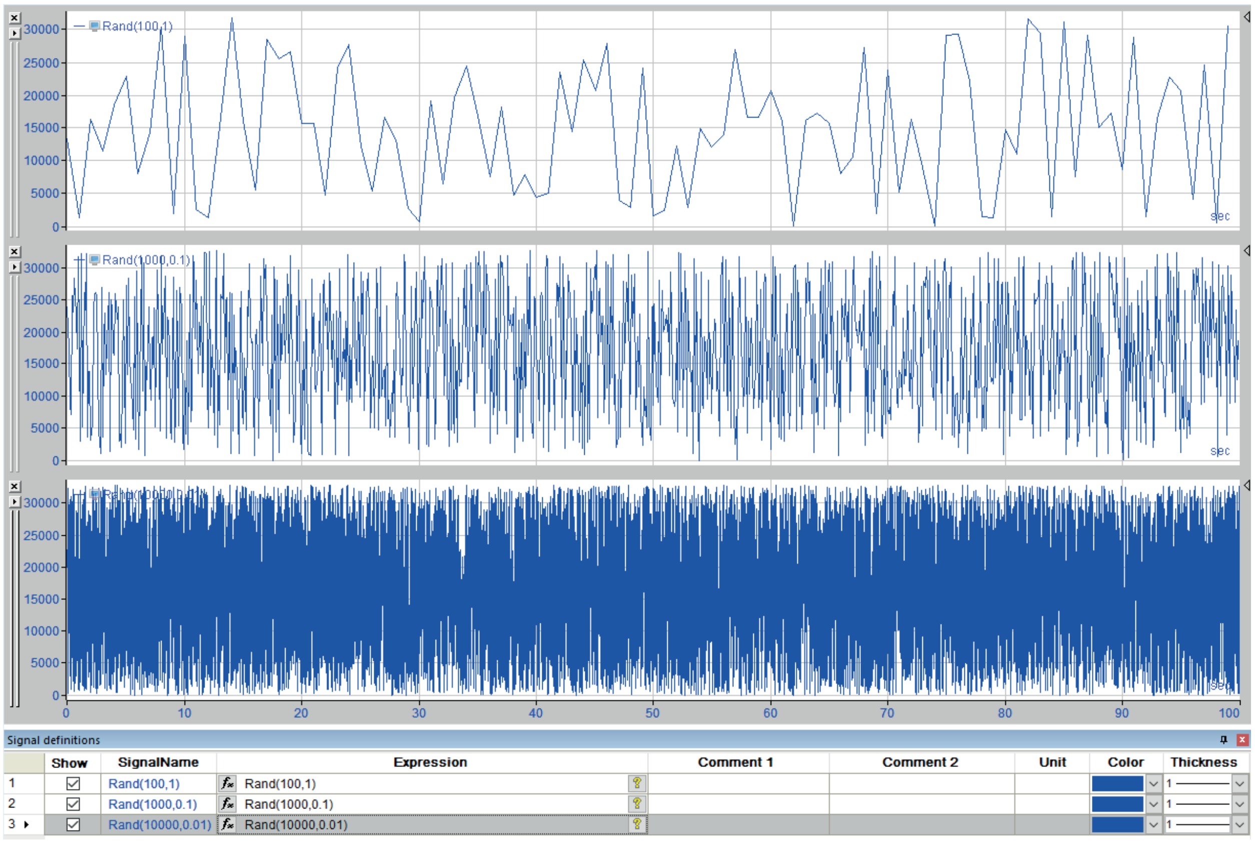
11.12 Rand

```
Rand('Count','XBase')
```

Description

This function generates a signal consisting of random numbers within the range of 0 to 32767. The parameter 'Count' defines the length of the graph (in seconds or meters). The parameter 'XBase' defines the distances between the samples.

The following figure shows three signals which are all 100 seconds long, but which consist of different numbers of points. The time basis 'XBase' is 1 s, 100 ms and 10 ms.



Signal definitions

	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☑	Rand(100,1)	Rand(100,1)				🟦	1
2	☑	Rand(1000,0.1)	Rand(1000,0.1)				🟦	1
3	☑	Rand(10000,0.01)	Rand(10000,0.01)				🟦	1

Tip

If you want to limit the random numbers by a range "a to b", you can use the following formula:

Rand('Count','XBase') * (b-a) - a

11.13 SampleAndHold

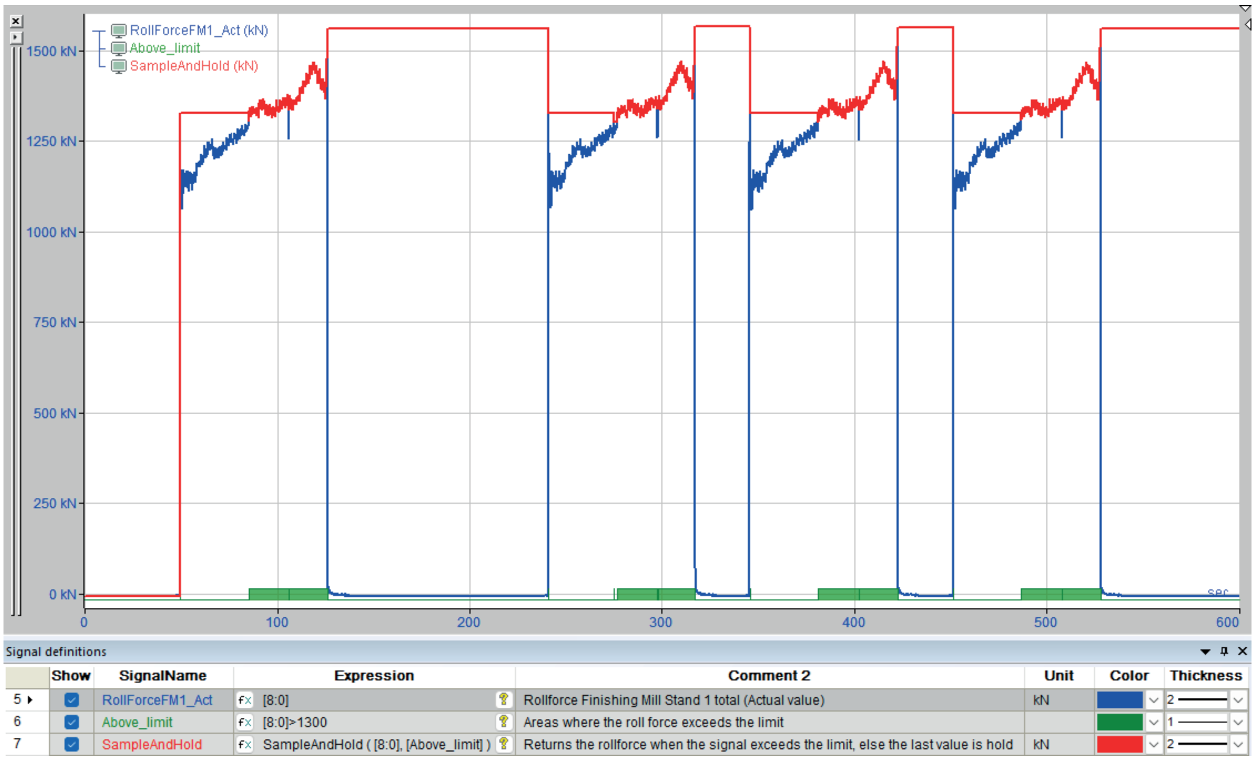
`SampleAndHold('Expression','Sample')`

Arguments

'Expression'	Signal or expression
'Sample'	Parameter that determines whether the function follows the signal (1) or holds the last measured value (0). 'Sample' can be a condition itself or be determined by a different function.

Description

This function is a sample-hold function. The output follows 'Expression' when 'Sample' = TRUE. It remains unchanged when 'Sample' = FALSE.



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	RollForceFM1_Act	[8:0]	Rollforce Finishing Mill Stand 1 total (Actual value)	kN	🟦	2
6	✓	Above_limit	[8:0]>1300	Areas where the roll force exceeds the limit		🟩	1
7	✓	SampleAndHold	SampleAndHold ([8:0], [Above_limit])	Returns the rollforce when the signal exceeds the limit, else the last value is hold	kN	🟥	2

11.14 SampleOnce

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

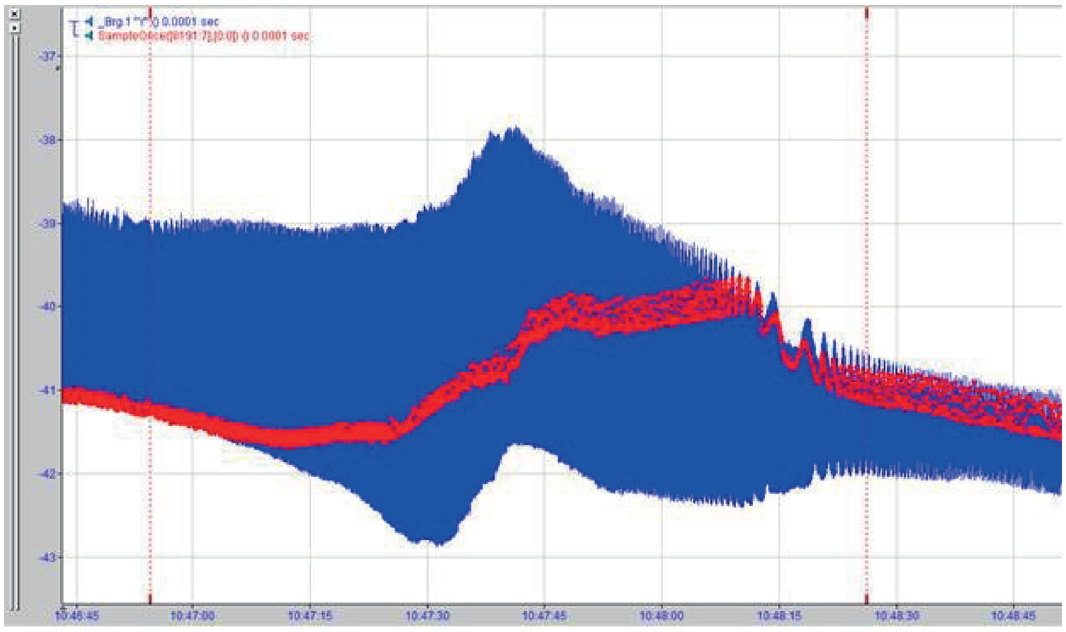
Arguments

'Expression'	Signal or expression
'Sample'	Digital signal whose rising edges determine the sampling points

Description
This function resamples an 'Expression' input signal at individual points determined by the rising edges of the 'Sample' digital signal. The result has one measuring point per rising edge and is invalid in the ranges in between.

Example
This function can be used to display a phase sensor signal (key phasor) in the time domain. Whenever the phase sensor jumps to TRUE, the original signal is sampled. By overlaying both representations, the times of the phase sensor are suitably represented. In the example below, a 180° phase shift can be detected when passing a resonance.

The following illustration shows the *SampleOnce* function applied to a vibration signal.



11.15 Sign

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

Description
This function returns the sign of 'Expression'.

- 'Expression' > 0 --> +1
- 'Expression' = 0 --> 0
- 'Expression' < 0 --> -1

11.16 Technostring

```
Technostring('Index','Begin',End')
```

Arguments

'Index'	Index of the data file
'Begin'	Start of the range to be read out
'End'	End of the range to be read out

Description

This function extracts the string from the data file index 'Index' between 'Begin' and 'End'. The default start index is 0. This means that it is possible to interpret information from the technostring as signals (numerical characters only).

The technostring information displayed in the *Info* branch in the signal tree window is evaluated. The precondition is, of course, that *ibaPDA* saved the technostring information in the data file.

'Begin' and 'End' correspond to the position of the characters in the technostring which limit the range to evaluate as a signal. Only numerical characters can be evaluated. Leading zeros are ignored.

The 'Index' only has to be entered if several data files are open at the same time. The file in the topmost position in the signal tree window has the index 0. All the other files, from top to bottom, then have the index 1, 2, and so forth. The index must always be 0 if only one file is open.

11.17 YatX and SetYatX

YatX

```
YatX('Expression','X','Continuous'=FALSE)
```

Arguments

'Expression'	Signal or expression
'X'	Position at which to read the value
'Continuous'	Optional parameter for permitting variable values of 'X'

Description

This function returns the Y-value of 'Expression' at position 'X' on the X-axis. You can use this function for time based signals as well as for length based signals.

In default mode, the 'Continuous' parameter is not set (or FALSE or 0). Then the function expects a constant X-value and returns a constant Y-value.

The 'X' parameter can also be variable, i.e. it can be a function itself. In this case, you have to activate the continuous mode by setting the 'Continuous' parameter to TRUE or 1. The function then determines the suitable Y-value for every value of 'X'.

SetYatX

```
SetYatX('Expression','SampleValue','X')
```

Arguments

'Expression'	Signal or expression to change
'SampleValue'	Value to insert at the position 'X'
'X'	X-position where to insert the value 'SampleValue'

Description

Using this function, you can create a copy of a signal in which a value has been changed. As a result, the function provides a copy of the signal 'Expression' where the value 'SampleValue' was inserted at the position 'X'.

You can also use this function to insert text.

The function behaves differently depending on whether it is applied to an equidistant signal. In the case of equidistant signals, a distinction is made between the following cases:

- If 'X' is smaller than the offset of the signal, the signal is returned unchanged.
- If 'X' corresponds to the size of the signal (see *XSize*) plus the sampling size, the signal is extended to include a sample with the value 'SampleValue'.
- In all other cases, the new value is inserted at the position 'X' or at the next smaller sample position.

For non-equidistant signals, the function replaces the value at the position 'X', if present, or inserts a new sample.

12 Filter functions

In the system, you can save digital filters that you configured using the filter editor. These filters are then also available as filter functions in the Expression builder.

12.1 LP

```
Lp('Expression','Omega')
```

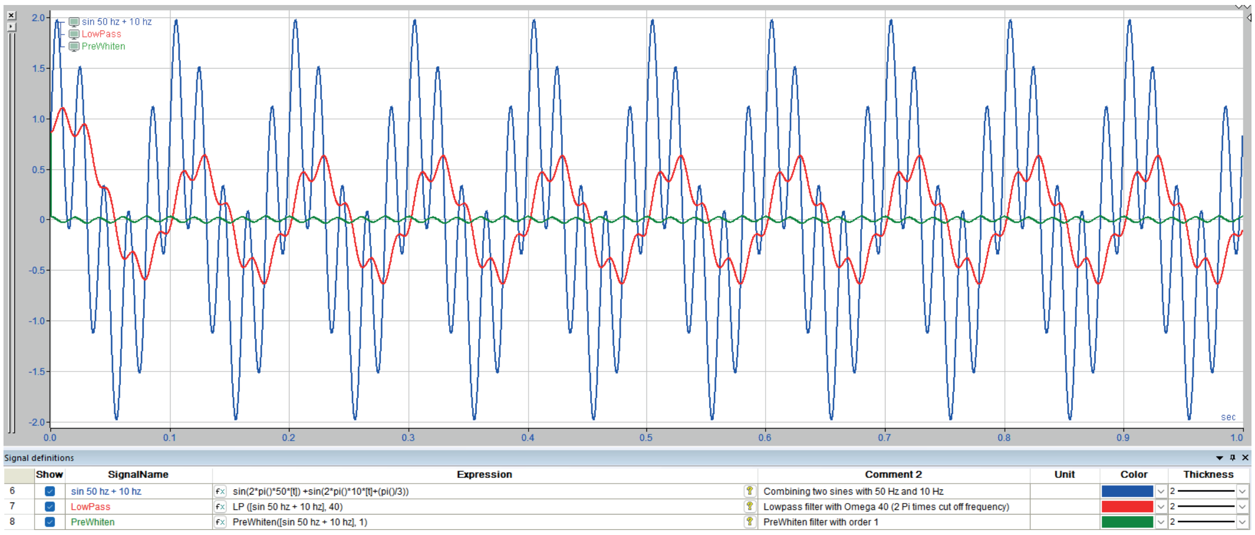
Arguments

'Expression'	Signal or expression
'Omega'	2 Pi times the cut off frequency for the lowpass filter

Description

This function applies a very simple digital lowpass filter with a limit frequency 'Omega'/(2*Pi). When applied to a signal 'Expression', the function returns a signal where the frequencies above the limit frequency are attenuated.

Note that due to the simple algorithm the attenuation is rather small and it is recommended to use the filter designer if specific filters are required.



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
6	✓	sin 50 hz + 10 hz	sin(2*pi()*50*[t]) +sin(2*pi()*10*[t]+(pi()/3))	Combining two sines with 50 Hz and 10 Hz		🟦	2
7	✓	LowPass	LP ([sin 50 hz + 10 hz], 40)	Lowpass filter with Omega 40 (2 Pi times cut off frequency)		🟥	2
8	✓	PreWhiten	PreWhiten([sin 50 hz + 10 hz], 1)	PreWhiten filter with order 1		🟩	2

12.2 PreWhiten

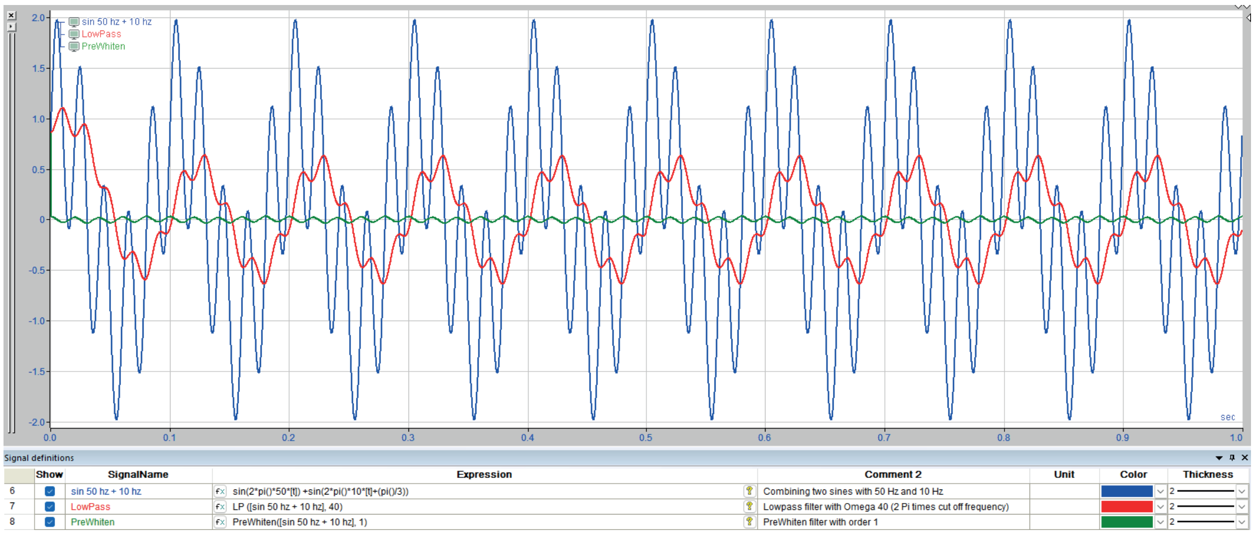
`PreWhiten('Expression','Order')`

Arguments

'Expression'	Signal or expression to be filtered
'Order'	Order of the FIR filter

Description

This function applies an FIR filter with coefficients that are determined using the Yule-Walker equation. It is a high-pass filter that only leaves white noise and the pulse components of the signal.



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
6	✓	sin 50 hz + 10 hz	sin(2*pi()*50*[t]) +sin(2*pi()*10*[t]+(pi()/3))	Combining two sines with 50 Hz and 10 Hz		🟦	2
7	✓	LowPass	LP ([sin 50 hz + 10 hz], 40)	Lowpass filter with Omega 40 (2 Pi times cut off frequency)		🟥	2
8	✓	PreWhiten	PreWhiten([sin 50 hz + 10 hz], 1)	PreWhiten filter with order 1		🟩	2

12.3 Vold-Kalman filter

```
VoldKalmanFilter('Expression','Position','Speed','Bandwidth','Filterpoles',
'Order','OTS'=FALSE)
```

Arguments

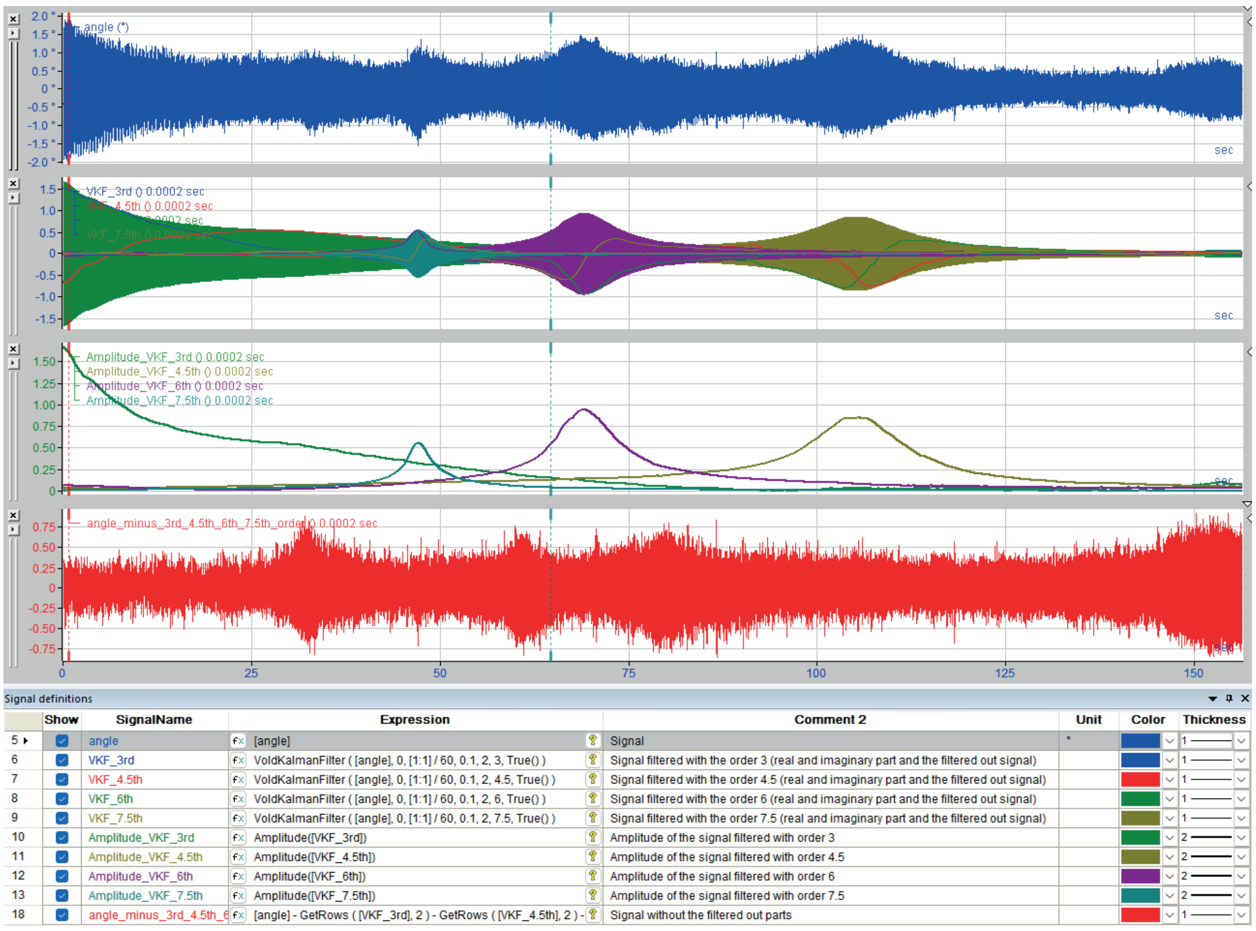
'Expression'	Signal or expression
'Position'	Position signal (can be 0 if 'Speed' is provided)
'Speed'	Speed signal (can be 0 if 'Position' is provided)
'Bandwidth'	Bandwidth of the filter
'Filterpoles'	1, 2, or 3 (3 offers highest accuracy but longest computation time)
'Order'	Order of the filtered signal
'OTS'	Optional output of the order-time signal

Description

This function applies the Vold-Kalman filter to the signal 'Expression' in order to extract specific sinusoidal components. The function returns a vector consisting of the real and imaginary parts of the complex result. If you set 'OTS' to TRUE, the extracted oscillation can also be returned as a third component.

Note

You can use the resulting vector directly in the *Amplitude* and *Phase* functions to obtain the amplitude and phase of the oscillation.


angle (°)
sec
VKF_3rd () 0.0002 sec
VKF_4.5th () 0.0002 sec
VKF_6th () 0.0002 sec
VKF_7.5th () 0.0002 sec
Amplitude_VKF_3rd () 0.0002 sec
Amplitude_VKF_4.5th () 0.0002 sec
Amplitude_VKF_6th () 0.0002 sec
Amplitude_VKF_7.5th () 0.0002 sec
angle_minus_3rd_4.5th_6th_7.5th_order () 0.0002 sec
0
25
50
75
100
125
150

Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5 ▸	✓	angle	[angle]	Signal	°	🟦	1
6	✓	VKF_3rd	VoldKalmanFilter ([angle], 0, [1:1] / 60, 0.1, 2, 3, True())	Signal filtered with the order 3 (real and imaginary part and the filtered out signal)		🟦	1
7	✓	VKF_4.5th	VoldKalmanFilter ([angle], 0, [1:1] / 60, 0.1, 2, 4.5, True())	Signal filtered with the order 4.5 (real and imaginary part and the filtered out signal)		🟥	1
8	✓	VKF_6th	VoldKalmanFilter ([angle], 0, [1:1] / 60, 0.1, 2, 6, True())	Signal filtered with the order 6 (real and imaginary part and the filtered out signal)		🟩	1
9	✓	VKF_7.5th	VoldKalmanFilter ([angle], 0, [1:1] / 60, 0.1, 2, 7.5, True())	Signal filtered with the order 7.5 (real and imaginary part and the filtered out signal)		🟫	1
10	✓	Amplitude_VKF_3rd	Amplitude([VKF_3rd])	Amplitude of the signal filtered with order 3		🟩	2
11	✓	Amplitude_VKF_4.5th	Amplitude([VKF_4.5th])	Amplitude of the signal filtered with order 4.5		🟫	2
12	✓	Amplitude_VKF_6th	Amplitude([VKF_6th])	Amplitude of the signal filtered with order 6		🟪	2
13	✓	Amplitude_VKF_7.5th	Amplitude([VKF_7.5th])	Amplitude of the signal filtered with order 7.5		🟦	2
18	✓	angle_minus_3rd_4.5th_6	[angle] - GetRows ([VKF_3rd], 2) - GetRows ([VKF_4.5th], 2) -	Signal without the filtered out parts		🟥	1

13 Technological functions

13.1 Amplitude and Phase

Amplitude

`Amplitude('Vector')`

Description

This function calculates the absolute value of a vector 'Vector', which describes the real and imaginary part of a complex number or a complex signal. The components of the vector can in turn be signals.

Note

The function always interprets the first entry of 'Vector' as the real part and the second as the imaginary part. If the vector contains more than two entries, only the first two are considered.

Phase

`Phase('Vector')`

Description

This function calculates the phase of a vector 'Vector', which describes the real and imaginary part of a complex number or a complex signal. The components of the vector can in turn be signals.

13.2 ChebyCoef

`ChebyCoef('Vector','Beginsegment','Endsegment','Order','CoverFactor'=1)`

Arguments

'Vector'	Measured values that are to be approximated
'Beginsegment'	Vector segment to be applied first
'Endsegment'	Vector segment to be applied last
'Order'	Order of the Chebyshev polynomial
'CoverFactor'	Optional parameter for determining the cover factor

Description

The *ChebyCoef* function calculates the coefficient of the Chebyshev polynomial of the order 'Order' across the cross profile of a vector 'Vector'. In the process, only the entries of the vector between the segments 'Beginsegment' and 'Endsegment' are considered. An optional cover factor 'CoverFactor' determines the behavior at the edges.

Example
The Chebyshev polynomial, named after the Russian mathematician Tschebyschow, is suitable for describing the profile of a roll gap in a mathematical way. Regarding the roll gap approximation, the orders 0 to 6 of the polynomial are relevant. The function provides the corresponding coefficients for this.

In real life, the coefficients can be derived from the measured values of a flatness measuring roll. The measured values of every zone are collected in a multidimensional signal 'Vector'. Each array field corresponds to a segment in terms of the cross profile of the gap.

13.3 CubicSpline

```
CubicSpline('Expression','X','Y')
```

Arguments

'Expression'	Auxiliary signal to determine the sampling rate of the result
'X'	X-coordinates (sampling points) that define the spline
'Y'	Y-coordinates that define the spline

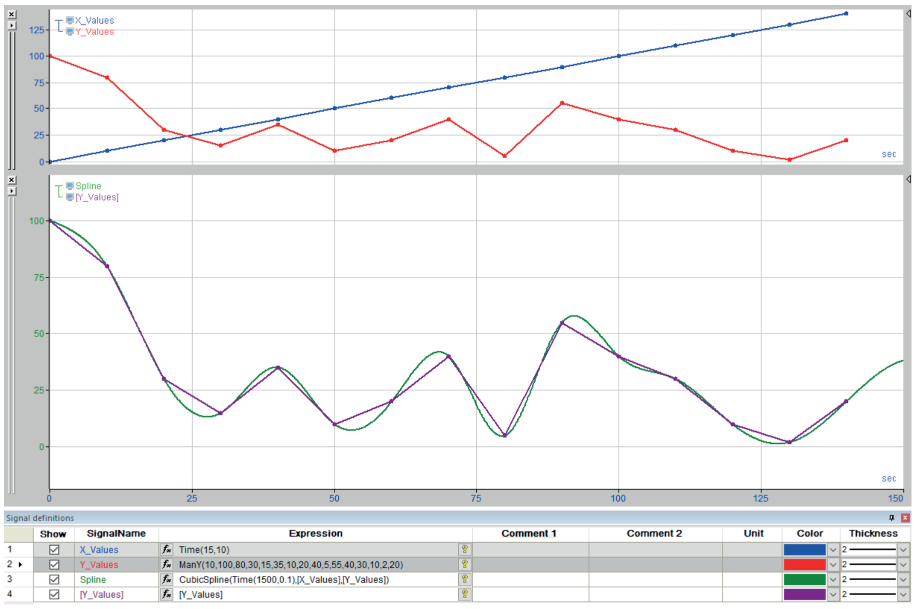
Description
As a result, this function returns a cubic spline that is aligned with the sampling points 'X' with the associated values 'Y'. The sampling rate and the weighting points of the result are determined by 'Expression'.

The X-coordinates do not have to be unambiguous and sorted. If there are several value pairs with the same X-coordinate, only the last value pair will be used for calculating the spline. The remaining value pairs are automatically sorted by X-coordinates.

Example

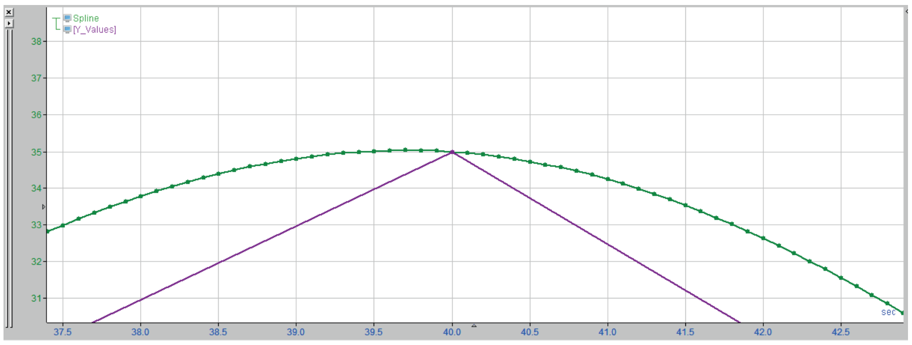
For a series of points, the function delivers a smoothed signal along the calculated spline as a result. The function can be used for interpolating a compensation curve for a signal with few samples:

A curve has only 17 samples over a time of 5000 s (Y-values, red curve). The corresponding X-coordinates – also only 17 values – are depicted as blue curve. The compensation curve as smoothed signal is to receive a significantly higher resolution (more samples). Therefore, the *CubicSpline* function is transferred a linear function with 5000 samples at an interval of 1 s as 'Expression' parameter.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	✓	X_Values	Time(15,10)				🟦	2
2	✓	Y_Values	ManY(10,100,80,30,15,35,10,20,40,5,55,40,30,10,2,20)				🟥	2
3	✓	Spline	CubicSpline(Time(1500,0.1),[X_Values],[Y_Values])				🟩	2
4	✓	[Y_Values]	[Y_Values]				🟪	2

With a high zoom level, the calculated samples of the compensation curve can be seen (green). The original X-Y coordinates form the knots of the splines (purple).



13.4 Exponential

```
Exponential('Coefs','X')
```

Arguments

'Coefs'	Vector with coefficients, e.g. the result of *LSQExponentialCoef*
'X'	X-coordinates (sampling points) where the exponential function is evaluated

Description

This function calculates the value of an exponential function a*exp(b*X) for every sample of 'X' with the coefficients a and b given as vector 'Coefs'. The function can visualize the results of *LSQExponentialCoef*.

13.5 LSQExponentialCoef

```
LSQExponentialCoef('X','Y')
```

Arguments

'X'	X-coordinates (sampling points) that define the interpolating exponential function
'Y'	Y-values that define the interpolation exponential function

Description

This function calculates the coefficients of an interpolating polynomial a*exp(b*x) for the function given by X-coordinates 'X' and corresponding function values 'Y' using the least squares method. The result of the function is a vector containing the coefficients.

13.6 LSQPolyCoef

`LSQPolyCoef('X','Y','Degree','PolynomialType'=0)`

Arguments

'X'	X-coordinates (sampling points) that define the interpolation polynomial
'Y'	Y-coordinates that define the interpolation polynomial
'Degree'	Polynomial degree (0 = average, 1 = linear, 2 = square, 3 = cubic, etc.)
'PolynomialType'	Defines the base polynomials to use; 0 = Lagrange (default), 1 = Chebyshev I, 2 = Chebyshev II, 3 = Legendre

Description

This function calculates the coefficient of an interpolation polynomial of the degree 'Degree' for value pairs 'X' and 'Y' according to the least squares method.

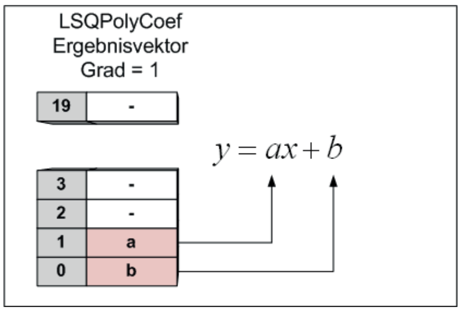
The result of the function is a vector (multidimensional signal, array) containing the coefficients. The array field with the index 0 contains the constant share or offset of the polynomial. The coefficients are written in array fields with ascending index according to their ascending degrees.

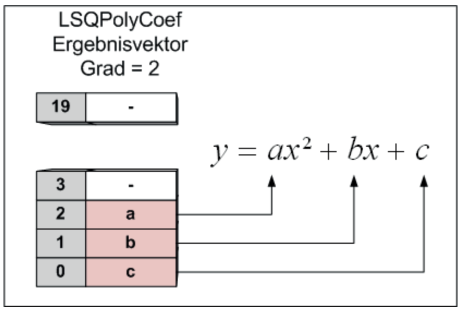
The function *Polynomial* can evaluate the polynomial.

With the optional parameter 'PolynomialType', you can use different base polynomials. The function supports the types Lagrange (default), Chebyshev I, Chebyshev II and Legendre.

Example

A quadratic approximation yields a polynomial of the form y = ax² + bx + c, i.e. the result is a vector with a total of 3 coefficients.





In principle, the function is based on an X-Y relationship, i.e. the operands X and Y can also be two different signals. If you want to calculate only one regression curve for a signal over time, the time values also have to be in the form of a signal, e.g. using *Xvalues*([signal]). You can use this time signal, whose Y-values are identical to the time along the X-axis, as operand 'X' in the *LSQPolyCoef* function.

13.7 Polynomial

```
Polynomial('Coefs','X','PolynomialType'=0)
```

Arguments

'Coefs'	Vector with coefficients, e.g. as result of *LSQPolyCoef*
'X'	X-coordinates (sampling points) at which to evaluate the polynomial
'PolynomialType'	Defines the base polynomials to use; 0 = Lagrange (default), 1 = Chebyshev I, 2 = Chebyshev II, 3 = Legendre

Description

This function calculates the polynomial value for every sample of 'X' on the basis of a coefficient vector 'Coefs'. The function is required to display regression lines or compensation curves whose coefficients were calculated with the *LSQPolyCoef* function prior to this.

With the optional parameter 'PolynomialType', you can use different base polynomials. The function supports the types Lagrange (default), Chebyshev I, Chebyshev II and Legendre.

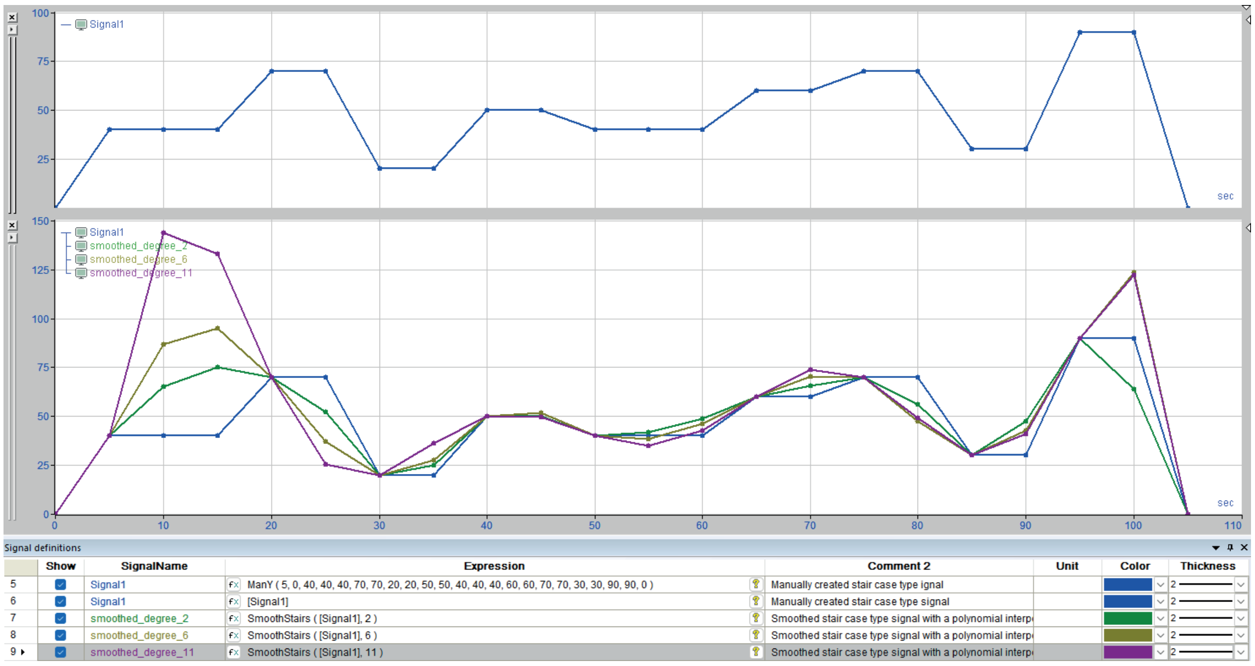
13.8 SmoothStairs

```
SmoothStairs('Expression','N')
```

'Expression'	Input signal in staircase form
'N'	Degree of the interpolation polynomial

Description

This function is used to smooth a staircase signal. For this purpose, an interpolation polynomial of degree 'N' is formed, the interpolation points of which are the first values of a staircase step.



	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
5	✓	Signal1	ManY (5, 0, 40, 40, 40, 70, 70, 20, 20, 50, 50, 40, 40, 40, 60, 60, 70, 70, 30, 30, 90, 90, 0)	Manually created stair case type ignal		🟦	2
6	✓	Signal1	[Signal1]	Manually created stair case type signal		🟦	2
7	✓	smoothed_degree_2	SmoothStairs ([Signal1], 2)	Smoothed stair case type signal with a polynomial interp		🟩	2
8	✓	smoothed_degree_6	SmoothStairs ([Signal1], 6)	Smoothed stair case type signal with a polynomial interp		🟫	2
9	✓	smoothed_degree_11	SmoothStairs ([Signal1], 11)	Smoothed stair case type signal with a polynomial interp		🟪	2

14 Spectral analysis (FT operations)

ibaAnalyzer can carry out the spectral analysis in the form of the Fast Fourier Transformation (FFT). With the FFT operations available, a time or length based signal cannot only be displayed in FFT mode, but also made available as a calculated expression and used for further analyses.

For most of the functions described here, you can either display an amplitude or a power trend. This is indicated by the suffix "Ampl" or "Power" in the function names.

14.1 AWeighting and DbScale

AWeighting

`AWeighting('Spectrum','type'=0)`

Arguments

'Spectrum'	Spectrum
'Type'	Specification of the type

Description

This function weights a spectrum according to the so-called A-weighting. This is a weighting filter that corresponds to human hearing.

Example

After applying this function, a spectrum can be assessed with regard to the perceptible noise emission. The rating has the following types:

- A: levels from approx. 20 phon to 40 phon
- B: levels from approx. 50 phon to 70 phon
- C: levels from approx. 80 phon to 90 phon
- D: for high sound pressures

DbScale

`DbScale('Spectrum','ref'=1)`

Arguments

'Spectrum'	Spectrum to scale logarithmically
'Reference'	Optional

Description

This function provides a logarithmic scaling in dB for a signal spectrum. For a meaningful result, the input must be the amplitude of a spectrum.

Tip

The following functions calculate such an amplitude: *FftAmpl, FftInTimeAmpl, FftOrderAnalysisAmpl*

14.2 FftAmpl and FftPower

```
e.g. FftAmpl('Expression','Samples','Window'=0,'SuppressDC'=FALSE)
```

Arguments

'Expression'	Signal or expression for which to calculate the Fourier transformation
'Samples'	Number of measured values to be considered and implied determination of the used time or length interval, depending on the sampling rate.
'Window'	Window type: 0 = Square 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
'SuppressDC'	DC suppression

Description
These functions calculate the amplitude or power of the Fourier transformation of the signal. The used time section is determined by rounding the number of 'Samples' to a power of 2.

Note

The parameter 'Samples' is rounded up. At least 128 measuring points must be used.

You can use the 'Window' parameter to set the window type that is used for the calculation. Optionally, you can activate DC suppression with the 'SuppressDC' parameter.

14.3 FftComplex

```
FftComplex('Expression','Inverse'=FALSE,'Normalize'=FALSE)
```

Arguments

'Expression'	Signal or expression for which to calculate the Fourier transformation
'Inverse'	Optional parameter to enable an inverse Fourier transformation
'Normalize'	Optional parameter to select a normalization

Description

This function performs a Fourier transformation for a complex signal across the entire expression and returns a vector with a real and imaginary part of the Fourier transformation. The input signal may consist both of an individual signal or a vector consisting of a real and imaginary part. A square window is used for the computation here.

If the parameter 'Inverse' is set to True() or 1, then an inverse Fourier transformation is computed. In this case, the function expects an input signal being either frequency-based or 1/length-based. The result of the operation then is a time based or length based signal accordingly.

The following values are permissible for the 'Normalize' parameter:

- **0**: No normalization is carried out.
- **1**: The result is divided by the number of samples. For an inverse transformation, the result is not changed.
- **2**: The result is divided by the square root of the number of samples. This applies both to a normal and an inverse transformation
- **Other values**: Function as with value 1.

The number of frequency samples is determined by the number of samples of the input signal. If N is even, N/2+1 frequency points are calculated; the first (DC component) and last point are purely real. If N is odd, (N+1)/2 frequency points are calculated; for those, the DC component is purely real.

14.4 FftInTimeAmpl and FftInTimePower

```
e.g. FftInTimeAmpl('Expression','Samples','#Freq','Min frequency'=0,'Max
frequency'=Sampling/2,'Window'=0,'Overlap'=0,'SuppressDC'=FALSE,'ZeroPad'=TRUE)
```

Arguments

'Expression'	Signal or expression for which to calculate the Fourier transformation
'Time'	Determination of the time or length intervals used. This rounds to an interval containing 2^N samples.
'#Freq'	Number of frequencies displayed
'Min frequency'	Minimum frequency
'Max frequency'	Maximum frequency
'Window'	Window type: 0 = Square 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
'Overlap'	Overlap factor
'SuppressDC'	DC suppression
'ZeroPad'	Adding zeros

Description

These functions calculate amplitude or power of the Fourier transformation of 'Expression' for sections with 2^N samples each. N is determined by rounding the product 'Time' × sampling frequency to a power of 2.

The result is a vector that contains '#Freq' equally divided frequencies between 'Min frequency' and 'Max frequency' per section. You can use the 'Window' parameter to set the window type that is used for the calculation.

The overlap factor determines the overlapping of the time segments and can be between 0 (no overlap) and 1 (complete overlap). Optionally, you can activate DC suppression with the 'SuppressDC' parameter.

If the parameter 'ZeroPad' is set to 1 or TRUE(), the last window is filled with zeros before calculating the FFT. If 'ZeroPad'=False, the last window is discarded.

Example

You can use the *FftInTime* function to display fluctuating frequencies over time. The resulting vector can be displayed in a 2D view for this purpose.

14.5 FftOrderAnalysisAmpl and FftOrderAnalysisPower

```
e.g. FftOrderAnalysisAmpl('Expression','Time','Freq','MinOrder'=0,'MaxOrder',
'Order subdivision'=1,'Window'=0,'Overlap'=0,'SuppressDC'=FALSE)
```

Arguments

'Expression'	Signal or expression for which to carry out the order analysis
'Time'	Determination of the time or length intervals used
'Freq'	Basic frequency for the order analysis (rotational frequency)
'MinOrder'	Minimum displayed order
'MaxOrder'	Maximum displayed order
'Order subdivision'	Grid width between the integer orders
'Window'	Window type: 0 = Square 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
'Overlap'	Overlap factor
'SuppressDC'	DC suppression

Description

This function calculates the orders (i.e. multiples of a basic frequency 'Freq') for a signal and returns a vector with the orders between 'MinOrder' and 'MaxOrder'. The number of data points per order is determined by the parameter 'Order subdivision'.

You can use the 'Window' parameter to set the window type that is used for the calculation. The overlap factor determines the overlapping of the time segments and can be between 0 (no overlap) and 1 (complete overlap). Optionally, you can activate DC suppression with the 'SuppressDC' parameter.

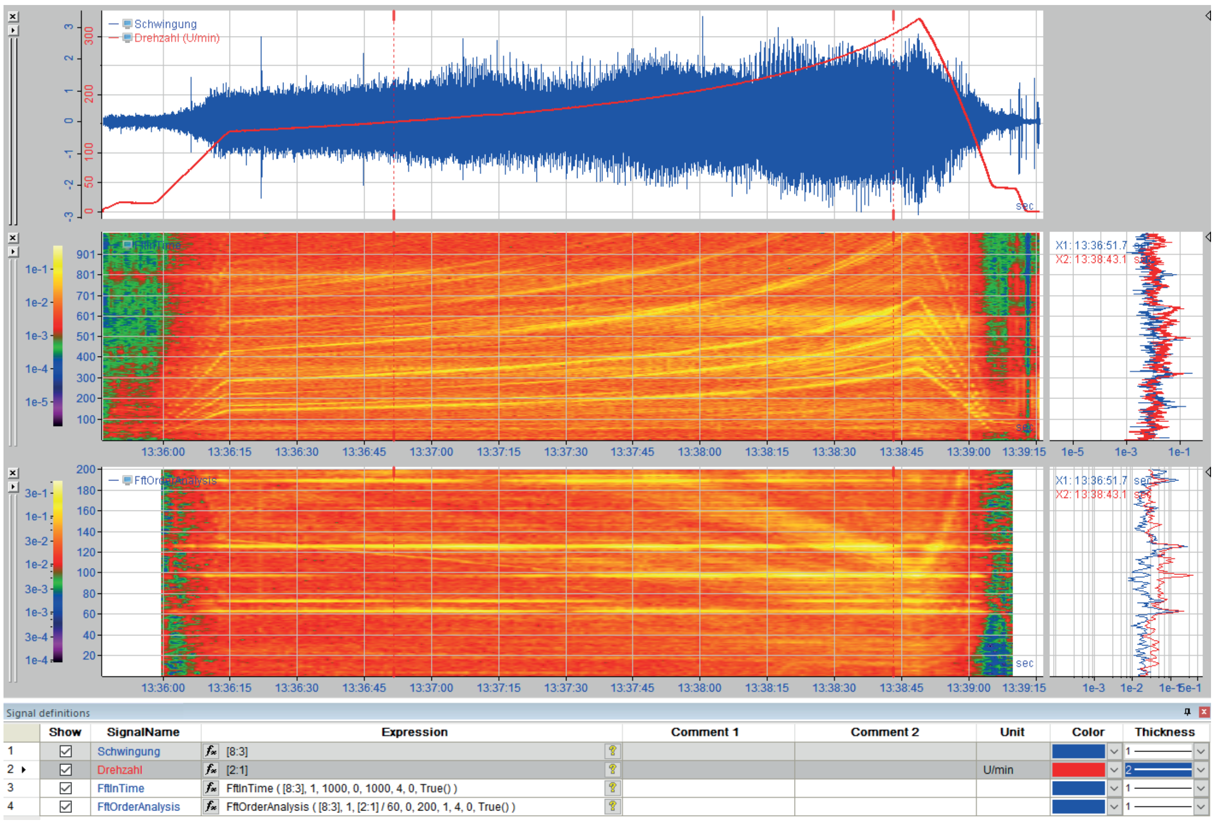
Contrary to the *FftInTime* function, the time/frequency is no longer displayed on the Y-axis, but the rotational frequency and its multiple, i.e. the orders. The frequency axis is distorted in accordance with the current revolutions per minute so that the orders are no longer displayed as a curve, but as straight lines. Depending on the function, either an amplitude trend (*FftOrderAnalysisAmpl*) or a power trend (*FftOrderAnalysisPower*) is calculated.

Note

The function does not yield results if the number of signal points per revolution is more than twice as high as the selected parameter 'Time'.

Example

With the *FftOrderAnalysis* function, you can calculate an order analysis. Frequencies corresponding to the motor speed or its multiple are called orders. The first order complies with the frequency of the motor speed, the second order complies with the frequency of the first order multiplied by the factor 2, etc. The order analysis calculates the level or the level curve of this order.



Signal definitions

	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☑	Schwingung	[8:3]				🟦	1
2	☑	Drehzahl	[2:1]			U/min	🟥	2
3	☑	FftInTime	FftInTime ([8:3], 1, 1000, 0, 1000, 4, 0, True())				🟦	1
4	☑	FftOrderAnalysis	FftOrderAnalysis ([8:3], 1, [2:1] / 60, 0, 200, 1, 4, 0, True())				🟦	1

An interpolation is carried out between the individual signal points to calculate the *FftOrderAnalysis*.

14.6 FftPeaksInTimeAmpl and FftPeaksInTimePower

```
e.g. FftPeaksInTimeAmpl('Expression','Time','#Peaks','Min frequency'=0,'Max
frequency'=Sampling/2,'Window'=0,'Overlap'=0,'SuppressDC'=FALSE,'ZeroPad'=FALSE)
```

Arguments

'Expression'	Signal or expression for which to evaluate the frequency spectra
'Time'	Determination of the time or length intervals used. This rounds to an interval containing 2^N samples.
'#Peaks'	Number of peaks displayed
'Min frequency'	Optional parameter for the minimally considered frequency, i.e. peaks at lower frequency are not shown
'Max frequency'	Optional parameter for the maximum considered frequency, i.e. peaks at higher frequency are not shown
'Window'	Window type: 0 = Square 1 = Bartlett 2 = Blackman 3 = Hamming 4 = Hanning 5 = Blackman-Harris 6 = Flat top
'Overlap'	Overlap factor
'SuppressDC'	DC suppression
'ZeroPad'	Adding zeros

Description

This function is used to calculate frequency peaks across smoothed time intervals, which are determined by the parameter 'Time'. In the process, the '#Peaks' are calculated, which are the highest peaks between the 'Min frequency' and 'Max frequency' frequencies. The timeline of the frequency and the pairs of peak values are returned as a vector.

The entries are sorted according to the following pattern:

- Index 0: Frequency with the highest peak
- Index 1: Amplitude/power of the highest peak
- Index 2: Frequency with the second highest peak
- Index 3: Amplitude/power of the second highest peak
- Etc.

Tip

To read out the requested values from the result array, you can use the *GetRows* function.

You can use the 'Window' parameter to set the window type that is used for the calculation. The overlap factor determines the overlapping of the time segments and can be between 0 (no overlap) and 1 (complete overlap). Optionally, you can activate DC suppression with the 'SuppressDC' parameter.

If the parameter 'ZeroPad' is set to 1 or TRUE(), the last window is filled with zeros before calculating the FFT. If 'ZeroPad'=False, the last window is discarded.

14.7 FftReal and FftRealInverse

```
e.g. FftReal('Expression','Normalize'=FALSE)
```

Arguments

'Expression'	Signal or expression for which to calculate the Fourier transformation
'Normalize'	Optional parameter to enable the normalization

Description of FftReal

This function performs a Fourier transformation for a real signal across the entire expression and, as a result, returns a vector with a real and imaginary part of the Fourier transformation. A square window is used here.

If the parameter 'Normalize' is set to TRUE() 1, a normalization is performed. If the number of samples (N) of the signal is odd, all frequency values except for the DC component are divided by N/2. If N is even, all frequency values except for the DC component and the last value are divided by N/2.

The number of frequency samples is determined by the number of samples of the input signal. If N is even, N/2+1 frequency points are calculated; the first (DC component) and last point are purely real. If N is odd, (N+1)/2 frequency points are calculated; for those, the DC component is purely real.

Description of FftRealInverse

This function calculates the inverse Fourier transformation, as created with *FftReal*. The result is real here and the input signal must accordingly be a vector consisting of a real and imaginary part. Other than that, the function works just like *FftReal*.

14.8 IntSpectrum

```
IntSpectrum('Spectrum')
```

Description

This function integrates a given spectrum. In this way, the vibration speed can be calculated from the frequency of an acceleration sensor.

15 Text functions

15.1 InfoFieldText, ChannelInfoFieldText and ModuleInfoFieldText

These functions read out information from an info field of a data file, a signal or a module available as a text channel.

Arguments

'Index'	Index of the data file, the signal or module
'InfoField'	Info field to read; put in quotation marks.
'Begin'	Optional: First character of the field content to be read. If you do not specify a value, the entire content is read.
'End'	Optional: Last character of the field content to be read. If you do not specify a value, the content is read from 'Begin' to the last character.

InfoFieldText

```
InfoFieldText('FileIndex',"'InfoField'",'Begin'=0,'End'=Text end)
```

Description

This function returns the content of an info field as a text signal.

Tip

If you double-click on the desired info field, *ibaAnalyzer* automatically inserts the corresponding function as new signal into the signal table. If required, you then only have to customize the signal name and beginning or end.

If you want to read out the content of an info field as numerical value, use the *InfoField* function.

ChannelInfoFieldText

```
ChannelInfoFieldText('ChannelIndex',"'InfoField'",'Begin'=0,'End'=Text end)
```

Description

This function returns the content of an info field of a signal as text.

ModuleInfoFieldText

```
ModuleInfoFieldText('FileIndex','ModuleIndex',"'InfoField'",'Begin'=0,
'End'=Text end)
```

Note

You have to specify 2 indices for this function: The index of the data file as the first argument and the index of the module as the second. All other arguments remain the same.

Arguments

'FileIndex'	Index of the data file to which the module belongs
'ModuleIndex'	The index of the module
'InfoField'	Info field to read from the module; put in quotation marks.
'Begin'	Optional: First character of the field content to be read. If you do not specify a value, the entire content is read.
'End'	Optional: Last character of the field content to be read. If you do not specify a value, the content is read from 'Begin' to the last character.

Description
This function works as the *InfoFieldText* and *ChannelInfoFieldText* functions, however, it refers to the info fields of a module and not the data file or signal. The function returns a text signal with the content of the specified info field.

15.2 CharValue

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

Arguments

'Text'	Text signal or input text (input in quotation marks)
'CharNumber'	Zero-based position of the character to evaluate

Description
This function returns the ASCII value of the character at position 'CharNumber' in 'Text'. As default the first character is used, which has the 'CharNumber' zero.

15.3 CompareText

```
CompareText('Text1','Text2','CaseSensitive'=TRUE,'NumericalSort'=FALSE)
```

Arguments

'Text1/2'	Strings to compare
'CaseSensitive'	Optional parameter to enable case sensitivity during comparison
'NumericalSort'	Optional parameter to specify whether digits are compared individually or whether combinations of digits are compared as numbers.

Description
With this function, you can compare the text information lexicographically. The function works with contents of text signals as well as with plain text that you can enter directly in the signal definition using quotation marks.

Comparison and results:

- Result -1: The first text comes before the second text lexicographically.
- Result 0: Both texts contain the same information.
- Result +1: The first text comes after the second text lexicographically.

Example

The following table shows the general functionality and the impact of the 'CaseSensitive' parameter:

Text1	**Text2**	**Result for**		**Comment**
		CompareText ("Text1","Text2",0)	**CompareText ("Text1","Text2",1)**	
1234 abcd	1234 abcd	0	0	Text1 = Text2
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 before "a"
12340 abcd	1234 abcd	1	1	Text1 > Text2 "0" comes after " " (space)
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"

The following table shows the impact of the 'NumericalSort' parameter:

Text1	**Text2**	**Result for**	
		CompareText ("Text1","Text2",0,0)	**CompareText ("Text1","Text2",0,1)**
12034 abcd	1234 abcd	-1 Text1 < Text2 "0" comes before "3"	1 Text1 > Text2 "12034" comes after "1234"

15.4 ConcatText

`ConcatText('t1','t2',…)`

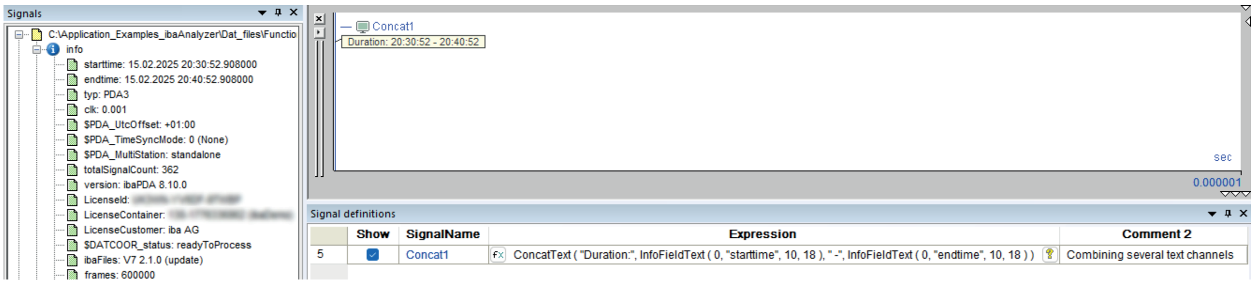
Description

You can use this function to merge several text signals. Numerical signals are also permissible here. These are automatically converted into text.

If the X-positions of the individual signals do not match, a new entry is created for each existing X-position and the missing entry is replaced by the left neighbor, if present.

Example

The file ID is available as a text signal for signals in series. There is a product counter for the end products. The result brings both pieces of information together.

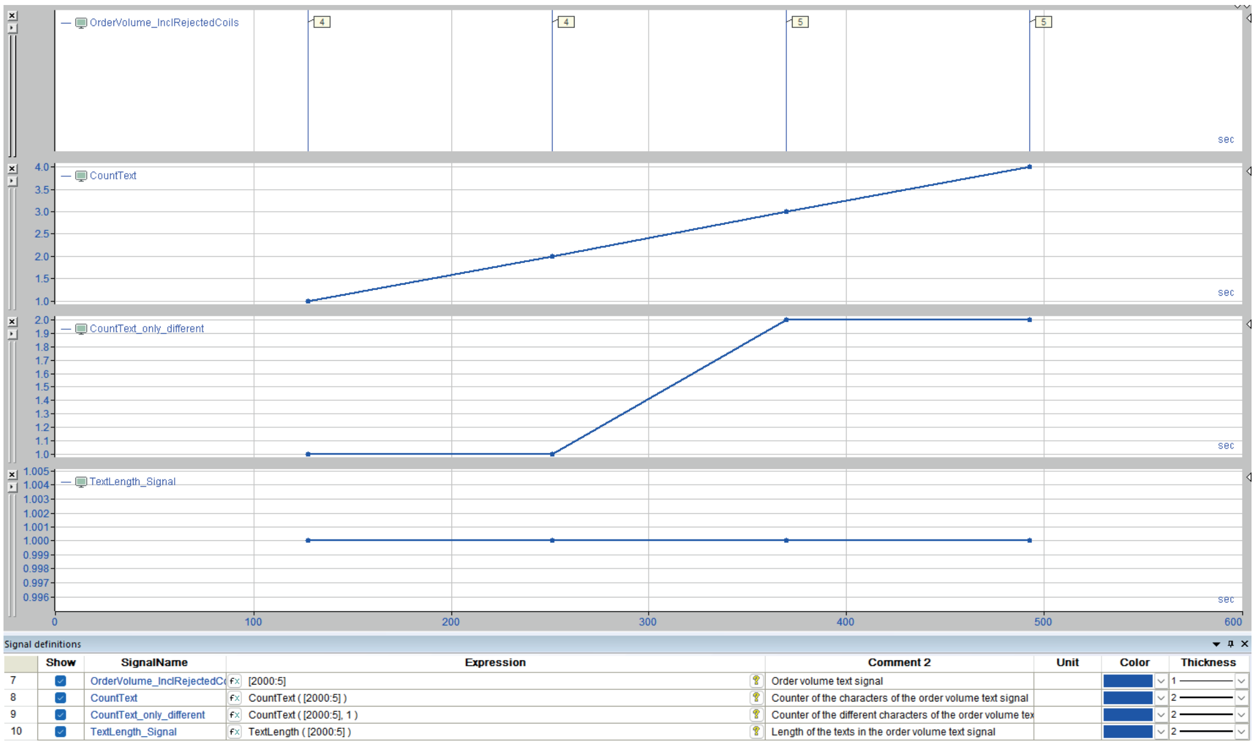


```
C:\Application_Examples_ibaAnalyzer\Dat_files\Functio
  info
    starttime: 15.02.2025 20:30:52.908000
    endtime: 15.02.2025 20:40:52.908000
    typ: PDA3
    clk: 0.001
    $PDA_UtcOffset: +01:00
    $PDA_TimeSyncMode: 0 (None)
    $PDA_MultiStation: standalone
    totalSignalCount: 362
    version: ibaPDA 8.10.0
    LicenseId: [redacted]
    LicenseContainer: [redacted]
    LicenseCustomer: iba AG
    $DATCOOR_status: readyToProcess
    ibaFiles: V7 2.1.0 (update)
    frames: 600000
```

Duration: 20:30:52 - 20:40:52

	Show	SignalName	Expression	Comment 2
5	✓	Concat1	ConcatText ("Duration:", InfoFieldText (0, "starttime", 10, 18), " -", InfoFieldText (0, "endtime", 10, 18))	Combining several text channels

15.5 CountText and TextLength



Signal definitions

	Show	SignalName	Expression	Comment 2	Unit	Color	Thickness
7	✓	OrderVolume_InclRejectedC	[2000:5]	Order volume text signal		🟦	1
8	✓	CountText	CountText ([2000:5])	Counter of the characters of the order volume text signal		🟦	2
9	✓	CountText_only_different	CountText ([2000:5], 1)	Counter of the different characters of the order volume tex		🟦	2
10	✓	TextLength_Signal	TextLength ([2000:5])	Length of the texts in the order volume text signal		🟦	2

CountText

```
CountText('Text','CountOnlyDifferent'=0,'Reset'=0)
```

Arguments

'Text'	Text signal or input text (input in quotation marks)
'CountOnlyDifferent'	Optional parameter to count only text which is different from the previous text flag
'Reset'	Optional parameter to reset the counter

Description

This function returns the number of texts within a text signal 'Text'. If the parameter 'CountOnlyDifferent' is set to TRUE, only texts different from the previous text flag are counted. Using the digital signal 'Reset', the counter can be reset to zero.

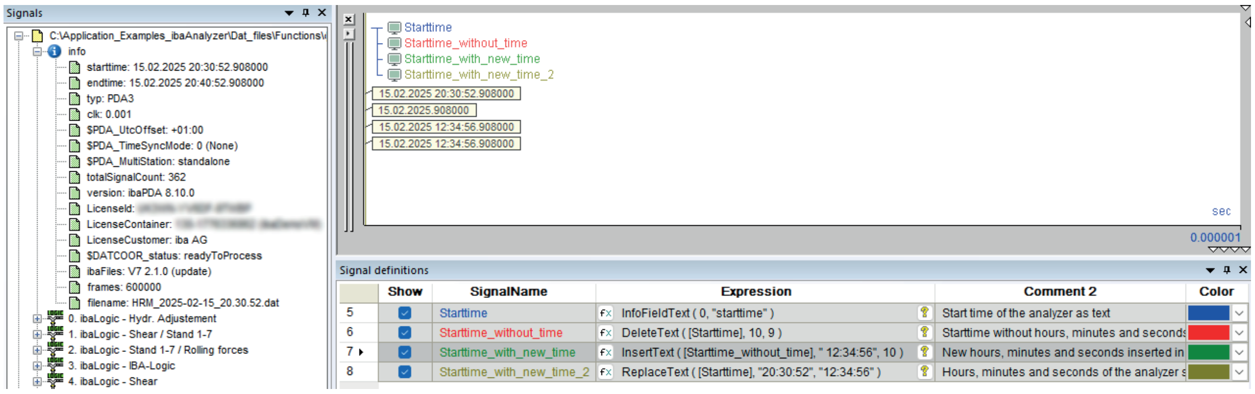
TextLength

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

Description

This function returns the number of characters in 'Text'.

15.6 DeleteText, InsertText and ReplaceText



Signals

C:\Application_Examples_ibaAnalyzer\Dat_files\Functions\
- info
  - starttime: 15.02.2025 20:30:52.908000
  - endtime: 15.02.2025 20:40:52.908000
  - typ: PDA3
  - clk: 0.001
  - $PDA_UtcOffset: +01:00
  - $PDA_TimeSyncMode: 0 (None)
  - $PDA_MultiStation: standalone
  - totalSignalCount: 362
  - version: ibaPDA 8.10.0
  - LicenseId: [redacted]
  - LicenseContainer: [redacted]
  - LicenseCustomer: iba AG
  - $DATCOOR_status: readyToProcess
  - ibaFiles: V7 2.1.0 (update)
  - frames: 600000
  - filename: HRM_2025-02-15_20.30.52.dat
- 0. ibaLogic - Hydr. Adjustement
- 1. ibaLogic - Shear / Stand 1-7
- 2. ibaLogic - Stand 1-7 / Rolling forces
- 3. ibaLogic - IBA-Logic
- 4. ibaLogic - Shear

Starttime
Starttime_without_time
Starttime_with_new_time
Starttime_with_new_time_2

15.02.2025 20:30:52.908000
15.02.2025.908000
15.02.2025 12:34:56.908000
15.02.2025 12:34:56.908000

Signal definitions

	Show	SignalName	Expression	Comment 2	Color
5	✓	Starttime	InfoFieldText (0, "starttime")	Start time of the analyzer as text	🟦
6	✓	Starttime_without_time	DeleteText ([Starttime], 10, 9)	Starttime without hours, minutes and seconds	🟥
7	✓	Starttime_with_new_time	InsertText ([Starttime_without_time], " 12:34:56", 10)	New hours, minutes and seconds inserted in	🟩
8	✓	Starttime_with_new_time_2	ReplaceText ([Starttime], "20:30:52", "12:34:56")	Hours, minutes and seconds of the analyzer s	🟫

DeleteText

`DeleteText('Text','StartPos','Length')`

Arguments

'Text'	Text signal or input text (input in quotation marks)
'StartPos'	Zero-based index of the first deleted character
'Length'	Number of characters to delete

Description

This function removes a number of 'Length' characters from 'Text'. The first character to delete is at position 'StartPos' within the text.

InsertText

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

Arguments

'Text1'	Original text
'Text2'	Text which is inserted in the original text
'Pos'	Zero-based character position where 'Text2' is inserted

Description

This function inserts 'Text2' into 'Text1' at the zero-based character position 'Pos'. If 'Pos' is less or equal to zero, then 'Text2' is prepended to 'Text1'. If 'Pos' is larger or equal to the length of 'Text1', then 'Text2' is appended to 'Text1'.

ReplaceText

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

Arguments

'Text'	Text signal or input text (input in quotation marks)
'SearchText'	Text to be replaced within 'Text'
'ReplaceText'	If 'SearchText' is found, it is replaced by 'ReplaceText'

Description

This function replaces all occurrences of 'SearchText' within 'Text' with 'ReplaceText'.

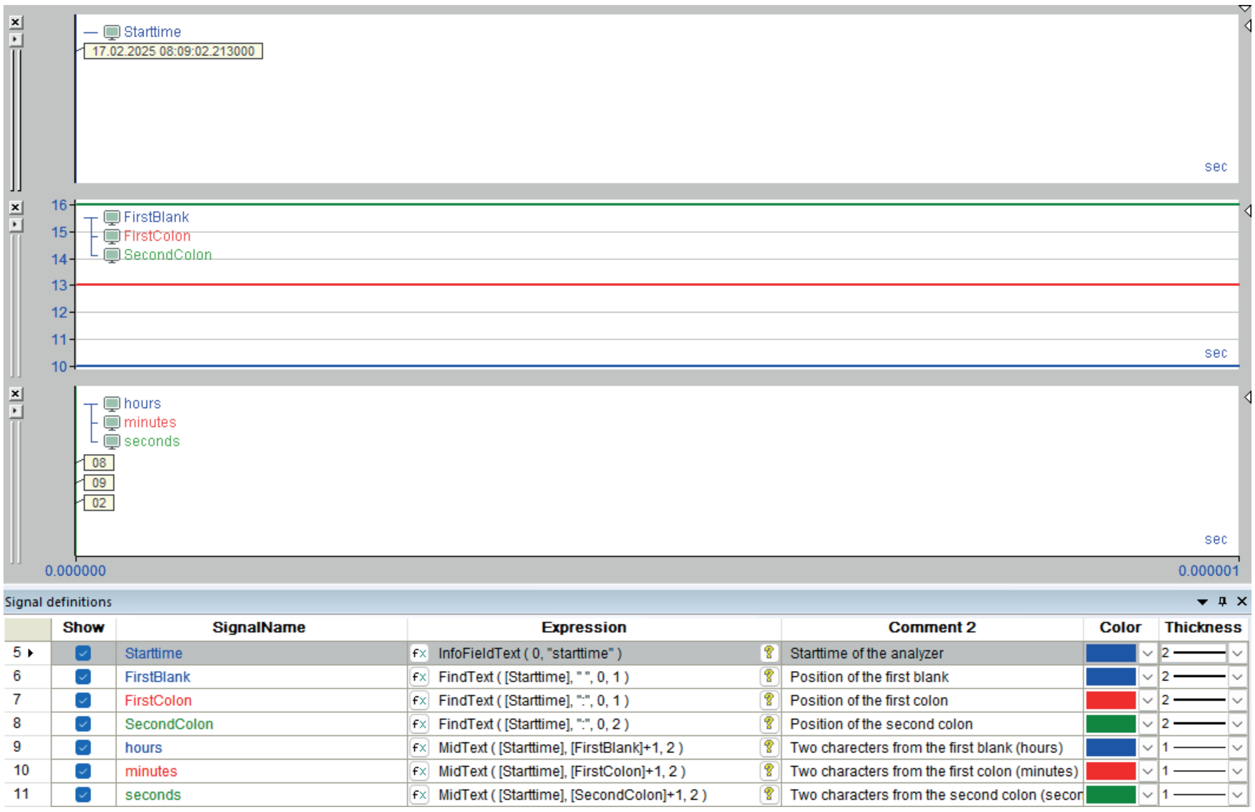
15.7 Merge

```
Merge('t1','t2',…)
```

Description

This function combines two or more text signals into one by displaying the entries of all signals in a common graph. If two entries of different signals have the same X-value, only the text of the signal listed first is displayed.

15.8 MidText and FindText



Signal definitions

	Show	SignalName	Expression	Comment 2	Color	Thickness
5	✓	Starttime	InfoFieldText (0, "starttime")	Starttime of the analyzer	🟦	2
6	✓	FirstBlank	FindText ([Starttime], " ", 0, 1)	Position of the first blank	🟦	2
7	✓	FirstColon	FindText ([Starttime], ":", 0, 1)	Position of the first colon	🟥	2
8	✓	SecondColon	FindText ([Starttime], ":", 0, 2)	Position of the second colon	🟩	2
9	✓	hours	MidText ([Starttime], [FirstBlank]+1, 2)	Two charecters from the first blank (hours)	🟦	1
10	✓	minutes	MidText ([Starttime], [FirstColon]+1, 2)	Two characters from the first colon (minutes)	🟥	1
11	✓	seconds	MidText ([Starttime], [SecondColon]+1, 2)	Two characters from the second colon (secor	🟩	1

MidText

```
MidText('Text','StartPos','Length')
```

Arguments

'Text'	Text signal or input text (input in quotation marks)
'StartPos'	Zero-based index of the first character to extract from the text
'Length'	Number of characters to extract from the text

Description

This function returns a number 'Length' characters from within 'Text'. The resulting string starts with the character at the zero-based position 'StartPos'.

FindText

```
FindText('Text1','Text2','CaseSensitive'=FALSE,'N'=1)
```

Arguments

'Text1'	Text signal or input text (input in quotation marks)
'Text2'	Text which needs to be identified within the input text
'CaseSensitive'	Optional parameter to enable case sensitivity during text search
'N'	Parameter to find not the first but rather the 'N'th occurrence of 'Text2'

Description

This function returns a zero-based index of the position of the 'N'th occurrence of 'Text2' within 'Text1'. If the parameter 'CaseSensitive' is set to TRUE, a case sensitive search is conducted. If the 'Text2' is not found, the result is -1.

15.9 SortText and XSortText

SortText

```
SortText('Expression','Descending'=FALSE,'CaseSensitive'=TRUE,
'NumericalSort'=FALSE)
```

Arguments

'Expression'	Text signal for which to sort the samples
'Descending'	Optional parameter for reversing the sort order
'CaseSensitive'	Optional parameter to enable case sensitivity during comparison
'NumericalSort'	Optional parameter to specify whether digits are compared individually or whether combinations of digits are compared as numbers.

Description

This function sorts entries of a text signal ('Expression') in ascending order.

Default settings:

- Sorting in ascending order ('Descending'=FALSE).
- Case sensitivity is applied ('CaseSensitive' = TRUE): Uppercase letters come before lowercase letters.
- Numbers are compared as text ('NumericalSort' = FALSE):
  If FALSE, 12034 comes before 1234 because "0" comes before "3".
  If TRUE, 12034 comes after 1234 because 12034 > 1234.

To sort the text entries in descending order, set 'Descending' to TRUE. To ignore upper and lower case when sorting (so that "A" = "a"), set 'CaseSensitive' to FALSE. To compare numbers as numeric values, set 'NumericalSort' to TRUE.

XSortText

```
XSortText('Expression','Descending'=FALSE,'CaseSensitive'=TRUE,
'NumericalSort'=FALSE)
```

Arguments

'Expression'	Text signal for which to sort the samples
'Descending'	Optional parameter for reversing the sort order
'CaseSensitive'	Optional parameter to enable case sensitivity during comparison
'NumericalSort'	Optional parameter to specify whether digits are compared individually or whether combinations of digits are compared as numbers.

Description

This function sorts all (X,Text) pairs of a text signal ('Expression') based on their text entries in ascending order and returns the corresponding X-values. This means that the last Y-value of the sorted signal corresponds to the X-value at which 'Expression' reaches its maximum. The X-axis and Y-axis of the sorted signal are of equal length.

Default settings as in *SortText*.

To sort the text entries in descending order, set 'Descending' to TRUE. To ignore upper and lower case when sorting (so that "A" = "a"), set 'CaseSensitive' to FALSE. To compare numbers as numeric values, set 'NumericalSort' to TRUE.

15.10 ToText and FromText

ToText

```
ToText('Expression','Format'="%g",'datatype'=0)
```

Arguments

'Expression'	Signal or expression whose content is to be converted into a text channel
'Format'	Optional parameter for a format string
'datatype'	Optional parameter that determines the floating-point format ■ 'datatype'=0: Default value ■ 'datatype'=1: Output is formatted as signed 16-bit integer ■ 'datatype'=2: Output is formatted as unsigned 16-bit integer ■ 'datatype'=3: Output is formatted as signed 32-bit integer ■ 'datatype'=4: Output is formatted as unsigned 16-bit integer

Description

This function converts a numerical signal value into a text signal. In case of equidistant samples of the 'Expression' input signal and a constant Y-value, only the value of the first signal point is entered and displayed as sample in the text signal. If the Y-value changes, a sample is entered and displayed in the text signal for every new Y-value.

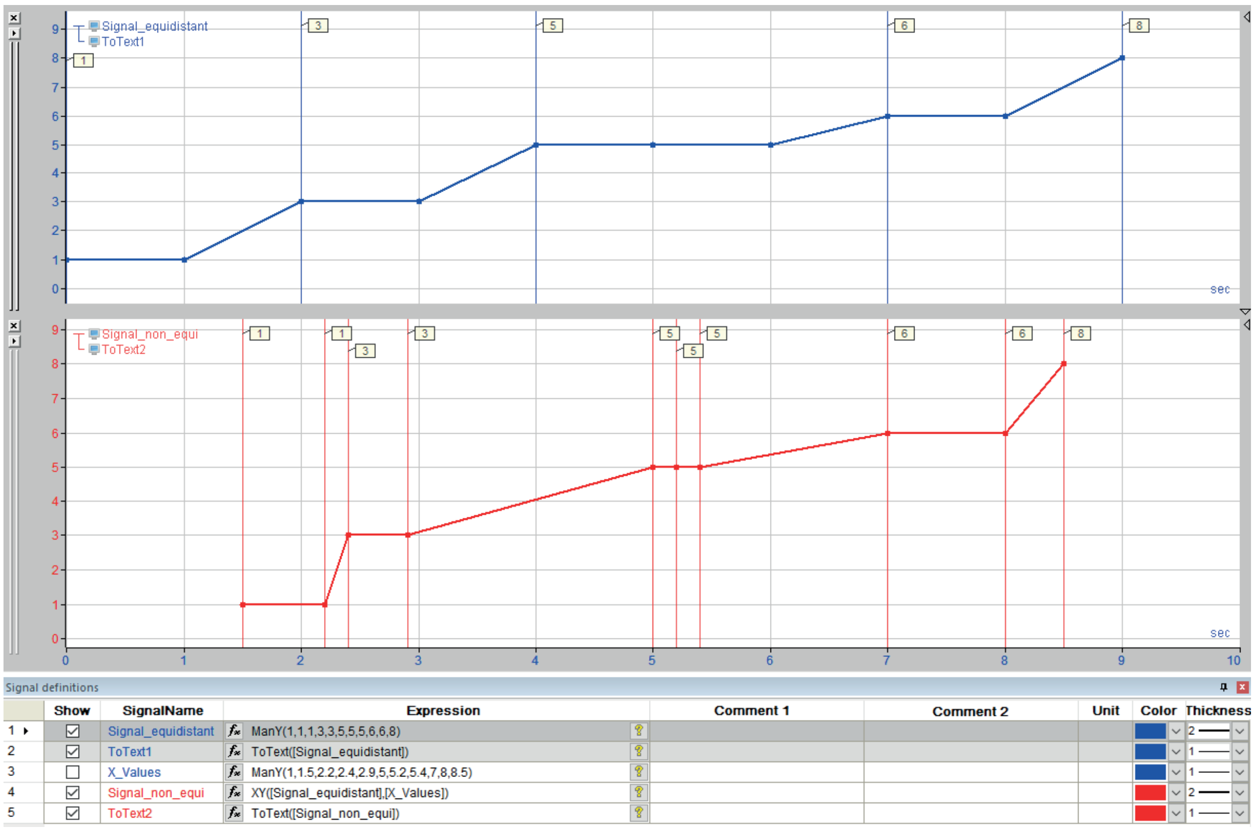
If the 'Expression' input signal does not contain equidistant samples, a sample is entered and displayed in the text signal for each input sample.

Enter the optional parameter 'Format' according to C printf syntax. You can only indicate a parameter (%) complying with an IEEE 32 bit floating point value. Default value is %g. This value is also used if you do not indicate the optional parameter.

Examples:
%g = conversion of the floating point value into a text
%. 4f = text/number with 4 decimal places, etc.

Example

The *ToText* function can be helpful, e.g. if trends are visualized containing vast amounts of data. Without constantly changing between the marker and signal view, the numerical values can be easily displayed. The following illustration shows the use of the *ToText* function.



	Show	SignalName	Expression	Comment 1	Comment 2	Unit	Color	Thickness
1	☑	Signal_equidistant	ManY(1,1,1,3,3,5,5,5,6,6,8)				🟦	2
2	☑	ToText1	ToText([Signal_equidistant])				🟦	1
3	☐	X_Values	ManY(1,1.5,2.2,2.4,2.9,5,5.2,5.4,7,8,8.5)				🟦	1
4	☑	Signal_non_equi	XY([Signal_equidistant],[X_Values])				🟥	2
5	☑	ToText2	ToText([Signal_non_equi])				🟥	1

FromText

```
FromText('Text','Begin','End')
```

Arguments

'Text'	Text signal to convert
'Begin'	Optional: First character of the field content to be read. If you do not specify a value, the entire content is read.
'End'	Optional: Last character of the field content to be read. If you do not specify a value, the content is read from 'Begin' to the last character.

Description

This function converts the content of the text signal 'Text' into a numerical value. You can use the 'Begin' and 'End' parameters optionally as indices to not convert the entire string. By default, 'Begin'=0 and 'End'=length of the string is used.

15.11 TrimText

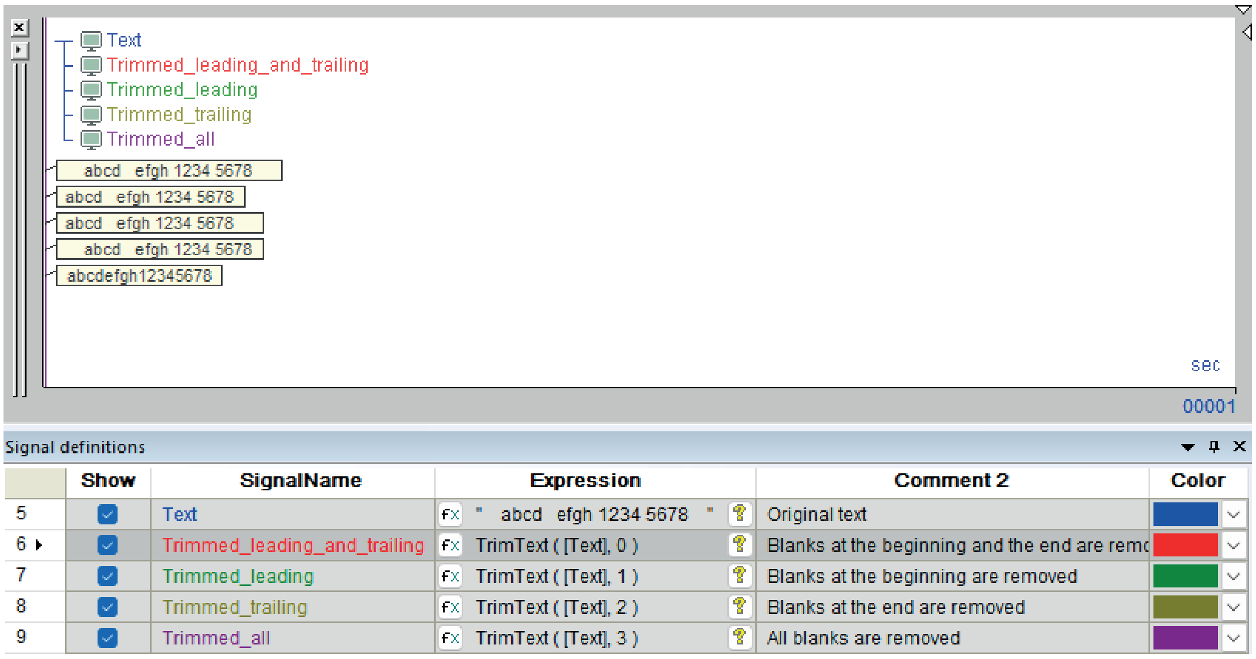
`TrimText('Text','RemoveOption'=0)`

Arguments

'Text'	Text signal or expression from which to remove spaces
'RemoveOption'	Parameter for setting the operating mode: 0: Removing spaces before and after the text (default) 1: Removing spaces before the text only 2: Removing spaces after the text only 3: Removing all spaces, also in the text

Description

With this function, you can delete the spaces from texts. You can use this function for text signals that are already part of data files and for results of the *InfoFieldText* and *ToText* functions.



Text
Trimmed_leading_and_trailing
Trimmed_leading
Trimmed_trailing
Trimmed_all

```
   abcd   efgh 1234 5678   
abcd   efgh 1234 5678
abcd   efgh 1234 5678   
   abcd   efgh 1234 5678
abcdefgh12345678
```

sec
00001

Signal definitions

	Show	SignalName	Expression	Comment 2	Color
5	✓	Text	" abcd efgh 1234 5678 "	Original text	🟦
6	✓	Trimmed_leading_and_trailing	TrimText ([Text], 0)	Blanks at the beginning and the end are remo	🟥
7	✓	Trimmed_leading	TrimText ([Text], 1)	Blanks at the beginning are removed	🟩
8	✓	Trimmed_trailing	TrimText ([Text], 2)	Blanks at the end are removed	🟫
9	✓	Trimmed_all	TrimText ([Text], 3)	All blanks are removed	🟪

16 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
Koenigswarterstrasse 44
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