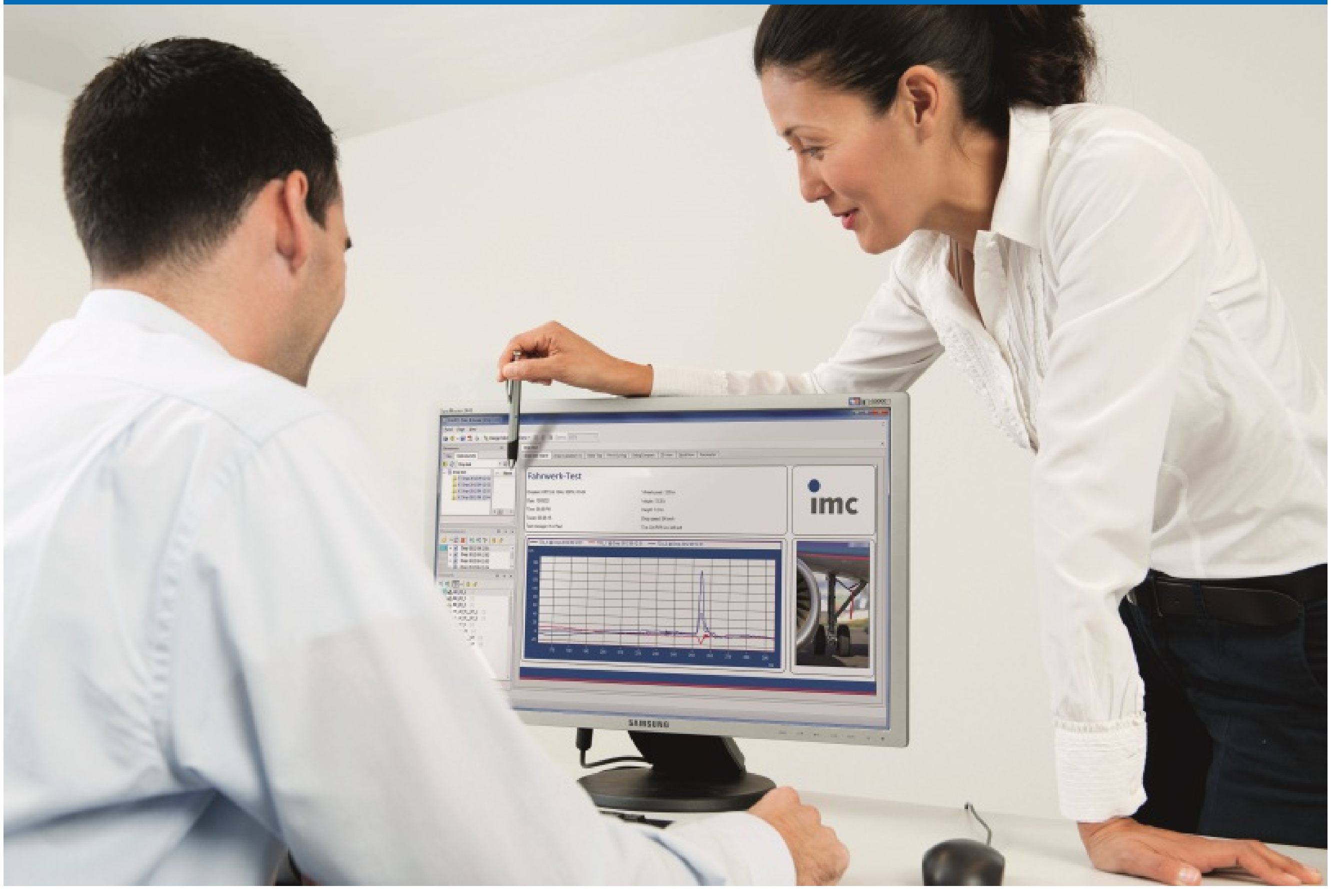
An Axiometrix Solutions Brand

imc FAMOS

Function Reference

Version 2024



© 2024 imc Test & Measurement GmbH

imc Test & Measurement GmbH • Voltastraße 5 • 13355 Berlin • Germany

Overview

This chapter provides a detailed reference of all functions, operators, constants and sequence commands available in imc FAMOS.

-

Point-by-point subtraction

Declaration:

```
Minuend - Subtrahend -> Difference
```

Parameter:

Minuend	First parameter, Minuend
Subtrahend	Second parameter, Subtrahend
Difference	Difference, results of point-by-point subtraction.

Description

The difference of two variables is calculated. Data sets are subtracted value for value, independent of their x-axis scaling. When a single value is subtracted from a data set, it is subtracted from each point of the data set.

Use the function Sub() for time-correct or x-correct subtraction.

Remarks

- Under certain circumstances, structured channels (segments/events) can also be subjected to this operator. For this purpose, either one of the two parameters must be a single value, or both parameters must have exactly the same structure (i.e. same total and segment length, same event count and event length).
- For meaningful division, the channels involved must have the same x-scaling. If they are not the same, a warning message is generated and the information from the first complex variable is used.
- Not all combinations of complex data types can be used for subtraction. To subtract a decibel-phase (DP) data set, it first must be converted to a different type; this is most convenient with the idB function. To subtract a complex and a real channel, the real data must first be converted to complex data, easily done using the Compl function.
- Complex data sets are subtracted value for value, according to the conventions of complex calculation. Note that when complex numbers are expressed in polar coordinates are subtracted, the magnitudes and phases will not be subtracted separately.
- If the specified data sets have different lengths, the length of the result corresponds to that of the shorter data set.

Examples:

Offset correction, subtraction of a fixed number:

```
NDcorr = NDdata - offset
XYdata.Y = XYData.Y- offset
```

Difference between two spectra:

```
MPdiff = MPspec1 - MPspec2
```

Difference between two spectra, where one is expressed in dB:

```
MPdiff = MPspec1 - idB(DPspec2)
```

Two methods of subtracting a real data set from a complex data set RiSpec having the components RiSpec.R and RiSpec.I:

```
RIdiff = Compl(RIspec.R - NDreal, RIspec.I)
RIdiff = RIspec - NDreal
```

Two methods of subtracting a single value from a complex data set RiSpec having the components RiSpec.R and RiSpec.I:

```
RIdiff = Compl(RIspec.R - offset, RIspec.I)
RIdiff = RIspec - offset
```

See also:

+(Addition), Sub, Append

>

Comparison operator, "greater than"

Declaration:

```
Operand1 > Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

Comparison of two numbers. The result is 1 if the first operand is greater than the second operand. Otherwise, the result is 0.

The operator can be applied to single values or to data sets. Comparison of data sets is performed data point by data point.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Examples:

A data set's maximum value is found and checked against a value limit.

```
Channel1 = ...
Maximum = max(Channel1)
IF Maximum > 21 AND Maximum <= 34
   BoxMessage("Attention", "Maximum near limit!", "!1")
ELSE
   IF Maximum > 34
      BoxMessage("Attention", "Limit is exceeded", "!1")
   END
END
```

In a data set, all values beyond a certain threshold are set to 0.

```
Channel1 = ...
Result = Channel1 * (Channel1 > 20)
```

In a data set, all values below 15 are set to the fixed value 10 and all values above 15 to the fixed value 20.

```
Channel1 = ...
Result = (Channel1 <= 15) * 10  + (Channel1 > 15) * 20
```

See also:

<, >=, UpperValue

>=

Comparison operator, "greater or equal"

Declaration:

```
Operand1 >= Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

Comparison of two numbers. The result is 1 if the first operand is greater than or equal to the second. Otherwise the result is 0.

The operator can be applied to single values or to data sets. Comparison of data sets is performed data point by data point.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Examples:

A data set's maximum value is found and checked against a value limit.

```
Channel1 = ...
Maximum = max(Channel1)
IF Maximum > 21 AND Maximum < 34
   BoxMessage("Attention", "Maximum near limit!", "!1")
ELSE
   IF Maximum >= 34
      BoxMessage("Attention", "Limit is exceeded", "!1")
   END
END
```

In a data set, all values beyond a certain threshold are set to 0.

```
Channel1 = ...
Result = Channel1 * (Channel1 >= 20)
```

In a data set, all values below 15 are set to the fixed value 10 and all values above 15 to the fixed value 20.

```
Channel1 = ...
Result = (Channel1 < 15) * 10  + (Channel1 >= 15) * 20
```

See also:

<=, >, UpperValue

<

Comparison operator, "less than"

Declaration:

`Operand1 < Operand2 -> ZeroOrOne`

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

Comparison of two numbers. The result is 1 if the first operand is less than the second operand. Otherwise, the result is 0.

The operator can be applied to single values or to data sets. Comparison of data sets is performed data point by data point.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Examples:

A data set's maximum value is found and checked against a value limit.

```
Channel1 = ...
Maximum = max(Channel1)
IF Maximum >= 21 AND Maximum < 34
   BoxMessage("Attention", "Maximum near limit!", "!1")
ELSE
   IF Maximum >= 34
      BoxMessage("Attention", "Limit is exceeded", "!1")
   END
END
```

In a data set, all values below a certain threshold are set to 0.

```
Channel1 = ...
Result = Channel1 * (Channel1 < 20)
```

In a data set, all values below 15 are set to the fixed value 10 and all values above 15 to the fixed value 20.

```
Channel1 = ...
Result = (Channel1 < 15) * 10  + (Channel1 >= 15) * 20
```

See also:

<=, >, LowerValue

<>

Comparison operator, "unequal"

Declaration:

```
Operand1 <> Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value/data set/text/text array to compare
Operand2	Second single value/data set/text/text array to compare
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

Test for inequality. The result is 1 if the two operands aren't equal. Otherwise, the result is 0.

This operator can be applied to single values, data sets, texts and text arrays.

With data sets, the comparison is performed sample point by sample point.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Two text arrays are considered identical if they share the same dimensionand the respective texts having the same index are the same.

Comparison of texts is not case-sensitive.

Examples:

A check is made of whether a data set contains an even number of samples.

```
Channel1 = ...
len = Leng?(Channel1)
IF mod(len, 2) <> 0
   BoxMessage("Error", "Invalid data set length", "!1")
END
```

Two digital data sets are to be compared. The result data set is 1 everywhere that the operand data sets have different values.

```
Result = (DigChannel1 <> DigChannel2)
```

The data set's units are checked for validity.

```
unit = UNIT?(Pressure_1, 1)
IF (unit <> "Bar" AND unit <> "Pa")
   BoxMessage("Attention", "Invalid unit (pressure)", "!1")
END
```

See also:

=, <=, >, TComp

<=

Comparison operator, "less than or equal to"

Declaration:

```
Operand1 <= Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

Comparison of two numbers. The result is 1 if the first operand is less than or equal to the second operand. Otherwise, the result is 0.

The operator can be applied to single values or to data sets. Comparison of data sets is performed data point by data point.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Examples:

A data set's maximum value is found and checked against a value limit.

```
Channel1 = ...
Maximum = max(Channel1)
IF Maximum > 21 AND Maximum <= 34
   BoxMessage("Attention", "Maximum near limit!", "!1")
ELSE
   IF Maximum > 34
      BoxMessage("Attention", "Limit is exceeded", "!1")
   END
END
```

In a data set, all values below a certain threshold are set to 0.

```
Channel1 = ...
Result = Channel1 * (Channel1 <= 20)
```

In a data set, all values below 15 are set to the fixed value 10 and all values above 15 to the fixed value 20.

```
Channel1 = ...
Result = (Channel1 <= 15) * 10  + (Channel1 > 15) * 20
```

See also:

<, >=, LowerValue

*

Point-by-point multiplication

Declaration:

```
Factor1 * Factor2 -> Product
```

Parameter:

Factor1	First parameter, Factor
Factor2	Second parameter, Factor
Product	Product; result of the point-by-point multiplication

Description

The product of two variables is calculated. Data sets are multiplied value for value, independent of their x-axis scaling.

Use the function Mult() for time-correct or x-correct multiplication.

With multiplication of single values and data sets , the same single value is multiplied to each of the data set's values.

Complex data types can be used interchangeably for multiplication. The result is always the same, regardless in which type the parameters are specified: as soon as at least one complex data set is involved in a multiplication, all units are checked for dB and treated accordingly. Multiplication of dB data is performed by adding the dB values.

Remarks

- Under certain circumstances, structured channels (segments/events) can also be subjected to this operator. For this purpose, either one of the two parameters must be a single value, or both parameters must have exactly the same structure (i.e. same total and segment length, same event count and event length).
- For meaningful division, the channels involved must have the same x-scaling. If they are not the same, a warning message is generated and the information from the first complex variable is used.
- Complex data sets are multiplied value for value, according to the conventions of complex calculation. Note that when complex numbers expressed in cartesian representation are multiplied, the real and imaginary parts will not be multiplied separately. Note also that in polar coordinate display, the phases and dB-numbers are added.
- If the specified data sets have different lengths, the length of the result corresponds to that of the shorter data set.

Examples:

A channel is normalized with a reference value:

```
NWnormalized = NWdata * SvReference
XYdata.Y = XYdata.Y * SvReference
```

The spectrum of a system's output variable is calculated by multiplying the transfer function by the spectrum of the input variable.

```
MPspec2 = MPspec1 * RItransfer
```

The spectrum of a system's output variable is calculated by multiplying the transfer function (in dB) by the spectrum of the input variable.

```
MPspec2 = MPspec1 * DPtransfer
```

Normalizes a spectrum using a reference value. If the unit of the reference value is dB, it is interpreted as a value in decibels and calculated accordingly.

```
MPtransfer = MPspec * SvReference
```

Multiplication of two real number data sets, with one expressed in dB. Since no complex data set is involved, the dB are not recognized automatically. Therefore, the inverse dB function is performed on the data set in dB.

```
NWdata = idB(NWdb) * NWnodb
; or alternatively:
NDdatdb = NDdb + dB(NDnodb)
```

See also:

/(Division), Mult, MatrixMult

/

Point-by-point division

Declaration:

```
Dividend / Divisor -> Quotient
```

Parameter:

Dividend	First parameter, dividend
Divisor	Second parameter, divisor
Quotient	Quotient, result of point-by-point division

Description

The quotient of two variables is computed. Data sets are divided point-by-point independently of their x-axes' scales.

Use the function Div() for time-correct or x-correct division.

With division of data set values by a single value, the same single value is the divider for each of the data set's values.

Complex data types can be used interchangeably for division. The result is always the same, regardless in which type the parameters are specified: as soon as at least one complex data set is involved in a division, all units are checked for dB and treated accordingly. Division of dB data is performed by subtracting the dB values.

Remarks

- Under certain circumstances, structured channels (segments/events) can also be subjected to this operator. For this purpose, either one of the two parameters must be a single value, or both parameters must have exactly the same structure (i.e. same total and segment length, same event count and event length).
- For meaningful division, the channels involved must have the same x-scaling. If they are not the same, a warning message is generated and the information from the first complex variable is used.
- Complex data sets are divided value for value, according to the conventions for complex calculation. Note that when complex numbers in rectangular coordinates are divided, the real and imaginary parts are not divided separately. Also note that for displays in polar coordinates, phase and dB numbers will be subtracted.
- If the specified data sets have different lengths, the length of the result corresponds to that of the shorter data set.

Examples:

A channel is normalized with a reference value:

```
DataNormalized = Data / SvReference
XYdata.Y = XYdata.Y / SvReference
```

Transfer function as the quotient of two spectra:

```
MPTransfer = MPSpectrum1 / RISpectrum2
```

Transfer function (in dB) as quotient of two spectra expressed in dB:

```
DPTransfer = DPSpectrum1 / DPSpectrum2
```

Normalizes a spectrum using a reference value. If the unit of the reference value is dB, it is interpreted as a value in decibels and calculated accordingly.

```
MPTransfer = MPSpectrum / SvReference
```

Division of two real data sets, where one is expressed in dB. Since a complex data set is not involved, the dB are not automatically recognized. Therefore, the data set present in dB should be converted using the inverse dB function.

```
NDdat = idB(NDdb) / NDnodb
; or alternatively:
NDdatdb = NDdb - dB(NDnodb)
```

See also:

*(Multiplikation), Div

^

Power operator (exponentiation)

Declaration:

```
Base ^ Exponent -> Result
```

Parameter:

Base	First parameter, base
Exponent	Second parameter, exponent
Result	Power (base raised to the exponent)

Description

The power function calculates the base to the power of the exponent. The power function processes data sets point-by-point regardless of the x-scaling.

When a single value is specified with a data set, the single value is applied to every value of the data set.

Remarks

- If the base value is positive, the exponent can assume any value.
- If the base value is negative, the exponent may only be an integer.
- If the base value is zero, the exponent must be positive.
- When possible, the unit of the result is determined from the unit of the base value and the value of the exponent. This is possible only when the exponent is a single value. Otherwise, the unit of the base value remains unchanged.
- The ^ character has a higher priority in sequences than basic arithmetic operators. For example, 4 * 3 ^ 2 is interpreted as 4 * (3 ^ 2), yielding a result of 36.
- Under certain circumstances, structured channels (segments/events) can also be subjected to this operator. For this purpose, either one of the two parameters must be a single value, or both parameters must have exactly the same structure (i.e. same total and segment length, same event count and event length).
- If the specified data sets have different lengths, the length of the result corresponds to that of the shorter data set.

Examples:

SVpower is assigned the value 8 V3.

```
SVpower = 2 'V' ^ 3
```

The reciprocal of the cubic root of the data set is calculated. If the data set unit is V^3, the result's unit will be 1/V.

```
NDpower = NDdata ^ (-1/3)
```

See also:

*(Multiplikation), sqr, sqrt, exp

+

Point-by-point addition

Declaration:

```
Summand1 + Summand2 -> Sum
```

Parameter:

Summand1	First summand
Summand2	Second summand
Sum	Sum; result of the point-by-point addition.

Description

The sum of two variables is calculated. Data sets are added value for value, regardless of their x-axis scaling.

A single value is added to a data set by adding the value to each point of the data set.

Use the function Add for time correct or x-correct addition.

This operator can also be used to append texts to each other (same as TAdd).

Remarks

- Under certain circumstances, structured channels (segments/events) can also be subjected to this operator. For this purpose, either one of the two parameters must be a single value, or both parameters must have exactly the same structure (i.e. same total and segment length, same event count and event length).
- When working with XY-data sets, the desired components must always be specified. You can add a single value directly to an XY-data set; each Y-value of the data set is then increased by the single value.
- For meaningful division, the channels involved must have the same x-scaling. If they are not the same, a warning message is generated and the information from the first complex variable is used.
- Not all combinations of complex data types can be added. To add a Dp data set (complex data set with polar coordinates, magnitude in dB), it must first be changed to a different type, for example using the idB function.
- Complex data sets are added value for value, according to the rules of complex arithmetic. Note that when complex numbers represented with polar coordinates are added, the magnitudes and phases are not added individually.
- If the specified data sets have different lengths, the length of the result corresponds to that of the shorter data set.

Examples:

Offset correction, addition of a fixed number:

```
NWcorrected = NWdata + SvReference
```

The amplitude of an XY-data set is corrected:

```
XYcorr.Y= XYcorr.Y+2
```

Superposition of two spectra:

```
MPsuperpos = MPspec1 + MPspec2
```

Superposition of two spectra, where one is given in dB:

```
MPsuperpos = MPspec1 + idB (DPspec2)
```

Two ways of adding a real number data set to a complex data set RiSpec with the components RiSpec.R and RiSpec.I.

```
RIsum = Compl(RIspec.R + NWreal, RIspec.I)
RIsum = RIspec + NWreal
```

Two ways of adding a single value to a complex data set RiSpec with the components RiSpec.R and RiSpec.I:

```
RIsum = Compl (RIspec.R + SvReference, RIspec.I)
RIsum = RIspec + SvReference
```

See also:

-(Subtraktion), Add, MatrixAdd, Append

=

Comparison operator, "equal"

Declaration:

```
Operand1 = Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value/data set/text/text array to compare
Operand2	Second single value/data set/text/text array to compare
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

Test for equality. The result is 1 if both operands are equal. Otherwise, the result is 0.

This operator can be applied to single values, data sets, texts and text arrays.

With data sets, the comparison is performed sample point by sample point.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Two text arrays are considered identical if they share the same dimensionand the respective texts having the same index are the same.

Comparison of texts is not case-sensitive.

Examples:

A check is made of whether a data set contains an even number of samples.

```
Channel1 = ...
Len = leng?(Channel1)
IF mod(len, 2) = 1
   BoxMessage("Error", "Invalid data set length", "!1")
END
```

Two digital data sets are to be compared. The result data set is 1 everywhere that the operand data sets have the same value.

```
Result = (DigChannel1 = DigChannel2)
```

Any of a data set's units stated in "Bar" are converted to "Pascal".

```
IF Unit?(Pressure_1, 1) = "Bar"
   Pressure_1 = Pressure_1 * 1e5
   SetUnit(Pressure_1, "Pa", 1)
END
```

See also:

<>, <=, >, TComp

ABCRating

A, B or C frequency rating in accordance with DIN EN 61672-1 (DIN IEC 651)

Declaration:

```
ABCRating ( Signal, SvType, SvTimeRating, SvReduction, Zero ) -> Rated
```

Parameter:

Signal	Signal to be rated [NW]
SvType	Type of rating
	1 : A-Rating
	2 : B-Rating
	3 : C-Rating
SvTimeRating	Time weighting
	0 : No time weighting
	>=0 : Time constant for averaging
	-1 : FAST weighting
	-2 : SLOW weighting
	-3 : IMPULSE weighting
	-4 : PEAK weighting
SvReduction	Reduction/Resampling
	0 : Equal-weighted RMS along the entire frequency-rated signal. TimeWeighting is ignored.
	1 : No resampling
	>1 : Factor for resampling
Zero	Reserved parameter. Always 0.
Rated	
Rated	Frequency rated signal

Description:

This function provides you with the standardized A, B and C frequency rating curves in accordance with DIN IEC 651 (sound level measurements).

Time weighting (moving RMS with exponential mean) and reduction (postsampling) are also available.

The A-weighting conforms to IEC 61672-1, 1st edition, 2002-05, Class1 and DIN IEC 651, 1981, class 0.

The cut-off frequencies of the rating curve band passes are as follows:

Rating	Lower cut-off	Upper cut-off
A	500 Hz	11 kHz
B	160 Hz	8 kHz
C	31.5 Hz	8 kHz

For A rating, the sampling frequency of the signal must be higher than 3.34kHz, for B and C rating higher than the lower threshold frequency of the filter. Ideally, the sampling frequency would be substantially higher than the upper threshold frequency.

The standardized weighting times are as follows:

FAST	Time constant = 0.125 s.
SLOW	Time constant = 1 s.
Impulse	For increasing amplitudes the time constant is 35 ms, for decreasing amplitudes 1.5 s. Thus impulse-shaped signals are captured quickly, the response decays slowly.
Peak	Extreme response for very short impulses; ensuring capture of the peak value. Time constant is zero during increasing amplitude (can be performed exactly by computer, by analog operation only in approximation); during decreasing amplitude 3 s.

A table of the frequency rating curves for the individual ratings is to be found in the description of the third-octave analysis - OctA().

Examples:

```
SignalA = ABCRating(Signal, 1, 0.2, 2, 0)
```

The signal is subjected to evaluation for an A-rating. The signal is time weighted with a time constant of 0.2s and resampled with the factor 2.

```
SignalCRms = ABCRating(Signal, 3, 0, 0, 0)
```

The equal-weighted RMS of the C-rated signal is determined.

See also:

OctA, ExpoRMS, RMS

Abs

Absolute magnitude

Declaration:

```
Abs ( Parameter ) -> Result
```

Parameter:

Parameter	Parameter. Allowed types: [ND],[XY].
Result	
Result	Absolute magnitude of the parameter

Description:

This operator determines the absolute value of real numbers. Positive numbers remain unchanged; the sign of negative numbers is inverted. This is equivalent in operation to an ideal rectifier.

Remarks

- The x-coordinate(s) of the parameter and the result are the same.
- The unit remains unchanged.
- The parameter may be structured (events/segments).

Examples:

Rectification of a sine wave:

```
NDhalfWave = Abs(NDsinus)
```

See also:

Round, Mod

ACF

Autocorrelation function

Declaration:

```
ACF ( InputData ) -> Result
```

Parameter:

InputData	Data set to be correlated with itself
Result	
Result	Result of the autocorrelation

Description:

The autocorrelation function is the special case of a cross-correlation function of two identical data sets. Autocorrelation means that a data set is correclated with itself.

In the implementation presented, the data set passed as the parameter is imagined to be extended periodically in both directions. if the data set (the signal) represents a single impulse, then the signal is interpreted as if it were many impulses strung together in succession, all the same length and same shape. This signal is compared to a copy of itself shifted once by an amount in the x-direction.

For any such shift, the autocorrelation function indicates how similar the signals are to each other. A return value of 1 means that both are identical. This is the case for a shift of 0, or also by one period length. A value of -1 means that both signals are oppositely equal; if one signal is positive, the other is just as large but negative.

A value of 0 indicates that the signal is not at all correlated with its shifted copy. A values between -1 and +1 may occur.

The data set generated is calculated over one period. The autocorrelation function is itself periodic, so that it is not necessary to calculate it over a larger range. The data set generated is axisymmetric around its center (in x-direction). The reason is that when comparing two identical functions, it does not matter whether a function is shifted by 0.9 periods in one direction or by 0.1 periods in the other, or even by 0.1 periods in the same direction. One half of the data set generated thus contains redundant data which you can discard.

To achieve acceptable calculation speed, the autocorrelation function is calculated with the help of the functions FFT(), iFFT(). Besides the immense savings on calculation time, this has two important consequences especially for long data sets:

For pre-processing, you can use the window function which you have set for the FFT. If you don't wish to use windowing, select the recatangle window; see the section on FFT. For another thing, the length of the processable data sets is limited to exponents of two up to 2^27. If a data set's length is different, it is truncated (even before windowing) to the next lower exponent of two. But if no values may be truncated, first use the function Red2(); see the examples.

- The ACF implemented here always yields normalized values, thus, no units are required. The values are normalized using the sum of the squares of all data set values. This means that the value of the non-normalized auto-correlation function is normalized at the position zero. This value is the square of the true RMS.
- When the length of the data set exceeds 2^27, an error message is generated. The ACF cannot be computed. Shorten the data set using either of the imc FAMOS functions Leng or Red2. data sets with a maximum length of 134.217.728 can be processed.
- If the length of the data set specified as a parameter is not a power of two, a warning message is generated. The data set is then automatically truncated..
- Because the ACF function uses the FFT function internally, temporary memory is required in the working memory. If insufficient memory is available, an error message is generated and calculation is canceled.
- If the data set to be processed has too large a mean value (y-offset), it is recommended to remove the mean value from the data set. Otherwise, the mean value influences the correlation result stronger than the actual signal. This means the auto-correlation function is not calculated, rather the auto-covariance function.

Examples:

```
acf_result = ACF(data)
```

Simplest application, periodic ACF; the data set may be truncated down to a power of two in length.

```
acf_result = ACF(Red2(data - Mean(data)))
```

Periodic ACF, with the whole data set affected through appropriate resampling. The mean value of the data set is subtracted since quite a large y-offset is present.

```
help = Red2(data)
help = Leng(help, 2 * Leng?(help))
acf_result = ACF(Red2(help))
acf_result = Leng(acf_result, 0.5 * Leng?(acf_result))
```

This is an example of non-periodic ACF: the signal is extended with zeros ("appending zero" in the literature), simulating a nonrepetitive signal, even though ACF calculation is periodic. After calculation, the second half of the result is discarded. Calling Red2 function twice ensures that each power of two of valid data is padded with an equal number of zeros. The rectangular window function should always be set for this sequence (see FFT).

See also:

CCF, CorrCoeff, FFT, Red2

acos

Arcsine; inversion of sin

Declaration:

```
acos ( Data ) -> Angle
```

Parameter:

Data	Data; allowed types: [ND],[XY].
Angle	
Angle	Arcsine of the parameter (angle in radians)

Description:

The arcsine function is computed.

This function returns an angle in radians without units.

The x-coordinate(s) of the results and the parameter are the same.

Remarks

- The argument should not have any unit; if it does, a warning is issued and the unit is adopted unchanged.
- The permittedd value range of the parameter is between -1 and +1.
- The parameter may be structured (events/segments).

Examples:

acos(0) = PI/2, acos(1) = 0, acos(-1) = PI

```
pihalf = acos(0.0)
```

A serrated triangular data set is generated from a sinusoidal data set:

```
NDspikes = acos(NDcos)
```

With the pre-defined constant PI or INDEGR, the angle returned is converted from radians to degrees:

```
value_90 = acos(0.0) * 180 '°' / PI
value_90 = acos(0.0) * INDEGR
```

See also:

cos, asin, atan2

Add

Time- or x-correct addition

Declaration:

```
Add ( Summand1, Summand2, SvOption ) -> Sum
```

Parameter:

Summand1	First summand. Allowed types: [ND],[XY].
Summand2	Second summand. Allowed types: [ND],[XY].
SvOption	Option
	0 : The trigger time of the two summands is ignored.
	1 : Time-correct superposition with regard to trigger-time
Sum	
Sum	Sum; result of the addition [XY]

Description:

Two channels undergo time-correct or x-correct addition, meaning that the y-values for each common xvalue of time are added.

The result is defined only within the x-range which is shared by both data sets. Within this range a resultvalue is determined for every point at which at least one of the data sets possesses a value. If no value exists for the other data set, one is determined by linear interpolation.

The x-tracks of both parameter data sets must be monotonous, i.e. the x-coordinates must increase continuously.

The addition operator, by contrast, performs point-by-point addition of the values of both data sets.

Examples:

Two channels are measured; one between 11:00 and 13:00, and the other between. 12:00 and 14:00.

```
superpos = Add(voltage11_13h, voltage12_14h, 1)
```

A time-correct superposition of the two data sets with respect to the trigger time is performed. The result is defined for the time between 12:00 and 13:00 hours.

See also:

+(Addition), Sub, Mult, Div, Append

All0

Returns the x-positions of all of a data set's zero-crossings

Declaration:

```
All0 ( Data ) -> XZeroes
```

Parameter:

Data	Data set examined. Allowed types: [ND],[XY].
XZeroes	
XZeroes	X-coordinates of all zero crossings located in the data set NDData

Description:

This function returns a data set with the x-values of all of the zero-crossings, i.e. points in the argument data set where the y-value is zero. The All0 function does not interpolate between x-values - if a zero lies between two samples, the larger of the x-coordinates is returned.

If NyData is an XY-data set, it must have a monotonous time or x-track.

- If the first value of a data set is zero it won't be interpreted as a zero-crossing.
- If no zeros are found, an empty data set is returned.
- To search for interpolating zeros in XY-data sets, you can use the function SearchLevel.

Examples:

```
xnull = All0(data)
```

The result is the x-coordinates of all zeros in the data set.

See also:

SearchLevel, Top, xMax, PolynomRoots, PosiEx

AmpSpectrumPeak

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Magnitude spectrum (harmonics determined as peak values or magnitudes) with a moving window and linear averaging. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
AmpSpectrumPeak ( InputData, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [, Base2] ) ->
Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
AveragingType	method of summarizing all spectra
	0 : no averaging
	1 : averaging (arithmetic mean or linear averaging of the magnitude spectra). The number of spectra over which the average is taken is determined by the parameter 'Reduction'.
	2 : Peak Hold Max, from beginning. Maximum values, based on the spectra calculated thus far in the algorithm.
	3 : Peak Hold Max, interval. Maximum values, based on the number of spectra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the spectra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of spectra which the parameter 'Reduction' dictates.
	6 : Root mean square (RMS) taken over all magnitude spectra computed
	7 : Root mean square (RMS) taken over all magnitude spectra computed. The result is divided by sqrt(ENBW=Equivalent noise bandwidth) according to the window type used. E.g. division by sqrt(1.5) in the case of a Hanning window.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

Examples:

```
Spectra = AmpSpectrumPeak ( Channel, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%.

```
Spectra = AmpSpectrumPeak ( Channel, 2048, 1, 0, 10, 6, 0 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

AmpSpectrumPeak_exp, AmpSpectrumPeak_1, AmpSpectrumRMS

AmpSpectrumPeak_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

An averaged magnitude spectrum is computed (harmonics determined as peak values or magnitudes). The averaging is taken of as many spectra as there are windows within the waveform.

Declaration:

```
AmpSpectrumPeak_1 ( InputData, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all spectra
	1 : averaging (arithmetic mean or linear averaging of the magnitude spectra). The mean is taken over all spectra computed.
	2 : Peak Hold Max, maximum values, based on the spectra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the spectra calculated thus far in the algorithm
	6 : Root mean square (RMS) taken over all magnitude spectra computed
	7 : Root mean square (RMS) taken over all magnitude spectra computed. The result is divided by sqrt(ENBW=Equivalent noise bandwidth) according to the window type used. E.g. division by sqrt(1.5) in the case of a Hanning window.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged magnitude spectrum

Description:

Examples:

```
Spectrum = AmpSpectrumPeak_1 ( Channel, 1000, 0, 50, 6, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values.

See also:

AmpSpectrumPeak_exp, AmpSpectrumPeak, AmpSpectrumRMS_1

AmpSpectrumPeak_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Magnitude spectrum (harmonics determined as peak values or magnitudes) with a moving window and exponential averaging. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
AmpSpectrumPeak_exp ( InputData, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [, Base2] ) ->
Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

Examples:

```
Spectra = AmpSpectrumPeak_exp ( Channel, 1000, 0, 50, 2, 40.0, 0 )
```

The channel has a sampling time of 10ms. Therefore, a 1000 point-spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

AmpSpectrumPeak_1, AmpSpectrumPeak, AmpSpectrumRMS_exp

AmpSpectrumRMS

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Magnitude spectrum (harmonics determined as RMS (root-mean-square) values) with a moving window and linear averaging. Computed using FFT. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
AmpSpectrumRMS ( InputData, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
AveragingType	method of summarizing all spectra
	0 : no averaging
	1 : averaging (arithmetic mean or linear averaging of the magnitude spectra). The number of spectra over which the average is taken is determined by the parameter 'Reduction'.
	2 : Peak Hold Max, from beginning. Maximum values, based on the spectra calculated thus far in the algorithm.
	3 : Peak Hold Max, interval. Maximum values, based on the number of spectra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the spectra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of spectra which the parameter 'Reduction' dictates.
	6 : Root mean square (RMS) taken over all magnitude spectra computed
	7 : Root mean square (RMS) taken over all magnitude spectra computed. The result is divided by sqrt(ENBW=Equivalent noise bandwidth) according to the window type used. E.g. division by sqrt(1.5) in the case of a Hanning window.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

Examples:

```
Spectra = AmpSpectrumRMS ( Channel, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%.

```
Spectra = AmpSpectrumRMS ( Channel, 2048, 1, 0, 10, 6, 2 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

AmpSpectrumRMS_exp, AmpSpectrumRMS_1, AmpSpectrumPeak

AmpSpectrumRMS_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

An averaged magnitude spectrum is returned (harmonics determined as RMS (root-mean-square) values). The averaging is taken of as many spectra as there are windows within the waveform. Calculated by means of FFT.

Declaration:

```
AmpSpectrumRMS_1 ( InputData, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all spectra
	1 : averaging (arithmetic mean or linear averaging of the magnitude spectra). The mean is taken over all spectra computed.
	2 : Peak Hold Max, maximum values, based on the spectra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the spectra calculated thus far in the algorithm
	6 : Root mean square (RMS) taken over all magnitude spectra computed
	7 : Root mean square (RMS) taken over all magnitude spectra computed. The result is divided by sqrt(ENBW=Equivalent noise bandwidth) according to the window type used. E.g. division by sqrt(1.5) in the case of a Hanning window.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged magnitude spectrum

Description:

Coefficient of the window functions: Hamming: 1, -0.46/0.54; Hanning: 1, -1; Blackman: 1, -0.50/0.42, 0.08/0.42; Blackman-Harris: 1, -0.48829/0.35875, 0.14128/0.35875, -0.01168/0.35875; Flat top: 1, -1.93, 1.29, -0.388, 0.0322

Examples:

```
Spectrum = AmpSpectrumRMS_1 ( Channel, 1000, 0, 50, 6, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values.

See also:

AmpSpectrumRMS, AmpSpectrumRMS_exp, AmpSpectrumPeak_1

AmpSpectrumRMS_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Magnitude spectrum (harmonics determined as RMS (root-mean-square) values) with a moving window and exponential averaging. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
AmpSpectrumRMS_exp ( InputData, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [, Base2] ) ->
Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

Examples:

```
Spectra = AmpSpectrumRMS_exp ( Channel, 1000, 0, 50, 2, 40.0, 0 )
```

The channel has a sampling time of 10ms. Therefore, a 1000 point-spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

AmpSpectrumRMS_1, AmpSpectrumRMS, AmpSpectrumPeak_exp

AND

Logical "AND"-operator

Declaration:

```
Operand1 AND Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

"AND"-operation on two numbers. The result is 1 if neither of the operands is 0. Otherwise the result is 0.

The operator can be applied to single values or data sets. With data sets, the operation is applied data point by data point..

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Examples:

A data set's maximum value is found and checked against a value limit.

```
Channel1 = ...
Maximum = max(Channel1)
IF Maximum > 21 AND Maximum < 34
   BoxMessage("Attention", "Maximum near limit!", "!1")
ELSE
   IF Maximum >= 34
      BoxMessage("Attention", "Limit is exceeded", "!1")
   END
END
```

Two data sets with same time base are examined. All times, at which both data sets are above a given limit, are calculated:

```
t = Top((Channel1 > 3) OR (Channel2 > 3), 0.9)
```

The operation is applied to two digital data sets. The result data set's value is 1 wherever both operand data sets have the value 1.

```
Result = (DigChannel1 AND DigChannel2)
```

See also:

NOT, OR, XOR

Append

Time- or x-correct merging of data set values

Declaration:

```
Append ( Data1, Data2, SvOption ) -> Merged
```

Parameter:

Data1	First data set; allowed types: [ND],[XY].
Data2	Second data set; allowed types: [ND],[XY].
SvOption	Option
	0 : The trigger time of the two summands is ignored.
	1 : Time-correct superposition with regard to trigger-time
Merged	
Merged	Resulting data set [XY]

Description:

This function enables time- or x-correct appending or merging of data sets.

The resulting data set contains points at all nodes which are defined in either [Data1] **or** [Data2]. If a node is defined in both data sets, the mean value of the corresponding y-values is used.

Both data sets must have a monotonic time/x-axis.

By contrast, the function Join() appends the data sets point-by-point.

Examples:

Two measurement channels having the same physical size are recorded between 11:00 and 13:00, and 12:00 and 14:00 respectively.

```
voltage11_14h = Append(voltage11_13h, voltage12_14h, 1)
```

The two data sets are merged. The result is defined for the time between 11:00 and 14:00 hours. Between 11:00 and 12:00 and between 13:00 and 14:00 the values of the first and of the second channels, respectively, are adopted. Between 12:00 and 13:00 the measurement values are ordered by time. If both data sets share a measurement time, the resulting y-value is the mean of the two values.

See also:

Join, JoinEx, AppendLoop, Add, Sub, Mult, Div

AppendLoop

Available in: Professional Edition and above

Appends additional data/values to a data set. The function is optimized for calling within a loop, in which small data volumes are repeatedly appended.

Declaration:

```
AppendLoop ( Dataset, Append )
```

Parameter:

Dataset	Data are appended to this data set. The function changes the variable passed.
Append	Data to be appended

Description:

All values of the parameter Append are appended to the data set.

The data set's scaling (units, x0, dx etc.) remains intact.

The data set's numerical format (e.g. digital or integer with value range, or real numbers) remains intact; but not if the data set is empty and non-empty data are appended. Then, the numerical format of the appended data is applied.

The data set can be equidistant, or an XY-data set, or complex. It may not have any events. The data to append are of the same type.

For segmented data, the following applies: If the data set contains segments, the parameter Append may also have segments of the same length. Alternatively, the parameter Append may not have any segments if its length matches the data set's segment length.

As the data set, only a complete variable can be specified: Portions (e.g. A.y, A[2]) or formulas (e.g. 2*A) are not permitted.

The function serves to increase the execution speed with increased memory allocated to the data set. The higher execution speed typically becomes noticeable in a loop where the function is repeatedly called for the same data set many times. If the data volume to be appended is small (e.g. less than 1000 samples), the execution speed is significantly higher.

Once appending to a data set has been completed, typically following a loop, the increased memory allocated to the data set may persist. With a call of AppendLoopEnd() the additional memory is removed. The call of AppendLoopEnd() remains without effect if no increased memory exists (any longer). The call of AppendLoopEnd() is omitted, if the increased memory demands don't matter for the work to be performed.

If copies of the data set or of its components are made, the copies also have increased memory requirements.

The function can provide time savings especially whenever the program has sufficient RAM available.

For TSA-data, TsaAppend() is used.

Examples:

A = empty

xdelta A 0.1

xunit A V

```
for i = 1 to 2000
   y=3*i
   AppendLoop(A, y)
end
AppendLoopEnd(A)
```

A = xyof (empty, empty)

```
for i = 1 to 2000
   x=i
   y=2*i
   AppendLoop(A, xyof(x, y))
end
AppendLoopEnd(A)
```

See also:

AppendLoopEnd, Join, Append, TsaAppend, EventAppend

AppendLoopEnd

Available in: Professional Edition and above

Once the function AppendLoop has been called (repeatedly) in a loop, the extra required memory is finally deallocated.

Declaration:

```
AppendLoopEnd ( Dataset )
```

Parameter:

Dataset	The data set to which new data was appended using AppendLoop

Description:

All constraints as with AppendLoop

Multiple calls or a call without previous AppendLoop do no harm.

Examples:

A = empty

xdelta A 0.1

xunit A V

```
for i = 1 to 2000
   y=3*i
   AppendLoop(A, y)
end
AppendLoopEnd(A)
```

See also:

AppendLoop

APPLICATION

Start Windows application or DOS-program
This command is obsolete, instead of it; the more powerful function Execute() should be used.

Declaration:

```
APPLICATION Mode Name Parameter
```

Parameter:

Mode	Window mode in which the program should launch
	NORMAL : Start program in normal size
	VOLLBILD : Start program in fullscreen mode (maximized)
	SYMBOL : Start program minimized (icon)
Name	Name (program path) of the program to start
Parameter	Command line parameter for the program to be started

Description

This command starts an application or a DOS program. You can supply a command-line parameter and specify the mode, in which to start the application in Windows.

Examples:

```
APPLICATION NORMAL Notepad.exe
APPLICATION MAXIMIZED c:\windows\Notepad.exe
```

Two methods of stating the Windows application "Notepad"

See also:

Execute

Appro

Approximation of a data set's values using the user's choice of functions

Declaration:

```
Appro ( Data, Fct1, Fct2, Fct3, Fct4, Fct5, Fct6, Fct7, Fct8 ) -> Coefficients
```

Parameter:

Data	Data set which is to be approximated; allowed data types: [ND]
Fct1	Data set for the 1st coefficient [ND]
Fct2	Data set for the 2nd coefficient [ND]
Fct3	Data set for the 3rd coefficient [ND]
Fct4	Data set for the 4th coefficient [ND]
Fct5	Data set for the 5th coefficient [ND]
Fct6	Data set for the 6th coefficient [ND]
Fct7	Data set for the 7th coefficient [ND]
Fct8	Data set for the 8th coefficient [ND]
Coefficients	
Coefficients	Coefficients of the approximating function

Description:

An approximation of the data set passed is generated using a function which is known except for its coefficients. The function supplied ca consist of up to eight partial functions, each having an unknown coefficient. The Appro function returns the coefficients by using the method of least squares.

The procedure for using the Appro function is as follows: for a measurement channel with the values g(x[i]) (i = 0, ..., data set length - 1), a function, known except for its coefficients, is to be approximated. This function consists of several partial functions, each of which has its own coefficient. The coefficients are determined by the function Appro. First, the data sets of the function values are determined for the known partial functions f0, ..., fm-1 for all x[i]. The Appro function determines the m coefficients c[0], ..., c[m-1] according to the method of least squares, until the condition:

$$g(x[i]) \approx k[0] \cdot f_0(x[i]) + k[1] \cdot f_1(x[i]) + \dots + k[m-1] \cdot f_{m-1}(x[i])$$

is met for all i.

The coefficients belonging to the partial functions are returned in the order in which the data sets [Fct] of the partial functions were specified.

If the function to be approximated consists of fewer than eight partial functions, the data sets of partial functions [Fct], whose coefficient should be determined, are entered and the remaining parameters of the function Appro must then be set to any single value (0 is recommended).

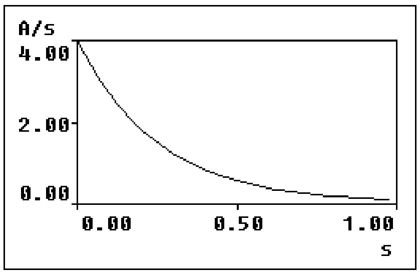
- If the data sets entered are not compatible (different lengths, units, x0 or deltaX), a warning message is generated. If the data sets have different lengths, all data sets are set to the length of the shortest data set before approximation. If differing units, x0 or deltaX are present, the calculations are performed using the values of the data set to be approximated, in this case NDData.
- Several data sets may be entered as measurement channels. The Appro function calculated the corresponding coefficients (see example).
- The Appro function is a generalized version of the function Poly().
- On the topic of precision, see the remarks in the section on the Poly function.

Examples:

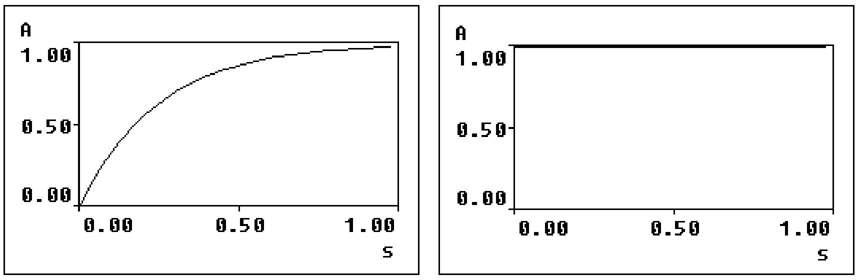
The measurement channel of the derivative of function y and the measurement channel of function y exist. The parameters c[0] and c[1] are to be determined according to the method of least squares, so that the condition

$$y'(x[i]) \approx k[0] \cdot y(x[i]) + k[1]$$

is fulfilled as closely as possible for all i (i = 0, ..., data set length - 1). The data set NwData in the following illustration is the measurement channel for y', and contains 50 values:



The data set NDData1 belonging to the first coefficient is the measurement channel for y, displayed at the left. The data set NDData2 belonging to the second coefficient consists only of values of one, displayed at the right. Both data sets contain 50 values:



```
NwCoeff = Appro(NwData, NwData1, NData2, 0, 0, 0, 0, 0, 0)
```

In imc FAMOS, the data set NDCoeff (coefficient) is output as the result. The data set NDCoeff contains the values: coeff0 = -4.0000 and coeff1 = 4.0000.

This is how to obtain the individual coefficients::

```
SvCoeff0 = NwCoeff[1]
SvCoeff1 = NwCoeff[2]
```

See also:

AppronNonLin, Poly, eFit, LFit, Value

AppроNonLin

Available in: Professional Edition and above

Approximation by a function which can be non-linear in its coefficients. A function described by a formula is miminized in the sensoe of least squares. The coefficients of the formula are determined in the process.

Declaration:

```
ApproNonLin ( Input, Formula [, TimeOut] [, Error handling] [, Limits] [, Weight] [, Variant] ) -> Coefficients
```

Parameter:

Input	Input
Formula	Freely defineable formula
	"A1+A2*exp(A3*x)-y" : Decay function
	"A1+A2*exp(A3*x)*sin(A4*x+A5)-y" : Damped oscillation
	"sqrt((X-A1)^2+(Y-A2)^2)-A3" : Circle
	"A1*exp(-((x-A2)^2/(2*A3^2)))-y" : Density of normal distribution
	"A1*exp(A2*x)-y" : Decay function without offset
	"A1+x*(A2+x*A3)-y" : 2nd-degree polynomial
	"A1+x*(A2+x*(A3+x*A4))-y" : 3rd-degree polynomial
TimeOut	TimeOut in seconds. If the algorithm is not finished within this time, the system cancels and posts a Timeout-error. 0 for no verification. (optional , Default value: 0)
Error handling	Determines the system response to an error. (optional , Default value: 0)
	0 : Cancel and post error message
	1 : Return empty data set
Limits	Limits for the coefficients. Conditions separated by commas, e.g. "A1 >= 0, A2 > 0, A2 < 1e3", which limit the value range of the coefficients. Initializations beginning with init are initial values for the iteration, e.g. "init A1 = 3". The parameter may also be empty. Initializations and limitations may be combined, but may not contradict each other. (optional , Default value: "")
Weight	Weight for different weighting of the individual measurement values (optional)
Variant	According to which variant of the algorithm is the calculation performed? (optional , Default value: 0)
	0 : Automatic. The best variant of the algorithm available in the current version is used. The algorithm may be improved in subsequent versions of FAMOS.
	1 : Like in FAMOS 7.2. In later versions of FAMOS the algorithm will also behave the same.
	2 : Like in FAMOS 7.3. In later versions of FAMOS the algorithm will also behave the same.
Coefficients	
Coefficients	Coefficients A1, A2, ...

Description:

The formula supplied is minimized in the sense of least squares: The square of the formula is minimized over all measurement values supplied, so as close to zero as possible.

For example, if the formula is "exp(A1*x)-y", then "Sum (exp(A1*x)-y)^2" is minimized.

In general, the difference between the original y-values and the approximated function is to be found. A term -y is then required in the formula.

If for instance a measurement is to be approximated by a function "y=exp(A1*x)", then this function may not itself be supplied as the formula. Instead, the deviations from the measurement values y which are to be minimized must be specified, thus "exp(A1*x)-y".

Minimization

The local minimum found does not need to match the global minimum.

This was illustrated by Dr. Knopp in an accessible way: Supposing you are in a hilly landscape. It is foggy and not possible to see very far. The ground slopes downwards and you follow the downward slope until it is no longer possible to go downward. That is the destination. But whether there is a deeper valley beyond the next hill remains unknown.

In principle the search is conducted according to the Gauss-Newton algorithm. The procedure works iteratively and improves the solution gradually from the initial values. However, the increment is governed in a similar way to Levenberg-Marquardt.

A minimum is then easily found if the coefficients are in front of terms which differ strongly in their character or form.

Coefficients

In the formula, the coefficients A1, A2, ... A16 are specified continually starting with A1. Only as many as are required.

Variables

In the formula, the variables representing the measured values are generally stated as x and y.

Input	Application
equidistant	This function can be applied to equidistant input data. In that case, x and y are used in the formula.
XY	The function can be applied to XY-input data. In that case, x and y are used in the formula.
Matrix	The function can be applied to segmented data. In that case, x , y and z are used in the formula. As usual with FAMOS, y is the amplitude, x and z are the matrix dimensions and are given explicitly by the initial value and increment. y=y(x,z).
List	This function can be applied to segmented data, where each segment is interpreted as a separate variable. In the formula, x, y1, y2, .. y16 are used, where x is the common x-coordinate for all segments. y1 are the values of the 1st segment, y2 those of the 2nd, etc.

Initial values

Initial values are supplied by means of the parameter Limits.

Beginning with the initial values, the function approaches a local minimum.

The user determines appropriate initial values before the function call.

For an unfortunate choice of initial values, the function will not converge and find any result.

In some cases it is not necessary to specify any initial values. In that case the system tries to find appropriate initial values. The initial values found automatically, and thus the reuslts, are not always good. The results need to be checked in each case.

When using sin() or cos() with equidistant data and arguments of the form "A1*x+A2", the function automatically finds the initial values by means of an internally performed FFT without observing weighting factors. However, this generally only achieves success when the frequency is constant and the amplitude dominant in the signal.

With formulas of the type "A1+A2*exp(A3*x)-y", it is well possible to dispense with specifying initial values.

If initial values are specified, they must lie within a range which makes sense. Any straying outside of the numerical range must be avoided. With "exp(A1*x)-y" and x in the range 10 and an initial value A1=100, the formula will produce an overflow and its derivative with respect to A1. A1=-100 produces a permanent underflow.

For coefficients which appear linearly in the fomula, it is never necessary to specify an initial value. Before the beginning of the iteration, the system will automatically find initial values for these coefficients anyway.

With trigonometric functions, there is typically a truly large number of local minima, which also are all very close to each other. As well, for a good selection of initial values, the procedure can easily jump to a nearby minimum.

Formula

Along with the basic operators, the formula may also contain the functions sin, cos, tan, exp, asin, acos, atan, ln, log, sqrt, abs.

All trigonometry functions work with radians. In case a phase in degrees is desired, e.g. "sin(A1*x+A2*PI/180)". PI and PI2 are defined.

Coefficients can be scaled in a suitable way, e.g. the frequency in a sin() function, "sin(2*PI*A1*x+A2)".

The procedure includes taking the derivatives of the formula with respect to the coefficients. The derivative must be continuous. Thus, the coefficients can certainly appear in a function such as exp(), but not in abs(). Functions such as abs() an be used at other locations in the formula such as abs(x).

Linear dependence must be avoided, for instance +(A1-A2). That would usually cause an error to occur during the iteration or a divergence.

Linearly applied coefficients

The formula is considered linear in a coefficient if the derivative with respect to that coefficient is independent of any linear coefficient.

Coefficients applied linearly are treated preferentially. Thus "A1*x+..." should be preferred over "x/A1+..."

Similarly, "A1*x+A2+..." is also to be preferred over "A1*(x+A2)+...". In the first expression, A1 and A2 are plugged in linearly; in the second expression only one of them.

Thus, "A2*(A1+x)^2..." is also to be preferred over "((A1+x)/A2)^2...". In the first expression, A2 is plugged in linearly.

Also in arguments for functions, the linearly plugged-in coefficient must be given absolute preference, e.g. "exp(A1*x)" and "sin(A1*x+A2)" instead of "exp(x/A2)" and "sin(x/A1+A2)".

input data

A sufficient number of independent input data must be provided. If the input data do not contain sufficient information, it causes the same difficulties as linear dependent coefficients.

Limits

For each coefficient used in the formula, an upper and a lower limit may be specified.

If no start value is specified for a coefficient, the function attempts to find an appropriate initial value within the range from the lower to the upper boundary.

In accordance with the formula it makes sense to form as narrow a range as possible in order to shorten the search for initial values.

If an initial value is specified for a coefficient, it will also be used.

The upper and lower boundaries are not checked during the iteration. Thus they are not true auxiliary conditions of the minimization procedure.

However, the lower and upper boundaries are used to check the result. If it is out of bounds, this is classified as an error.

The boundaries and initialization values can also be given by a calculation, as long as it leads to a constant, e.g. "A1 > 10-3; A1 < 10+3"

Interpretation of the coefficients determined

The result needs to be verified in any case.

If the coefficients in a formula can be interchanged without changing the content, then the coefficients may also appear interchanged in the result. E.g. in "exp(A1*x)+exp(A2*x)-y", the result may sometimes be [-3,-7], sometimes [-7,-3]. But such arrangements can be problematic anyway with regard to convergence.

With coefficients in trigonometry functions, unexpected results can also occur: E.g. with the formula "A1*tan(A2*x+A3)-y", A1 and A2 may both be negative instead of positive, A3 can become multiples of PI larger or smaller.

With coefficients within sin() and cos() such as for example in "A1*sin(A2*x+A3)-y", A1 and A2 are determined preferentially positive, A3 preferentially in the range -PI..+PI.

Results with a value of 1e35 or even -1e35 may possibly be limited to FAMOS value range. Actually their magnitude may even be higher.

Speed

To improve the speed of program execution when the data volume is large, it is possible to achieve an intermediate result by means of an appropriately selected small but representative portion of data. This is used as the inital value of the actual larger approximation task.

When there is a large number of coefficients and no initial values specified, the internal search for appropriate initial values is especially demanding.

The internal search for initial values demands calculation time especially for coefficients which are plugged in non-linearly. Thus in A1+A2*exp(A3*x)-y, A1 and A2 are used linearly, while A3 is non-linear and costs much search time.

Multiple processor cores may be utilized.

A data set containing 100000 samples is considered quite large.

Weight

If no weighting is specified, all measured values are weighted equally.

The data set Weight contains a weighting factor for each measured value. The value 1.0 is neutral. But values >1 and <1 including 0 are possible.

The data set Weight is equally long as the input data. It is never segmented. If the input data are segmented and contain a list, the data set Weight is equally long as one segment.

If the data set Weight is empty, no weight is specified.

General

If called with incorrect coefficients, syntax errors in the formula, or insufficient memory, the system always cancels operation and posts the usual error message.

If there are problems with the convergence, the error handling takes effect.

If the valid range is exceeded too often, e.g. exp(1000) or division by zero, there will be problems with the convergence.

With equidistant input data, an x-offset x0 which is large (in absolute value) can lead to numerical problems if the sampling interval dx is too small. For instance, if the attenuation constant in a decay function is to be determined, then inthe formula the calculation must treat x. However, the desired attenuation constant does not depend on the x-offset. In this case, the x-offset of the input data should first be set to zero.

This function works at the precision level for 64-bit real numbers, which limits its applicability when handling extreme numerical analysis.

When working with a timeout, it may not be set for too brief a time frame. The time needed for calculations depends on many factors such as the current number values, the processor type and the demands it faces.

Examples:

Equidistant input data without specified initial values, decay function

```
t = ramp(0,0.01, 10)
data = 4 + -3 * exp ( t * -20 ) ; test data
A = ApproNonLin(data,"A1 + A2 * exp ( x * A3 ) - y")
A1 = A[1]
A2 = A[2]
A3 = A[3]
```

Equidistant input data without specified initial values, damped osciallation, advantageous configuration of the coefficients

```
t = ramp(0,0.0001, 2000)
A1 = 4 ;offset [V]
A2 = 3 ; amplitude[V]
A3 = -1/0.05
A4 = 120 ; f[Hz]
A5 = 70 ; phase [degrees]
y = A1 + A2 * exp ( t * A3 ) * sin ( 2 * PI * A4 * t - A5 * PI/180 ) ; test data
A = ApproNonLin(y, "A1 + A2 * exp ( x * A3 ) * sin ( 2 * PI * A4 * x - A5 * PI/180 )  - y" )
Tau=-1/A[3] ; time constant
```

Equidistant input data with initial values, polynomial

```
t = ramp(0,0.001, 50000)
test = 4 * t*t + 10 ; test data
start="init A1 = 3, init A2 = 7"
```

```
A = ApproNonLin(test,"A1 * x*x + A2 - y",0,0,start)
```

2nd-degree polynomial from equidistant data

```
x = ramp ( 0.01, 0.001, 1000 )
y = 3 + 8 * x - 5 * x^2
A = ApproNonLin ( y, "A1 + A2 * x + A3 * x^2 - y" )
```

3rd-degree polynomial from XY values

```
x = 0.001 * random ( 1000, 2, 0, 0, 8 )
y = 3 + 8 * x - 5 * x^2 + 2 * x^3
input_xy = xyof ( X, Y )
A = ApproNonLin ( input_xy, "A1 + A2 * x + A3 * x^2 + A4 * x^3 - y" )
```

Fit of a 2-dimensional 2-degree polynomial: z = f(x,y)

z="A1+A2*x+A3*x^2 + A4*y+A5*x*y+A6*x^2*y + A7*y^2+A8*x*y^2+A9*x^2*y^2

```
Ai=[2,3,4, -2,-3,-4,1.5,1.7,-1.3]
data=leng(0,300)
setseglen(data, 100)
x=round(ramp(-4.9,1,200),10)/200
y=mod(ramp(0,1,200),10)/10
z=(Ai[1]+x*(Ai[2]+x*Ai[3])) + y*((Ai[4]+x*(Ai[5]+x*Ai[6])) + y*(Ai[7]+x*(Ai[8]+x*Ai[9])))
data[1]=x
data[2]=y
data[3]=z
A = ApproNonLin(data,"(A1+y1*(A2+y1*A3)) + y2*((A4+y1*(A5+y1*A6)) + y2*(A7+y1*(A8+y1*A9))) - y3")
```

Density of normal distribution

```
x = ramp(0,0.00001, 10000)
A1 = 10 ; amplitude
A2 = 0.03 ; center
A3 = 0.01 ; sigma
y = A1 * exp ( -( (x-A2)^2 / (2*A3^2) ) )
A = ApproNonLin(y,"A1*exp(-((x-A2)^2/(2*A3^2)))-y")
center = A[2]
sigma = A[3]
```

Equidistant input data without initial values, but with weighting, (straight) line

```
t = ramp(0,0.001, 50000)
test = 4 * t + 10 ; test data
Weight=test*0+1
Weight[1] = 0.1
A = ApproNonLin(test,"A1 * x + A2 - y",0,0,"", Weight)
```

XY input data with initial values, circle

```
t = ramp(0,0.001, 1000)*30
A1 = 3
A2 = 10
A3 = 30
X = A1 + A3 * sin(t) + 1*random ( 1000, 2, 0, 0, 0 )
Y = A2 + A3 * cos(t)+ 1*random ( 1000, 2, 0, 0, 0 )
input_xy = xyof ( X, Y ) ; test data
start = "init A1 = 0, init A2 = 0"
A = ApproNonLin ( input_xy, "sqrt ( (X-A1)^2+(Y-A2)^2 )- A3",0,0, start )
```

XY input data with value range and initial value, circle

```
t = ramp(0,0.001, 1000)*30
A1 = 3
A2 = 10
A3 = 30
X = A1 + A3 * sin(t)
Y = A2 + A3 * cos(t)
input_xy = xyof ( X, Y ) ; test data
limits = "init A1 = 0, A1 < 100, A2 < 100, A2 >= 0"
A = ApproNonLin ( input_xy, "sqrt ( (X-A1)^2+(Y-A2)^2 )- A3",0,0, limits )
```

Segmented input data, curved surface

```
A1 = 4
A2 = -0.1
A3 = 0.002
A4 = -0.3
```

```
A5 = 0.2
y=leng(0,1000) ; test data
setseglen(y, 100)
x=ramp(0,1,100)
for i = 1 to 10
   z=i-1
   y[i] = A1 + A2 * x + A3 * x * x + A4 * z + A5 * z * z
end
formel = "A1 + A2 * x + A3 * x * x + A4 * z + A5 * z * z - y"
A = ApproNonLin(y,formel)
```

Many input channels y1 through y4

```
A1 = 3
A2 = -0.3
A3 = 0.2
A4 = -0.3
x=ramp(0,1,100)
y1 = 0.0003 * x * x - 0.03*x+1 ; test data
y2 = 3 * sin ( x * 0.1 )
y3 = 5 * exp ( -0.03 * x )
y4 = A1 * y1 + A2 * y2 + A3 * y3 + A4
y=leng(0,400)
setseglen(y, 100)
y[1] = y1
y[2] = y2
y[3] = y3
y[4] = y4
A = ApproNonLin(y, "A1 * y1 + A2 * y2 + A3 * y3 + A4 - y4")
```

Decay function without offset. Provides better compensation than eFit().

```
t = ramp(0,0.001, 1000)
data = 5 * exp ( t * -3 ) + 0.1* random ( 1000, 2, 0, 0, 13 )
A = ApproNonLin(data,"A1 * exp ( A2 * x ) - y")
data_A = A[1] * exp ( A[2] * t )
data_B = eFit( data )
```

Best fit plane: y = A1 + A2 * x1 + A3 * x2. Triplets (x1,x2,y) are given, which in a display of y with respect to x1 and x2 are supposed to be in one plane.

```
; test data
A1 = 7
A2 = 3
A3 = 0.5
x1 = 1 + 5 * sin ( ramp(0,1,100) * 0.2 ) * ramp(3,1,100)/100 ; x direction
x2 = 2 + 3 * cos ( ramp(0,1,100) * 0.2 + 0.1 ) * ramp(5,1,100)/100 ; z direction
y = A1 + A2 * x1 + A3 * x2 + 0.0 * random ( 100, 2, 0 , 0, 3 ) ; amplitudes
;calculation
data = MatrixInit(leng?(y),3)
data[1] = x1
data[2] = x2
data[3] = y
A = ApproNonLin( data, "A1 + A2 * y1 + A3 * y2 - y3")
```

See also:

Appro, LFit, eFit, Poly

ASCII

Sets the ASCII format for subsequent loading of files using the command LOAD

Declaration:

```
ASCII SvSkipRange SvType SvDivider SvCount SvOption Leader-string
```

Parameter:

SvSkipRange	Length of the leader; specified in either values or Bytes (depending on the 2nd parameter [Type])
SvType	Indicates how the parameter [Leader] is to be interpreted.
	0 : The Leader specifies the number of Bytes to skip over.
	1 : The Leader specifies the number of numerical values to skip over.
SvDivider	Specifies "n", where every n-th value is to be read from the values in the file.
SvCount	Indicates the number of values to be read. Zero means that all values are read until the end of the file.
SvOption	Option
	0 : Numbers with a decimal point are expected. The data set generated has the data format "Float" (4 Byte Real).
	1 : Numbers with a decimal comma are expected. The data set generated has the data format "Float" (4 Byte Real).
	10 : Numbers with a decimal point are expected. The data set generated has the data format "Double" (8 Byte Real).
	11 : Numbers with a decimal comma are expected. The data set generated has the data format "Double" (8 Byte Real).
Leader-string	A string which the system searches for in the file, before beginning to read the leader and the numerical values. The string may not contain spaces. If no string is supplied as the parameter, then reading begins with the leader immediately at the start of the file.

Description

The file format for the command LOAD is set to ASCII format.

If no parameters are specified for the "ASCII" command, the currently defined ASCII format is selected.

A completely new ASCII format is defined when parameters are specified.

- You must either specify all parameters or none. It is not possible to just specify one or a few of them. Each parameter, with the exception of the string, may be a fixed numerical value or a single value (Sv) variable. The use of variables as parameters allows variable file formats to be read.
- The "ASCII" command represents an automation of the dialog box "Options / File - Load/Import / ASCII". This dialog box is accessed by selecting the "Options" button in "Load file" when the ASCII format is selected. A detailed description and further examples are found in the Chapter 'File Management'.
- An alternative to ASCII import with the command combination ASCII/LOAD (espcially for complx formats) is to use the imc File Assistant ('Load File'-dialog, format: "ASCII/Excel Import", button 'Options') or the imc File Assistant for creating an import filter and the use of the filter with the command FASLOAD or the function FileOpenFAS.
- To read text files line-by-line, you can use the function FileOpenASCII().
- Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
ASCII
LOAD DATA.ASC
```

The last ASCII format specified is set again; files were loaded in a different format, e.g. FAMOS format, since it was last used. Then the ASCII-file DATA.ASC is loaded.

```
ASCII 0 0 2 0 0 test
LOAD DATA.ASC
```

One column is read from a file containing a table of ASCII values. The header of this table is "Costs". The first of the two columns is read.

```
Vleng = 10
Data = 100
ASCII Vleng 1 1 Data 11
LOAD DATA.ASC
```

100 values are read from a ASCII file. A leader with a length of 10 bytes is skipped at the beginning of the file. The numbers are read with a decimal comma. The result data set has the data format 8 Byte Real.

See also:

LOAD, BINARY, FileOpenASCII, FileOpenFAS

ASCOPTION

Specifies the file format for the command ASCSAVE (storing of files in ASCII format).
Instead of the commands ASCOPTION and ASCSAVE, you can also use the function FileOpenASCII2. This offers much more flexible file export capability and can also be use to save multiple waveforms in multi-column ASCII files.

Declaration:

```
ASCOPTION SvNumberformat SvDecimalSepar SvPrecision SvWidth SvWithHeader SvWithXvalues
```

Parameter:

SvNumberformat	NumericalFormat
	0 : All values are saved in exponential notation
	1 : All values are saved in fixed point notation.
	2 : The shortest possible notation for the value is selected.
SvDecimalSepar	DecimalSeparator
	0 : Decimal point
	1 : Decimal comma
SvPrecision	In number formats 0 and 1, the number of places after the decimal; in number format 2, the total number of significant places Value range 0..7
SvWidth	Minimum output width; if the width of the value is smaller than this parameter, the value is filled with zeros from the left Value range 0..99
SvWithHeader	Specifies whether the file is to be prefixed with a standardized file header.
	0 : No header
	1 : A standardized header, containing significant characteristics for the variable, entered at the beginning of the file.
SvWithXvalues	Specifies whether a two-column output including x-coordinates is to be outputted.
	0 : Y-values are written each below the last in a vertical column.
	1 : Each line in the file contains the x-value, then a tab and then the corresponding y-value.

Description

The ASCII format for saving files (command: ASCSAVE) is specified here.

The command ASCOption effectively automates the dialog box 'Options / File -Save/Export / ASCII '. You can do this, for example, by selecting the 'Options' button in the 'Save file' dialog when the ASCII format is selected. Please refer to Chapter 'File Management'. for further information and further examples. The header is also described here.

The parameters can be fixed numbers or single value variables.

Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

A file in imc/FAMOS-format is loaded and saved again as a two-column ASCII-file.

```
FAMOS
LOAD sintest1.dat data
ASCOPTION 1 0 5 10 0 1
ASCSAVE data "c:\my data\sintest1.txt"
```

See also:

ASCSAVE, FileSave, FileOpenASCII2, FileLineWrite

ASCSAVE

Saves files in the ASCII format
Instead of the commands ASCOPTION and ASCSAVE, you can also use the function FileOpenASCII2. This offers much more flexible file export capability and can also be use to save multiple waveforms in multi-column ASCII files.

Declaration:

```
ASCSAVE VariableName Filename
```

Parameter:

VariableName	Variable to be saved
Filename	Filename (or also complete path), under which the variable is to be saved.

Description

The variable specified as a parameter is saved as an ASCII file; the variable can be a single value or a normal waveform. If no file name is specified, the variable is saved under its own name in the directory currently selected for storing variables. The standard file name extension is ".ASC".

When the file name is specified, the variable is saved under this name. If a complete directory is specified, this is used for storing the variable.

The filename may also specified to contain quotation marks. This can be necessary, if, for instance, the path contains spaces.

Configuration of the file format is performed in accordance with the settings under "Options"/"Save File/Export"/"ASCII (FAMOS 3.2 compatible)" or by means of the command ASCOPTION.

This command cannot be used to store complete data groups. Use this command to save individual channels. Storing texts is also not possible with this command; use the function FileSave() with the format identifier "imc/Text" or the function FileLineWrite for saving texts.

Examples:

```
Data = Int(Data)
ASCSAVE Data
```

The variable Data is integrated and stored in the file "DATA.ASC".

```
Hist = Histo(Data1, 10, 6)
ASCSAVE Hist c:\data\histogr\hist001
Hist = Histo(Data2, 10, 6)
ASCSAVE Hist c:\data\histogr\hist002
```

A histogram is calculated for each of the waveforms Data1 and Data2 and are stored in the specified directory under the names 'HIST001.ASC" and 'HIST002.ASC' in ASCII format.

```
ASCSAVE Hist "c:\data\histogr\My Histogram"
```

The filename contains spaces and therefore it must be written inside of quotation marks.

See also:

ASCOPTION, FileSave, FileOpenASCII2, FileLineWrite

asin

Arcsine; inverse of sine

Declaration:

```
asin ( Data ) -> Angle
```

Parameter:

Data	Data; allowed types: [ND],[XY].
Angle	
Angle	Arcsine of the parameter (angle in radians)

Description:

The arcsin function is computed.

This function returns an angle in radians without units.

The x-coordinate(s) of the results and the parameter are the same.

Remarks

- The argument should not have any unit; if it does, a warning is issued and the unit is adopted unchanged.
- The permittedd value range of the parameter is between -1 and +1.
- The parameter may be structured (events/segments).

Examples:

asin(0)= 0, asin(1)= PI/2, asin(-1)= -PI/2)

```
pihalf = asin(1)
```

A serrated triangular data set is generated from a sinusoidal data set:

```
NDspikes = asin(NDsine)
```

With the pre-defined constant PI or INDEGR, the angle returned is converted from radians to degrees:

```
value_90 = asin(1.0) * 180 '°' / PI
value_90 = asin(1.0) * INDEGR
```

See also:

sin, acos, atan2

atan

Arctan; inverse of tangent

Declaration:

```
atan ( Data ) -> Angle
```

Parameter:

Data	Data; allowed types: [ND],[XY].
Angle	
Angle	Arctangent of the parameter (angle in radians)

Description:

This function is only included for compatibility reasons. In general, the funuction atan2() is more suitable.

The aectan function is computed.

This function returns an angle in radians without units.

The x-coordinate(s) of the results and the parameter are the same.

Remarks

- The argument should not have any unit; if it does, a warning is issued and the unit is adopted unchanged.
- The parameter may be structured (events/segments).

Examples:

atan(0)= 0, atan(1)= P1/4, atan(-1)= -PI(4))

```
pi025 = atan(1)
```

Correction of Data by lookup of the tangent; e.g. saturation effects:

```
NDcorr = atan(NDdata)
```

With the pre-defined constant INDEGR, the angle returned is converted from radians to degrees:

```
value_45 = atan(1.0) * INDEGR
```

See also:

tan, atan2, asin, acos

atan2

Arctan; inverse of tangent

Declaration:

```
atan2 ( y, x ) -> Angle
```

Parameter:

y	y-coordinate(s) in the cartesian plane. Allowed types: [ND].
x	x-coordinate(s) in the cartesian plane. Allowed types: [ND].
Angle	
Angle	Arctangent of the parameter (angle in radians)

Description:

In contrast to the atan() function, this function returns the angle in the range from -PI to +PI.

The signs of [x] and [y] are used to determine the correct quadrants for this result.

The function is defined for each point except the origin (0,0). The result's unit is radians ("Rad"). For conversion to "Degrees", see the example below.

If one of the parameters is a single value, all values belonging to the other parameter will be calculated on the basis of the single value. If both parameter sets have more than one point, then the arctan is calculated value-by-value until the end of the shorter data set is reached.

Both parameters may be structured (events/ segments), however, in that case, the respective other parameter must have exactly the same structure (same segment length, event-count and -length), or be a single value.

Examples:

```
RectAngle = atan2(1, 0) * INDEGR   ;=> 90 degree
Angle_Minus90 = atan2(-1, 0) * INDEGR  ;=> -90 degree
Angle_180 = atan2(0, -1) * INDEGR  ;=> 180 degree
```

2 ways to find the Cartesian representation of the phase in a complex data set:

```
phase1 = atan2(CplxRect.I, CplxRect.R)
phase2 = Cmp2(Pol(CplxRect)) * INDEGR
```

See also:

tan, atan2, asin, acos

BEGIN_PARALLEL

Available in: Professional Edition and above

Opens a code-block with sequence functions to be executed in parallel.

Declaration:

```
BEGIN_PARALLEL
```

Description

Within a parallel-block (meaning all lines between a BEGIN_PARALLEL and END_PARALLEL-keyword), all the instructions are executed in parallel. Only direct calls of sequence functions are allowed as instructions in a parallel-block. The sequence functions are each started in a separate execution context (thread), then the system jumps immediately to the next line without waiting for the end of execution.

The end of such a block is indicated by the END_PARALLEL-keyword. For END_PARALLEL, the sequence waits until all previously initiated sequence functions are concluded and only then resumes execution.

At least 2 threads are always active in FAMOS. The Main thread takes care of the Display of the program interface and the processing of user input. By default, all sequence instructions in FAMOS are processed in the Standard execution thread, which is also always active and, if necessary, waits for tasks. The current execution thread of a sequence function in a BEGIN_PARALLEL block is temporary and is terminated at the end of the processing of the sequence function.

The operating system ensures that the parallel executions are optimally distributed among the available processor cores. In contrast to sequential execution, which uses only one processor core, this makes significantly faster execution of the sequence possible.

Application Tips

- Only truly time demanding calculations and analyses should be run in parallel. When the execution time is short, the losses due to resource demands from extra administration are often greater than the time gains from parallel execution.
- Please ensure that you only use parallel execution for tasks which are truly mutually independent. The order in which the partial tasks are executed is random, so the results of one parallel sequence function (not only explicit results variables, but also newly created files) may thus not be required as the input parameters or as implicit prerequisites for other parallel sequence functions. Approach the question this way: If executed sequentially, would it be possible to arbitrarily change the order of the calls in the PARALLEL-block? If not, then parallel execution is not possible.
- Sequence functions called in PARALLEL-blocks should only use local variables to the extent possible, since global variables could be changed if they are ever accessed in any unrelated operation. Example: the same sequence function is called two times in a parallel-block, with different parameters. In the sequence, a non-local loop variable i is used in a FOR-loop. Both threads access this variable in parallel and change its value. Thus, how often the loop is run in each oft sequence is actually random.

Scope of presettings and initial values

- Some functions define default settings or initial values which are then used by subsequent function calls. These settings can be global or thread-local.
- Global means here that a single global memory is available and this memory is valid and used in all execution threads. Examples are e.g. functions such as SetOption(), SelMeasListSetName(), FFTOPTION and all functions for Panel and Report remote control.
- Thread-local means that the internal memory is managed separately for each execution thread, so it may also have to be reinitialized in each sequence function called in parallel. Examples are e.g. Stat(), FileOpenDSF(), SelUseMeasurement(), TerzI(), CwSelectWindow(), OtrTachoMode() and the functions for Excel and Powerpoint remote control. The specific behavior is described in the respective function help.

Constraints

- The maximum count of sequence functions executed in parallel in a PARALLEL-block is **64**. The count of virtual processor cores in the system serves as a practical and sensible upper boundary. This number can be obtained using the function GetSystemInfo().
- PARALLEL-blocks can not be nested. Within a PARALLEL-block, any calls of BEGIN_PARALLEL, for example, within the sequence function called are ignored.
- Some functions cannot be used within BEGIN_PARALLEL blocks. Examples of this are, for example, the functions of the Database kit or the ASAM-ODS kit. Other functions can be called anywhere, but are pushed internally into the FAMOS main thread and executed from there, e.g. the functions of the R- and Pythons kits. Such restrictions are described in the respective function help.
- PARALLEL blocks are not permitted within Timer Event sequences (panels).
- PARALLEL-blocks are ignored during execution in Debug-mode (e.g. "Single Step", "Execute (Observe Breakpoints)".
- Upon Suspend/Resume of execution (e.g. because an error occurred), the system waits until all active parallel executions have concluded.

Locking of access to variables

The quasi-simultaneous execution of lines of code is possible with PARALLEL blocks. When a line of code is executed in a parallel sequence function, the variables used (e.g. function parameters) are exclusively reserved internally for the time of execution of a partial function in this line. If another sequence line is to be executed in parallel, in which variables that have already been reserved are to be used, processing is delayed until all the required variables are available again. This prevents a variable used from being unexpectedly changed from elsewhere during processing. The variables are normally released after the sequence line has been completely processed.

There is an exception when calling sequence functions. Since when jumping into the sequence function in general. If a local copy of the parameters is created, the reserved parameters can already be released when processing the function begins. This also makes it possible for sequence functions that work on the same variable(s) to actually be executed in parallel within a BEGIN_PARALLEL block. But beware: if a data group is transferred as a parameter and the data type of the corresponding sequence function parameter is not defined as a data group, a loop is executed internally over all channels of the group. The parameter can then only be released when entering the last loop pass.

When duplicating a parameter for the local copy, however, there is the special feature that the original and the copy initially refer to the same internal memory area for the actual data. This optimization, which is common in FAMOS (which is also used with a simple assignment "a = b"), of course, saves storage space and time when duplicating, especially with large data sets, but can be disadvantageous with parallel calls and lead to unwanted delays! I f parallel executions want to access the data of the local copies of the same parameter at the same time, an appropriate lock must take effect because of the shared internal memory area. So if a parameter is used in a longer-running function, parallel accesses to the same parameter's data in other sequence functions must wait until the current function has finished its work.

So if the execution time of a PARALLEL block, in which the same data set is used repeatedly as a parameter for the called sequence functions, does not meet your expectations, you should try passing a temporary copy with its own (unshared) memory area instead of the original parameter. The easiest way to force a true copy with its own memory space is to add a 0. This procedure is demonstrated in Example 3.

Examples:

```
An analysis is applied to three data sets which returns a single characteristic number as its result. Subsequently, the mean value of the results is calculated.
BEGIN_PARALLEL
   t1 = !CalcSpecificValue(channel_1)
   t2 = !CalcSpecificValue(channel_2)
   t3 = !CalcSpecificValue(channel_3)
END_PARALLEL
t = (t1+t2+t3)/3
```

Suppose a text array contains a large amount of filenames. We want to load and evaluate the associated files.

```
allFiles = FsGetFileNames("c:\data", "*.raw", 0, 1, 0)
FOREACH ELEMENT file in allFiles
   !WorkWithFile(file) ; loads the file and carries out the evaluation
END
```

This routine is to be accelerated by means of parallel execution:

```
allFiles = FsGetFileNames("c:\data", "*.raw", 0, 1, 0)
count = TxArrayGetSize(allFiles)
chunksize = floor(count/4)
BEGIN_PARALLEL
   !WorkWithFiles(TxArrayPart(allFiles, 1,              chunksize))
   !WorkWithFiles(TxArrayPart(allFiles, 1+ chunksize,   chunksize))
   !WorkWithFiles(TxArrayPart(allFiles, 1+ chunksize*2, chunksize))
   !WorkWithFiles(TxArrayPart(allFiles, 1+ chunksize*3, count-3*chunksize))
END_PARALLEL
```

For this purpose, the sequence function !WorkWithFiles is defined as follows:

```
; Declaration: !WorkWithFiles(Par1 [Data type: Textarray])
FOREACH ELEMENT file in Par1
   !WorkWithFile(file)
END
```

```
testdata= ...
f = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 500, 1000, 5000, 10000]* 1e-3
```

A long data set is to be subjected to high-pass filtering with different cut-off frequencies one after the other.

```
FOR i = 1 TO lang?(f)
   Name = "Filt "+ TForm(i, "")
   <Name> = FiltHP(testdata, 0, f[i])
END
```

This routine is to be accelerated by means of parallel execution:

```
count = lang?(f)
chunksize = Floor(count/4)
BEGIN_PARALLEL
   !DoHighPass_Chunk(testdata+0, f+0, 1, chunksize)
   !DoHighPass_Chunk(testdata+0, f+0, 1+  chunksize, chunksize)
   !DoHighPass_Chunk(testdata+0, f+0, 1+2*chunksize, chunksize)
   !DoHighPass_Chunk(testdata+0, f+0, 1+3*chunksize, count-3*chunksize)
END_PARALLEL
```

The sequence function !DoHighPass_Chunk is defined as follows:

```
;   Declaration:
;   !DoHighPass_Chunk (data [Datatype:Normal], freq, index, count  [Single values])
;   (Option "Newly created variables are local by default" is deactivated.)
LOCAL i
FOR i = index TO index+count-1
   LOCAL Name = "Filt_"+ TForm(i, "")
   <Name> = FiltHP(data, 0, 0, freq[i])
END
```

Passing "testdata+0" instead of "testdata" to the sequence function ensures that each call is executed with a complete, self-contained copy of "testdata". The same applies to the "freq" parameter. The FiltHP() calls in the sequence function are thus completely decoupled and can be executed in parallel. Without this trick, all "data" and "f" instances in the sequence functions would internally share the same data memory and the FiltHP() calls would have to be executed practically one after the other. The only parameters to be taken into account are those that are passed directly to functions that can potentially take a long time and therefore have to be blocked for a longer period of time. The variable "chunksize" in the example (local name "count") is only latched very briefly when reading, so the "+0" trick is not necessary here. See also the above explanation under "Locking of variable accesses".

For a group consisting of many channels, the FFT of each channel is to be calculated.

```
GrFFTs = FFT(GrInput, 0, 0)
```

This calculation is to be accelerated by means of parallel execution:

```
count = GrChanNum?(GrInput)
chunksize = floor(count/4)
BEGIN_PARALLEL
   GrFFT1 = !MyFFT(GrPart(GrInput, 1,              chunksize))
   GrFFT2 = !MyFFT(GrPart(GrInput, 1+ chunksize,   chunksize))
   GrFFT3 = !MyFFT(GrPart(GrInput, 1+ chunksize*2, chunksize))
   GrFFT4 = !MyFFT(GrPart(GrInput, 1+ chunksize*3, count-3*chunksize))
END_PARALLEL
GrFFTs = GrConcat(GrFFT1, GrFFT2, GrFFT3, GrFFT4)
```

For this purpose, the sequence function !MyFFT is defined as follows:

```
; declaration:
; !MyFFT(Par [Datatype: Normal]) => Result [Complex]
Result = FFT(Par, 0, 0)
```

See also:

END_PARALLEL

Supported since:

Version 2022, Editionen: Professional, Enterprise, Runtime

BINARY

Sets the BINARY format for subsequent loading of files using the command LOAD

Declaration:

```
BINARY SvFormat SvBytes SvSkipRange SvSpace SvCount SvBits SvSign SvMinimum SvMaximum String
```

Parameter:

SvFormat	Format
	1 : Real floating-point numbers in Intel format
	2 : Whole numbers in Intel-format
	3 : Real floating-point numbers in the Motorola format
	4 : Whole numbers in the Motorola-format
SvBytes	A number's Byte count
	1 : 1 Byte (only integer)
	2 : 2 Bytes (only integer)
	4 : 4 Bytes (integer or real)
	6 : 6 Bytes (only real)
	8 : 8 Bytes (only real)
SvSkipRange	Specifies the length of the leader to be skipped, in Bytes.
SvSpace	Indicates how many Bytes should be skipped between the individual numbers to be read.
SvCount	Indicates how many numbers are to be read. Zero means that all values to the end of the file should be read.
SvBits	Only for Type = 2 or 4 (integer): The number of valid bits in an integer. Used together with minimum and maximum for scaling the integers.
SvSign	Only for type = 2 or 4 (integer): Indicates whether integers are to be interpreted as signed (two's complement display) or unsigned. A value of 1 means "signed", zero means "unsigned".
SvMinimum	Only for Type = 2 or 4 (integer): Real value of the smallest number displayable in the specified format, is a real number; also interpreted as the lower limit of the measurement range.
SvMaximum	Only for Type = 2 or 4 (integer): Real value of the largest displayable number in the specified format, is a real number; also interpreted as the upper limit of the measurement range.
String	A string which the system searches for in the file, before beginning to read the leader and the numerical values. The string may not contain spaces. If no string is supplied as the parameter, then reading begins with the leader immediately at the start of the file.

Description

The file format for the command LOAD is set to a binary format

If no parameters are added to the command BINARY, the Binary format currently set is selected.

If parameters are specified, a binary format is completely redefined. The parameter to be specified depends on the type of binary format to be read. If any parameter is specified, all parameters must be specified.

The command BINARY is essentially the automation of the dialog box 'Options'/ 'Load file'/ 'Binary'. One way to access this dialog box is to click on the button 'Options' in the 'Load File'-dialog when Binary format is set. For details and more complete examples, see the chapter 'File Management' in the user's manual.

- Each parameter, with the exception of a string, may be a fixed numerical value or a single value (SV) variable. Using variables as parameters allows variable file formats to be read.
- As an alternative to BINARY import by means of the command combination ASCII/BINARY (especially for more complex formats), it is possible to use the imc File Assistant to create an import filter to be used with the command FASLOAD or the function FileOpenFAS().
- Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
BINARY
LOAD DATA.BIN
```

The binary format used is set again; files were loaded in a different format, e.g. FAMOS format, since it was last used. The file DATA.BIN is then loaded.

```
BINARY 1 4 20 8 100 Data:
LOAD DATA.BIN
```

The format for reading the following binary format is defined: the beginning of the file contains information about an experiment; the actual binary data are introduced by the string "Data:". Then real numbers in 4-bytes display are read. The first 5 numbers (= 20 bytes) are skipped, and then only every third number (i.e.: 2 * 4 = 8 bytes space between the numbers) is read. 100 numbers are read; the waveform generated in FAMOS has a length of 100.

```
Mini = -10.0
Maxi = 10.0
Bits = 12
BINARY 2 2 0 0 0 Bits 0 Mini Maxi
LOAD DATA.BIN
```

All numbers are to be read from a file. The beginning of the file contains no information to be skipped, only numbers in as 12-bit values in 2-byte display are contained. These unprocessed data originate in an AD-converter, and have no sign. The measurement range of the converter is -10V to 10V.

See also:

LOAD, ASCII, FileOpenASCII, FileOpenFAS

BitAnd

Bitwise "AND" logic operation

Declaration:

```
BitAnd ( Data, Mask, SvDataformat ) -> Result
```

Parameter:

Data	First value/data set to be linked. Allowed data types: [NW],[XY].
Mask	Second value/data set (mask) to be linked [NW]
SvDataformat	Assumed (integer) data format. Allowed values include:
	64 : 64-bit, unsigned
	32 : 32-bit unsigned
	16 : 16-bit unsigned
	8 : 8-bit unsigned
	1 : 1 bit (digital)
	-8 : 8-bit signed
	-16 : 16-bit signed
	-32 : 32-bit signed
	-64 : 64-bit, signed
Result	
Result	Result of bitwise logic operator

Description:

Bitwise 'AND'-operation applied to two numbers. Both operands are converted to the specified integer data format and a logical 'AND'-operation is performed on each pair of corresponding bits. The result bit is 1 if both bits are 1, else 0.

The second parameter must either be a single value or a data set with exactly identical structure (length, segmenting, events) as the first parameter, in this case the operator is applied point-by-point to corresponding values.

Negative integers are represented as a two's complement.

When converting to the data type 'Digital', all values which do not equal 0 are regarded as 1. The result is identical to that of the logical AND-operator.

The result has the format specified in the parameter [SvDataFormat].

Examples:

In a 1-Byte data set (value range: 0..255), the upper 4 bits are set to 0.

```
Erg = BitAnd(Data, 0x0F, 8)
```

In a 4-Byte wide data set, the value of the 2nd Byte is determined.

```
Byte2 = BitShift(BitAnd(Data, 0x00FF0000, 32), -16, 32)
```

All points are found in which the 7th bit in both data sets is 1 at the same time.

```
Result = BitAnd(BitGet(Port1, 7), BitGet(Port2, 7), 1)
; is compatible with:
; Result = BitGet(Port1, 7) AND BitGet(Port2, 7)
```

See also:

BitOr, BitNot, BitShift, BitGet, BitSet

BitGet

Querying a bit

Declaration:

```
BitGet ( Data, SvBitNumber ) -> ZeroOrOne
```

Parameter:

Data	Single value/data set to be queried. Allowed data types: [NW],[XY].
SvBitNumber	Number of the bit to be set. The bit on the far right (LSB) has the number 1. Permitted range: 1<= [SVBitNumber] <= 64.
ZeroOrOne	
ZeroOrOne	Result of query

Description:

Gets a bit in an integer's bit pattern. The values of the data set provided are converted to an integer and the content (0/1) of the associated bit is returned.

Negative integers are represented as a two's complement.

the result has the data format 'Digital'.

Examples:

All points are found in which the 7th bit in both data sets is 1 at the same time.

```
Result = BitAnd(BitGet(Port1, 7), BitGet(Port2, 7), 1)
; is compatible with:
; Result = BitGet(Port1, 7) AND BitGet(Port2, 7)
```

In a 1-Byte data set, the 2nd bit is set to the value of the 1st bit.

```
Bit1 = BitGet(Data, 1, 8)
Result = BitSet(Data, Bit1, 2)
```

See also:

BitAnd, BitOr, BitNot, BitShift, BitSet

BitNot

Bitwise inversion

Declaration:

```
BitNot ( Parameter, SvDataformat ) -> Result
```

Parameter:

Parameter	Single value/data set to be inverted. Allowed data types: [NW],[XY]
SvDataformat	Assumed (integer) data format. Allowed values include:
	64 : 64-bit, unsigned
	32 : 32-bit unsigned
	16 : 16-bit unsigned
	8 : 8-bit unsigned
	1 : 1 bit (digital)
	-8 : 8-bit signed
	-16 : 16-bit signed
	-32 : 32-bit signed
	-64 : 64-bit, signed
Result	
Result	Result of bitwise logic operator

Description:

Bitwise inversion of the value of the parameter. Each value is converted to the specified integer data format and a logical inversion is performed for each bit. The result bit is 0 if the input bit is 1, else 0.

Negative integers are represented as a two's complement.

When converting to the data type 'Digital', all values which do not equal 0 are regarded as 1. The result is identical to that of the logical NOT-operator.

The result has the format specified in the parameter [SvDataFormat].

Examples:

All bits of a 2-Byte wide data set are inverted.

```
Result = BitNot(Port, 16)
```

Inversion of a digital data set

```
Result = BitNot(DigChannel1, 1)
; is compatible with:
; Result = Not(DigChannel)
```

Implementation of a bitwise XOR operation

```
Result = BitOr(BitAnd(x, BitNot(y, 32), 32), BitAnd(BitNot(x, 32), y, 32), 32)
```

See also:

BitAnd, BitOr, BitShift, BitGet, BitSet

BitOr

Bitwise "OR" operator

Declaration:

```
BitOr ( Data, Mask, SvDataformat ) -> Result
```

Parameter:

Data	First value/data set to be linked. Allowed data types: [NW],[XY].
Mask	Second value/data set (mask) to be linked [NW]
SvDataformat	Assumed (integer) data format. Allowed values include:
	64 : 64-bit, unsigned
	32 : 32-bit unsigned
	16 : 16-bit unsigned
	8 : 8-bit unsigned
	1 : 1 bit (digital)
	-8 : 8-bit signed
	-16 : 16-bit signed
	-32 : 32-bit signed
	-64 : 64-bit, signed
Result	
Result	Result of bitwise logic operator

Description:

Bitwise 'OR'-operator applied to two numbers. Both operands are converted to the specified integer data format and a logical 'OR' operation is applied to each pair of corresponding bits. The result bit is 0, if both bits are 0, else 1.

The second parameter must either be a single value or a data set with exactly identical structure (length, segmenting, events) as the first parameter, in this case the operator is applied point-by-point to corresponding values.

Negative integers are represented as a two's complement.

When converting to the data type 'Digital', all nonzero values are considered to be 1. The result is identical with that of the logical OR-operator.

The result has the format specified in the parameter [SvDataFormat].

Examples:

In a 1-Byte data set, the lower 4 bits are set to 1.

```
Erg = BitOr(Data, 0x0F, 8)
```

All points are found in which either the 2nd or the 7th bit is set in a data set.

```
Result = BitOr(BitGet(Port1, 2), BitGet(Port1, 7), 1)
; is compatible with:
; Result = BitGet(Port1, 2) OR BitGet(Port1, 7)
```

Implementation of a bitwise XOR operation

```
Result = BitOr(BitAnd(x, BitNot(y, 32), 32), BitAnd(BitNot(x, 32), y, 32), 32)
```

See also:

BitAnd, BitNot, BitShift, BitGet, BitSet

BitSet

Sets a bit

Declaration:

```
BitSet ( Data, ZeroOrOne, SvBitNumber ) -> Result
```

Parameter:

Data	Single value.data set to be edited. Allowed data types: [NW],[XY].
ZeroOrOne	Value(s) to be set [NW]
SvBitNumber	Number of the bit to be set. The bit on the far right (LSB) has the number 1. Permitted range: 1<= [SVBitNumber] <= 64.
Result	
Result	Result of the operation

Description:

Sets a bit in the bit pattern of an integer. The values of the input data set are converted to integers if appropriate and the content of the specified bit is set to 0/1. Subsequently, the resulting bit pattern is converted back to the data format of the input data set. Values of magnitude > 10^14 can not be converted without losses; in this case a warning is issued.

The second parameter must either be a single value or a data set having exactly the same structure (length, segmenting, events) as the first parameter, in this case a point-by-point logical operation on the corresponding values is performed. If the 2nd parameter has a value of 0, then the corresponding bit is set to 0, else to 1.

Negative integers are represented as a two's complement.

The result has the same data format as the input parameter.

Examples:

In a 1-Byte data set, the 7th bit is set to 1.

```
Result = BitSet(Data, 1, 7)
; compatible with:
; Result = BitOr(Data, 0x40, 8)
```

In a 1-Byte data set, the 2nd bit is set to the value of the 1st bit.

```
Bit1 = BitGet(Data, 1, 8)
Result = BitSet(Data, Bit1, 2)
```

See also:

BitAnd, BitOr, BitNot, BitShift, BitGet

BitShift

Shift of the bit pattern of an integer by a specified number of positions.

Declaration:

```
BitShift ( Parameter, SvDigits, SvDataformat ) -> Result
```

Parameter:

Parameter	Single value.data set to be edited. Allowed data types: [NW],[XY].
SvDigits	Number of positions by which to shift the pattern. A number above 0 means shifting to the left; a number below 0 means shifting to the right.
SvDataformat	Assumed (integer) data format. Allowed values include:
	64 : 64-bit, unsigned
	32 : 32-bit unsigned
	16 : 16-bit unsigned
	8 : 8-bit unsigned
	-8 : 8-bit signed
	-16 : 16-bit signed
	-32 : 32-bit signed
	-64 : 64-bit, signed
Result	
Result	Result of bit shifting

Description:

The parameter is converted to the specified integer data format and a bit pattern shift by the specified number of positions is performed.

A positive specification for the number of positions means shifting to the left, zeroes are appended from the right side. Any bits shifted leftward past the edge will disappear. Shifting left by 1 position is equivalent to multiplying by 2.

Specifying a negative position number means shifting the bits to the right; the new bit values placed in the corresponding positions on the left side are either 0 or 1 depending on the numerical format and sign. Bits which are pushed to the right beyond the position limit disappear. For unsigned numerical formats, the bit positions becoming open on the left side as the bits move to the right are filled with zeroes. For signed numerical formats, the sign bit is used to fill the newly open bit positions. Thus if the number is positive, 0 is used; and if the number is negative, 1 is used. A rightward shift by 1 position corresponds to division by 2 (with rounding to the next smaller number if appliable).

The absolute value of the position count must be lower than the number's bit width as defined by the third parameter; for instance, for [SVDataFormat] = 16, the value for [SVPositions] must lie in the range -15..15.

Negative integers are represented as a two's complement.

The result has the format specified in the parameter [SvDataFormat].

Examples:

Two data sets which are each 1 Byte wide are combined to a 2-Byte wide data set, where the 2nd data set is shifted into the upper 8 bits.

```
WordData = BitOr(BitShift(ByteData2, 8, 16), ByteData1, 16)
```

In a 4-Byte wide data set, the value of the 2nd Byte is determined.

```
Byte2 = BitShift(BitAnd(Data, 0x00FF0000, 32), -16, 32)
```

See also:

BitAnd, BitNot, BitOr, BitGet, BitSet

BoxMessage

Text output box with prompt buttons

Declaration:

```
BoxMessage ( TxTitle, TxText, TxOption ) -> SvReturn
```

Parameter:

TxTitle	Title of window
TxText	Text in window
TxOption	Determines type of buttons and icons
	"!1" : Exclamation mark, OK
	"!2" : Exclamation mark, OK/Cancel
	"!3" : Exclamation mark, Repeat/Cancel
	"!4" : Exclamation mark, Yes/No
	"?1" : Question mark, OK
	"?2" : Question mark, OK/Cancel
	"?3" : Question mark, Repeat/Cancel
	"?4" : Question mark, Yes/No
	"S1" : Stop sign, OK
	"S2" : Stop sign, OK/Cancel
	"S3" : Stop sign, Repeat/Cancel
	"S4" : Stop sign, Yes/No
	"I1" : Info(i)-symbol, OK
	"I2" : Info(i)-symbol, OK/Cancel
	"I3" : Info(i)-symbol, Repeat/Cancel
	"I4" : Info(i)-symbol, Yes/No
SvReturn	
SvReturn	Which button was pressed?
	1 : The button "Yes", "OK" or "Repeat" was pressed.
	0 : The button "No"/"Cancel" was pressed.

Description:

This function generates a window with a title, text icons and buttons, which can only be exited by pressing one of the buttons. The particular button pressed is reflected in the return value ["SvReturn"].

- Querying the return value can be skipped in imc FAMOS (which makes sense if there is only one response key).
- The position at which the box appears can be set using the function SetBoxPos.
- You can also manually change the output box's position (e.g. by shifting it with the mouse). The position is kept for the duration of the session.
- Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

For a window with a "Question mark" symbol and "Yes" and "No" option screen buttons:

```
TxQuestion = "Do you really want to stop the evaluation?"
Stop = BoxMessage("Evaluation", TxQuestion, "?4")
```

See also:

BoxOutput, BoxValue?, BoxText?, SetBoxPos, DlgFileName, Dialog

BoxOutput

Output of text and/or a number in an output window

Declaration:

```
BoxOutput ( TxOutput, SvValue, TxFormat, SvOption )
```

Parameter:

TxOutput	Text to be outputted
SvValue	Number to be outputted
TxFormat	Formatting for the value
SvOption	What manner to design output?
	0 : Output in a separate output window. The message must be acknowledged.
	1 : Display in the output box of the imc FAMOS main window

Description:

If [SvValue] displays a single number, it is converted by the format specified in [TxFormat] and appended to the output text. If no single number is displayed, specify the pseudo-constant 'EMPTY' and [TxFormat] is ignored.

Otherwise the following options may be selected for the format "TxFormat". The first letter is the required format identification, then specify one or two numbers, according to the desired format.

TxFormat	Description	Example
"e.N"	Floating point format N: Number of post-decimal digits	3.456 with "e.2" -> "3.46E+00"
"FV.N"	Fixed-point format V: Minimum count of pre-decimal digits N: Count of post-decimal digits Excess pre-decimal places are filled with zeroes.	3.456 with "f2.2" -> " 3.46"
"gV.N"	Fixed-point format V: Count of pre-decimal digits N: Count of post-decimal digits Excess pre-decimal places are filled with zeroes.	3.456 with "g2.2" -> "03.46"
"a.N"	Automatic N: Maximum count of outputted digits The shortest possible notation is used for the respective numerical value. Concluding zeroes after the decimal point are omitted, or even the decimal if applicable. Thus, ideal for integers.	3.40056 with "a.2" -> "3.4"
""	Corresponds to "a.6"	
"xG"	Hexadecimal format G: Total count of digits	340056 with "x8" -> "340056"
"bG"	Binary format G: Total count of digits	34.56 with "b4" -> "100011"

Comma or period

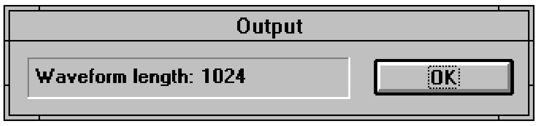
If the period is replaced by a comma in these format strings, the output features a decimal comma instead of a decimal point. Otherwise, the period can be omitted (Example: "f23" is equivalent to "f2.3"; however, "f2,3" forces the use of a decimal comma).

If the period is replaced by a semicolon in these format strings, the output is formatted in accordance with the global presetting for the preferred decimal separator for real numbers ("Extra"/"Options"/"Display & Curve Window", or according to SetOption("Display.DecimalSeparator",...).

- The output box in the main window is emptied before any formulas are executed. When a sequence is processed in single-step mode or a formula is executed using the operation box directly, only the output of the last BoxOutput command is visible.
- The position at which the dialog box appears can be specified with the help of the function SetBoxPos().
- The position and size of the output box can also be changed manually; the program memorizes the current position and size for the duration of each session.
- If an endless loop was accidentally programmed, the sequence can be interrupted by the combination of keys "CTRL" and "Break".
- Multithreading: The function may only be called in the standard execution thread with [SvOption] = 0 (ie separate output box). A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

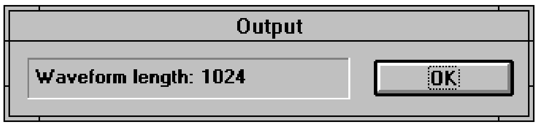
```
leng = Leng?(data)
BoxOutput("Length of data: ", leng, "", 0)
```



Output

Waveform length: 1024

```
VMean = Mean(data)
BoxOutput("Calculated Mean : ", VMean, "f23", 0)
DELETE data
BoxOutput("Calculation Completed!", EMPTY, "", 1)
```



Output

Waveform length: 1024

See also:

BoxMessage, BoxValue?, BoxText?, SetBoxPos, DlgFileName, Dialog

BoxText?

Calls a dialog box for entering text

Declaration:

```
BoxText? ( TxTitle, TxInit, SvOption ) -> TxInput
```

Parameter:

TxTitle	Title of box
TxInit	Suggested input in box upon calling
SvOption	Defines the size of the window.
	0 : Position and size as recently used or defined with SetBoxPos().
	1 : The size of the box is calculated on the required size for a complete display of [TxTitle]. Position of the box as recently used or defined with SetBoxPos() . Explicit line breaks in [TxTitle] (with CarriageReturn/LineFeed combination "~013~010") are observed.
TxInput	
TxInput	Text entered

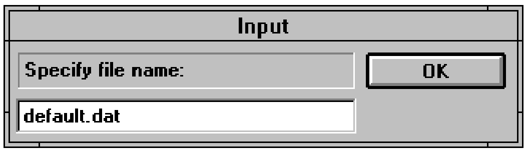
Description:

A small dialog box to enter a text is called. After successful entry and confirmation with [OK], the entered value is returned.

- The position at which the dialog box appears can be specified with the help of the function SetBoxPos().
- The position and size of the output box can be changed manually; the program memorizes the current position and size for the duration of each session and is valid also for the BoxValue? function.
- The maximum length of the returned text is 255 characters.
- If an endless loop was accidentally programmed, the sequence can be interrupted by the combination of keys "CTRL" and "Break".
- Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

```
TxName = BoxText?("Please enter the file name:", "default.dat", 0)
```



With automatic sizing (2 lines):

```
TxName = BoxText?("Please enter the file name:~013~010(EXCEL format required)", "default.xls", 1)
```

See also:

BoxValue?, BoxOutput, BoxMessage, SetBoxPos, DlgFileName, Dialog

BoxValue?

Calls a dialog box for entering a number.

Declaration:

```
BoxValue? ( TxTitle, SvInit, SvOption ) -> SvInput
```

Parameter:

TxTitle	Title of box
SvInit	Entry in the input box upon calling
SvOption	Defines the size of the window.
	0 : Position and size as recently used or defined with SetBoxPos().
	1 : The size of the box is calculated on the required size for a complete display of [TxTitle]. Position of the box as recently used or defined with SetBoxPos() . Explicit line breaks in [TxTitle] (with CarriageReturn/LineFeed combination "~013~010") are observed.
SvInput	
SvInput	Entered value

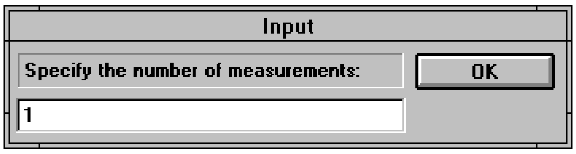
Description:

A small dialog box is called to enter a value; after input and confirmation with [OK], the value entered is assigned to a variable. If the entry cannot be converted to a value, the dialog is not ended.

- The position at which the dialog box appears can be specified with the help of the function SetBoxPos().
- The position and size of the output box can be changed manually; the program memorizes the current position and size for the duration of each session and is also valid for the BoxText?() function.
- If an endless loop was accidentally programmed, the sequence can be interrupted by the combination of keys "CTRL" and "Break". Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

```
Count = BoxValue?("Please specify the number of measurements:", 1, 0)
```



With automatic sizing (2 lines):

```
Count = BoxValue?("Please specify the number of measurements.~013~010Must be smaller than 10.", 1, 1)
```

See also:

BoxText?, BoxOutput, BoxMessage, SetBoxPos, Dialog

BoxVarSelector

A dialog window with a list of the variables present in FAMOS at the time of the call is displayed. The function returns the names of the variables selected by the user.

Declaration:

```
BoxVarSelector ( SelectionType [, TxNameFilter] [, TxTypeFilter] [, TxTitle] [, TxDescription] ) -> TxOrTxaVariablesSelection
```

Parameter:

SelectionType	Specifies whether the user can select exactly 1 variable (single selection) or any number of variables (multiple selection). Also determines the data type of the result (text or textarray).
	"single" : Single selection
	"multi" : Multiple selection
TxNameFilter	Only variables whose names correspond to the name pattern defined here are displayed. The wildcard characters "*" (any number of arbitrary characters) and "?" (exactly 1 arbitrary character) can be used in their usual interpretation can be used. A preceding "!" negates the condition, so all variables that do not match the pattern are displayed. Empty text means no filtering by name will be performed. The filter set here can be changed by the user in the dialog window. (optional , Default value: "")
TxTypeFilter	Only variables whose data type corresponds to the constraint defined here are displayed. Empty text means no filtering by type is performed. (optional , Default value: "")
	"" : No filtering by type
	"SV" : Single value (normale data set of length 1)
	"ND" : Normal data set
	"ND(..)" : Normal data set, Length > 1
	"ND(->)" : Normal data set; no events, no segments
	"ND(..,->)" : Normal data set; no events, no segments, Length > 1
	"CX" : Complex data set
	"!CX" : Data set, not complex
	"CX(->)" : Complex data set; no events, no segments
	"RI" : Complex data set in Real-/Imaginary-part representation
	"MP" : Complex data set in Magnitude/Phase-representation
	"DP" : Complex data set in Decibel/Phase-representation
	"XY" : XY-data set
	"!XY" : Data set; not XY
	"XY(->)" : XY-data set; no events, no segments
	"XY(/)" : XY-data set with monotonically increasing x-track
	"XY(->,/)" : XY-data set with monotonically increasing x-track; no events, no segments
	"TSA" : Data set, TimeStamp-ASCII
	"!TSA" : Data set; not TimeStamp-ASCII
	"TSA(->)" : TimeStamp-ASCII; no events, no segments
	"DS" : Data set (any type), therefore no text, text array or data group
	"Evn" : Data set (any type) with events
	"Seg" : Data set (any type) with segments
	"SegEvn" : Data set (any type) with segments and events
	"!Evn" : Data set (any type) without events
	"!Seg" : Data set (any type) without segments
	"!SegEvn" : Data set (any type); no events, no segments
	"->" : Abbreviation for "!SegEvn"; data set (any type), no events, no segments
	"Monotone" : Single value, normal data set or XY-data set with monotonically increasing x-track
	"/" : Abbreviation for "Monotonic". Single value, normal data set or XY-data set with monotonically increasing x-track
	"Digital" : Digital data set
	"!Digital" : Data set; not digital
	"TXT" : Text
	"!TXT" : No text; therefore data set, text array or data group
	"TXA" : Text array
	"!TXA" : No text array; therefore data set, text or data group
	"TX*" : Text or text array
	"!TX*" : No text or text array; therefore data set or data group
	"Group" : Data group
	"!Group" : No data group
	"Empty" : Variable is empty. Length 0 for data sets; text length 0 for texts; element count 0 for text arrays and TSA.
	"!Empty" : Variable is not empty
	"@M" : The variable is assigned to a measurement.
	"!@M" : The variable is not assigned to any measurement.
TxTitle	The text displayed in the title bar of the dialog window. An empty text means that a standard title ("Select Variables") is used. (optional , Default value: "")
TxDescription	Additional text that may be displayed above the selection list. (optional , Default value: "")

TxOrTxaVariablesSelection	
TxOrTxaVariablesSelection	If [SelectionType]="multi" a textarray with the selected variable names. If the dialog box was canceled or no variable was selected, the dimension of the array is 0. For [SelectionType]="single" a text with the selected variable name. If the dialog window was canceled, an empty text variable is returned.

Description:

The function displays a dialog window in which the user can select from a list of the variables available at the time of call. The names of the selected variables are returned.

- Local variables are not displayed.
- In order for a variable name to be specified in formulas, it must obey certain rules (e.g. no spaces, first character no number etc.). If this is not the case, the name must also be enclosed in curly brackets {...} The names supplied by this function are automatically supplemented with the curly brackets if necessary.
- The position at which the box appears can be specified using the SetBoxPos() function. You can also change the position of the box manually (e.g. by moving it with the mouse). The position is remembered for the duration of the current session.
- Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in their own thread) is not permitted.

Examples:

```
; Single selection from all variables
txName = BoxVarSelector("single")

; Single selection from all variables whose name begins with "channel"
txName = BoxVarSelector("single", "channel*")

; Multiple selection from all variables whose name does not start with "_".
txaNames = BoxVarSelector("multi", "!_*")

; Multiple selection from all variables that are of data type "XY",
; have a monotonically growing x-track and have neither events nor segments
txaNames = BoxVarSelector("multi", "", "XY(->,/)")

; Multiple selection from all variables that belong to the measurement "0001" and are sampled equidistantly
; and do not have any events or segments. Window title and description are specified.
txaNames = BoxVarSelector("multi", "*@0001", "ND(->)", "Spectral Analysis", "Please select the data for which you want to perform a spectral analysis.")
```

The user selects a variable from which the spectrum is then calculated. Only variables are displayed whose name begins with "chan" and whose type is permitted as a parameter for which the following calculation is carried out.

```
txaName = BoxVarSelector("single", "chan*", "ND(->)")
IF txName <> ""
spectrum = Spec(<txName>)
   SHOW <txName>
   SHOW spectrum
END
```

The user makes a selection from the variables associated with the measurement, whose name contains the subtext "engine". The selected variables are smoothed.

```
txaNames = BoxVarSelector("multi", "*engine*", "@M", "Smooth data")
FOREACH ELEMENT txName IN txaNames
   <txName> = Smo5(<txName>)
END
```

The user selects variables that are then to be saved together in a file. Texts and text fields are not offered for selection.

```
txaNames = BoxVarSelector("multi", "", "!TX*", "", "Please select the data to be saved.")
c = TxArrayGetSize(txaNames)
IF c > 0
   fh = FileOpenDSF("result.dat", 1)
   IF fh <>> 0
      FOR i = 1 TO c
         txaName = txaNames[i]
         FileObjWrite(fh, <txaName>)
      END
      FileClose(fh)
   END
END
```

See also:

VarGetInit, VarGetInit2, VarGetName?, VarExist?, BoxText?, SetBoxPos, Dialog

Supported since:

Version 2024

BREAK

The command interrupts the processing of the higher-level loop. Running of the sequence resumes after the end of the loop body.

Declaration:

`BREAK`

Description

The command can be used within a WHILE-, FOR- or FOREACH-loop.

Examples:

The first text object located in a file containing multiple data objects is read out.

```
fh = FileOpenDSF("test.dat", 0)
IF fh > 0
   n = FileObjNum?(fh)
   FOR i = 1 to n
       IF FileObjType?(fh, i) = 2
         text = FileObjRead(fh, i)
         BREAK ; exit loop
      END
   END
   FileClose(fh)
END
```

See also:

WHILE, FOR, FOREACH, CONTINUE

BSave

Saves data sets, single values or text as a binary file.

Declaration:

```
BSave ( SvOption, Data, TxFile ) -> SvError
```

Parameter:

SvOption	Configuration of the output file's format. See description.
Data	Variable to be saved. Data set, single value or text
TxFile	Name of the file. Unless a full pathname is specified, the default loading folder is used. Unless a file extension is specified, ".BIN" is used.
SvError	
SvError	Error code
	0 : The file has been saved successfully.
	1 : Insufficient working memory available
	2 : Error in creating the file. Please check the filename specified.
	3 : Unable to write a file. Available space on the data carrier may not be sufficient.

Description:

Meaning of the parameter [SvOption]:

0	Create new file
1	Append to existing file
add these:	
0	Intel-format (LSB first)
10	Motorola format (MSB first)
add these:	
0	save as Bytes (8-bit, Range: 0..255)
100	save as Words (16-bit; Range: 0..65535)
200	save as Longs (32-bit; Range: 0..4.2E9)
300	as 4-Byte Floating-Point number as per IEEE
400	as 8-Byte Floating-Point number as per IEEE
add these:	
1000	The values are saved in two's complement. With Bytes, Words and Longs, the ranges consequently change as follows: Bytes: -128..127, Words: -32768..32767, Long: -2.1E9..2.1E9

Unless a full filename is specified, the current default folder is used. Upon starting FAMOS, the current default folder is set to the preset (dialog: 'Options'/ 'Folders'). It can be changed using the function SetOption().

Examples:

Supposing a file "Curve.bin" is created, which contains at its beginning "Length:xxxxxxxx". xxxxxxxx is the number of number of values following, as a 4Byte-Integer number, which are saved as 2Byte-Integer numbers.

```
BSave(0, "Length:", "Curve")
Len = Leng?(Data)
BSave(201, Len, "Curve")
BSave(101, Data, "Curve")
```

See also:

FileSave

CASE

Designates one alternative in a case distinction initialized by the command SWITCH (multiple branching). The expression supplied here is compared with the value specified with the SWITCH block. If they are equal, the instructions belonging to the associatedd CASE block are carried out.

Declaration:

```
CASE TestExpression
```

Parameter:

TestExpression	Value or value list with which the SWITCH comparison value is compared.

Description

The end if the instructions belonging to this CASE is determined by another CASE, a DEFAULT-command or an END-command.

Once the CASE-block has run all the way through, execution of the sequence is resumed at the corresponding END. Subsequent CASE instructions are thus no longer executed, even if they would meet the entry condition.

Resolution of the test expressino must result in either a number or a text, and its type must match that of the comparison value specified for the SWITCH.

Here, you can specify a constant, a variable, or an arithmetical expression:

```
CASE 0
CASE "Monday"
CASE var
CASE var+1
```

A list of expressions separated by commas is also allowed. The comparison condition is met if at least one of the values is identical to the comparison value.

```
CASE 0, 2, 5
CASE "Monday", "Tuesday", "Friday"
CASE var1, var2
CASE var1+1, var+2
```

If the comparison value is numerical, it is also possible to specify a range by using the keyword TO ('LowerLimit TO UpperLimit'). The comparison condition is met if the comparison value lies within the specified range (including edges).

```
CASE 0 TO 10
CASE var1 TO var1+10
```

Text comparison does not distinguish between upper and lower case.

Examples:

A descriptive text is formed for a value normally lying between 0 and 100.

```
SWITCH Round(value, 1)
CASE 0
   Tx = "Lower limit"
CASE 1 TO 48
    Tx = "Lower half"
CASE 49,50,51
    Tx = "Center"
CASE 52 To 99
   Tx = "Upper half"
CASE 100
   Tx = "Upper limit"
DEFAULT
   Tx = "Invalid Value"
END
```

The user is prompted to select a file to open. based on the file extension, the file format is recognized and the corresponding function for opening it is called.

```
TxFileName = DlgFileName("", "", "",0)
TxFileExt = FsSplitPath(TxFileName, 3)
SWITCH TxFileExt
CASE ".dat", ".raw"
   ; Load imc data file
   fh = FileOpenDSF(TxFileName, 0)
   ;...
CASE ".xls"
   ; Load EXCEL file
   fh = FileOpenXLS(TxFileName, 0)
   ;...
DEFAULT
   PAUSE ==> Invalid file format
END
```

See also:

SWITCH, DEFAULT, IF

CCF

Cross correlation of a data set with a reference data set

Declaration:

```
CCF ( ReferenceData, TestData ) -> Result
```

Parameter:

ReferenceData	Reference data set
TestData	Test data set
Result	
Result	Results of the cross correlation

Description:

The cross-correlation function (CCF) indicates how similar two data sets are for different shifts in the x direction. The cross-correlation function returns values between -1 and +1.

A value of 1 means that the signals are identical for that shift; a value of -1 means that both signals are the inverse of each other. If one of the signals is positive, the other signal has the same amplitude but a negative sign. A value of 0 means that both signals have nothing in common (are not correlated). All values between -1 and +1 are possible.

Cross-correlation allows the user to determine whether a signal is hidden in another signal, and what the time shift is with respect to the other data set. The maximum of the cross-correlation function is especially interesting: its position indicates the exact time shift in the other data set. Its height indicates how similar the two signals are.

In order to be able to correctly interpret the signals' delay, it is essential to list the parameters of the CCF function in the correct order. The first parameter is the reference channel, which contains the original signal. The second parameters contains the test channel which contains a delayed (and often distorted) signal. The x-coordinate of the maximum in the cross correlation function indicates directly the time shift of the test channel with respect to the reference channel. In this implementation, cross-correlation is calculated using the FFT and iFFT functions to achieve a reasonable calculation time. Aside from saving computing time, this has important consequences:

First, the shorter of two data sets is extended with zeros so that the data sets have exactly the same length. Then both data sets are shortened to the next power of two points, if their length is not already a power of two. Use the Red2 function to avoid losing any data. Using the FFT function also means that the data set are treated as periodic, i.e. they are treated as if they were extended periodically in both xdirections. If a data set represents one impulse, the signal is interpreted as if a series of these impulses were contained.

The cross-correlation function itself is calculated for the complete length of the shortened, supposedly periodic waveforms. It is not of use to calculate the cross-correlation function for a different range, since the function is periodic and all ranges will show the same period. If a large shift of 0.9 periods results from a cross-correlation function, this means the same as a small shift in the negative direction, i.e. -0.1 periods.

The window function set for FFT is used for cross-correlation. Following any necessary truncation, the channels to be correlated are weighted according to the selected FFT window function. Use this to suppress edge effects and large signal jumps introduced by the periodic extension of the signals. See the FFT function description for more detailed information. If window functions are not desired, select the rectangular window function.

- The x-scaling of the CCF is the scaling of the specified waveforms; thus the two specified data sets should have the same x-scaling. Otherwise, a warning message is generated and the scaling of the reference channel is used.
- Because it is normalized, the CCF function has no y-unit.
- If the length of a parameter exceeds 2^27, an error message is generated and calculation is not possible. Shorten the relevant data set using the Leng or Red2 functions. The maximum length of a data set to be processed is 134.217.728.
- If the length of a data set is not a power of two, an error message is usually generated. If necessary, the data set is then automatically truncated (cut off).
- The CCF function uses the FFT function internally. This requires temporary (working) memory; if insufficient memory is available, calculation is stopped.
- The CCF function is normalized, so that it is independent of the amplitudes of the specified data sets. One of the data sets can be multiplied by a constant before correlation without any influence on the result. If the data sets to be correlated have a high mean value (y-offset), it is better to subtract this mean value from the data sets before correlation, as the mean value would influence the result more than the actual signal. In fact, it would no longer be the cross correlation function but the cross covariance function which is calculated.
- The CCF implemented here is normalized to the product of the RMS values of the specified data sets, so that the created data set has no y-unit. The RMS is the square root of the ACF function at the zero position.

Examples:

```
NDccf = CCF(NDref, NDtest)
```

Simplest application of CCF: data sets are shortened to the next power of two points, if necessary.

```
NDccf= CCF(NDref-Mean(NDref),NDtest-Mean(NDtest))
```

Simple application of CCF: data sets are shortened to the next power of two points, if necessary. To allow better comparison of data sets with a high y-offset (mean value), the mean value is subtracted from both data sets before the CCF function is executed.

```
NDtest1 = Red2(NDtest)
NDref1 = RSamp(NDref, NDtest1)
NDccf = CCF(NDref1, NDtest1)
maxi = Max(NDccf)
```

It is to be checked whether a short reference signal (NwRef) can be found in a longer, slightly distorted data set NwTest. The longer data set is truncated to a power of two using the Red2 function, so that no values are cut off. The shorter data set is sampled using the function RSamp so that the sampling rates match exactly. These manipulated data sets are then correlated. If the maximum of the CCF is greater than 0.8, it can be concluded that the test signal significantly resembles the reference signal.)

See also:

ACF, CorrCoeff, FFT, Red2

CFCFilter

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Calculation of a CFC-filters as per SAE J211/1.

Declaration:

```
CFCFilter ( Signal, CFC_value, Filter start-up ) -> Result
```

Parameter:

Signal	Time-based data to be filtered
CFC_value	CFC-value, e.g. 60, 180, 600, 1000.
Filter start-up	Handling of filter start-up transients:
	0 : Polar symmetry around 1st value: The method suggested by SAE, of extending the signal in both directions with polar symmetry. The symmetry center is the respective endpoint value. (Data transposed about end points)
	1 : Polar symmetry around 0.0: The method suggested by SAE, of extending the signal in both directions with polar symmetry. The symmetry pole is then the point 0.0.(Data transposed about zero magnitude)
	2 : Constant 0.0: The signal is extrapolated beyond its margins with a value of zero.
	3 : Axially symmetric across the end value: the signal is mirrored across an axis given by x = end value.
Result	
Result	Filtered waveform

Description:

The CFC-filter is a digital, non-causal filter. this filter is applied to the time-based signal. The filter has a smoothing effect, but attempts to avoid displacing the edges in time.

Algorithm as per the standard SAE J211/1, March 1995.

On the basis of the CFC-value, a low-pass filter with the following cutoff frequency is formed:

- fg = 2.0775 * CFC_Value

The filter's cutoff frequency must be below half of the sampling frequency in order for the filter to be calculated.

The filter only begins to work properly (have a truly smoothing effect) once the CFC-value is well below 1/8 of the sampling frequency, and preferably far lower.

Examples:

```
Smoother = CFCFilter ( Signal, 600, 0 )
```

A signal is smoothed with a CFC-value of 600. The transient effect as per SAE is used.

See also:

FilterAnalog, FiltLP, dFilt, Smo

CharToSv

Returns the ASCII code from first character in a text.

Declaration:

```
CharToSv ( TxText ) -> SvCode
```

Parameter:

TxText	The text whose first character's ASCII-code is to be determined
SvCode	
SvCode	ASCII-code of the character

Description:

A text is specified as the parameter. The ASCII-code of the text's first character is determined and returned.

- The ASCII-codes permitted are 1..255.
- For an empty text, a 0 is returned.
- For example, tabulator character's code = 9, carriage return = 13, line feed = 10.
- Not all ASCII codes can be displayed in Windows. Above all be sure that ANSI character set is used. ASCII tables made for the DOS environment display the characters in the OEM character set.

Examples:

The ASCII codes for a lower-case and an upper-case letter are determined:

```
aLower = CharToSv("a")
aUpper = CharToSv("A")
```

See also:

SvToChar, TtoSv, TReplace, TConv

Chrct

Correction by characteristic curve

Declaration:

```
Chrct ( Data, CharacteristicData ) -> Corrected
```

Parameter:

Data	Waveform with raw data to be corrected using characteristic curve Allowed data types: [ND],[XY]
CharacteristicData	Characteristic data set Allowed data types: [ND],[XY]
Corrected	
Corrected	Corrected data set

Description:

This function can be used for correcting the data values in a data set with a characteristic (e.g. from a sensor). If a raw data value lies between two characteristic values, linear interpolation is performed.

- The characteristic curve data set may contain a maximum of 8388608 values.
- If the characteristic curve data set is of the type XY, the X-track must increase monotonically.
- The correction data set may be structured (events/segments).
- Because linear interpolation is performed between the two characteristic values, approximation errors should be reduced by using sufficiently large characteristic data sets, especially for non-linear characteristics.

Examples:

A data set [sensor_data] is corrected according to a characteristic curve data set.

```
sensor_corr = Chrct(sensor_data, characteristic)
```

See also:

LFit, eFit, MatrixGet

Clip

Imposes a boundary on the Y-value range to a specified band.

Declaration:

```
Clip ( Data, SvTop, SvLow ) -> Result
```

Parameter:

Data	Data set to be clipped; allowed types: [ND],[XY].
SvTop	Upper boundary
SvLow	Lower boundary
Result	
Result	Clipped data set. Y-value range lies between [SvBottom] and [SvTop]

Description:

This function clips all Y-values in the data set between down to the value range specified by the second and third parameters. Any values which exceed these thresholds are set to the respective limit value. The upper limit is always the greater value of the second and third parameter, so when calling the function it isn't necessary to ensure the matching of parameter to respective limit.

- The first parameter may be structured (events/segments).
- The units of the single values should match those of the data set, but are not checked, which makes calling the function particularly convenient when numerical values are used instead of variables.
- Band-limiting in the sense applicable to frequencies is not performed.

Examples:

Data are limited to a range between positive and negative supply voltage to simulate the saturation effect:

```
NDsaturated = Clip(NDdata, Uplus, Uminus)
```

See also:

Cut, STri, Scale

CLIPBOARD

Copies curve window to the Windows Clipboard
This command is obsolete; instead of it, the more powerful function CwAction("Clipboard.Copy") should be used.

Declaration:

```
CLIPBOARD VariableName
```

Parameter:

VariableName	Variable whose curve graph is to be copied to the Windows Clipboard.

Description

With this command, a curve graph is copied to the Clipboard. From there, it can be pasted into other Windows applications (DTP, text processing, graphics program).

Examples:

```
CLIPBOARD slope
```

The curve-graph of the variable "slope" is copied to the Clipboard. If the data set wasn't yet displayed, then it will first be displayed in accordance with the valid presettings, then copied and the curve window subsequently deleted,

See also:

CwAction

ClsLvlCr

Available in: Enterprise Edition and above (ClassCounting-Kit)

Classification by level crossing counting

Declaration:

```
ClsLvlCr ( Channel, MaxValue, MinValue, Number of bins, Reference, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Reference	Reference level
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : automatic reference level
	3 : open-ended outer bins, automatic reference level
Result	
Result	Distribution

Description:

Level-crossing procedure as per DIN 45667.

In the level-crossing-counting method, the tally increases in classes whose upper levels are at or above the reference line whenever the signal crosses the upper level from below.

The tally increases in classes whose upper levels are below the reference line when the signal crosses the upper level from above.

Examples:

```
Distribution = ClsLvlCr( Data, 3, -3, 12, 0, 0.25, 1 )
```

See also:

ClsTimeAtLevel

ClsMaxSt

Available in: Enterprise Edition and above (ClassCounting-Kit)

Classification by the Maximum value storage

Declaration:

```
ClsMaxSt ( Channel, MaxValue, MinValue, Number of bins, Distance between samples, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Distance between samples	
Options	
	0 : no options
	1 : open-ended outer bins
Result	
Result	Distribution

Description:

Class-counting of measured data by the maximum-value-storage method as per DIN 45667.

Measured data are classed according to the maximum-value-storage method, which is a variation of the sampling method.

The largest sample value occurring within a given time interval is found and tallied in a bin.

In the Class-counting Kit, the maximum value is the highest valued sample among the number of samples given by the parameter SvDistance (sampling distance).

Examples:

```
Distribution  = ClsMaxSt( Data, 12, 0, 12, 5, 1 )
```

See also:

ClsTimeAtLevel

ClsOff2ChannelHistogram

Available in: Enterprise Edition and above (ClassCounting-Kit)

Calculation of the mutual density, 2D histogram.

Declaration:

```
ClsOff2ChannelHistogram ( Channel1, Channel2, Min1, Max1, Classes1, Min2, Max2, Classes2, Options ) -> Result
```

Parameter:

Channel1	The first input channel. Its classes are plotted along the matrix' x-axis, in other words, its classes are arrayed in a column.
Channel2	The second input channel. Its classes are plotted along the matrix' z-axis, in other words, its classes are arrayed in a row.
Min1	Minimum of range of 1st channel
Max1	Maximum of range of 1st channel
Classes1	Number of bins for 1st channel
Min2	Minimum of range of 2nd channel
Max2	Maximum of range of 2nd channel
Classes2	Number of bins for 2nd channel
Options	
	0 : Closed outer bins, axes labelled by class
	1 : Open-ended outer bins, axes labelled by class
	2 : Closed outer bins, use physical units
	3 : Open-ended outer bins, use physical units
Result	
Result	

Description:

The values in Channel1 and Channel2 are allotted to classes defined by the specified number of classes and the limits of the input range.

Each matrix cell represents a combination of a Channel1 class and a Channel2 class, and contains a tally of such combinations which occur.

For each pairing of one measured value from Channel1 and one from Channel2, the tally in a matrix' cell is raised by 1.

The outer bins (classes) can be open-ended or closed.

In the case of open-ended outer bins, the values outside of the range Min..Max are included in the outer bins.

If the outer bins are closed, values outside of the range Min..Max are not counted.

The labelling of the axes (x-axis and z-axis) can either refer to classes (Class 0, Class 1, Class 2, ...) or to the input channels' physical units (e.g., -0.1 Nm .. +0.1 Nm).

Min1, Max1, Classes1: The first channel's input range. The bins (classes) extend from Min1 (<Max1) .. Max1 and their number is given as the parameter Classes1 (>=4).

Min2, Max2, Classes2: The second channel's input range. The bins (classes) extend from Min2 (<Max2) .. Max2 and their number is given as the parameter Classes2 (>=4).

Examples:

```
Min1 = -0.1
Max1 = 0.1
Classes1 = 30
Min2 = -10.0
Max2 = 10.0
Classes2 = 40
Options = 3
MutualDensity = ClsOff2ChannelHistogram ( a1, a2, Min1, Max1, Classes1, Min2, Max2, Classes2, Options)
```

Mutual density class-counting of two channels a1 and a2.

The first channel is divided into 30 classes extending through -0.1 .. 0.1, the second channel into 40 classes through -10.0 .. +10.0.

The outer bins are open-ended. The matrix axes are labelled in terms of physical units.

See also:

ClsTimeAtLevel, ClsOffRevolutionsHistogram, ClsOffRevolutionsMatrix

ClsOffFromRainflowGetLevelCrossing

Available in: Enterprise Edition and above (ClassCounting-Kit)

level-crossing-count from Rainflow-matrix

Declaration:

```
ClsOffFromRainflowGetLevelCrossing ( ClsHandle, Reference ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Reference	signal amplitude which is to serve as the reference line (zero-line).
Result	
Result	Distribution

Description:

A level-crossing-count is determined from the current Rainflow-matrix and the current residue.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix)
ClsOffRainflowFeedResidue ( ClsHandle, Residue, 1.0)
_Reference = 0.0 ; reference level used for various DIN-based procedures
DurchgangOff = ClsOffFromRainflowGetLevelCrossing( ClsHandle, _Bezug)
```

The Rainflow-analysis is first initalised.

An already existing matrix, togther with its residue, is fed in.

The DIN-based procedure is performed on the basis of these data.

See also:

ClsOffRainflowInit1, ClsLvlCr, ClsOffFromRainflowGetRangePair

ClsOffFromRainflowGetMinMaxPeak

Available in: Enterprise Edition and above (ClassCounting-Kit)

Peak-count (Method 3) from Rainflow-matrix

Declaration:

```
ClsOffFromRainflowGetMinMaxPeak ( ClsHandle, Min_Or_Max ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Min_Or_Max	What shall be counted?
	0 : All minima are class-counted.
	1 : All maxima are class-counted.
Result	
Result	Distribution

Description:

A peak-count (Method 3) is determined from the current Rainflow-matrix and the current residue.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix)
ClsOffRainflowFeedResidue ( ClsHandle, Residue, 1.0)
Peak3PosOff = ClsOffFromRainflowGetMinMaxPeak( ClsHandle, 1)
Peak3NegOff = ClsOffFromRainflowGetMinMaxPeak( ClsHandle, 0)
```

The Rainflow-analysis is first initalised.

An already existing matrix, togther with its residue, is fed in.

The DIN-based procedure is performed on the basis of these data.

The results for the minima and for the maxima are determined separately.

See also:

ClsOffRainflowInit1, ClsPeak3, ClsOffFromRainflowGetZeroCrossingPeak, ClsOffFromRainflowGetLevelCrossing

ClsOffFromRainflowGetPeak

Available in: Enterprise Edition and above (ClassCounting-Kit)

Peak-count (Method 2) from Rainflow-matrix

Declaration:

`ClsOffFromRainflowGetPeak ( ClsHandle, Reference ) -> Result`

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Reference	signal amplitude which is to serve as the reference line (zero-line).
Result	
Result	Distribution

Description:

A peak-count (Method 2) is determined from the current Rainflow-matrix and the current residue.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix)
ClsOffRainflowFeedResidue ( ClsHandle, Residue, 1.0)
_Reference = 0.0 ; reference level used for various DIN-based procedures
Peak2Off = ClsOffFromRainflowGetPeak( ClsHandle, _Reference)
```

The Rainflow-analysis is first initalised.

An already existing matrix, togther with its residue, is fed in.

The DIN-based procedure is performed on the basis of these data.

See also:

ClsOffRainflowInit1, ClsPeak2, ClsOffFromRainflowGetZeroCrossingPeak, ClsOffFromRainflowGetLevelCrossing

ClsOffFromRainflowGetRangePair

Available in: Enterprise Edition and above (ClassCounting-Kit)

Range-pair-count from Rainflow-matrix

Declaration:

```
ClsOffFromRainflowGetRangePair ( ClsHandle ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Result	
Result	Distribution

Description:

A range-pair-count is determined from the current Rainflow-matrix and the current residue.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix)
ClsOffRainflowFeedResidue ( ClsHandle, Residue, 1.0)
RangePairOff = ClsOffFromRainflowGetRangePair( ClsHandle )
```

The Rainflow-analysis is first initalised.

An already existing matrix, togther with its residue, is fed in.

The DIN-based procedure is performed on the basis of these data.

See also:

ClsOffRainflowInit1, ClsRngPr, ClsOffFromRainflowGetLevelCrossing

ClsOffFromRainflowGetReconstruction

Available in: Enterprise Edition and above (ClassCounting-Kit)

Reconstruction of time-based signal from Rainflow-matrix

Declaration:

```
ClsOffFromRainflowGetReconstruction ( ClsHandle, EdgeValue, PosMin, PosMax, Stretch, SV_1, SV_2, SV_3 ) ->
Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
EdgeValue	This is the value at the beginning and at the end of the signal. If a number >= 1e30 is specified, then no supplementary value will be added. Caution: adding supplementary values can cause the class-count to be changed significantly, since under some circumstances new extrema can result.
PosMin	the relative location of a relative minimum between the bin (class) limits. A value between 0.0 and 1.0. For instance these values: 0.0 the local minimum is positioned at the lower limit of the bin. 1.0 the local minimum is positioned at the upper limit. 0.5 in the exact center of the bin. When in doubt, set to 0.5. Values near 0.0 and 1.0 are risky, because the values might be counted in an adjacient bin in the process of any subsequent class-count. For a minimum, the value could well be set at 0.1. This makes the ranges somewhat larger. If a small value for the hysteresis opens the possibility of the starting and the target classes of an half-cycle being the same, then PosMax should be set as > PosMin.
PosMax	as PosMin, but for a relative maximum. When in doubt, set to 0.5. The value 0.9 can also be used, in order to differentiate from a minimum located in the same class.
Stretch	a local extremum in the result waveform is lengthened by this many data points. No lengthening for the setting 0. In that case, an extremum is one sample in size. When in doubt, set to 1. If a signal reversal (extremum) is only of short duration in the signal, then the procedure might miss it. By lengthening the reversal this is avoided. 0 <= ExStretch <= 100
SV_1	Reserved. = 0
SV_2	Reserved. = 0
SV_3	Reserved. = 0
Result	
Result	Reconstruction

Description:

A time-based signal which would represent similar stresses on the examined system is reconstructed from the current Rainflow-matrix and the current residue.

Class-counting the reconstructed signal with an hysteresis value of 0 would return the identical matrix and residue.

The signal course in time can deviate.

The function only produces correct results if the matrix and the residue are derived from the same class-count. If one of the two was unilaterally changed, the result will also be undefined.

The exact amount of cycles recorded in the matrix will be reconstructed. The amount can be too large to fit in a result-waveform.

In that case, the values in the matrix should be reduced by dividing them by a fixed factor > 1.

The residue should be available separately rather than already being tallied into the matrix.

Under certain circumstances, the residue of the result will differ from the original one.

In some cases, the edge value can cause extra ranges to be identified and counted.

This is because the linkup with the edge value can require an additional extremum in order that adjoining extrema are recognized as such.

An attempt is made to preserve the orientation of ranges. I.e., a range extending from Bin (Class) 5 to Bin 7 is different from one which goes from Bin 7 to Bin 5. However, this is not always possible to accomplish, especially if both directions occur.

Reconstruction is returned as the reconstructed waveform. The sampling rate and units must be set subsequently.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix)
ClsOffRainflowFeedResidue ( ClsHandle, Residue, 1.0)
_PosMin = 0.2 ; 0 .. 1, standard 0.5
_PosMax = 0.8 ; 0 .. 1, standard 0.5
_Stretch = 0
_EdgeVal = 1e30 ; 1e30 if not desired. Otherwise, the value of EdgeVal
```

```
Reconstruction = ClsOffFromRainflowGetReconstruction ( ClsHandle, _EdgeVal, _PosMin, _PosMax, _Stretch, 0, 0, 0)
yUnit Reconstruction Nm
xdelta Reconstruction 0.001
```

A Rainflow analysis performed earlier has produced a Rainflow-matrix and residue.

Both are entered into the reconstructing function.

A compact, reconstructed time-based waveform is returned.

See also:

ClsOffRainflowInit1

ClsOffFromRainflowGetSpans

Available in: Enterprise Edition and above (ClassCounting-Kit)

Number of occurences of spans from the Rainflow matrix. For each class (level of span) the total number of cycles will be determined.

Declaration:

```
ClsOffFromRainflowGetSpans ( ClsHandle ) -> Span count
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Span count	
Span count	Span count

Description:

The function determines for each span (amplitude) the total number of cycles. The information is extracted from the Rainflow matrix.

A vector is constructed from the Rainflow matrix.

With Amplitude-Mean counting, all cycles that have been counted for a certain amplitude are added together. The information on the mean value is lost. The x-axis is scaled as amplitudes.

With Span-Mean counting, all cycles that have been counted for a certain span are added up. The information on the mean value is lost. The x-axis is scaled as spans.

With start- and target class counting, the span is the difference between start and target class. For each span, all cycles from the matrix are added up. The information on the slope is lost. The x-axis is scaled as spans.

The number of cycles actually measured is determined from the Rainflow-matrix. The residue is not taken into account. If you wish it to be taken into account, it must first be included in the matrix count.

If classification by physical units is selected with ClsOffRainflowInit1 ( ), then the x-axis will be scaled in physical units. Otherwise in classes.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
ClsOffRainflowAddResidue ( ClsHandle, 1 )
Spans = ClsOffFromRainflowGetSpans ( ClsHandle )
```

The Rainflow-analysis is first initalised.

A Rainflow counting will be performed. The number of spans for each class shall be determined.

See also:

ClsOffRainflowInit1, ClsOffRainflowSetMatrix, ClsOffRainflowFeedSamples

ClsOffFromRainflowGetZeroCrossingPeak

Available in: Enterprise Edition and above (ClassCounting-Kit)

Zero-crossing-peak-count from Rainflow-matrix

Declaration:

```
ClsOffFromRainflowGetZeroCrossingPeak ( ClsHandle, Reference ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Reference	signal amplitude which is to serve as the reference line (zero-line).
Result	
Result	Distribution

Description:

A zero-crossing-peak-count is determined from the current Rainflow-matrix and the current residue.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix)
ClsOffRainflowFeedResidue ( ClsHandle, Residue, 1.0)
_Reference = 0.0 ; reference level used for various DIN-based procedures
Peak1Off = ClsOffFromRainflowGetZeroCrossingPeak( ClsHandle, _Reference)
```

The Rainflow-analysis is first initalised.

An already existing matrix, togther with its residue, is fed in.

The DIN-based procedure is performed on the basis of these data.

See also:

ClsOffRainflowInit1, ClsPeak1, ClsOffFromRainflowGetPeak, ClsOffFromRainflowGetLevelCrossing

ClsOffMarkov

Available in: Enterprise Edition and above (ClassCounting-Kit)

Transition counting; calculation of the Markov matrix

Declaration:

```
ClsOffMarkov ( Input data, Min, Max, Classes, Hysteresis, UnitOption ) -> Result
```

Parameter:

Input data	Data set of input data, from which the Markov-matrix is calculated
Min	Range minimum
Max	Range maximum
Classes	Bin count
Hysteresis	Hysteresis for suppressing small oscillations; stated in physical units
UnitOption	
	0 : axes labelled by class
	1 : use physical units
	2 : Axes labelled by class, small spans within one class are counted.
Result	
Result	Markov matrix

Description:

The transition from a start class to a target class is calculated for each span between two neighboring extreme values. The extreme values are assigned to classes. All classes have the same size. Their width is determined by the range (Min, Max) and the bin count.

An alternative name for the Markov matrix is transition matrix.

The result is a segmented waveform representing an N * N matrix, where N is the number of classes.

The segments of the result represent the columns of the matrix. The first segment corresponds to the first column.

To each column, a start class is assigned; to each row a target class. E.g. the first column (first segment) contains all transitions from the first start class to any target class.

The x-coordinate of the segmented waveform represents the target class. The z-coordinate represents the start class.

Rising spans are counted into the lower triangular matrix. For rising spans the row index is greater than the column index. Falling spans are counted into the upper triangular matrix.

The diagonal of the matrix always contains only zeroes with UnitOption =1 or =2.

If extrema are located outside of the range, they will be limited to the bounding values. This is equivalent to open-ended outer bins.

The hysteresis is specified in physical units of the input data. A hysteresis of about one class width makes sense.

Examples:

```
MarkovMatrix = ClsOffMarkov ( Data, 0.0, 1000.0, 100, 10.0, 1 )
```

The range from 0.0 to 1000.0 is divided into 100 classes. Each has a width of 10.0. A hysteresis of 10.0 is used which equals 100% of the class width.

These rising spans have been counted: 4 from class 1 to 2, 5 from 1 to 3 and 2 from 2 to 3. Furthermore, these falling spans have been counted: 6 from 2 to 1, 3 from 3 to 1 and 1 from 3 to 2:

Input matrix:

0 6 3

4 0 1

5 2 0

Segment 1 = Column 1: 0 4 5

Segment 2 = Column 2: 6 0 2

Segment 3 = Column 3: 3 1 0

E.g.. MarkovMatrix[1,3] = 5. Thus 5 transitions from 1 to 3.

ClsOffMatrixSum1Triangle

Available in: Enterprise Edition and above (ClassCounting-Kit)

makes triangle matrix

Declaration:

```
ClsOffMatrixSum1Triangle ( Matrix, Uo ) -> Result
```

Parameter:

Matrix	Matrix, e.g.. Rainflow matrix
Uo	Lower or upper triangle matrix
	0 : The upper triangle matrix (above the main diagonal) is filled. Only the first cell of the first column (1st segment of the resulting waveform) is filled. The last column is completely filled.
	1 : the lower triangle matrix (below the main diagonal) is filled. The matrix' first coumn (1st segment of the resulting waveform) is then complete. The last column has only its last cell filled.
Result	
Result	TriangleMatrix

Description:

A triangle matrix is produced by taking a square matrix, subtracting the value in each of its cells on one side of the main diagonal (thus leaving zero in those cells) and adding each such value to the value in the corresponding cell on the other side.

This function is mainly used on Rainflow matrices, in which the columns and rows represent the starting-class and the target-class, respectively. After the function has been executed, the direction of the signal range is no longer recognizable.

Algorithm:

For all i > k

x[i,k] := x[i,k] + x[k,i]

x[k,i] := 0

This function can only be applied to square matrices.

Examples:

```
RainflowTriangle = ClsOffMatrixSum1Triangle ( RainflowMatrix, 1 )
```

All cycles are tallied into the lower triangle.

The diagonal remains unchanged.

Input matrix:

2 2 2

4 1 6

5 2 3

Segment1 = Column 1: 2 4 5

Segment2 = Column 2: 2 1 2

Segment3 = Column 3: 2 6 3

Output matrix:

2 0 0

6 1 0

7 8 3

Segment1 = Column 1: 2 6 7

Segment2 = Column 2: 0 1 8

Segment3 = Column 3: 0 0 3

See also:

ClsOffRainflowGetMatrix, MatrixSumLines

ClsOffMatrixSumLines

Available in: Enterprise Edition and above (ClassCounting-Kit)

Sum of the contents of a columns or rows
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and its predecessors. Please use instead the function MatrixSumLines().

Declaration:

```
ClsOffMatrixSumLines ( Matrix, Rows ) -> Result
```

Parameter:

Matrix	Matrix, e.g.. Rainflow matrix
Rows	Sum of the contents of a columns or rows
	0 : rows. Sum of the contents of a row. The resulting vector's first row contains the sum of all input segments' first elements.
	1 : columns. Sum of the contents of a column, in other words, of one segment's elements.
Result	
Result	Sum

Description:

The function produces a vector consisting of the sums of either the matrix' rows or its columns.

The matrix does not need to be a square matrix.

Examples:

```
ColumnSum = ClsOffMatrixSumLines ( Matrix, 1 )
```

Input matrix:

2 1 0

4 1 1

5 2 0

Segment 1 = Column 1: 2 4 5

Segment 2 = Column 2: 1 1 2

Segment 3 = Column 3: 0 1 0

This vector is the result:

(11, 4, 1)

Each element is the sum of one matrix column's cells.

```
RowSum = ClsOffMatrixSumLines ( Matrix, 0 )
```

this returns (3, 6, 7).

See also:

MatrixSumLines

ClsOffRainflowAddResidue

Available in: Enterprise Edition and above (ClassCounting-Kit)

The current residue is tallied into the matrix and subsequently emptied.

Declaration:

```
ClsOffRainflowAddResidue ( ClsHandle, WeightingFactor )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
WeightingFactor	0... 1: When in doubt, set to 1. The function will not increment the count in the matrix cells by 1, but rather by the factor specified. This allows the residue to make a weakened impact on the class-count.

Description:

The residue is tallied into the matrix. However, each 'oscillation' of the 'signal' is weighted by the specified WeightingFactor.

The weighting factor will be used to increment the corresponding places in the matrix.

WeightingFactor = 0.0: the matrix remains unchanged

WeightingFactor = 1.0: worst case scenario. Every 'oscillation' is counted at full strength. The result of this is that uncompleted cycles will be entered into the tally as if they had been completed.

WeightingFactor = 0.5: A compromise. The ranges in the residue are interpreted as half-cycles experienced by the physical system.

The WeightingFactor can be set arbitrarily, typicallly within the range 0...1.

In order to tally the residue into the matrix, it is fed into the matrix as if it were a time-plotted waveform.

To make sure that the complete residue is counted, it may be necessary to insert intermediary values into it.

A residue of length 1 is ignored.

Afterwards, the residue is emptied.

A discontinuity is inserted at the beginning and at the end.

With the ASTM E1049 option this function must be called! The residue is counted into the matrix using a weighting of 0.5. The option "WeightingFactor" is ignored and can be set to 0.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
ClsOffRainflowAddResidue ( ClsHandle, 1 )
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
```

A Rainflow-count is performed on the waveform 'data_chan1'.

After all range-pairs have been tallied into the matrix, a residue is left. The residue is not to be stated separately, rather it is to be included in the matrix.

So some of the matrix cells are incremented.

Afterwards, the residue is empty and its data no longer available.

See also:

ClsOffRainflowInit1, ClsOffRainflowFeedResidue, ClsOffRainflowGetResidue, ClsOffRainflowFeedMatrix

ClsOffRainflowClearMatrix

Available in: Enterprise Edition and above (ClassCounting-Kit)

The current matrix is set to zero.

Declaration:

```
ClsOffRainflowClearMatrix ( ClsHandle )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()

Description:

All previously counted values are deleted. The current residue remains unchanged.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
 ... ; additional operations
ClsOffRainflowClearMatrix ( ClsHandle )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix2)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
```

The current matrix is emptied using *ClearMatrix(). Afterwards, another matrix is added to it using *FeedMatrix().

See also:

ClsOffRainflowInit1, ClsOffRainflowSetMatrix, ClsOffRainflowClearResidue

ClsOffRainflowClearResidue

Available in: Enterprise Edition and above (ClassCounting-Kit)

The current residue is emptied.

Declaration:

`ClsOffRainflowClearResidue ( ClsHandle )`

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()

Description:

Rather than being tallied into the matrix, it's contents are deleted and lost.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
ClsOffRainflowClearResidue ( ClsHandle )
...
```

The measured values of data_chan1 are tallied into a matrix.

The resulting residue is then deleted.

Afterwards, other operations are performed to which the residue was not intended to contribute.

See also:

ClsOffRainflowInit1, ClsOffRainflowGetResidue, ClsOffRainflowFeedResidue, ClsOffRainflowAddResidue, ClsOffRainflowClearMatrix

ClsOffRainflowFeedDiscontinuity

Available in: Enterprise Edition and above (ClassCounting-Kit)

A discontinuity is inserted.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and its predecessors.

Declaration:

```
ClsOffRainflowFeedDiscontinuity ( ClsHandle )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()

Description:

Measured values fed in subsequently are regarded as data from a new measurement which is not to be directly appended to the other data sets.

See also:

ClsOffRainflowFeedSamples

ClsOffRainflowFeedMatrix

Available in: Enterprise Edition and above (ClassCounting-Kit)

The current matrix' count is increased by the values of the paramter Matrix. The current residue remains unchanged.

Declaration:

```
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Matrix	The added matrix must have the same dimensions as the internal, current matrix.

Description:

The internal matrix is incremented value by value.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix1)
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix2)
ClsOffRainflowFeedResidue ( ClsHandle, Residue2, 1.0)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
Residue = ClsOffRainflowGetResidue ( ClsHandle )
```

Various class-counting results are already available for a certain channel. The Rainflow-matrix and residue of the old values is always available.

The matrices and the residues are all fed to the function. Everything is counted together.

Where there are residues, they are also fed in. In this case, there was only a residue for the 2nd measurement.

The total mechanical stress is then obtained.

See also:

ClsOffRainflowInit1, ClsOffRainflowFeedResidue, ClsOffRainflowSetMatrix

ClsOffRainflowFeedResidue

Available in: Enterprise Edition and above (ClassCounting-Kit)

A residue which was previously determined is entered into the current matrix. This alters both the current matrix and the current residue.

Declaration:

```
ClsOffRainflowFeedResidue ( ClsHandle, Residue, WeightingFactor )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Residue	the waveform of the residue
WeightingFactor	0... 1: When in doubt, set to 1. The function will not increment the count in the matrix cells by 1, but rather by the factor specified. This allows the residue to make a weakened impact on the class-count.

Description:

Note: The hysteresis is ignored if "_Precise" < 6. The y-scaling of the residue is interpreted according to the value of UnitUse in the function ClsOffRainflowInit1().

The residue including any necessary artificial intermediate values (applicable if "Precise" < 5) is fed to the function. If the residue internally available is empty, no intermediate values are inserted.

If the residue is expressed in terms of physical units, class-midpoints should be used. In fact, where the residue takes physical units, the bins shift position downward by 1/1000 of a class width if "_Precise" < 5.

The residue must be valid, i.e., its values must all lie within the permitted range, neighboring values must be located in different classes. Invalid residues will produce unexpected results! A residue of length 1 is ignored, if "_Precise" < 6.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix1)
ClsOffRainflowFeedResidue ( ClsHandle, Residue1, 1.0)
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix2)
ClsOffRainflowFeedResidue ( ClsHandle, Residue2, 1.0)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
Residue = ClsOffRainflowGetResidue ( ClsHandle )
```

Various class-counting results are already available for a certain channel. The Rainflow-matrix and residue of the old values is always available. The matrices and the residues are all fed to the function. Everything is counted together. The total mechanical stress is then obtained.

See also:

ClsOffRainflowInit1, ClsOffRainflowFeedMatrix, ClsOffRainflowGetResidue, ClsOffRainflowFeedSamples

ClsOffRainflowFeedSamples

Available in: Enterprise Edition and above (ClassCounting-Kit)

New measured data are tallied into the matrix. The matrix and the residue are updated accordingly.

Declaration:

```
ClsOffRainflowFeedSamples ( ClsHandle, Samples )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Samples	one or more new measurement values; load vs. time; load spectrum

Description:

It is especially effective to first join multiple measurement values to a single waveform in order to apply this function to them.

The function ClsOffTM() can be used to improve precision.

The new measurement values are interpreted as if they are appended directly onto the most recent value. This ensures that extrema located at the transition point are detected.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_test1)
ClsOffRainflowFeedSamples ( ClsHandle, data_test2)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
delete ClsHandle
```

A Rainflow-analysis is initialised and performed. The measurement files were separate, because each contained only the data collected in 1 hour. Each of the separate waveforms is appended and counted.

See also:

ClsOffRainflowInit1, ClsOffRainflowInit2, ClsOffRainflowInit3, ClsOffRainflowGetMatrix, ClsOffTM

ClsOffRainflowGetMatrix

Available in: Enterprise Edition and above (ClassCounting-Kit)

Returns the current matrix.

Declaration:

```
ClsOffRainflowGetMatrix ( ClsHandle ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Result	
Result	Matrix

Description:

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
ClsOffRainflowAddResidue ( ClsHandle, 1 )
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
```

A Rainflow-count is conducted.

The resulting matrix is then collected.

Since no separate residue is called for here, the residue was first tallied into the current matrix.

See also:

ClsOffRainflowInit1, ClsOffRainflowSetMatrix, ClsOffRainflowGetResidue

ClsOffRainflowGetResidue

Available in: Enterprise Edition and above (ClassCounting-Kit)

Returns the current residue.

Declaration:

```
ClsOffRainflowGetResidue ( ClsHandle ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Result	
Result	Residue

Description:

If the residue takes physical units, then bin (class)-midpoints are returned.

Otherwise, the residue is expressed in classes (0, 1, 2, ...).

The residue's x-axis has no significance. The values are simply arrayed along it.

Two adjacent residue values are denoted as a span (range).

With Precise = 4 the values of the residue are not rounded. They can have any values between the boundaries.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
Residue = ClsOffRainflowGetResidue ( ClsHandle )
```

A Rainflow-count is performed on the waveform 'data_chan1'.

The matrix and residue are determined.

The residue is not tallied into the matrix, instead it is stated separately.

The residue is expressed as a vector in which all remaining signal extrema are recorded. The ranges between the extrema can be read off directly from a graphical representation.

See also:

ClsOffRainflowInit1, ClsOffRainflowGetResidue, ClsOffRainflowGetResidueMtx, ClsOffRainflowFeedResidue, ClsOffRainflowAddResidue

ClsOffRainflowGetResidueMtx

Available in: Enterprise Edition and above (ClassCounting-Kit)

The current residue is returned in matrix format.

Declaration:

```
ClsOffRainflowGetResidueMtx ( ClsHandle ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Result	
Result	Matrix

Description:

The returned matrix is structured in the same manner as a class-count matrix.

In this case, however, the tallies in the matrix cells state the number of occurences in the residue of a particular range.

The value, then, is 0 if the corresponding range does not appear in the residue.

And the value is 1 if, it appears once.

'n' for 'n' times etc.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
ResidueMtx = ClsOffRainflowGetResidueMtx ( ClsHandle )
```

A Rainflow-count is performed on the waveform 'data_chan1'.

The matrix and residue are determined.

The residue is not tallied into the matrix, instead it is stated separately.

The residue is returned as a matrix which records the ranges it contains.

See also:

ClsOffRainflowInit1, ClsOffRainflowGetResidue, ClsOffRainflowFeedResidue, ClsOffRainflowAddResidue, ClsOffRainflowGetMatrix

ClsOffRainflowInit1

Available in: Enterprise Edition and above (ClassCounting-Kit)

Initialises a Rainflow-count. Always to be called prior to the Rainflow function.

Declaration:

```
ClsOffRainflowInit1 ( Classes, UnitUse, UnitColumn, UnitRow, UnitCounter, Unit_Y_Residue, SV_0 ) -> Result
```

Parameter:

Classes	the number of classes, e.g. 32 or 64. >= 4 and <= 1000
UnitUse	This determines how the matrix and other results class widths and the residue values are scaled.
	0 : scaling by classes (0, 1, ...)
	1 : scaling by the physical units of the input data
UnitColumn	This unit is marked along the x-direction of the matrix. It characterizes the elements arrayed along a column.
UnitRow	This unit is marked along the z-direction of the matrix. It characterizes the elements arrayed along a row.
UnitCounter	the unit for the matrix y-axis, which counts the number of oscillations.
Unit_Y_Residue	The unit for the residue's y-axis. If other counting methods are derived from the Rainflow-matrix, this unit is used for the x-axis.
SV_0	always set to 0
Result	
Result	The return value is used in all other ClsOffRainflow* functions and may not be changed. However, it can be changed by the function itself.

Description:

All units can also be set as empty strings.

Examples:

```
_NumberClasses = 50
_TypeOfUnit = 1 ; 0 Classes, 1 physical unit
_UnitX = "Mean [Nm]"
_UnitZ = "Span [Nm]"
_UnitCount = "Count"
_UnitRes = "ResClasses"
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitX, _UnitZ, _UnitCount, _UnitRes, 0)
_Min = -5
_Max = 5
_Hysteresis = ( _Max - _Min ) / _NumberClasses
_Axis = 1 ; 0 x is dest class or Amplitude, 1 x is start class or mean
_Type = 2 ; 0 start and target class, 1 amplitude and mean, 2 span and mean
_CalcOptions = 0 ; 0 base algorithm, 1 Clormann Seeger correction, 2 ASTM E1049, 3 HCM
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
delete ClsHandle
```

A Rainflow analysis is initialized and performed.

See also:

ClsOffRainflowInit2, ClsOffRainflowInit3

ClsOffRainflowInit2

Available in: Enterprise Edition and above (ClassCounting-Kit)

Continued initialisation of a Rainflow-count. Always use immediately after ClsOffRainflowInit1 (). Sets the current matrix to zero and empties the residue.

Declaration:

```
ClsOffRainflowInit2 ( ClsHandle, Min, Max, Hysteresis, Axes, Type, OuterBins, Calculation )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Min	Lower limit of input range
Max	Upper limit of the input range of the signal to be processed. This determines the matrix' input range. Max > Min. This range should be made to cover the input signal's whole amplitude range, or even provide extra 'room'. Example: Min = -1Nm and Max = +1Nm, with 32 classes, lead to very unwieldly lass widths, namely 2/32 Nm. If the class width is to take a handy value, then round off Min and Max somewhat, e.g. +- 1.28 Nm or +- 1.6Nm.
Hysteresis	hysteresis, scaled in physical units. Hysteresis value to be used for finding the extrema. Must be >= 0. The value of one class width is recommended. Set to zero when in doubt.
Axes	x-Axis use?
	0 : the target class or amplitude is marked along the columns (x-direction). Thus, the x-axis can be designated as the amplitude or target class. The index for an element within a column states which target class or amplitude is referred to. The mean or start class decides which column is referred to. A column of the matrix is a segment.
	1 : The mean or start class is marked along the columns (x-direction). The opposite case from '0'.
Type	specifies significance of the x- and z-axes.
	0 : start- and target class. This refers to the counting algorithm. The start class is the bin into which the first extreme value of the signal is alloted, the target class is the bin for the second extreme value. The cases of a signal sweep from Bin 10 to Bin 16 and from Bin 16 to Bin 10 are differentiated.
	1 : amplitude and mean. The mean is the midpoint of an oscillation, i.e. (start class + target class) / 2. The amplitude is the absolute value of (start class - target class)/ 2. Since there are equal amounts of mean-classes and amplitude-classes (square matrix), the resolution of the amplitudes is twice as good. The amplitude is half of the value of (peak-to-peak).
	2 : The mean is the midpoint of an oscillation, i.e. (start class + target class) / 2. The span is the absolute value of (start class - target class).
OuterBins	Treatment of outside values
	0 : Closed. Before the extrema are found, the signal is truncated to the range Min..Max.
	1 : Open: Before the extrema are found, the signal is truncated to the range Min..Max. Recommended
Calculation	Algorithm
	0 : standard-algorithm 4 points
	1 : Clormann Seeger correction. Gives consideration to the zero-level dependencies. Counts transitions from x[1] -> x[2], if 1) on of x[1] and x[2] is > 0 and one is < 0; 2) abs. value of (x[2]) <= abs. value of (x[1]); 3) abs. value of (x[1]) <= abs. value of (x[3]).
	2 : Calculation according to ASTM E1049, reapproved 1990.
	3 : RAINFLOW-HCM method. U.H. Clormann, T. Seeger; Stahlbau 3/1986

Description:

With the ASTM E1049 option, the sequence of numbers before step (6) will be treated as the residue. The function ClsOffRainflowAddResidue must be called to get the matrix according to the standard!

With the ASTM E1049 option, the residue is counted into the matrix using a weighting of 0.5. The option "Precise" must be 5 or higher.

With RAINFLOW-HCM, the option "Precise" must be 5 or higher.

Examples:

```
_NumberClasses = 50
_TypeOfUnit = 1 ; 0 Classes, 1 physical unit
_UnitX = "Mean [Nm]"
_UnitZ = "Span [Nm]"
_UnitCount = "Count"
_UnitRes = "ResClasses"
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitX, _UnitZ, _UnitCount, _UnitRes, 0)
```

```
_Min = -5
_Max = 5
_Hysteresis = ( _Max - _Min ) / _NumberClasses
_Axis = 1 ; 0 x is dest class or Amplitude, 1 x is start class or mean
_Type = 2 ; 0 start and target class, 1 amplitude and mean, 2 span and mean
_CalcOptions = 0 ; 0 base algorithm, 1 Clormann Seeger correction, 2 ASTM E1049, 3 HCM
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1)
RainflowMatrix1 = ClsOffRainflowGetMatrix ( ClsHandle )

ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan2)
RainflowMatrix2 = ClsOffRainflowGetMatrix ( ClsHandle )
delete ClsHandle
```

A Rainflow-analysis is initialised and performed. Subsequently, another channel is processed.

See also:

ClsOffRainflowInit1, ClsOffRainflowInit3

ClsOffRainflowInit3

Available in: Enterprise Edition and above (ClassCounting-Kit)

Continued initialisation of a Rainflow-count. Always use immediately after ClsOffRainflowInit2 ().

Declaration:

```
ClsOffRainflowInit3 ( ClsHandle, IgnoreSmallSpans, Precise, CountStartEnd, SV_Null1, SV_Null2, SV_Null3 )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
IgnoreSmallSpans	specifies whether small spans which fit within one class are to be ignored
	0 : Count the spans.
	1 : Ignore: small spans are not counted.
Precise	Precise calculation of the spans? Highest value recommended
	0 : compatible: Extreme values are first related to classes. Next, in the calculation of amplitudes/mean values, the amplitude or span is determined by subtraction, which is not very precise.
	1 : Precise calculation of the spans and mean values and subsequent relating to classes.
	2 : Precise calculation of the spans and mean values and subsequent allocation to classes. Additionally, more precise removal from the residue.
	3 : Precise calculation of the spans and mean values and subsequent allocation to classes. Additionally, more precise removal from the residue even with residue fading away.
	4 : Like Precise 3, but additionally precise (unrounded) residue, without rounding even with multiple calls of ClsOffRainflowFeedSamples.
	5 : Like Precise 4, but more precise
	6 : Like Precise 5, but more precise at boundary values
CountStartEnd	Count boundary time values?
	0 : Boundary time values are not counted.
	1 : The boundary time values are counted as extreme values. Recommended!
SV_Null1	0
SV_Null2	0
SV_Null3	0

Description:

Precise = 6 is recommended.

With the ASTM E1049 or HCM option, the parameter Precise must be set to 5 or higher.

Examples:

```
_IgnoreSmallSpans = 1 ; 1 ignore, 0 count
_Precise = 6
_CountStartEnd = 0 ; 1 start- and end values count as extreme values. 0 does not.
ClsOffRainflowInit3 ( ClsHandle, _IgnoreSmallSpans, _Precise, _CountStartEnd, 0, 0, 0 )
```

See also:

ClsOffRainflowInit1, ClsOffRainflowInit2, ClsOffRainflowFeedSamples

ClsOffRainflowSetMatrix

Available in: Enterprise Edition and above (ClassCounting-Kit)

The current matrix is replaced by a new matrix.

Declaration:

```
ClsOffRainflowSetMatrix ( ClsHandle, Matrix )
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
Matrix	the new matrix. Must take the same dimensions as the internal, current matrix.

Description:

All tallied values in the old matrix will be lost.

The current residue remains unchanged.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
 ... ; additional operations
ClsOffRainflowSetMatrix ( ClsHandle, Matrix1)
ClsOffRainflowFeedMatrix ( ClsHandle, Matrix2)
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
```

Here, *SetMatrix() is used to set the current matrix. Then *FeedMatrix() is used to increase its cell values by the contents of another.

See also:

ClsOffRainflowInit1, ClsOffRainflowGetMatrix, ClsOffRainflowFeedMatrix

ClsOffRevolutionsHistogram

Available in: Enterprise Edition and above (ClassCounting-Kit)

The number of turns is counted in reference to a channel.

Declaration:

```
ClsOffRevolutionsHistogram ( Channel, RPM, Min, Max, Classes, OuterBins, UnitOption ) -> Result
```

Parameter:

Channel	The input channel (e.g. plot of torque over time). This channel's values are allotted to bins (classes).
RPM	The turns are calculated on the basis of this channel. The rpms's is given in rotations per min: "rpm".
Min	Minimum of 1st channel
Max	Maximum of 1st channel
Classes	Number of bins for 1st channel
OuterBins	
	0 : closed outer bins
	1 : open-ended outer bins
UnitOption	
	0 : axes labelled by class
	1 : use physical units
Result	
Result	Histogram containg number of revolutions

Description:

The number of turns is derived from the revolutions signal.

Each measured value of a channel is allotted to one of the bins (classes), and the actual number entered into the bin is the number of turns derived from the measured rpm's.

The classes themselves are defined by the specified number of classes and the limits of the input range, given by the range Min .. Max.

The rpm measurement is converted to an absolute value.

Subsequent to this, the turns are calculated.

A turn denotes a complete revolution of the turning axis.

If the axis turns at 6000 rpm, this corresponds to 100 rotations per second.

Within one sampling interval both channels' values are assumed as constant.

Min, Max, Classes: The first channel's input range. The bins (classes) extend from Min .. Max and their number is given as the parameter Classes (>=4).

The outer bins (classes) can be open-ended or closed.

In the case of open-ended outer bins, the values outside of the range Min..Max are included in the outer bins.

If the outer bins are closed, values outside of the range Min..Max are not counted.

The labelling of the axes (x-axis and z-axis) can either refer to classes (Class 0, Class 1, Class 2, ...) or to the input channels' physical units (e.g., -0.1 Nm .. +0.1 Nm).

Examples:

```
MinTo = 0   ;Nm
MaxTo = 200 ; Nm
ClassesTo = 40
OpenEnd = 1 ; outer bins: 0 closed, 1 open
UnitOption = 1 ; 0 in classes, 1 in physical units
RevHistogram = ClsOffRevolutionsHistogram ( Torque, RPMs, MinTo, MaxTo, ClassesTo, OpenEnd, UnitOption)
```

An rpm-curve 'RPMs' and a torque curve 'Torque' (in Nm) are the input channels for the count of wheel turns.

The range of torques is divided into 40 bins (classes) through the range 0Nm .. 200Nm.

Open-ended outer bins and the labelling of the resulting histogram in terms of physical units are specified.

See also:

ClsTimeAtLevel, ClsOff2ChannelHistogram, ClsOffRevolutionsMatrix

ClsOffRevolutionsMatrix

Available in: Enterprise Edition and above (ClassCounting-Kit)

The number of turns is determined on the basis of the input data from Channel and RPM and put into a matrix.

Declaration:

```
ClsOffRevolutionsMatrix ( Channel, RPM, Min, Max, Classes, MinRPM, MaxRPM, ClassesRPM, Options ) -> Result
```

Parameter:

Channel	The input channel (e.g. plot of torque over time). This channel's values are allotted to bins (classes).
RPM	The turns are calculated on the basis of this channel. The rpms's is given in rotations per min: "rpm". The rpm values are allotted to bins.
Min	Minimum of 1st channel
Max	Maximum of 1st channel
Classes	Number of bins for 1st channel
MinRPM	Minimum of RPM
MaxRPM	Maximum of RPM
ClassesRPM	Classes for RPM
Options	Options for outer bins, unit and orientation
	0 : outer bins closed, axes labelled by class, rpm's in z-direction
	1 : outer bins open, axes labelled by class, rpm's in z-direction
	2 : outer bins closed, use physical units, rpm's in z-direction
	3 : outer bins open, use physical units, rpm's in z-direction
	4 : outer bins closed, axes labelled by class, rpm's in x-direction
	5 : outer bins open, axes labelled by class, rpm's in x-direction
	6 : outer bins closed, use physical units, rpm's in x-direction
	7 : outer bins open, use physical units, rpm's in x-direction
Result	
Result	mutual density, 2D histogram

Description:

The number of turns is derived from the revolutions signal.

Each measured value of Channel is allotted to one of the bins (classes), and each measured value of RPM is allotted to a rpm-bin, where the actual bin entry is the number of turns calculated from the measured rpm. The function returns a matrix as the result.

The rpm measurement is converted to an absolute value.

Subsequent to this, the turns are calculated.

A turn denotes a complete revolution of the turning axis.

If the axis turns at 6000 rpm, this corresponds to 100 rotations per second.

Within one sampling interval both channels' values are assumed as constant.

The outer bins (classes) can be open-ended or closed.

In the case of open-ended outer bins, the values outside of the range Min..Max are included in the outer bins.

If the outer bins are closed, values outside of the range Min..Max are not counted.

The labelling of the axes (x-axis and z-axis) can either refer to classes (Class 0, Class 1, Class 2, ...) or to the input channels' physical units (e.g., -0.1 Nm .. +0.1 Nm).

It is also possible to assign the channels to the desired coordinate in the resulting matrix.

For instance, if the rpm's are designated for the matrix z-axis, the classes of the input channel 'Channel' are plotted along the matrix x-axis.

A column of the matrix is equivalent to a segment of the waveform returned as the function's result.

Min, Max, Classes:The first channel's input range. The bins (classes) extend from Min (<Max) .. Max and their number is given as the parameter Classes (>=4).

MinRPM, MaxRPM, ClassesRPM: The input range for the rpm-channel. The bins (classes) extend from MinRPM (<MaxRPM) .. MaxRPM and their number is given as the parameter ClassesRPM (>=4).

Examples:

```
MinTo = 0   ;Nm
MaxTo = 200 ; Nm
ClassesTo = 40
MinRPM = 0   ;rpm
MaxRPM = 6000 ; rpm
ClassesRPM = 30
Option = 3 ; outer bins open, use physical units, rpm's in z-direction
RevMatrix = ClsOffRevolutionsMatrix ( Torque, RPMs, MinTo, MaxTo, ClassesTo, MinRPM, MaxRPM, ClassesRPM, Option )
```

An rpm-curve 'RPMs' and a torque curve 'Torque' (in Nm) are the input channels for the count of wheel turns.

The torque input range of 0 .. 200Nm is divided into 40 classes.

The rpm values are sorted in 30 classes in the range 0 .. 6000 rpm.

See also:

ClsTimeAtLevel, ClsOff2ChannelHistogram, ClsOffRevolutionsHistogram, ClsOffRevolutionsMatrix2

ClsOffRevolutionsMatrix2

Available in: Enterprise Edition and above (ClassCounting-Kit)

Determines the number of revolutions as a function of Channels 1 and 2.

Declaration:

```
ClsOffRevolutionsMatrix2 ( Channel1, Channel2, RPM, MinChannel1, MaxChannel1, ClassesChannel1, MinChannel2,
MaxChannel2, ClassesChannel2 ) -> Result
```

Parameter:

Channel1	1st input channel (e.g. torque over time)
Channel2	2nd input channel (e.g. temperature over time).
RPM	the revolutions are computed from this RPM channel's data.
MinChannel1	lower limit of the 1st channel's class range
MaxChannel1	upper limit of the 1st channel's class range
ClassesChannel1	number of bins (classes) for the 1st channel, >= 4
MinChannel2	lower limit of the 2nd channel's class range
MaxChannel2	upper limit of the 2nd channel's class range
ClassesChannel2	number of bins (classes) for the 2nd channel, >= 4
Result	
Result	RevolutionsMatrix

Description:

Determines the number of revolutions as a function of Channels 1 and 2.

The number of turns is derived from the revolutions signal.

Each of a channel's measurement values is assigned to one of the classes, and the actual entry in a class is the amount of revolutions derived from the corresponding RPM-measurement.

This procedure produces a matrix.

The rpm measurement is converted to an absolute value.

Then the revolutions are computed.

A turn denotes a complete revolution of the turning axis.

If the axis turns at 6000 rpm, this corresponds to 100 rotations per second.

The RPM-value is always scaled in revolutions per minute: "revs/min" or "rpm".

Within one sampling interval both channels' values are assumed as constant.

The outer bins (classes) are closed, i.e., values lying outside of the range Min..Max are discarded, and don't affect the count.

The matrix is labeled both in the x- and z-direction in terms of the input channels' physical units.

The matrix's x-axis (column, segment) is scaled in terms of the physical quantity represented by the 1st channel.

The z-axis (row) is scaled in terms of the 2nd channel's quantity.

Examples:

```
Min1 = 0   ; Nm
Max1 = 200 ; Nm
Classes1 = 40
Min2 = 0   ; Nm
Max2 = 300 ; Nm
Classes2 = 60
RevMatrix = ClsOffRevolutionsMatrix2 ( Torque1, Torque2, RPMs, Min1, Max1, Classes1, Min2, Max2, Classes2 )
```

An rpm-signal RPMs (expressed in revolutions per minute) and 2 torque signals Torque1 and Torque2 (in Nm) serve as the input channels for a count of revolutions.

The torque range extends through 0 .. 200Nm and is divided into 40 classes in the 1st channel and in the 2nd channel through 0 .. 300Nm in 60 classes.

See also:

ClsTimeAtLevel, ClsOff2ChannelHistogram, ClsOffRevolutionsHistogram, ClsOffRevolutionsMatrix

ClsOffTM

Available in: Enterprise Edition and above (ClassCounting-Kit)

Application of the imc TrueMax procedure.

Declaration:

```
ClsOffTM ( Input ) -> Result
```

Parameter:

Input	Input data
Result	
Result	Filtered signal

Description:

If a signal is measured by sampling, it is not certain that any sample will coincide with the true reversals (extrema) in the physical signal.

This fact is especially noticeable if the signal frequency is high.

Maxima often are found (far) too low and minima too high.

Subjecting such distorted raw data to the Rainflow class-count then causes significant errors.

For band-limited signals (signals sampled with a correctly-adjusted anti-aliasing filter), the complete signal course can be reconstructed and the true signal reversals thus determined.

The procedure gives a good approximation and improves the signal's course by (slightly) changing the local extreme values. The waveform which results is only suitable for class-counting.

The time plot of the corrected signal can appear unnatural.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
ClsOffRainflowInit2 ( ClsHandle, _Min, _Max, _Hysteresis, _Axis, _Type, 1, _CalcOptions)
ClsOffRainflowInit3 ( ClsHandle, 1, 6, 1, 0, 0, 0 )
data_chan1_TM = ClsOffTM ( data_chan1 )
ClsOffRainflowFeedSamples ( ClsHandle, data_chan1_TM )
RainflowMatrix = ClsOffRainflowGetMatrix ( ClsHandle )
```

The local extrema of a measurement are corrected befor the measurement is subjected to Rainflow-analysis.

This increases the accuracy of the class-count.

See also:

ClsOffRainflowInit1, ClsOffRainflowFeedSamples

ClsOffWoehlerSN

Available in: Enterprise Edition and above (ClassCounting-Kit)

Cumulative fatigue damage from the Rainflow-matrix. The Palmgren / Miner rule is used.

Declaration:

```
ClsOffWoehlerSN ( ClsHandle, S-N diagram, class relation, Interpolation ) -> Result
```

Parameter:

ClsHandle	data returned by ClsOffRainflowInit1 ()
S-N diagram	Data set describing the S-N curves in the XY-format (stress over lifetime [cycles]). The Y-coordinate contains the nominal values of (mechanical) stress. These values are ordered by decreasing value. The X-coordinate contains the values for the associated lifetime, a number of values which is usually increasing, typically about: 10, 1000, 1000000, ...For X and Y only values > 0 are allowed.
class relation	How is the corresponding Y-coordinate of the S-N curve derived from the value range of a Rainflow-matrix class, from which in turn the lifetime will be according to the S-N curve?
	0 : Automatic: the assumption that all cycles in a class of the Rainflow-matrix are distributed equally over the entire extent of the class.
	1 : The maximum value (upper edge of a class) is assumed for all cycles of the class.
	2 : The median (midpoint between a class's upper and lower limits) is assumed for all of the class's cycles.
Interpolation	Which interpolation is to be used to connect the individual points of the S-N curve?
	0 : Automatic. The S-N curve is plotted over a double logarithmic scale. With this scaling, the stated points of the line are connected with straight lines. For metals.
	1 : The S-N curve is plotted over a scale in which the lifetime scaling is logarithmic and the stress linear. In this scale, the stated points on the line are connected with straight lines. For concrete.
Result	
Result	The result is the cumulative fatigue damage. The typical value range lies between 0 and 1, where 0 means no damage, and 1 means damage leading to breakage. Values above 1 are also possible.

Description:

The number of cycles actually measured is determined from the Rainflow-matrix. The residue is not taken into account. If you wish it to be taken into account, it must first be included in the matrix count.

The Rainflow matrix must previously have been created with the functions ClsOffRainflowInit1() etc. In the process, such functions as ClsOffRainflowFeedSamples () can be used, if there is a time history of stress, or functions such as ClsOffRainflowSetMatrix(), if there is already a Rainflow-matrix available.

The Y-coordinate of the S-N curve is always expressed as a range. The range is the difference between the maximum value and the minimum value of a vibration, in other words twice the amplitude. It does not matter whether the Rainflow count consisted of amplitudes, ranges or start-target classes.

The Y-coordinate of the S-N curve must take the same unit as the time history of stress. If the Rainflow count was made with a stress expressed in N/mm^2, then the S-N curve must also be expressed in N/mm^2.

The Y-coordinate of the S-N curve is typically given as a stress, force or torque. Please note that the S-N curve for a certain material is often represented in a normalized form. Then this normalized curve needs to be adapted to the unit of the time history used for rainflow counting.

The S-N curve is often given by 2 points. The underlying equation is S = a * N ^ b, with S = stress, N= lifetime [cylces], a and b as constants. With double-logarithmic scaling, this forms a straight line. Since the function is interpolated adequately, stating just these two points is sufficient.

The S-N curve should be defined over a sufficiently wide stress range, generally at least as wide as the classification range.

If the stress exceeds the largest value stated in the S-N curve (meaning the first Y-value), the lifetime assigned to the largest value is assumed.

If the stress falls below the smallest value stated in the S-N curve, an infinitely long lifetime is assumed.

Please note that the S-N curve only approximates a real workpiece. The Palmgren / Miner rule is also only an approximation technique which ignores, for example, the mean values of the cycles and their order. Consequently, the accumulated damage calculated here is only an approximation of the damage sustained by the workpiece.

A return value of 0.1, for instance, means that 10% of the workpiece's lifetime has elapsed.

Suppose that 10 cycles at a certain stress level have been counted into the rainflow, and that you can read from the S-N curve a lifetime of 10000 cylces for that stress level, then 0.1% of the liefetime has elapsed due to this stress.

Examples:

```
ClsHandle = ClsOffRainflowInit1 ( _NumberClasses, _TypeOfUnit, _UnitRow, _UnitColumn, _UnitCount, _UnitRes, 0)
.... various others such as ClsOffRainflowFeedSamples() or ClsOffRainflowSetMatrix
```

```
Y = join ( 2000, 20 ) ; stress
X = join (1e3, 1e8 ) ; lifetime [cycles]
SN = xyof ( X, Y) ; S-N curve
Damage = ClsOffWoehlerSN ( ClsHandle, SN, 0, 0 )
```

The Rainflow analysis is first performed and then the Rainflow matrix calculated.

Then the S-N curve is created from 2 points for a steel workpiece and the damage is determined.

See also:

ClsOffRainflowInit1, ClsOffRainflowSetMatrix, ClsOffRainflowFeedSamples

ClsPeak1

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by zero-crossing (Peak-counting Procedure 1)

Declaration:

```
ClsPeak1 ( Channel, MaxValue, MinValue, Number of bins, Reference, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Reference	Reference level
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : automatic reference level
	3 : open-ended outer bins, automatic reference level
Result	
Result	Distribution

Description:

Class-counting of measured data by the peak-counting Method 1 (Zero-crossing peak counting), as per DIN 45667.

This class-counting method tallies the most extreme value within each interval between two crossings of the signal over the reference line. The extrema identified are tallied in bins which reflect their values.

Examples:

```
Distribution = ClsPeak1( Data, 15, 5, 20, 0, 0.5, 3 )
```

See also:

ClsTimeAtLevel, ClsPeak2, ClsPeak3, ClsOffFromRainflowGetZeroCrossingPeak

ClsPeak2

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by peak-counting Procedure 2

Declaration:

```
ClsPeak2 ( Channel, MaxValue, MinValue, Number of bins, Reference, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Reference	Reference level
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : automatic reference level
	3 : open-ended outer bins, automatic reference level
Result	
Result	Distribution

Description:

Class-counting of measured data by the peak-counting Method 2, as per DIN 45667.

This method tallies all local maxima lying above the reference line and all local minima lying below the reference line in bins.

Examples:

```
Distribution = ClsPeak2( Data, 28, 12, 32, 0, 0.5, 1 )
```

See also:

ClsTimeAtLevel, ClsPeak1, ClsPeak3, ClsOffFromRainflowGetPeak

ClsPeak3

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by peak-counting Procedure 3

Declaration:

```
ClsPeak3 ( Channel, MaxValue, MinValue, Number of bins, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : negative peaks
	3 : open-ended outer bins, negative peaks
Result	
Result	Distribution

Description:

Class-counting of measured data by the peak-counting Method 3, as per DIN 45667.

Peak-counting Method 3 entails tallying all local maxima, and separately, all local minima in bins.

The results of this method, then are organized in two distributions.

This function in the Class-counting Kit, however, produces a distribution either for only the minima or only the maxima. As the default, the results are based on the maxima.

Examples:

```
Distribution = ClsPeak3( Data, 0.6, 0.1, 25, 0.02, 3 )
```

See also:

ClsTimeAtLevel, ClsPeak1, ClsPeak2, ClsOffFromRainflowGetMinMaxPeak

ClsPkSmp

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by peak value/sampling procedure

Declaration:

```
ClsPkSmp ( DataSet1, DataSet2, MaxValue, MinValue, Number of bins, Hysteresis, Options ) -> Result
```

Parameter:

DataSet1	1st set of data to be classified
DataSet2	2nd set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
Result	
Result	Distribution

Description:

Class-counting of measured data by the Peak /Sample method

Waveform 1 is processed according to peak-counting Method 3.

All local extrema are given regard.

Wherever a local extremum is found in Waveform 1, the sample in Waveform 2 at the corresponding location is tallied in the appropriate bin.

If, for instance, the fifth value in Waveform 1 is a local minimum, its value is classed and so is the fifth value of Waveform 2.

In the second stage, the reverse procedure is carried out; the local extrema of Waveform 2 are tallied in bins, as well as the corresponding data points of Waveform 1.

The results of the class-counting method are arranged in a correlation matrix; the matrix columns represent the bins of Waveform 1 and the rows the bins of Waveform 2. A tally is made to that cell of the matrix where one coordinate corresponds to the value of a local extremum in one waveform, and the other coordinate corresponds to the value of the contemporaneous data point in the other waveform.

Examples:

```
Distribution = ClsPkSmp( Data, 10, -10, 40, 0.5, 1 )
```

See also:

ClsTimeAtLevel, ClsPeak3, ClsSmSmp

ClsQuantile

Available in: Enterprise Edition and above (ClassCounting-Kit)

Quantile of a frequency distribution. Up to which x-coordinate a particular percentage of all values are located.

Declaration:

```
ClsQuantile ( Frequency distribution, Proportion, Rounding ) -> Result
```

Parameter:

Frequency distribution	Frequency distribution for which the quantile is to be determined
Proportion	Proportion, stated in percent, 0 to 100
Rounding	Rounding
	0 : If the quantile does not lie exactly on a class boundary, an intermediate value is calculated. The underlying assumption is that the values within a class are uniformly distributed. This corresponds to a display of the signal in stair-steps.
	1 : If the quantile does not lie exactly on a class boundary, then the next lower class boundary is returned. A somewhat smaller actual proportion may be contained up to that point.
	2 : If the quantile does not lie exactly on a class boundary, then the next higher class boundary is returned. A somewhat larger actual proportion may be contained up to that point.
Result	
Result	Quantile

Description:

The sum of all y-values is regarded as an aggregate and declared to be 100%.

The function summates all y-values from the beginning of the data set (i.e. from the left) onwards, until the sum reaches the desired percentage of the total.

Returns the x-coordinate at this location.

The summation correctly applies the values' signs.

The quantile divides a distribution into 2 regions.

Thus, for instance, a 75%-quantile is the x-coordinate from where 75% of the surface is on the left side and 25% on the right.

The input data are treated as stair-steps/columns.

For instance, if the distribution has values at the x-positions 3, 4, 5, then there are three stair-steps with frequencies, each of width 1. The last step extends from 5 to 6. The 100%-quantile is thus 6, unless the classes contain zero-values.

Available from imc FAMOS 7.1 onwards

The median is the 50% quantile. With tertiles (terciles) the range is split into 3 equal intervals. There are 4 quartiles, each of which has a width of 1/4 of the range. There are 100 percentiles of width 1%.

Examples:

```
Quantile = ClsQuantile ( Histogram, 95, 0 )
```

```
Percentile90 = ClsQuantile ( Histogram, 90, 0 ) - ClsQuantile ( Histogram, 89, 0 )
```

```
MedianValue = ClsQuantile ( Histogram, 50, 0 )
```

See also:

ClsTimeAtLevel

ClsRange

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by range-counting

Declaration:

```
ClsRange ( Channel, MaxValue, MinValue, Number of bins, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : negative ranges
	3 : open-ended outer bins, negative ranges
Result	
Result	Distribution

Description:

Class-counting of measured data by the Range-counting method as per DIN 45667.

Ranges in this context are defined to be the sweeps of the signal between two adjacent local extema. Ppositive ranges are defined as the signal between a local minimum and the following local maximum, negative ranges are defined as the signal between a local maximum and the following local minimum.

The function will tally either all positive ranges or all negative ranges.

Countint positive ranges is the default option.

Examples:

```
Distribution  = ClsRange( Data, 13, -1, 28, 0.5, 3 )
```

See also:

ClsTimeAtLevel, ClsRMean, ClsRngPr

ClsRFlow

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by the Rainflow-procedure

Declaration:

```
ClsRFlow ( Channel, MaxValue, MinValue, Number of bins, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : Amplitude-Mean-representation
	3 : open-ended outer bins, Amplitude-Mean-representation
Result	
Result	Histogram

Description:

Before the function returns, the residue is tallied into the matrix.

Examples:

```
Rainflow  = ClsRFlow( Data, 25, 10, 30, 0.5, 3 )
setseglen ( Rainflow, 30 )
```

; Standard function call. The matrix is not scaled.

```
_NumberClasses = 30
_Min = -5
_Max = 5
_option = 2+1 ;( +1 open end, +2 Amp-Mean )
_Hysteresis = ( _Max - _Min ) / _NumberClasses
RainflowMatrix = ClsRFlow ( data_chan1, _Max, _Min, _NumberClasses, _Hysteresis, _option)
setseglen ( RainflowMatrix, _NumberClasses )
xoffset RainflowMatrix _Min
_dx = ( _Max - _Min ) / _NumberClasses
xdelta RainflowMatrix _dx
if _option >= 2
   ; mean plotted along x-axis
   ; amplitude plotted along z-axis
   setzdel ( RainflowMatrix, _dx / 2 )
   setzoff ( RainflowMatrix, 0 )
   setunit ( RainflowMatrix, "Mean [Nm]", 0 ) ; x
   setunit ( RainflowMatrix, "Ampl [Nm]", 2 ) ; z
else
   setzdel ( RainflowMatrix, _dx )
   setzoff ( RainflowMatrix, _Min )
   setunit ( RainflowMatrix, "Nm", 0 ) ; x
   setunit ( RainflowMatrix, "Nm", 2 ) ; z
end
setunit ( RainflowMatrix, "Count", 1 ) ; y
```

The Rainflow-matrix is determined.

A segmented waveform with scaling is generated.

See also:

ClsOffRainflowInit1

ClsRMean

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by range-mean counting

Declaration:

```
ClsRMean ( Channel, MaxValue, MinValue, Number of bins, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : Amplitude-Mean-representation
	3 : open-ended outer bins, Amplitude-Mean-representation
Result	
Result	Distribution

Description:

Class-counting of measured data by the range-mean-counting method as per DIN 45667.

A range refers to the difference in signal value between two adjacent extrema.

A range located between a minimum and a subsequent maximum is termed a positive range, and the opposite case is termed a negative range.

In contrast to the case of single-parameter range-based counting methods, the positive and negative ranges are not treated separately but are lumped together, and also, the mean values of the ranges are weighted.

This function offers both the correlation matrix and amplitude-mean representation as options for the output of the results. The default setting is the correlation matrix.

Where the correlation matrix is used, each range is tallied in the matrix at the location where one coordinate corresponds to the bin where the range began and the other coordinate corresponds to the bin where the range ended.

If amplitude-mean representation is applied, each range is evaluated in terms of its magnitude and its mean value.

The range is then tallied in the result matrix at the location where one coordinate corresponds to the magnitude of the range, measured in classes (bin widths), and the other coordinate corresponds to the class (bin) where the range's mean value is situated.

Ranges which are between 0 and 1 bin width in magnitude are tallied in the matrix where the magnitude coordinate is Bin 1.

Ranges larger than 1 and up to 2 bin widths in magnitude are tallied in the matrix where the magnitude coordinate is Bin 2, etc.

Examples:

```
Distribution = ClsRMean( Data, 50, 0, 50, 1, 1 )
```

See also:

ClsTimeAtLevel, ClsRange, ClsRFlow

ClsRngPr

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by range-pair counting

Declaration:

```
ClsRngPr ( Channel, MaxValue, MinValue, Number of bins, Hysteresis, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Hysteresis	Hysteresis for suppressing small oscillations
Options	
	0 : no options
	1 : open-ended outer bins
	2 : Precise calculation
	3 : Precise calculation with start value
	4 : Spans between extreme values
Result	
Result	Distribution

Description:

Class-counting of measured data by the range-pair class-counting method as per DIN 45667.

The system recognizes positive ranges as signal sweeps subsequent to minima and negative ranges as signal sweeps subsequent to maxima.

A tally is only then occasioned, when a positive range is matched to a negative range of equal magnitude; in other words, when a range-pair has occurred.

A small range-pair can be counted several times, before a bigger one will be counted.

With the option "Precise calculation", the algorithm is optimised. Open-ended outer bins are applied.

If the precise calculation is not selected, the algorithm will be compatible with previous versions.

If the option contains "with start value", the first value of the input data will be considered as an extreme value. Otherwise the algorithm starts with the search for the first relative extreme value.

With the option "Spans between extreme values", a precise calculation with start value is performed. But spans are only counted between extreme values. If, for instance, the input data contain a single cycle, then only a single "1" is returned instead of all "1"s up to the corresponding class. This does not conform to DIN 45667.

Examples:

```
Distribution  = ClsRngPr( Data, 24, -12, 36, 1, 1 )
```

See also:

ClsTimeAtLevel, ClsRFlow, ClsRange, ClsOffFromRainflowGetRangePair

ClsSampl

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by sampling

Declaration:

```
ClsSampl ( Channel, MaxValue, MinValue, Number of bins, Distance between samples, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Distance between samples	
Options	
	0 : no options
	1 : open-ended outer bins
	2 : random sampling
	3 : open-ended outer bins, random sampling
Result	
Result	Distribution

Description:

Class-counting of measured data by the Sampling method as per DIN 45667.

In this procedure,samples of the momentary value of the measured waveform are tallied into classes (bins). The samples can be spaced at either fixed or randomly fluctuating intervals.

Examples:

```
Distribution = ClsSampl( Data, 20, -20, 40, 3, 3 )
```

See also:

ClsTimeAtLevel, ClsMaxSt

ClsSmSmp

Available in: Enterprise Edition and above (ClassCounting-Kit)

Class-count by sample/sample procedure

Declaration:

```
ClsSmSmp ( DataSet1, DataSet2, MaxValue, MinValue, Number of bins, Distance between samples, Options ) -> Result
```

Parameter:

DataSet1	1st set of data to be classified
DataSet2	2nd set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Distance between samples	
Options	
	0 : no options
	1 : open-ended outer bins
	2 : random sampling
	3 : open-ended outer bins, random sampling
Result	
Result	Distribution

Description:

Class-counting of measured data by the Sample-/sample method.

Waveform 1 is treated according to the Sampling method,

A tally is made in the bin corresponding to the value of those samples of Waveform 2 located where Waveform 1 has tallied values.

For instance, if Data Point 5 of Waveform 1 is tallied, then so is Data Point 5 of Waveform 2.

The results of the class-counting method are arranged in a correlation matrix; the matrix columns represent the bins of Waveform 1 and the rows the bins of Waveform 2. A tally is made to that cell of the matrix where one coordinate corresponds to the sample value in one waveform, and the other coordinate corresponds to the value of the contemporaneous data point in the other waveform.

Examples:

```
Distribution = ClsSmSmp( Data1, Data2, 3, -3, 24, 5, 3 )
```

See also:

ClsTimeAtLevel, ClsPkSmp, ClsSampl

ClsTAtLv

Available in: Enterprise Edition and above (ClassCounting-Kit)

Classification by time-at-level procedure

Declaration:

```
ClsTAtLv ( Channel, MaxValue, MinValue, Number of bins, Options ) -> Result
```

Parameter:

Channel	Set of data to be classified
MaxValue	Upper limit of the value range to be classified
MinValue	Lower limit of the value range to be classified
Number of bins	
Options	
	0 : no options
	1 : open-ended outer bins
Result	
Result	Histogram

Description:

Instead of this function, the newer function ClsTimeAtLevel() should be used!

Class-counting of measured data by the Time-at-level as per DIN 45667.

The cumulative time spent by the signal in each partial range is reflected in the bin corresponding to that range.

The Class-counting kit accomplishes this by tallying every sample of the waveform in the appropriate bin.

This function is equivalent to using the Sampling-method function and specifying the sampling distance as 1 (data point).

The time-at-level procedure is what is usually called 'histogram' or 'distribution of amplitudes'.

Examples:

```
Distribution = ClsTAtLv( Data, 10, -10, 40, 1 )
```

See also:

Histo, ClsOff2ChannelHistogram, ClsTimeAtLevel

ClsTimeAtLevel

Available in: Enterprise Edition and above (ClassCounting-Kit)

Histogram, time-at-level; determines the occurrence count of amplitudes

Declaration:

```
ClsTimeAtLevel ( Channel, MinValue, MaxValue, Number of bins, Interpolation, OuterBins, X-unit, Calculation ) -> Result
```

Parameter:

Channel	Set of data to be classified
MinValue	Lower limit of the value range to be classified
MaxValue	Upper limit of the value range to be classified
Number of bins	
Interpolation	
	0 : Stair-steps; standard-algorithm for a histogram. Each sample value applies to the entire sampling interval. The data set's sample values are counted by class.
	1 : Linear; the individual sample values are imagined to be connected by straight lines. The entire signal plot is counted, not just the sample values themselves.
OuterBins	
	0 : Closed outer bins; if any value is outside of the range, it is not counted.
	1 : Open-ended outer bins; if any value is outside of the range, it is counted in the corresponding outer bin.
X-unit	
	0 : axes labelled by class
	1 : use physical units
Calculation	
	0 : Count of measured values
	1 : Time expressed in the physical unit of the X-axis of the data set to be classified
	2 : Percent; as a proportion of the total duration/total count; value range: 0 .. 100%
	3 : Amplitude probability density. The integral over the entire range equals 1. The integral over any partial range equals the probability of the amplitude lying within that partial range.
	4 : Cumulative relative count. Representing the percent of values lying either in this or any lower range. Value range: 0 .. 100%
Result	
Result	Histogram

Description:

Classification of measured value by the Time-at-level procedure as per DIN 45667, in case of interpolation with stair-steps

The function replaces ClsTAtLv()

Examples:

Histogram, also DIN 45667

```
TimeAtLevel = ClsTimeAtLevel ( channel, -10.0, 10.0, 100, 0, 1, 0, 0 )
```

Time-at-level

```
TimeAtLevel = ClsTimeAtLevel ( channel, -10.0, 10.0, 100, 0, 1, 1, 1 )
```

Precise Time-at-level

```
TimeAtLevel = ClsTimeAtLevel ( channel, -10.0, 10.0, 100, 1, 1, 1, 1 )
```

Amplitude probability density

```
TimeAtLevel = ClsTimeAtLevel ( channel, -10.0, 10.0, 100, 0, 1, 1, 3 )
```

See also:

ClsTAtLv, ClsQuantile

Cmp1

First component of a complex data set

Declaration:

```
Cmp1 ( Data ) -> Component1
```

Parameter:

Data	Complex data set, whose first componente is to be returned [KX]
Component1	
Component1	First component of the complex data set

Description:

This function returns the first component of a complex data set. The first component is either the real part, the magnitude or the magnitude in dB, depending on the type of complex data set.

- Intermediate results in a formula cannot be used in combination with the name extensions ".R", ".M" , etc. to form a complex data set. In such cases, the Cmp1 function should be used instead.
- If the complex type is unknown, the Cmp1 function should not be used. First use the Pol or Rect functions to create a definite data type.
- The parameter may be structured (events/segments).

Examples:

The magnitude of the spectrum is to be determined; taking advantage of the fact that the Spec function always returns the complex data type MP:

```
magnitude = Cmp1(Spec(NDdata))
```

The real part of a complex data set is assigned to a variable. Both instructions are equivalent:

```
NDreal = Cmp1(RIdata)
NWreal = RIdata.R
```

See also:

Cmp2, Compl, IsCplx, Pol, Rect

Cmp2

Second component of a complex data set

Declaration:

```
Cmp2 ( Data ) -> Component2
```

Parameter:

Data	Complex data set whose second componente is to be returned [KX]
Component2	
Component2	Second component of the complex data set

Description:

This function returns the second component of a complex data set. The second component is either the imaginary part or the phase, depending on the complex data type.

- Intermediate results in a formula cannot be used in connection with the name extensions ".I", ".P", etc. to form a component of a complex data set. In such cases, use the Cmp2 function instead.
- If the complex data type is unknown, Cmp2 cannot be used. Use the Pol or Rect functions to create definite data types.
- The parameter may be structured (events/segments).

Examples:

A spectrum's phase is to be determined; it is used so that the function Spec() always returns the complex type MP (magnitude/phase):

```
Phase = Cmp2(Spec(NDdata))
```

The imaginary part of a complex data set is assigned to a variable. Both instructions are equivalent:

```
NDimag = Cmp2(RIdata)
WDimag = RIdata.I
```

See also:

Cmp1, Compl, IsCplx, Pol, Rect

CmpX

X-component of an XY-data set

Declaration:

```
CmpX ( Data ) -> ComponentX
```

Parameter:

Data	XY-data set whose X-component is to be returned [XY]
ComponentX	
ComponentX	X-component of the data set

Description:

The X-component of an XY-data set is returned.

- Use the suffixes .X and .Y for access to the components of an XY-data set.
- The parameter may be structured (events/segments).

Examples:

The X-component of an XY-data set is multiplied by 2. Both instructions are equivalent:

```
data.X = CmpX(data) * 3
data.X = data.X * 3
```

See also:

CmpY, XYof, IsXY

CmpY

Y-component of an XY-data set

Declaration:

```
CmpY ( Data ) -> ComponentY
```

Parameter:

Data	XY-data set whose Y-component is to be returned [XY]
ComponentY	
ComponentY	Y-component of the data set

Description:

The Y-component of a XY-data set is returned.

- Use the suffixes .X and .Y for access to the components of an XY-data set.
- The parameter may be structured (events/segments).

Examples:

The Y-component of an XY-data set is multiplied by 2. Both instructions are equivalent:

```
data.Y = CmpY(data) * 2
data.Y = data.Y * 2
```

See also:

CmpX, XYof, IsXY

CodeRange

Available in: Professional Edition and above

A list of numerical values (codes) is assigned to and returned for a list of ranges of Y-values of the input signal.

Declaration:

```
CodeRange ( input data, Lower boundaries, Upper boundaries, Codes [, Comparison] ) -> Result
```

Parameter:

input data	input data
Lower boundaries	List of the lower boundaries of all value ranges; or triplets
Upper boundaries	List of the upper boundaries of all value ranges; 0 with triplets
Codes	List of the codes for all value ranges; 0 with triplets
Comparison	Formulation of the comparison (optional , Default value: "L<=y<=H")
	"L<=y<=H" : Lower Boundary <= Input data <= Upper boundary
	"L<y<=H" : Lower Boundary < Input data <= Upper boundary
	"L<=y<H" : Lower Boundary <= Input data < Upper boundary
	"L<y<H" : Lower Boundary < Input data < Upper boundary
Result	
Result	Result

Description:

Encoding of a range

The 3 datasets "Lower, "Upper" and "Codes" have equal lengths. For each range (defined by the lower and upper boundaries) the corresponding code is definied.

The list of value ranges does not need to be sorted. The function works through the list from start to end. Upon the first match, the code found is returned.

An input value is considered to be assigned to a value range if the comparison condition is met according to its exact formulation (< or <=).

The function can be considered as a Switch instruction with Case instructions for each value range. The Switch instruction is run for each measurement start of the input signal.

If the value of the input data does not lie within any of the value ranges, a zero is returned. This can be regarded as the default situation of the Switch instruction.

The input data can have events and segments. If they are of the type XY, the range encoding is applied to the Y-component.

In a way which is compatible with imc Inline FAMOS, it is possible to specify a single data set containing the triplet consisting of lower boundary, upper boundary and code in succession; the remaining function parameters are all zero; see example.

Examples:

Gear recognition: rpm_in, rpm_out are the input and output RPM values at the transmission. Invalid ranges are somewhat widened.

```
Lo = [ 0.4, 0.7, 0.9, 1.1 ]
Hi = [ 0.5, 0.8, 0.93, 1.17 ]
Co = [ 1, 2, 3, 4 ]
Ratio = CodeRange ( rpm_out / rpm_in, Lo, Hi, Co )
Change1 = ( Ratio = 0 ) OR (rpm_in < 20 )
Change2 = Monoflop ( Change1, 0.5, "1 retrig", "0-1" )
Change3 = Monoflop ( Change2, 0.5, "1 retrig", "0-1", "reverse" ) ; =1 when gears are changed
Gear = Ratio * (1-Change3)
```

One data set provides all triplets; application like 1st example

Compatible with imc Inline FAMOS and imc Online FAMOS

The parameter Control contains the sequence of values low[1], high[1], code[1], low[2], high[2], code[2], ...

```
Control = [ 0.4, 0.5, 1, 0.7, 0.8, 2, 0.9, 0.93, 3, 1.1, 1.17, 4 ] ; 4 triples of low/high/code
Ratio = CodeRange ( rpm_out / rpm_in, Control, 0, 0 )
```

See also:

switch, RangeSet

Coherence

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

The coherence is determined through linear averaging of power spectra. It is calculated by means of FFT.

Declaration:

```
Coherence ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping [, Base2] ) -> Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Result

Description:

With x = input, y = output and G being a power spectrum, then | Gxy | ^ 2 / (Gxx * Gyy) is calculated. Value range: 0.0 ... 1.0

The averaging is performed on the real and imaginary parts separately.

This calculation becomes significant only if there are many values to be averaged.

Examples:

```
Coh = Coherence ( Force, Movement, 1000, 0, 50, 0 )
```

This calculates a sequence of 1000 point-power-spectra, which each overlap their neighbors by 50%. A force operates on a mechanical part. The part's movement is measured on the opposite end. The result is formed from the averaged power spectra.

See also:

FrequencyResponse, CrossPowerDS

Color?

Queries the color for a data set's curve window representation

Declaration:

`Color? ( Data ) -> SvColorValue`

Parameter:

Data	Data set whose color attribute is to be queried
SvColorValue	
SvColorValue	Color value; -1 means automatic color assignment

Description:

Normally, data sets are displayed in the curve window configuration's color for graphs. But you ca assign to the data set a fixed color in which it will appear in any curve window, regardless of the curve window's current color setting.

This function returns the color value with which the data set is displayed, or -1, if no fixed color has been assigned.

The color value is a so-called RGB-value in which the portions of the 3 primary colors red, green and blue are stated.

To achieve a color value from primary color components, you can use the function RGB().

Examples:

```
green = RGB(0, 255, 0)
clr = Color?(data)
IF clr = -1
   SetColor(data, green)
END
```

Unless a fixed color has already been assignedd to the data set, it will be colored green in the future.

See also:

SetColor, RGB

Comm?

Queries the comment on a data set, text or data group

Declaration:

```
Comm? ( Dataobject ) -> TxComment
```

Parameter:

Dataobject	Data set, text or data group whose comment is to be determined
TxComment	
TxComment	Comment

Description:

The comment on a data set, text or group is queried. A comment can be assigned to each of these data types, which can be specified either by means of a dialog or using the function SetComm().

Examples:

The comment on a data set is queried. If none has been specified (length: 0), the user is prompted to enter a comment, which will then be assigned to the data set.

```
txComment = Comm?(data)
IF TLeng(txComment) = 0
   txComment = BoxText?("Please enter comment:", "",0)
   SetComm(data, txComment)
END
```

See also:

SetComm, Name?

COMMENT

Begins a comment
This command is obsolete; for reasons of better legibility, a semicolon should be used instead to begin a line of comment.

Declaration:

```
COMMENT Comment
```

Parameter:

Comment	Arbitrary text for the comment

Description

This command begins a comment. All characters following this command are interpreted as comments. This command is useful for embedding commentary in sequences.

Comments initialized with a semicolon can also be positioned after a command or formula in the same code line.

Examples:

```
COMMENT -- This is a short sequence,
COMMENT -- which loads and displays a file.
; -- A parameter must be supplied.
LOAD PA1 ; Load parameter
SHOW PA1 ; Show parameter
```

A data set is loaded and displayed. The first three lines are comments.

See also:

SEQUENCE

Compl

Combines two real number data sets to one complex data set.

Declaration:

```
Compl ( Component1, Component2 ) -> ComplexData
```

Parameter:

Component1	Real part of magnitude of the complex data set [ND] to be formed
Component2	Imaginary part or phase of the complex data set [ND] to be formed.
ComplexData	
ComplexData	Resulting complex data set [BP],[DP] or [RI]

Description:

This function combines two real data sets (type Nw) to form one complex data set. One of the two specified parameters must be a equidistant data set, the other can be a single value. The single value is then automatically expanded to a equidistant data set of the required length.

Both parameters may be structured (events/ segments), however, in that case, both parameters must have exactly the same structure (same segment length, event-count and -length). The combination of a structured parameter with a single value is also allowed.

The complex data type created depends on the units of the specified data sets. The complex data type is determined according to the following criteria:

If the units of both data sets are the same, or if the data sets have no units, the type RI is created.

If the criteria for the type RI are not fulfilled, the type DP is created if the unit of the first specified data set is dB.

The type MP is created if the criteria for the types RI or DP are not met.

- Always specify a definite unit for phase, i.e. "Rad" or "Degr".
- Both of the specified real data sets should have the exact same x-scaling and length. Otherwise, a warning message is generated and if necessary, the waveforms are adjusted automatically.
- Use the extensions ".M" or ".P" etc. to change only one component of a complex data set.

Examples:

The following three formulas generate an imaginary data set, i.e. a complex data set with a real part of 0. Note that when a single value is specified, its unit must correspond to that of the specified data set to create a complex data set of the type RI.

```
RIdata = Compl(RIdata.R * 0, RIdata.I)
RIdata = Compl(0 'V', RIdata.I)
RIdata = Compl(0 'V', NDdata)
```

A complex data set is created with a magnitude which grows from 0 Hz to 511 Hz at a constant phase of 90°. This complex data set is completely imaginary, but can be used to represent the frequently used "i" (angular frequency).

```
MPjomega = Compl(Ramp(0, 1, 512) * 1'Hz', 90'°'* (1 - 0 * Ramp(0, 1, 512)))
```

See also:

Cmp1, Cmp2, IsCplx, Pol, Rect

ComplexSpectrum

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Complex spectrum (harmonics determined as RMS (root-mean-square) values) using a moving window and linear averaging. Calculated by means of FFT.

Declaration:

```
ComplexSpectrum ( InputData, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [, Base2] ) ->
Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
AveragingType	method of summarizing all spectra
	0 : no averaging
	1 : averaging (arithmetic mean or linear averaging of the real and imaginary parts separately). The number of spectra over which the average is taken is determined by the parameter 'Reduction'.
	2 : Peak Hold Max, from beginning. Maximum values, based on the spectra calculated thus far in the algorithm.
	3 : Peak Hold Max, interval. Maximum values, based on the number of spectra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the spectra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of spectra which the parameter 'Reduction' dictates.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is complex, i.e., it has a magnitude and a phase. The phase is computed in degrees. The result is a segmented waveform, where each segment represents a spectrum.

Description:

The linear averaging is performed on the real and imaginary parts separately. For the calculation of an averaged magnitude spectrum the use of the function AmpSpectrumRMS() is recommended.

For Peak Hold calculations, the Peak Hold is applied to the magnitude. The phase is averaged, but is thereby rendered irrelevant and should be ignored.

Examples:

```
Spectra = ComplexSpectrum ( Channel, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%.

```
Spectra = ComplexSpectrum ( Channel, 2048, 1, 0, 10, 1, 0 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

ComplexSpectrum_exp, ComplexSpectrum_1, AmpSpectrumRMS

ComplexSpectrum_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

An averaged complex spectrum is calculated (harmonics determined as RMS (root-mean-square) values). The averaging is taken of as many spectra as there are windows within the waveform. Calculated by means of FFT.

Declaration:

```
ComplexSpectrum_1 ( InputData, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all spectra
	1 : averaging (arithmetic mean or linear averaging of the real and imaginary parts separately). The mean is taken over all spectra computed.
	2 : Peak Hold Max, maximum values, based on the spectra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the spectra calculated thus far in the algorithm
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged spectrum. The result is complex, i.e., it has a magnitude and a phase. The phase is computed in degrees.

Description:

The linear averaging is performed on the real and imaginary parts separately. For the calculation of an averaged magnitude spectrum the use of the function AmpSpectrumRMS_1() is recommended.

For Peak Hold calculations, the Peak Hold is applied to the magnitude. The phase is averaged, but is thereby rendered irrelevant and should be ignored.

Examples:

```
Spectrum = ComplexSpectrum_1 ( Channel, 1000, 0, 50, 1, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values.

See also:

ComplexSpectrum, ComplexSpectrum_exp, AmpSpectrumRMS_1

ComplexSpectrum_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Complex spectrum (harmonics determined as RMS (root-mean-square) values) using a moving window and exponential averaging. Calculated by means of FFT.

Declaration:

```
ComplexSpectrum_exp ( InputData, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is complex, i.e., it has a magnitude and a phase. The phase is computed in degrees. The result is a segmented waveform, where each segment represents a spectrum.

Description:

The averaging is performed on the real and imaginary parts separately. For the calculation of an averaged magnitude spectrum the use of the function AmpSpectrumRMS_exp() is recommended.

Examples:

```
Spectra = ComplexSpectrum_exp ( Channel, 1000, 0, 50, 2, 40.0, 0 )
```

The channel has a sampling time of 10ms. Therefore, a 1000 point-spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

ComplexSpectrum, ComplexSpectrum_1, AmpSpectrumRMS_exp

CONTINUE

This command ends the current iteration of the loop. If the loop condition is still met, the loop is resumed upon the next iteration.

Declaration:

```
CONTINUE
```

Description

The command can be used within a WHILE-, FOR- or FOREACH-loop.

Examples:

All files in a specified folder and having the extension "*.dat" are found and enumerated in a loop. If the file time is later than a fixed deadline, the file is loaded and edited.

```
list  = FsFileListNew("c:\imc\dat", "*.dat", 0, 0, 0)
count = FsFileListGetCount(list)
deadline = TimeJoin(1, 1, 2012, 0, 0, 0)
FOR i = 1 TO count
   time = FsFileListGetTime(list, i)
   IF time < deadline
      CONTINUE
   END
   ; load and process file
   TxName = FsFileListGetName(list, i)
   fh = FileOpenDSF(TxName, 0)
   ; ...
END
FsFileListClose(list)
```

See also:

WHILE, FOR, FOREACH, BREAK

CONTROL

Available in: Professional Edition and above

Send command via DDE to another application
This command is obsolete; instead of it, the more powerful function DDESend() should be used.

Declaration:

```
CONTROL Application Topic Command VariableName
```

Parameter:

Application	Name of the DDE-application to be addressed
Topic	Designation of the DDE topic
Command	DDE-command in accordance with the format-specifications of the server application
VariableName	Name of the FAMOS-variable for receiving the return value

Description

FAMOS operates as a DDE-client, the application addressed as a DDE-server. The entries "Application" and "topic" cause FAMOS to select another DDE-capable application and request a conversation. If the request is complied with, FAMOS then transfers the desired command. FAMOS waits until the server either confirms having transferred the data or responds in the negative. Then FAMOS ends the conversation.

The optional variable specified under [VariableName] contains the return value of the DDE-application addressed.

- The individual parameters for the Control-command may not contain space characters.
- FAMOS does not check the syntax of the "Command"-entry. Please check that the format of the entry accords with the DDE-specification of the application to be addressed.
- FAMOS generates an error message when the DDE-application (server) addressed either does not respond or does not accept the command.
- If the application (server) addressed is busy, imc FAMOS waits until it is free.
- FAMOS-commands are case-insensitive (upper or lower-case letters are equally valid). However, some DDE applications do distinguish between upper and lower case. Have regard, therefore, for correct spelling.

Examples:

```
CONTROL Trans Timebase [dt=1.5e-3]
```

"Trans" is the name of the receiving DDE application, "Timebase" the name of the topic. The DDEcommand is "dt=1.5e-3" in this case. No return value is requested.

```
CONTROL Trans TimeBase [trigger][arm] ReturnValue
```

'Trans' is the name of the receiving DDE application, 'Timebase' the name of the topic. Two commands are being issued by Control simultaneously, and are written together. The return value of the DDE conversation is contained in the variable 'RetVal'. In this way, the value is made available for checking.

See also:

DDESend, DDEInq, DDESet, REQUEST

ConvertUnit

Converts a unit and changes the data set's numerical values/characteristic parameters accordingly.

Declaration:

```
ConvertUnit ( Data, TxUnit, SvChoice ) -> Result
```

Parameter:

Data	Data set to be converted
TxUnit	Unit into which the conversion is made. If special values such as "SI+", then conversion is performed from the current unit to an unsigned SI-unit.
SvChoice	Selection of the unit
	0 : X-unit for 1-component data. Unit of the X-component for XY-data. Unit of the phase/imaginary part for complex data.
	1 : Y-unit with 1-component data. Unit of the Y-component for XY-data. Unit of the magnitude/real part for complex data.
	2 : Z-unit
	3 : Unit of the parameter for 2-component data
	-1 : Only allowed for the purpose of standardization to SI-units (2nd parameter has a special value such as "SI+"). Then, all units existing in the data set are standardized.
Result	
Result	Converted data set

Description:

The function converts a unit in the data set and changes the numerical value accordingly.

Typical applications:

- Conversion of unit prefixes, e.g. 'mV' to 'V' => all numerical values are divided by 1000.
- Conversion of related units, e.g. '&deg;C' to 'K' (Kelvin) => 273.15 is added to all target values.

This function is particularly useful in mathematics operation for adapting the parameter before the actual calculation. In particular, this applies to basic arithmetic, but also to integration (see example), differentiation and FFT (time axis in s).

The current unit and the target unit must be mutually convertible in a sensible way. For example, conversion from 'mA' to 'V' is not sensible and returns an error.

Conversion of temperature units:

Temperatures are converted in conjunction with their offsets, e.g.

```
T [K] = T [&deg;C] + 273.15
T [&deg;F] = T [&deg;C] * 1.8 + 32
```

Temperature differentials are mutually converted without offsets:

```
DeltaT [K] = DeltaT [&deg;C]
DeltaT [&deg;F] = DeltaT [&deg;C] * 1.8
```

Once the fractions have been cancelled, the function can distinguish among temperature differentials according to the following indications: the temperature unit appears in the denominator, for instance 1/°C or mV//°C. The temperature unit appears in combination with other units, for instance with °C/ W. It is recommended when calculating temperature differentials, to assign the unit K right away even though °C is also allowed and customary.

Special values of the 2nd parameter

With all special values of the type "SI...", standardization to the appropriate SI-unit is performed. Such prefixes as milli, kilo etc. are eliminated and the numerical values are converted accordingly. The SI-units consist of the basic SI-units A, cd, K, kg, m, mol, s and the derivative SI-units Bq, &deg;C, C, F, Gy, H, Hz, J, kat, lm, lx, Ohm, N, Pa, rad, sr, S, Sv, T, V, W, Wb.

With "SI+" and "SI+D", additional units such as Bark, Bft, dB, Grad (Degree), phon, Scoville, sone are retained.

With the special values, the special handling of temperature is performed as follows:

"SI0"	The temperature unit is K. When a temperature is detected, then 0&deg;C = 273,15K, for example, is used.
"SI0D"	All temperatures detected are interpreted as a temperature differential, where 1K=1&deg;C is applied and K is always generated.
"SI+"	The temperature unit is &deg;C, but K is retained. When a temperature is detected, then 32&deg;F = 0&deg;C, for example, is used.
"SI+D"	All temperatures detected are interpreted as a temperature differential, where 1.8&deg;F=1&deg;C is applied and &deg;C is generated but K retained.

Recognition and conversion of units is governed by a number of presettings which can be set in FAMOS in the dialog 'Extras'/'Options'/'Units', or by means of the function SetOption().

The function can be used on normal (equidistantly sampled) data sets, as well as on XY- and complex data.

Examples:

```
; Signal has the Y-unit 'm/s'
Signal = ConvertUnit(Signal, "km/h", 1)
; => all Y-values are multiplied by 3.6

; Signal has the X-unit 's'
Signal = ConvertUnit(Signal, "min", 0)
; => Normal (1-component) data set: the characteristic parameters x-offset (pretrigger) and x-delta (sampling interval) are divided by 60.
; => XY-data set: all values of the X-component are divided by 60

Signal = ConvertUnit(Signal, "SI0", 1)
; Y-unit 'kV' => Result: 'V', all Y-values multiplied by 1000
; Y-unit 'min' => Result: 's', all Y-values multiplied by 60
; Y-unit '&deg;C' => Result: 'K', to every Y-value, 273.15 is added
; Y-unit 'km/h' => Result: 'm/s', all Y-values are divided by 3.6
; Y-unit 'V/A' => Result: 'Ohm', Y-values unchanged
```

Electric power is calculated. The voltage is measured in mV, the current in mA. For correct calculation, before multiplication, the two readings are converted to base units, and the result is automatically associated with the unit 'W'.

```
P = ConvertUnit(U, "SI0", 1) * ConvertUnit(I, "SI0", 1)
```

An acceleration signal has the Y-unit 'g' (standard gravity), the time axis is stated in seconds. For the calculation of the velocity by means of integration, the signal must first be converted to 'm/s^2'.

```
SetOption("Units.Read.g", "gravity") ; "g" as acceleration (instead of  mass [gram])
v = Int(ConvertUnit(a, "SI0", 1))
```

Conversion of a rotation speed signal (in revolutions per minute) into angular velocity and frequency:

```
tr = ConvertUnit(60'RPM', "Hz", 1) ; => 1Hz
tr = ConvertUnit(60'RPM', "rad/s", 1) ; => 6.2832 rad/s
tr = ConvertUnit(60'RPM', "Degr/s", 1) ; => 360 Degr/s
```

See also:

SetUnit, Unit?

CorrCoeff

Available in: Professional Edition and above

Correlation coefficient, also moving

Declaration:

`CorrCoeff ( Pattern, Test data [, Calculation] [, Parameter] ) -> Result`

Parameter:

Pattern	Data set x, the first of the two data sets to be correlated. For a moving calculation, the (short) pattern or reference data set, whose occurrence is to be located in the test data set.
Test data	Data set y, the second of the two data sets to be correlated. For a moving calculation of the (long) data set, in which the pattern is to be located.
Calculation	Calculation (optional , Default value: 0)
	0 : A correlation coefficient for the two data series is calculated. No moving calculation is performed.
	1 : A series of moving correlation coefficients is determined based on Pattern and on a moving window on Testdata.
	2 : Like "Moving correlation coefficient", but additionally, the result is weighted by a factor SF_rms.
	3 : Like "Moving correlation coefficient", but additionally, the result is weighted by a factor SF_span.
	4 : Like "Moving correlation coefficient", but additionally, the result is weighted by a factor SF_fred.
Parameter	The value required, depending on the calculation type. For "Correlation coefficient F_rms", the minimum RMS-value (>=0) of a window on Testdata. For "Correlation coefficient F_span", the minimum span (>=0) of a window on Testdata. For "Correlation coefficient F_fred", the minimum relative span from 0 to 1 for 0 to 100%. Else 0. (optional , Default value: 0)
Result	
Result	Result

Description:

Correlation coefficient

When calculating the correlation coefficient, the function is called only with the first two arguments.

The calculation of the correlation coefficient r is performed according to the following formula, where n is the number of samples, x means Pattern and y means Testdata.

$$r = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{\sqrt{(n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2)(n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2)}}$$

For a non-moving calculation, the shorter of the two input data Pattern and Testdata determines for how many points the calculation is performed.

For instance, if 2 normal data sets for Pattern and Testdata, each having 1000 readings, are available, then the correlation coefficient is determined over these 1000 values and returned as one value.

The calculation of the correlation coefficients always inclueds subtraction of the mean value. For instance, if Pattern has the values [0, 2, 2, 0], it is equivalent to [-1, 1, 1, -1].

The calculation is performed point-by-point. The sampling interval of Pattern is not taken into account.

The correlation coefficient's value range is from -1 to +1. For a value of +1/-1, there is complete positive/negative linear correlation. A value of 0 means there is absolutely no correlation between the two data sets.

The correlation coefficient is also referred to as the correlation value, product-moment-correlation, Bravais-Pearson correlation or Pearson correlation.

Moving correlation coefficient

In the moving calculation, a window is moved along the data set "Testdata". The width of the window is the length of the data set "Pattern". The first position is the left edge of Testdata. The window then shifts forward point-by-point. For each point, the correlation coefficient is determined. The window is only shifted forward until its edge reaches the right edge of Testdata.

Typically, the function is called with 3 parameters for this purpose, where Calculation is set to = 1.

The calculation of the correlation coefficient r is performed according to the following formula, where k represents the window shifting and n is the length of Pattern (x).

$$r_k = \frac{n\sum_{i=1}^{n} x_i y_{i+k} - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_{i+k})}{\sqrt{(n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2)(n\sum_{i=1}^{n} y_{i+k}^2 - (\sum_{i=1}^{n} y_{i+k})^2)}}$$

With moving calculations, the number of result data is calculated as [Length(Testdata) - Length(Pattern) + 1].

The result may be an empty data set, if it is impossible to calculate a result because the input data are too short.

For a moving calculation, the result has the time base of Testdata, but is shorter.

A moving calculation with a long Pattern data set, and much longer data set Testdata, can require considerable time to calculate.

Formats

The input data sets Pattern and Testdata must be equidistant.

Pattern and Testdata may possess events and segments. If they have events and segments, Pattern and Testdata must have the same structure. Then, when calculating each event/segment of Testdata, the associated event/segment of Pattern is used for the calculation.

Alternatively, Testdata may have events and segments, while Pattern is a normal data set without any events/segments. Then, in the calculation of each event/segment belonging to Testdata, the correlation is calculated with one and the same Pattern.

Special conditions

Optionally, the calculation of an additional factor F can be performed. The moving correlation coefficient is thus weighted (multiplied). The result is r', the correlation coefficient weighted with F:

$$r' = r * F \qquad 0 \leq F \leq 1$$

The purpose of the additional factor is to reduce the correlation for low values. As an illustration, consider a pattern which is to be located in a long signal. At all locations where the correlation coefficient is around 1, this pattern occurs. Now if the signal is a measured signal, it might be noisy, including in the form of chatter of the LSB due to the AD-converter. But exactly such noise, in an otherwise constant signal, can have the same shape as the pattern to be found. Since the definition of the correlation coefficient makes the result independent of the size, there is the appearance of a very high correlation coefficient.

This is where the extra factor plays a role. Below a specifiable boundary, the factor reduces the result, and above it, it retains the calculated correlation coefficient unchanged.

This avoids generating high correlation coefficient values in very small signal regions.

What is considered too "small" is determined according to the various ways of calculating the factor.

If one of the following calculations results in a factor > 1, the factor is limited to 1.

Correlation coefficient F_rms

Within the window under investigation, the mean value of Testdata is determined, and next subtracted from Testdata, and finally the RMS value is computed.

$$F_{rms} = \frac{RMS\big(window - Average(window)\big)}{Parameter}$$

The parameter is interpreted as an RMS-value. If the mean square deviation within the window is smaller than the reference value, reduction (F < 1) of the result is performed.

If the denonimator = 0, the resulting factor is 1.

Correlation coefficient F_span

Within the window under investigation, the amplitude span of Testdata is determined.

$$F_{span} = \frac{Max_{window} - Min_{window}}{Parameter}$$

The parameter is interpreted as a span. If the span within the window is smaller than the reference value, reduction (F < 1) of the result is performed.

If the denonimator = 0, the resulting factor is 1.

Correlation coefficient F_fred

This is an algorithm that works with spans between maximum and minimum. It calculates "F" in order to decrease the value of "r".

Within the window under investigation, the amplitude span of Testdata is determined. This returns Max_window and Min_window

The Minimum and Maximum over the entire data set passed are found. With segments and events, not over each individual segment/event, but rather once over all segments/events. This returns Max_total and Min_total.

$$F_{fred} = \frac{\dfrac{Max_{window} - Min_{window}}{Max_{total} - Min_{total}} - \dfrac{Parameter}{10}}{0.9 \cdot Paramter}$$

The parameter is interpreted as a relative span. The calculation is performed according to the formula. If one of the denominators is = 0, the resulting factor is 1.

Examples:

Calculation of the correlation coefficient between two time series and test for good conformance.

```
CoCo = CorrCoeff ( data1, data2 )
if CoCo > 0.9
end
```

A search for a pattern is to be conducted in a long data set.

```
pattern = [0,0,1,2,3,2,1,0,0]
mvc = CorrCoeff ( pattern, data, 1 )
interesting = xmaxi ( mvc, 0.9 )
```

A search for a pattern is to be conducted in a long data set. However, if it is chatter of the LSB of the measured data, it is to be rejected. The LSB chatter has an RMS-value of about 0.02

```
pattern = [0,0,1,2,3,2,1,0,0]
mvc = CorrCoeff ( pattern, data, 2, 0.05 )
```

A search for a pattern is to be conducted in a long data set. If a section of the data set has a span of less than 5% of the entire span, it is to be assigned less weighting.

```
pattern = [0,0,1,2,3,2,1,0,0]
mvc = CorrCoeff ( pattern, data, 0, 4, 0.05 )
```

See also:

CCF, ACF

cos

Cosine, trigonometric function

Declaration:

```
cos ( Parameter ) -> Result
```

Parameter:

Parameter	Input data (angle). Allowed types: [ND],[XY].
Result	
Result	Cosine of the parameter

Description:

The trigonometric function cos is calculated using radians or degrees, corresponding to the unit of the specified parameter.

The x-coordinate(s) of the results and the parameter are the same.

Remarks

- The parameter of the cos function should not have a unit or the units 'rad', 'degr', or '° '. Any other units will cause an error message to be generated and the unit will be used for the result.
- The parameter may be structured (events/segments).
- The corresponding inverse function is acos.

Examples:

cos(0)= 1, cos(PI/2)= 0, cos(PI)= -1

```
one = cos(0.0)
```

Creates a cosine shaped waveform:

```
NDcos = cos(Ramp(0, 0.2, 100))
```

For the corresponding unit, the parameter is interpreted in degrees.

```
zero = cos(90 '°')
zero = cos(90 'degree')
```

See also:

sin, tan, acos

CrossPowerDS

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Cross Power Density spectrum, using a moving window and linear averaging. Calculated by means of FFT.

Declaration:

```
CrossPowerDS ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [,
Base2] ) -> Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
AveragingType	method of summarizing all spectra
	0 : no averaging
	1 : averaging (arithmetic mean or linear averaging of the real and imaginary parts separately). The number of spectra over which the average is taken is determined by the parameter 'Reduction'.
	2 : Peak Hold Max, from beginning. Maximum values, based on the spectra calculated thus far in the algorithm.
	3 : Peak Hold Max, interval. Maximum values, based on the number of spectra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the spectra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of spectra which the parameter 'Reduction' dictates.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is a segmented waveform, where each segment represents a spectrum. The result is complex, i.e., it has a magnitude and a phase.

Description:

Conjugate complex RMS-spectrum of InputChannel, multiplied by the complex RMS-spectrum of OutputChannel, divided by the frequency line distance.

The linear averaging is performed on the real and imaginary parts separately.

For Peak Hold calculations, the Peak Hold is applied to the magnitude. The phase is averaged, but is thereby rendered irrelevant and should be ignored.

The result is divided by the ENBW (Equivalent noise bandwidth) according to the window type used. E.g. division by 1.5 in the case of a Hanning window.

Examples:

```
CrossPowerDensity = CrossPowerDS ( Force, Movement, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%. A force operates on a mechanical part. The part's movement is measured on the opposite end.

```
CrossPowerDensity = CrossPowerDS ( Force, Movement, 2048, 1, 0, 10, 1, 0 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

CrossPowerDS_exp, CrossPowerDS_1, CrossPowerNorm

CrossPowerDS_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

A mean cross power density is calculated. The averaging is taken of as many spectra as there are windows within the waveform.

Declaration:

```
CrossPowerDS_1 ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) ->
Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all spectra
	1 : averaging (arithmetic mean or linear averaging of the real and imaginary parts separately). The mean is taken over all spectra computed.
	2 : Peak Hold Max, maximum values, based on the spectra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the spectra calculated thus far in the algorithm
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged cross power density spectrum. The result is complex, i.e., it has a magnitude and a phase.

Description:

Conjugate complex RMS-spectrum of InputChannel, multiplied by the complex RMS-spectrum of OutputChannel, divided by the frequency line distance.

The averaging is performed on the real and imaginary parts separately.

For Peak Hold calculations, the Peak Hold is applied to the magnitude. The phase is averaged, but is thereby rendered irrelevant and should be ignored.

The result is divided by the ENBW (Equivalent noise bandwidth) according to the window type used. E.g. division by 1.5 in the case of a Hanning window.

Examples:

```
CrossPowerDensity = CrossPowerDS_1 ( Force, Movement, 1000, 0, 50, 1, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channels contains approx. 20000 measured values. A force operates on a mechanical part. The part's movement is measured on the opposite end.

See also:

CrossPowerDS, CrossPowerDS_exp, CrossPowerNorm_1

CrossPowerDS_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Cross Power Density using a moving window and exponential averaging. Calculated by means of FFT.

Declaration:

```
CrossPowerDS_exp ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [,
Base2] ) -> Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is a segmented waveform, where each segment represents a spectrum. The result is complex, i.e., it has a magnitude and a phase.

Description:

Conjugate complex RMS-spectrum of InputChannel, multiplied by the complex RMS-spectrum of OutputChannel, divided by the frequency line distance.

The result is divided by the ENBW (Equivalent noise bandwidth) according to the window type used. E.g. division by 1.5 in the case of a Hanning window.

Examples:

```
CrossPowerDensity = CrossPowerDS_exp ( Force, Movement, 1000, 0, 50, 2, 40.0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%. A force operates on a mechanical part. The part's movement is measured on the opposite end. Both channels have a sampling time of 10ms. Therefore, a spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

CrossPowerDS, CrossPowerDS_1

CrossPowerNorm

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Normalized cross power density spectrum using a moving window and linear averaging. Calculated by means of FFT. Arithmetical (linear) averaging is performed.

Declaration:

```
CrossPowerNorm ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping, Reduction [, Base2] ) ->
Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is a segmented waveform, where each segment represents a spectrum. The result is complex, i.e., it has a magnitude and a phase.

Description:

Conjugate complex RMS-spectrum of InputChannel, multiplied by the complex RMS-spectrum of OutputChannel.

The result is normalized to the product of the input signals' RMS values. Range of the result: -1 .. +1

The channel's RMS values are calculated after the moving window is applied.

The averaging is performed on the real and imaginary parts separately.

Arithmetical (linear) averaging is performed.

Examples:

```
NormCrossPower = CrossPowerNorm ( Force, Movement, 1000, 0, 50, 1, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%. A force operates on a mechanical part. The part's movement is measured on the opposite end.

```
NormCrossPower = CrossPowerNorm ( Force, Movement, 2048, 1, 0, 10, 0 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

CrossPowerNorm_1, CrossPowerDS

CrossPowerNorm_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Normalized cross power spectrum, using a moving window and linear averaging. Calculated by means of FFT.

Declaration:

```
CrossPowerNorm_1 ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping [, Base2] ) -> Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged cross power density spectrum. The result is complex, i.e., it has a magnitude and a phase.

Description:

Conjugate complex RMS-spectrum of InputChannel, multiplied by the complex RMS-spectrum of OutputChannel.

The result is normalized to the product of the input signals' RMS values. Range of the result: -1 .. +1

The channel's RMS values are calculated after the moving window is applied.

The averaging is performed on the real and imaginary parts separately.

Arithmetical (linear) averaging is performed.

The averaging is taken of as many spectra as there are windows within the waveform.

Examples:

```
NormCrossPower = CrossPowerNorm_1 ( Force, Movement, 1000, 0, 50, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values. A force operates on a mechanical part. The part's movement is measured on the opposite end.

See also:

CrossPowerNorm, CrossPowerDS_1

CURVESETUP

Displays a waveform in a curve window configuration
This command is obsolete; instead of it, the more powerful function CwLoadCCV() should be used.

Declaration:

```
CURVESETUP Variable Filename
```

Parameter:

Variable	Name of the variable to be displayed
Filename	Name of the curve configuration file to be loaded

Description

A waveform is displayed as a curve window in a predefined curve window configuration.

The file with the curve configuration (*.ccv) must have been created in a curve window using the menu option 'Save' (Curve Window documentation).

- A complete pathname can be specified when the file is not located in the default directory.
- The default folder for curve configurations can be set in the dialog "Options"/"Folders".

Examples:

```
CURVESETUP slope config1
CURVESETUP slope config1.ccv
CURVESETUP slope c:\imc\ccv\config1.ccv
```

Three options to load a curve window configuration and display in the configuration "Slope".

See also:

CwLoadCCV

Cut

Cuts a section of a data set at specified X-boundaries.

Declaration:

```
Cut ( Data, SvIntervalStart, SvIntervalEnd [, IntervalMeaning] ) -> Repl
```

Parameter:

Data	Data set to be cut; allowed types: [ND],[XY].
SvIntervalStart	Coordinate of the first value to be excised
SvIntervalEnd	Coordinate of the last value to be excised
IntervalMeaning	Sets how the specified interval boundaries are to be interpreted (as x-coordinates or as absolute time). (optional , Default value: 0)
	0 : As x-coordinate
	1 : As absolute time
Repl	
Repl	Portion of the parameter data set

Description:

A section of a long data set can be copied from out of the rest of the data set for further processing. Specify this section by entering its beginning and end as the second and third parameters. These boundaries are entered as x-coordinates.

Normal Data sets::

If the specified beginning or end of the section does not lie within the domain of the parameter data set, the data set is extended accordingly by filling with zeros.

If the front end interval boundary does not exactly coincide with a sample value belonging to the data set, the system rounds off to the previous sample value. A slight margin is applied. Thus if the value to be found is only slightly lower than a sample value, the value at this position is still returned. The purpose of this behavior is to compensate for numerical deviations in cases of previously computed interval boundaries.

For IntervalInterpretation == 0 (stated in x-coordinates): A specified x-value is also considered an exact match with a sample value if it is located less than 1/10000 of a sample interval before this sample value.

For IntervalInterpretation == 1 (stated in absolute time): In order to determine the difference by which a start value may deviate from a sample point, and still be counted as an exact match, both 1/10000 of a sampling interval, and the product of the front-end interval boundary with 1e-14 are calculated, and the greater of the two values is used. The second expression may take effect for values of high magnitude, if a current time is specified. LSB-errors cause deviations here in the range of microseconds, so that 1/10000 of the sampling interval as the margin may already be insufficient.

If the back-end interval boundary does not exactly coincide with a sample value belonging to the data set, the next adjacent sample value is used.

XY Data sets:

If the beginning and the end of the section no longer lie within the domain of the originally transferred data set, the results are confined to within the data set's bounds. If one of the x-bounds is not exactly a node of the original data set, its corresponding y-value is determined by linear interpolation.

- The units of the second and third parameters should correspond to the x-unit of the data set.
- The unit of the data set is not changed.
- The x-offset of the resulting data set is adjusted.
- The order of the second and third parameters is irrelevant; the smaller value is always interpreted as the lower boundary.
- Alternatively, you can use the function CutIndex which requires you to specify the indices of the points of the data set as the bounds of the section.
- Another alternative is the function CutDt(). The main functional difference from the function Cut() is that any sample point located at the end of the excerpted region is not included in the result. The function CutDt() is thus often easier to use, when successive sections of a data are to be extracted in a loop, since then no overlapping effects occur at the window boundaries.
- A section of a data set can also be isolated by creating a measurement value window for the curve window and exporting the section between the measurement cursors. The advantage of this method is that the section is selected from the graph; more convenient for direct operation, but inappropriate for automatic calculations in a sequence. Please refer to the Curve Manager manual, section 'Curve Window/Measure' for more information.

Examples:

The section between 10s and 20.5s is cut out from a data set with a domain of 0s ... 100s.:

```
NDpart = Cut(NDdata, 10 's', 20.5 's')
```

A data set with the range 0s ... 100s is extended with zeros on both ends to create a range from -10s to 200s.

```
NDlonger = Cut(NDdata, -10 's', 200 's')
```

The section representing one day within an equidistant data set extending over multiple days:

```
time1 = TimeJoin(18, 7, 2018, 0, 0, 0)
time2 = TimeJoin(19, 7, 2018, 0, 0, 0)
Day_18_07_2018 = Cut(datagroup, time1, time2, 1)
```

Attention: In the example presented, the result will in most cases contain the first measured value on the July 20th, since the next adjacent value is used if the far boundary does not exactly coincide with a sample value.

See also:

CutIndex, CutDt, Value2, ValueIndex, Repl, ReplIndex

CutDt

Excises a section of a data set by specifying the starting point and the (window) width.

Declaration:

```
CutDt ( Data, Start, Width [, StartInterpretation] [, StartRounding] [, Extrapolation] ) -> Repl
```

Parameter:

Data	Data set from which to excerpt. Equidistantly sampled. No events, no segments.
Start	Beginning of the excerpt in x-coordinates or absolute time
Width	Width of the excerpt in x-coordinates. The system rounds to a multiple of the sample interval or of the x-direction increment.
StartInterpretation	Specifies how to interpret the start-parameter (as x-coordinate or as absolute time statement). (optional , Default value: 0)
	0 : As x-coordinate
	1 : As absolute time
StartRounding	Governs how the first value to excise is determined when the specified start value does not exactly coincide with a sample point of the data set (meaning it lies between sample points in the signal). (optional , Default value: 0)
	0 : A sample point closer to the start value is used.
	1 : The sample point directly following the start value is used.
	2 : The sample point directly preceding the start value is used.
Extrapolation	Governs the behavior of the function when the start or end of the region to be excised is located outside of the data set's boundaries. (optional , Default value: 0)
	0 : No extrapolation is performed. Only real values actually lying within the region are returned. The result may be shorter than expected.
	1 : The missing auxiliary values at the beginning or end are initialized as 0.
	2 : The last or respectively first value of the data set is repeated forward/backward.
Repl	
Repl	Portion of the parameter data set

Description:

It is possible to cut out a portion of a data set for separate processing. To designate this section, a start point (stated either in x-coordinates or absolute time) and its length (in x-coordinates) are entered.

This function can only be applied to equidistantly sampled data.

The length specified will be rounded to a multiple of the data set's sampling time.

The length of the result is calculated as the quotient of the width (rounded to a multiple of the sampling interval) and the sampling interval. Thus, the sample value located exactly at the end of the region is not included.

Example: Data set [data]; Sampling interval: 1s; Offset: 0s

```
result = CutDt(data, 0, 10, 0, 0)
```

The result has 10 values (at the x-coordinates 0..9s). The sample value at the location 10s is no longer included.

- The x-offset of the data set generated is modified; the trigger time remains intact.
- The similar functions Cut() and CutIndex() offer alternative ways to specify the region to excise (by means of Start/End-coordinates or Start-Index/Value-Count).
- One main functional difference from the function Cut() is that a sample point located at the end of the excerpt region is not included in the results. The function CutDt() is thus often easier to use when successive sections of a data are to be extracted in a loop, since then no overlapping effects occur at the window boundaries.
- Compared to the function CutIndex(), the function CutDt() is superior when data sets having different trigger times/sampling rate or different x-offsets/x-increments are to be excerpted at the same time.
- You can also generate a data set excerpt by generating a measurement value window in a curve window and exporting the region between the measurement cursors there. This provides the advantage of letting you work in a graphical technique. For this reason, this approach is often more convenient, but not suitable for automatic calculations in a sequence. See the user's manual for the curve manager, chapter 'Curve window/Measure'.

If the parameter [StartRounding] is non-zero, meaning that the preceding or subsequent sample value may be intended for use, and the start value does not exactly coincide with a sample value belonging to the data set, the system applies a slight margin. Thus when [StartRounding] = 2, for instance, if the value to be found is only slightly lower than a sample value, the value at this position is still returned. The purpose of this behavior is to compensate for numerical deviations in cases of previously computed x-coordinates.

In order to determine the difference by which a start value may deviate from a sample point, and still be counted as an exact match, both 1/10000 of a sampling interval, and the product of the start value with 1e-14 are calculated, and the greater of the two values is used. The second expression may take effect for start values of high magnitude, if a current time is specified. LSB-errors cause deviations here in the range of microseconds, so that 1/10000 of the sampling interval as the margin may already be insufficient

Examples:

The data group [datagroup] in the xample below contains multiple channels of a multi-day endurance measurement. The channels' sampling time are different. The measured valued for 3 days in succession are excised:

```
time = TimeJoin(18, 7, 2018, 0, 0, 0)
secondsPerDay = 60*60*24
Day_18_07_2018 = CutDt(datagroup, time, secondsPerDay, 1)
Day_19_07_2018 = CutDt(datagroup, time+ secondsPerDay, secondsPerDay, 1, 1)
Day_20_07_2018 = CutDt(datagroup, time+ 2*secondsPerDay, secondsPerDay, 1, 1)
```

The data set [data] in the example below is to have Offset =0s, Sampling interval =1s.

```
x5_to_14 = CutDt(data, 5, 10)         ; cut 10 samples from x=5 to 14
x5_to_15 =   Cut(data, 5, 15)         ; cut 11 samples from x=5 to 15

x5_to_14 = CutDt(data, 5.2, 10)       ; cut 10 samples from x=5 to 14
x6_to_15 = CutDt(data, 5.2, 10, 0, 1) ; cut 10 samples from x=6 to 15
x5_to_14 = CutDt(data, 5.2, 10, 0, 2) ; cut 10 samples from x=5 to 14
x5_to_15 =   Cut(data, 5.2, 15.2)     ; cut 11 samples from x=5 to 15

x5_to_14 = CutDt(data, 4.7, 10)       ; cut 10 samples from x=5 to 14
x5_to_14 = CutDt(data, 4.7, 10, 0, 1) ; cut 10 samples from x=5 to 14
x4_to_13 = CutDt(data, 4.7, 10, 0, 2) ; cut 10 samples from x=4 to 13
x4_to_15 =   Cut(data, 4.7, 14.7)     ; cut 12 samples from x=4 to 15

x0_to_3  = CutDt(data, -1, 5, 0, 0, 0) ; cut 4 samples from x=0 to 3
xm1_to_3 = CutDt(data, -1, 5, 0, 0, 1); cut 5 samples from x=-1 to 3. xm1_to_3[1] = 0
xm1_to_3 = CutDt(data, -1, 5, 0, 0, 2); cut 5 samples from x=-1 to 3. xm1_to_3[1] = data[1]
```

See also:

CutIndex, Cut, Value2, ValueIndex, Repl, ReplIndex, MatrixPart

CutIndex

Cuts a section of a data set. Borders of the section are to be specified by indices of the data set's data points.

Declaration:

```
CutIndex ( Data, SvStartIndex, SvEndIndex ) -> Repl
```

Parameter:

Data	Data set to be cut; allowed types: [ND],[XY].
SvStartIndex	Index of the first value to be cut
SvEndIndex	Index of the last value to be cut
Repl	
Repl	Portion of the parameter data set

Description:

A portion of a data set is copied. The portion is defined by specifying the indices of its first and of its last point (position). The original data set remains intact.

The position (the index) [SvStartIndex] must lie between 1 and the length of the data set [Data]. Exclusively [SvEndIndex] is cut out. If the two specified indices are the same, then exactly 1 value is copied. If [SvEndIndex] is greater than the data set length, everything up to the end of [Data] is cut.

The data type of the first parameter and that of the result are the same. The x-offset of the data set generated is adapted.

The order of the second and third parameters is irrelevant; the smaller value is always interpreted as the lower boundary.

Alternatively, the functions Cut() or CutDt() can be used, in which the boundaries are specifed as x-coordinates.

In order to query a single value by means of its index int he data set, you can also address the data set's indices in formulas:

```
singleValue = Data[ Index ]
```

A section of a data set can also be isolated by creating a measurement value window for the curve window and exporting the section between the measurement cursors. The advantage of this method is that the section is selected from the graph; more convenient for direct operation, but inappropriate for automatic calculations in a sequence. Please refer to the Curve Manager manual, section 'Curve Window/Measure' for more information.

Examples:

The portion of a data set from its tenth value to the end is coped. The piece is smoothed and copied back into the data set:

```
DataPart = CutIndex(Data, 10, Leng?(Data))
DataPart = Smo5(DataPart)
Data = ReplIndex(Data, DataPart, 10)
```

The following formulas are equivalent:

```
Sample2 = CutIndex(Data, 2, 2)
Sample2 = ValueIndex(Data, 2)
Sample2 = Data[2]
```

See also:

Cut, CutDt, Value2, ValueIndex, Repl, ReplIndex, MatrixPart, SamplesGate

CvAppendMarker

Scope: Curve Windows

This function sets a marker in a curve window.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvAppendMarker ( NcWinChannel, SVx, SvY, Svr, SvCoordinates, SvCurveIndex, TXText, Sv1, Sv2 ) -> SvError
```

Parameter:

NcWinChannel	specifies the curve window
SVx	x-coordinate of the new marker
SvY	y-coordinate of the new marker
Svr	Svr
SvCoordinates	SvCoordinates
	1 : Svx and Svy in physical units
	2 : Svx and Svy in percent of the axis length
SvCurveIndex	1, 2, 3, ... : index of the line used to manage the marker. Important when displaying in several coordinate systems or rescaling the axes. If uncertain, set this value to 1.
TXText	This the text string which appears in the marker.
Sv1	Sv1
Sv2	Sv2
SvError	
SvError	Return is SvError (optional)

Description:

Please use CwNewElement with the parameter Marker; then CwMarkerSet..

The new marker is added to the list of any already existing markers.

Svr = Sv1 = Sv2 = 0, if not specified otherwise

Return:

0: No errors, function successful

1: Error, marker could not be created

Examples:

```
error = CvAppendMarker ( Channel1, 10, 20, 0, 2, 1, "Turn-on Transient", 0, 0 )
```

See also:

CwNewElement, CwMarkerSet

CvAskTitle

Scope: Curve Windows

get title
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvAskTitle ( NcWinChannel, SvTask ) -> TxTitle
```

Parameter:

NcWinChannel	specifies the curve window
SvTask	SvTask
	1 : If the channel NwChannel is displayed in a curve window, the title bar of the curve window is requested.
	2 : Requests the name of the variable NwChannel. The name can be changed using the function CvTitle(). Variable group names are ignored.
TxTitle	
TxTitle	TxTitle

Description:

Please use CwDisplayGetText with the parameter Title for Task 1, CwDataGetText for Task 2.

Examples:

```
Title = CvAskTitle ( x, 1 )
```

See also:

CwDisplayGetText

CvAttrib

Scope: Curve Windows

Specifies a display attribute for the curve window.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvAttrib ( NcWinChannel, SVAttribute )
```

Parameter:

NcWinChannel	specifies the curve window
SVAttribute	SVAttribute

Description:

Please use CwDisplaySet.

SVAttribute

0: standard

1: curves stacked

Add:

0: no grid

2: grid

Add:

0: axis labeling

4: no axis labeling

Add:

00: draw directly to screen

10: use bitmap character (reduces flicker)

Add:

00: do not show trigger

20: show trigger

Add:

00: automatic adaptation to curve

40: no automatic adaptation (only used during imc FAMOS 2.x-compatible operation)

Add:

000:normal x-axis

100: x-axis for scroll mode (only used during imc FAMOS 2.x-compatible operation)

Add:

000: x-axis in seconds

200: x-axis in relative time (hours, minutes,...)

Add:

000: x-axis in seconds

400: x-axis in absolute time (date, time)

Add:

1000*number of symbols (0..99) per display. Only for symbols.

0 is standard for symbols with each measurement value

Examples:

```
CvAttrib ( x, 1+400 )
```

See also:

CwDisplaySet, CwAxisSet

CvConfig

Scope: Curve Windows

Shows channel using configuration file specified in TxFilename.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvConfig ( NcWinChannel, TxFileName ) -> TxError
```

Parameter:

NcWinChannel	specifies the curve window
TxFileName	TxFileName
TxError	
TxError	The return value is an empty text or an error message if an error occurred. (optional)

Description:

Please use CwLoadCCV.

Examples:

```
Error = CvConfig ( x, "x.ccv" )
```

See also:

CwLoadCCV

CvCursor

Scope: Curve Windows

Returns the measurement crosshair position in a measurement window.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvCursor ( NcWinChannel, SvCursor ) -> SVxValue
```

Parameter:

NcWinChannel	specifies the curve window
SvCursor	SvCursor
	1 : x-coordinate of cursor from left mouse button
	2 : x-coordinate of cursor from right mouse button
	3 : the (current) left cursor (smaller parameter)
	4 : the (current) right cursor (larger parameter)
	5 : return value = 1, if measurement window exists, otherwise 0
	6 : return value = 1, if curve window exists, otherwise 0
	7 : return value = 1, if overview window exists, otherwise 0
	8 : return 1 (x axis chosen x-unit), 2 (Date/Time: Absolute), 3 (Days/Hours/Minutes), 4 (Third/Octave Labeling), > 4 for future expansion, 0 (not displayed)
	9 : return 1 (Default), 2 (y-Axes Stacked), 3(Waterfall Diagram), 4 (Last Value As Number), 5 (Color Card), > 5 for future expansion, 0 (not displayed)
	10 : return xmin
	11 : return xmax
	12 : y-coordinate of cursor from left mouse button
	13 : y-coordinate of cursor from right mouse button
	14 : z-coordinate of cursor from left mouse button
	15 : z-coordinate of cursor from right mouse button
	16 : parameter of cursor from left mouse button
	17 : parameter of cursor from right mouse button
SVxValue	
SVxValue	SVxValue

Description:

Please use CwDisplayGet, CwIsWindow for Option 5, CwAxisGet for Options 10, 11.

Returns 0 when no measurement window is present.

Examples:

```
xLeft = CvCursor ( x, 1 )
```

See also:

CwDisplayGet

CvLoadGlobalSetting

Scope: Curve Windows

This function loads a global setting from the file specified in TxFileName into the Curve Manager.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvLoadGlobalSetting ( TxFileName, SvSetting, SvParameter )
```

Parameter:

TxFileName	TxFileName
SvSetting	SvSetting
	1 : All curve windows are immediately displayed using the new color scheme. This includes all of the settings in the color dialog.
	2 : For all subsequent printing procedures, for transfer to the Report Generator, copying to the Clipboard, and graphics export, the new colors will be applied.
	3 : All settings belonging to the dialog <Clipboard settings> are applicable effective immediately. For all subsequent printing procedures, and transfer to the Report Generator, copying to the Clipboard and graphics export, these settings will be applied. However, any reports for which the current settings are not to be applied upon transferring are unaffected.
	4 : Some Settings made in the dialog <curve window presettings...> are loaded. These settings will be used for all curve windows. The font used for screen display is one of these settings.
SvParameter	SvParameter

Description:

Please use CwLoadSettings.

The file is usually expected in the CCV-directory (imc FAMOS). Some of the options affect all open windows. When not otherwise specified, set SvParameter = 0.

Examples:

The file c:\imc\set\color.set was previously saved.

```
CvLoadGlobalSetting ( "..\set\color.set", 1, 0 )
```

See also:

CwLoadSettings

CvPosi

Scope: Curve Windows

Settings for the position and size of a curve window
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvPosi ( NcWinChannel, SvX, SvY, SvdX, SvdY )
```

Parameter:

NcWinChannel	specifies the curve window
SvX	SvX
SvY	SvY
SvdX	SvdX
SvdY	SvdY

Description:

Please use CwPosition or CwDisplaySet with the parameters win.x, win.dx etc..

SvX, SvY: upper left corner of the window

SvdX, SvWdY: size of the window

Examples:

```
CvPosi ( x, 0, 0, 640, 480 )
```

See also:

CwPosition

CvRefDB

Scope: Curve Windows

Defines the reference value for decibel (dB) values.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvRefDB ( NcWinChannel, SvDB, SvTask )
```

Parameter:

NcWinChannel	specifies the curve window
SvDB	SvDB
SvTask	SvTask
	0 : NcWinChannel is set to 0. SvDB is stored and used as the decibel reference value for any new curve windows.
	1 : NcWinChannel specifies the curve window and SvDB receives the new decibel reference value

Description:

Please use CwDisplaySet.

Before this function is called for the first time, the default value set in the curve window preferences is used. Only numbers greater than zero may be specified. The new reference value is assigned to SvDB and is only implemented in curve windows. It is not used for mathematical calculations.

Examples:

```
CvRefDB ( 0, 0.075, 0 )
CvRefDB ( RISE, 378.0, 1 )
```

See also:

CwDisplaySet

CvReplaceChannel

Scope: Curve Windows

Replace channel
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvReplaceChannel ( NcWinChannel, NwReplacement, TxOldName ) -> NumberReplaced
```

Parameter:

NcWinChannel	specifies the curve window
NwReplacement	Replacing channel
TxOldName	Old name
NumberReplaced	
NumberReplaced	Returns the number of channels replaced. (optional)

Description:

Please use CwReplace.

In a curve window, NwWindowChannel, a channel called TxOldName is displayed. This channel is to be replaced by NwReplacement. As a result, the channel NwReplacement is displayed instead of TxOldName. If the channel is not currently present, the function has no effect. Separate group names from the channel using ":". This function is useful for replacing channels which are automatically loaded by a curve window configuration file.

Examples:

When the curve configuration x.ccv was created, channels x1 and x2 were displayed.

```
Error = CvConfig ( x, "x.ccv" )
Replaced = CvReplaceChannel ( x, Channel1, "x1" )
Replaced = CvReplaceChannel ( x, Channel2, "x2" )
```

These are replaced by Channel1 and Channel2 in the curve window

See also:

CwReplace

CvSave

Scope: Curve Windows

Saves the configuration of the curve window specified by NcWinChannel to the file indicated in TxFilename.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvSave ( NcWinChannel, TxFileName ) -> TxError
```

Parameter:

NcWinChannel	specifies the curve window
TxFileName	TxFileName
TxError	
TxError	The return value is an empty text or an error message if an error occurred. (optional)

Description:

Please use CwSaveCCV.

The standard directory is curves configuration directory (ccv) from imc FAMOS and the standard file extension is ".CCV". The return value is either an empty text or, when an error was made, an error message.

Examples:

```
Error = CvSave ( SLOPE, "an.ccv" )
```

See also:

CwSaveCCV

CvSetCursor

Scope: Curve Windows

A measurement cursor (crosshair) is set to a specific position in a previously opened measurement window.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvSetCursor ( NcWinChannel, SVx, SvPara1, SvPara2, SvRightLeft ) -> SvError
```

Parameter:

NcWinChannel	specifies the curve window
SVx	The new x-coordinate of the cursor rounded to the nearest pixel. The x-axis must already be scaled to display this coordinate. For example, if the x-axis is displayed in absolute time, then the x-coodinate also has to be specified in absolute time. With XY-plots, specify the parameter instead. In this latter case, the coordinate is rounded to a simple fraction of the parameter increment. NOTE: Do not specify coordinates in dB, even if the window is scaled in dB.
SvPara1	SvPara1
SvPara2	SvPara2
SvRightLeft	SvRightLeft
	1 : Cursor assigned to left mouse button
	2 : Cursor assigned to right mouse button
SvError	
SvError	Return is SvError (optional)

Description:

Please use CwDisplaySet with the parameters measure.*.

The curve window must not be frozen, i.e. do not use CvUpdate(0).

SvPara1 = SvPara2 = 0, if not specified otherwise

On colormap display SvPara1 is the y coordinate. If the window is scaled in dB, the value will not be given in dB.

Return:

0: No errors, function successful

1: Measurement cursor could not be set to the specified value.

Examples:

```
error = CvSetCursor ( Channel1, 0.1, 0, 0, 1 )
```

See also:

CwDisplaySet

CvTitle

Scope: Curve Windows

set title
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvTitle ( NcWinChannel, TxTitle, SvTask )
```

Parameter:

NcWinChannel	specifies the curve window
TxTitle	title
SvTask	SvTask
	1 : If the channel NcChannel is displayed in a curve window, the title bar is set to TxTitle. This setting is valid only as long as the curve window exists; when it is closed, the setting is lost.
	2 : The variable name of the channel specified by NcChannel is changed in all curve windows. The new name for NcChannel is used to save, display and export the variable in all further actions.

Description:

Please use CwDisplaySet with the parameter Title for Task 1, Rename for Task 2.

Examples:

```
CvTitle ( x, "Result of spectral analysis:", 1 )
```

See also:

CwDisplaySet

CvUpdate

Blocks updating of the curve window
This function is obsolete; the function CwUpdateEnable() should be used instead.

Declaration:

```
CvUpdate ( SvUpdate )
```

Parameter:

SvUpdate	Updating On/Off
	0 : Blocks updating of the curve window
	1 : Allow updating of the curve window (again)

Description:

If the [SVUpdate] is setting is non-0, then during running of a sequence, WM_PAINT- and other messages are allowed and imc FAMOS ist operational.

Otherwse this is not the case! Thus, it is possible for example to prevent repeated updating of a curve window when redesigning.

Attention:

[SvUpdate] should only be set to 0 before a group of curve wineow configuration functions and subsequently back to 1. Severe malfunctioning of the function due to inappropriate application is possible!!!

Examples:

```
CvUpdate(0)
CvYAxis(...)
CvYAxis(...)
CvUpdate(1)
```

See also:

CwUpdateEnable

CvVar

Scope: Curve Windows

swap variables
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvVar ( NcOld, NcNew )
```

Parameter:

NcOld	NcOld
NcNew	NcNew

Description:

First, the channel with the name specified by NcOld is displayed in a curve window. Then the channel specified by NcNew replaces the old channel. Hence, a new channel is assigned to the curve window.

Examples:

```
CvVar ( Slope, Arches )
```

CvWin

Scope: Curve Windows

Opens or closes measurement or overview windows affiliated with a given curve window.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvWin ( NcWinChannel, SvTask )
```

Parameter:

NcWinChannel	specifies the curve window
SvTask	SvTask
	1 : show measurement window
	2 : close measurement window
	3 : show overview window
	4 : close overview window
	5 : close curve window, not for curves in dialog!
	6 : reduce curve window to icon, not for curves in dialog!
	7 : maximize curve window, not for curves in dialog!
	8 : return curve window to normal size after reducing or maximizing
	9 : make measurement window transparent, only cursors visible
	10 : open navigator window
	11 : open communicator window

Description:

Please use CwAction.

Examples:

```
CvWin ( x, 1 )
```

See also:

CwAction

CvXAxis

Scope: Curve Windows

Defines the range of the x-axis to be displayed.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvXAxis ( NcWinChannel, SvxMin, SvxMax, SVAttribute )
```

Parameter:

NcWinChannel	specifies the curve window
SvxMin	xMin
SvxMax	xMax
SVAttribute	SVAttribute

Description:

Please use CwAxisSet.

SvAttribute

-1: retain old attribute setting

0: linear

1: logarithmic

Add:

00: fixed scaling with xMin, xMax

10: xMin, xMax rounded

20: x-axis automatic, display all curves; xMin, xMax are ignored

30: x-axis automatic, display all curves (rounded); xMin, xMax are ignored

40: like 20

50: like 30

Add:

100*markings (markings >= 2 und <= 99, markings=0 for automatic)

If the x-axis is displayed in absolute time, the function will work in compatibility with long-previous versions: if the variable identifying the window is a data set, the function adds its absolute time to the values specified for xMin and xMax. If this is not the desired behavior, then the absolute time can be set to zero. Or a text variable is used as the reference.

Examples:

```
CvXAxis ( x, 0, 1, 0+10+100*5 )
```

See also:

CwAxisSet

CvYAxis

Scope: Curve Windows

Defines the range of the y-axis to be displayed.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and predecessors!

Declaration:

```
CvYAxis ( NcWinChannel, Channel, NcComponent2, SVyMin, SVyMax, SVAttribute, SvPosition )
```

Parameter:

NcWinChannel	specifies the curve window
Channel	channel to be scaled or modified
NcComponent2	second component in X-Y plots.
SVyMin	yMin
SVyMax	yMax
SVAttribute	SVAttribute
SvPosition	SvPosition

Description:

Please use CwAxisSet.

SvAttribute

-1: keep old attribute setting

-2: channel is removed from curve window; position unimportant

otherwise:

0: real

1: y-axis in XY-display; x-axis is the last channel presently displayed in the window!

2: locus diagram, NcWinChannel must be complex, NcComponent2 = 0

3: X-Y plot: NcWinChannel is x-axis, NcComponent2 is y-axis

4: X-Y plot: NcWinChannel is y-axis, NcComponent2 is x-axis

Add:

00: linear

10: logarithmic

20: logarithmic with decibels

Add:

000: fixed scaling

100: yMin, yMax are rounded

200: automatic

300: automatic with zero line

400: like previous y-axis

Add:

0000: lines

1000: dots

2000: steps

3000: bars

4000: symbols (squares, etc.) joined by lines

5000: symbols not joined by lines

Add:

00000: square

10000: circle

20000: triangle1

30000: triangle2

40000: diamond

50000: +

60000: x

70000: large dot

80000: Horiz. Lines

Add:

100000*markings (markings >= 2 and <= 99, markings =0 for automatic)

SvPosition:

-1 keep old position

-2 append as last curve

otherwise:

1, 2, 3, 4... index of the curve; position at which the new channel appears (1=first curve)

Examples:

channel a in window a is to be displayed with 5 ticks from -10 ... +10

```
CvYaxis ( a, a, 0, -10.0, 10.0, 5*100000, 1 )
```

append channel b to curve window a

```
CvYaxis ( a, b, 0, 0, 0, 200, -2 )
```

append x and y from X-Y plot to curve window a

```
CvYaxis ( a, x, y, 0, 0, 200+3, -2 )
```

See also:

CwAxisSet, CwLineSet

CwAction

Scope: Curve Windows

Performs an action on the selected curve window.

Declaration:

`CwAction ( Action )`

Parameter:

Action	Which action is to be performed?
	"axes.fix" : Fix axes
	"clipboard.copy" : Copy to Clipboard
	"cosys.height.auto" : Sets the heights of all coordinate systems to automatic.
	"delete.harmonic" : Delete all markers which form the harmonics cursor
	"delete.lines" : Removes all lines and data from the curve window
	"delete.markers" : Remove all markers
	"dialog.display" : Starting the dialog: Display
	"dialog.lines" : Starting the dialog: Lines
	"dialog.more channels" : Starting the dialog: More waveforms
	"dialog.x-axis" : Starting the dialog: x-Axis and other axes
	"history.reset" : Delete history
	"link.remove" : Delete x-link to other curve windows
	"map.fit.axes" : With maps from Internet: Alignment of axes for better readability
	"map.load" : Obsolete, please use CwActoinP! Load a file with background picture. The filename has just been set using CwDisplaySet ("map.filename" ...).
	"measure.close" : close measurement window
	"measure.invisible" : An invisible measurement value window is opened but the measurement cursors are displayed
	"measure.show" : show measurement window
	"optimize" : Optimize (delete empty channels)
	"overview.close" : close overview window
	"overview.show" : show overview window
	"print" : Print
	"reset" : Reset
	"slavepointer.reset" : Slave pointer reset
	"start.link select" : Starting the mode: Link with other curve window
	"start.modify values" : Starting the mode: Modify values
	"start.zoom" : Starting the mode: Zoom
	"win.close" : Closes a free-floating curve window
	"win.disable" : Disables the curve window: nolonger operable by mouse (and keyboard
	"win.enable" : Enables the curve window: can be operated by mouse (and keyboard).
	"win.hide" : Hides a free-floating curve window
	"win.icon" : Displays a free-floating curve window as an icon.
	"win.maximize" : Displays a free-floating curve window in Maximize mode
	"win.show" : Shows a free-floating curve window if it was previously hidden.
	"win.sizenormal" : A free-floating curve window displayed as either an icon or maximized is restored to its normal size.
	"win.twin" : Creates a free-floating twin window

	"unzoom" : Rezoom

Description:

Examples:

Close curve window

```
CwSelectWindow("curve1")
CwAction("win.close")
```

See also:

CwActionP

CwActionP

Scope: Curve Windows

Performs an action (with additional parameter) on the selected curve window.

Declaration:

```
CwActionP ( Action, Parameter1, Parameter2 )
```

Parameter:

Action	Which action is to be performed?
	"export.graphics" : Exports the graphic to a file, e.g. .bmp, .png, .jpg or .pdf. The filename is specified in Parameter1.
	"export.pdf.append" : Attaches the graphic to an existing pdf-file. The filename is specified in Parameter1. If the file does not yet exist, it is created.
	"link.set" : Establish x-link to another curve window
	"map.load" : Loads a file with background image. The filename is specified as Parameter1.
Parameter1	Parameter1. Definition depends on the action selected
Parameter2	Parameter2 = 0, if not used

Description:

If graphics are exported, the curve window must be made visible in some cases. E.g. in order to create an exact copy of the screen, the definitive size must be known and therefore the window must be made visible.

Examples:

Establish link between 2 curve windows

```
CwSelectWindow("curve1")
CwActionP("link.set", "curve2", 0)
```

Create PNG-file

```
CwActionP("export.graphics", "c:\1.png", 0)
```

Create PDF-file and append 2nd page

```
CwActionP("export.graphics",   "c:\1.pdf", 0)
CwActionP("export.pdf.append", "c:\1.pdf", 0)
```

See also:

CwAction

CwAxisGet

Scope: Curve Windows

Get axis property

Declaration:

`CwAxisGet ( Property ) -> Value`

Parameter:

Property	Get which property?
	"range" : Range (1 auto, 2 auto with zero, 3 rounding, 4 fixed, 5 like previous)
	"scale" : Scale. (1 default linear, 2 dB, 3 logarithmic, 4 absolute time, 5 relative time, 6 1/3-octaves)
	"max" : Actual maximum value
	"min" : Actual current value of minimum
	"count.data" : Amount of data elements contained in this axis
	"count.line" : Amount of lines contained in this y-axis
	"count.userticks" : Count of user ticks present on this axis
	"description.color" : Color of the description; for format, see rgb(), (-1 automatic)
	"description.distance" : How far from the axis the description should be placed; in mm, also < 0.
	"description.font.size" : Font size for description, in pt, e.g. 8, (0: auto)
	"description.option" : Display description at the axis? (0: no, 1: fixed text, 2: unit, 3: [Unit], 4: unit name, 5: definable with placeholders)
	"description.orientation" : In which direction should the description extend? (0: auto, 1: parallel to axis, 2: perpendicular to axis)
	"description.pos" : Exactly where should the description be placed? (0: auto, 1: centered, 2: flush with axis beginning, 3: flush with axis end)
	"description.symbol" : Display the line's symbol as displayed in a legend? (0: no, 1: yes; in front)
	"description.text" : The text to be displayed; optionally with placeholders
	"direction" : Should the numerical values along the axis increase in reverse direction? Only in special display styles. (0: auto, 1: reverse)
	"exponent" : Power of 10 for scale labeling (-12, -9, ...12), 1000 for auto
	"font.color" : Font color (numerical value at the ticks); for format, see rgb(), (-1: automatic, -3: color of 1st line from axis)
	"font.size" : Font size, in pt, e.g. 8 (0 auto)
	"format.option" : Is the format valid? (0: Auto with three-row time format; 1: Auto, one-row time format; 2: Auto, two-row time format; 3: freely defined format, one row; 4: freely defined format, two rows)
	"labels.end" : Should the labels at the two axis ends be moved so that they do not extend past the end of the axis? (0: auto, 1: always unmoved, 2: move if necessary)
	"line.color" : Color of the axis line; for format see rgb(), (-1: automatic)
	"line.show" : Should the axis line be displayed? (0: auto, 1: yes, 2: no)
	"line.width" : Thickness of axis lines; in mm (0: auto)
	"max.nominal" : Nominal value of the maximum
	"min.nominal" : Nominal value of the minimum
	"orientation.logical" : Logical orientation, direction (1: x, 2: y, 3: z or color axis for Standard and Color Map, or Angle for Polar, 4: color axis for 3D)
	"places.right" : Decimal places: 0..14; -1: automatic
	"position.bot" : Relative position of the axis' bottom end. 0% means all the way down; 50% in the middle.
	"position.place" : Should the axis be positioned to the left or the right of the coordinate system? (0: auto, 1: left, 2: right)
	"position.top" : Relative position of the axis' top end. 100% means all the way up; 50% in the middle.
	"resolution" : Resolution of an axis; generally for y-axis (0: auto; 1: same resolution as x-axis)

	"small ticks.count" : Amount of small ticks, >= 0; -2 auto
	"ticks.count" : Large tick count, > 1. Only valid if the tick option is set to "At axis end with fixed count".
	"ticks.large.length" : Entire length of the large ticks; in mm (-1: auto)
	"ticks.large.width" : Line thickness of the large ticks (0: auto, 1: like axis line, 2: 75% of axis line, 3: 50% of axis line, 4: 25% of axis line)
	"ticks.option" : Tick option (1 count auto at axis end, 2 count fixed at axis end, 3 distance auto, 4 distance fixed)
	"ticks.orientation" : In what direction should the ticks be drawn? Outwards in the direction of the labeling, or inwards toward the coordinate system? (0: auto, 1: outward, 2 inward, 3: out- and inward)
	"ticks.small.length" : Entire length of small ticks in mm (-1: auto)
	"ticks.small.width" : Line thickness of the small ticks (0: auto, 1: like axis line, 2: 75% of axis line, 3: 50% of axis line, 4: 25% of axis line)
	"ticks.spacing" : The fixed distance between ticks. Only valid if the tick option is set to fixed distance.
	"unit.visible" : Units visibility (0: no, 1: auto=yes)
	"userticks" : User ticks; desired? (0: no, 1: additionally; 2: exclusively)
	"userticks.alignX" : User ticks; horiztontal alignment of text at reference point (0: auto; 1: left-aliged; 2: centered; 3: right-aligned; 4 left-aligned extended; 5: centered extended; 6: right-aligned extended; 7: other side). Access to the currently selected user tick
	"userticks.alignY" : User ticks; vertical alignment of text reference point (0: auto; 1: bottom-aligned; 2: centered; 3: top-aligned; 4 bottom-aligned extended; 5: centered extended; 6: top-aligned extended; 7: other side). Access to the currently selected user tick
	"userticks.clip" : User ticks; text clipping. Is labeling omitted when a tick lying outside is not itself drawn? (0: auto; 1: no). Access to the currently selected user tick
	"userticks.exclusive" : User ticks; exclusive. Draw exclusively user-defined tick at pixel position along axis? (0: no, a regular tick could also be drawn; 1: yes, do not draw regular tick there). Access to the currently selected user tick
	"userticks.grid" : User ticks; Gridlines visible? (0: auto; 1: when grid along window; 2: yes; 3: no). Access to the currently selected user tick
	"userticks.Label for raw data" : User ticks; derived from channel's user-defined property "Label for raw data". (0: No; 1: Yes; 2: template). Access to the currently selected user tick
	"userticks.position" : User ticks; Position. At which position (coordinate) is the tick to be attached? Stated as numerical value. Access to the currently selected user tick
	"userticks.position.type" : User ticks; how is the position defined? (0: auto = coordinate in physical units; 1: percentage from 0 to 100 along the axis in direction of increasing axis values). Access to the currently selected user tick
	"userticks.shiftx" : User ticks; shift of the reference point for drawing text, in the x-direction, in mm; positive in rightward direction, negative leftward. Access to the currently selected user tick
	"userticks.shifty" : User ticks; shift of the reference point for drawing text, in the y-direction, in mm; positive in upward direction, negative downward. Access to the currently selected user tick
	"userticks.text.angle" : User ticks; Angle of the text in degrees (-90 .. +90). What is the text's angle from horizontal? Access to the currently selected user tick
	"userticks.text.color" : User ticks; text color; for the format, see rgb(); -1: automatic. Access to the currently selected user tick
	"userticks.text.size" : User ticks; Font size, in pt, e.g. 8 (0: auto). Access to the currently selected user tick
	"userticks.text.style" : User tick font style (0: auto, 1: default, 2: bold, 3: italic, 4: bold and italic; 5: underlined; 6: bold and underlined; 7: italic and underlined; 8: bold and italic and underlined). Access to the currently selected user tick
	"userticks.tick" : User ticks; tick type (0: auto; 1: large tick; 2 small tick; 3: no large tick; 4: no small tick). Access to the currently selected user tick
	"visible" : Visibility of the axis. Only in special situations, e.g. polar plot or z-axis with color palette: 0 auto, 1 visible, 2 invisible
	"width" : Axis width in mm (0: auto)
Value	
Value	The property's value

Description:

Examples:

Get the x-axis value range

```
CwSelectByIndex("x-axis", 1)
```

```
xmin = CwAxisGet("min")
xmax = CwAxisGet("max")
```

See also:

CwAxisSet, CwAxisGetText

CwAxisGetText

Scope: Curve Windows

Get axis text property

Declaration:

```
CwAxisGetText ( Property ) -> Value
```

Parameter:

Property	Get which property?
	"description.text" : The text to be displayed; optionally with placeholders
	"format.text" : Format of tick labeling, e.g. for absolute or relative time: <hh:mm:ss.s> Valid with associated option
	"format.text2" : Format of the labeling of the second row in the time format, e.g. for absolute time: <DD.MM.YYYY>
	"userticks.text" : User ticks; displayed text. Access to the currently selected user tick
Value	
Value	The property's value

Description:

Examples:

Gets the supplemental text of the x-axis

```
CwSelectByIndex("x-axis", 1)
txt = CwAxisGetTxt("description.text")
```

See also:

CwAxisSet, CwAxisGet

CwAxisSet

Scope: Curve Windows

Set axis properties

Declaration:

```
CwAxisSet ( Property, Value )
```

Parameter:

Property	Which property is to be set?
	"range" : Range (1 auto, 2 auto with zero, 3 rounding, 4 fixed, 5 like previous)
	"scale" : Scale. (1 default linear, 2 dB, 3 logarithmic, 4 absolute time, 5 relative time, 6 1/3-octaves)
	"max" : Target value for the Maximum. Only applies if the range is also fixed; see "Range"
	"min" : Target value for the Minimum. Only applies if the range is also fixed; see "Range"
	"count.userticks" : Sets the number of user ticks; this many should be present on this axis.
	"description.color" : Color of the description; for format, see rgb(), (-1 automatic)
	"description.distance" : How far from the axis the description should be placed; in mm, also < 0.
	"description.font.size" : Font size for description, in pt, e.g. 8, (0: auto)
	"description.option" : Display description at the axis? (0: no, 1: fixed text, 2: unit, 3: [Unit], 4: unit name, 5: definable with placeholders)
	"description.orientation" : In which direction should the description extend? (0: auto, 1: parallel to axis, 2: perpendicular to axis)
	"description.pos" : Exactly where should the description be placed? (0: auto, 1: centered, 2: flush with axis beginning, 3: flush with axis end)
	"description.symbol" : Display the line's symbol as displayed in a legend? (0: no, 1: yes; in front)
	"description.text" : The text to be displayed; optionally with placeholders
	"direction" : Should the numerical values along the axis increase in reverse direction? Only in special display styles. (0: auto, 1: reverse)
	"exponent" : Power of 10 for scale labeling (-12, -9, ...12), 1000 for auto
	"font.color" : Font color (numerical value at the ticks); for format, see rgb(), (-1: automatic, -3: color of 1st line from axis)
	"font.size" : Font size, in pt, e.g. 8 (0 auto)
	"format.option" : Is the format valid? (0: Auto with three-row time format; 1: Auto, one-row time format; 2: Auto, two-row time format; 3: freely defined format, one row; 4: freely defined format, two rows)
	"format.text" : Format of tick labeling, e.g. for absolute or relative time: <hh:mm:ss.s> Valid with associated option
	"format.text2" : Format of the labeling of the second row in the time format, e.g. for absolute time: <DD.MM.YYYY>
	"labels.end" : Should the labels at the two axis ends be moved so that they do not extend past the end of the axis? (0: auto, 1: always unmoved, 2: move if necessary)
	"line.color" : Color of the axis line; for format see rgb(), (-1: automatic)
	"line.show" : Should the axis line be displayed? (0: auto, 1: yes, 2: no)
	"line.width" : Thickness of axis lines; in mm (0: auto)
	"places.right" : Decimal places: 0..14; -1: automatic
	"position.bot" : Relative position of the axis' bottom end. 0% means all the way down; 50% in the middle.
	"position.place" : Should the axis be positioned to the left or the right of the coordinate system? (0: auto, 1: left, 2: right)
	"position.top" : Relative position of the axis' top end. 100% means all the way up; 50% in the middle.
	"resolution" : Resolution of an axis; generally for y-axis (0: auto; 1: same resolution as x-axis)
	"small ticks.count" : Amount of small ticks, >= 0; -2 auto
	"ticks.count" : Large tick count, > 1. Only valid if the tick option is set to "At axis end with fixed count".
	"ticks.large.length" : Entire length of the large ticks; in mm (-1: auto)

	"ticks.large.width" : Line thickness of the large ticks (0: auto, 1: like axis line, 2: 75% of axis line, 3: 50% of axis line, 4: 25% of axis line)
	"ticks.option" : Tick option (1 count auto at axis end, 2 count fixed at axis end, 3 distance auto, 4 distance fixed)
	"ticks.orientation" : In what direction should the ticks be drawn? Outwards in the direction of the labeling, or inwards toward the coordinate system? (0: auto, 1: outward, 2 inward, 3: out- and inward)
	"ticks.small.length" : Entire length of small ticks in mm (-1: auto)
	"ticks.small.width" : Line thickness of the small ticks (0: auto, 1: like axis line, 2: 75% of axis line, 3: 50% of axis line, 4: 25% of axis line)
	"ticks.spacing" : The fixed distance between ticks. Only valid if the tick option is set to fixed distance.
	"unit.visible" : Units visibility (0: no, 1: auto=yes)
	"userticks" : User ticks; desired? (0: no, 1: additionally; 2: exclusively)
	"userticks.alignX" : User ticks; horiztontal alignment of text at reference point (0: auto; 1: left-aliged; 2: centered; 3: right-aligned; 4 left-aligned extended; 5: centered extended; 6: right-aligned extended; 7: other side). Access to the currently selected user tick
	"userticks.alignY" : User ticks; vertical alignment of text reference point (0: auto; 1: bottom-aligned; 2: centered; 3: top-aligned; 4 bottom-aligned extended; 5: centered extended; 6: top-aligned extended; 7: other side). Access to the currently selected user tick
	"userticks.clip" : User ticks; text clipping. Is labeling omitted when a tick lying outside is not itself drawn? (0: auto; 1: no). Access to the currently selected user tick
	"userticks.exclusive" : User ticks; exclusive. Draw exclusively user-defined tick at pixel position along axis? (0: no, a regular tick could also be drawn; 1: yes, do not draw regular tick there). Access to the currently selected user tick
	"userticks.grid" : User ticks; Gridlines visible? (0: auto; 1: when grid along window; 2: yes; 3: no). Access to the currently selected user tick
	"userticks.Label for raw data" : User ticks; derived from channel's user-defined property "Label for raw data". (0: No; 1: Yes; 2: template). Access to the currently selected user tick
	"userticks.position" : User ticks; Position. At which position (coordinate) is the tick to be attached? Stated as numerical value. Access to the currently selected user tick
	"userticks.position.type" : User ticks; how is the position defined? (0: auto = coordinate in physical units; 1: percentage from 0 to 100 along the axis in direction of increasing axis values). Access to the currently selected user tick
	"userticks.shiftx" : User ticks; shift of the reference point for drawing text, in the x-direction, in mm; positive in rightward direction, negative leftward. Access to the currently selected user tick
	"userticks.shifty" : User ticks; shift of the reference point for drawing text, in the y-direction, in mm; positive in upward direction, negative downward. Access to the currently selected user tick
	"userticks.text" : User ticks; displayed text. Access to the currently selected user tick
	"userticks.text.angle" : User ticks; Angle of the text in degrees (-90 .. +90). What is the text's angle from horizontal? Access to the currently selected user tick
	"userticks.text.color" : User ticks; text color; for the format, see rgb(); -1: automatic. Access to the currently selected user tick
	"userticks.text.size" : User ticks; Font size, in pt, e.g. 8 (0: auto). Access to the currently selected user tick
	"userticks.text.style" : User tick font style (0: auto, 1: default, 2: bold, 3: italic, 4: bold and italic; 5: underlined; 6: bold and underlined; 7: italic and underlined; 8: bold and italic and underlined). Access to the currently selected user tick
	"userticks.tick" : User ticks; tick type (0: auto; 1: large tick; 2 small tick; 3: no large tick; 4: no small tick). Access to the currently selected user tick
	"visible" : Visibility of the axis. Only in special situations, e.g. polar plot or z-axis with color palette: 0 auto, 1 visible, 2 invisible
	"width" : Axis width in mm (0: auto)
Value	In which way should this property be set?

Description:

Examples:

Select and parameterize an axis

```
CwSelectWindow("curve1")
CwSelectByIndex("y-axis", 1)
CwAxisSet("range", 4)
CwAxisSet("min", -10)
CwAxisSet("max", 10)
```

Select and parameterize the x-axis

```
CwSelectByIndex("x-axis", 1)
CwAxisSet("scale", 4)
CwAxisSet("range", 1)
```

User ticks at x-axis: The mode is defined first. Next the number of user ticks is defined. Then a user tick is selected and its properties are set.

```
CwSelectByIndex("x-axis", 1)
CwAxisSet("userticks", 1 )
CwAxisSet("count.userticks", 2)
CwSelectByIndex("usertick", 1)
CwAxisSet("userticks.position", 3.0)
CwAxisSet("userticks.text", "!")
CwSelectByIndex("usertick", 2)
CwAxisSet("userticks.position", 4.0)
CwAxisSet("userticks.text", "?")
```

See also:

CwAxisGet, CwAxisGetText

CwColorGet

Scope: Curve Windows

Get curve window colors.

Declaration:

```
CwColorGet ( Color, Individualized, Screen ) -> Color
```

Parameter:

Color	Which is the color to get?
	"back" : Background. Color = -2 for transparent background for printer
	"border.bot" : Coord. system (lower right)
	"border.top" : Coord. system (upper left)
	"cos.back" : Coord. system background
	"grid.1" : Main grid
	"grid.2" : Sec. grid
	"leg.back" : Legend background
	"leg.bot" : Legend border (lower right)
	"leg.text" : Legend text
	"leg.top" : Legend border (upper left)
	"num" : Numbers: foreground
	"num.back" : Numbers: background
	"text" : General text
	"trigger" : Trigger and auxiliary lines
	1 .. 24 : Curve index
Individualized	Set individualized color for only the selected curve window, or globally for all?
	"global" : global
	"individual" : Individualized
Screen	Screen or Printer
	"printer" : Printer
	"screen" : Screen
Color	
Color	Color; format see rgb()

Description:

It is only possible to get individualized colors if the curve window is set for individualized colors.

To get individualized colors, the applicable curve window must be selected.

To get a global color, no curve window needs to be selected.

Examples:

```
rgb = CwColorGet("back", "individual", "screen")
```

See also:

CwColorSet

CwColorSet

Scope: Curve Windows

Set curve window colors

Declaration:

```
CwColorSet ( Color, Individualized, Screen, Color )
```

Parameter:

Color	Which color is to be set?
	"back" : Background. Color = -2 for transparent background for printer
	"border.bot" : Coord. system (lower right)
	"border.top" : Coord. system (upper left)
	"cos.back" : Coord. system background
	"grid.1" : Main grid
	"grid.2" : Sec. grid
	"leg.back" : Legend background
	"leg.bot" : Legend border (lower right)
	"leg.text" : Legend text
	"leg.top" : Legend border (upper left)
	"num" : Numbers: foreground
	"num.back" : Numbers: background
	"text" : General text
	"trigger" : Trigger and auxiliary lines
	1 .. 24 : Curve index
Individualized	Set individualized color for only the selected curve window, or globally for all?
	"global" : global
	"individual" : Individualized
Screen	Screen or Printer
	"printer" : Printer
	"screen" : Screen
	"screen+printer" : Screen and Printer
Color	Set this color to which value? For the format, see rgb()

Description:

Individualized colors only take effect once the curve window is set for individualized colors. This can be done, for example, with CwDisplaySet() and "colors.screen.indiv" or "colors.printer.indiv".

If individualized colors are set, the affected curve window must be selected.

Global colors affect all curve windows. To set these, no curve window needs to be selected.

Examples:

Set individualized colors

```
CwDisplaySet("colors.screen.indiv", 1)
CwColorSet("back", "individual", "screen", rgb(255,0,0))
```

Set global colors for printout

```
CwColorSet(1, "global", "printer", rgb(0,0,0))
CwColorSet(2, "global", "printer", rgb(0,0,0))
```

See also:

CwColorGet

CwCosysGet

Scope: Curve Windows

Get property of a coordinate system

Declaration:

```
CwCosysGet ( Property ) -> Value
```

Parameter:

Property	Get which property?
	"count.axis" : Amount of axes contained in this coordinate system
	"count.data" : Amount of data elements contained in this coordinate system
	"count.line" : Amount of lines contained in this coordinate system
	"height.relative" : The configured relative height of the individual coordinate system, in proportion to the total height of all coordinate systems. Between 0 and 1. For Automatic height: -1.
	"pos.dx" : Width of the coordinate system on the screen, stated in pixels
	"pos.dy" : Height of the coordinate system on the screen, stated in pixels
	"pos.x" : Distance between left edge of the coordinate system to the left edge of the client area of the curve window on the screen, stated in pixels
	"pos.y" : Distance between top edge of the coordinate system to the top edge of the client area of the curve window on the screen, stated in pixels
Value	
Value	The property's value

Description:

Examples:

Get count of axes in the coordinate system

```
CwSelectByIndex("cosys", 1)
count = CwCosysGet("count.axis")
```

See also:

CwCosysSet

CwCosysSet

Scope: Curve Windows

Sets a coordinate system's property

Declaration:

```
CwCosysSet ( Property, Value )
```

Parameter:

Property	Which property is to be set?
	"height.relative" : The configured relative height of the individual coordinate system, in proportion to the total height of all coordinate systems. >= 0.001 and <= 1.0. Due to constraints regarding the display options, not any and all specifications can be implemented. In general, only whatever can be set by means of dragging the mouse. If no coordinate system has its individual height, all heights are determined automatically.
Value	In which way should this property be set?

Description:

Examples:

```
CwSelectByIndex("cosys", 1)
CwCosysSet("height.relative", 0.1)
```

See also:

CwCosysGet

CwDataGet

Scope: Curve Windows

Get data element property

Declaration:

`CwDataGet ( Property ) -> Value`

Parameter:

Property	Get which property?
	"component" : Channel component: 0 entire channel, 1 first component (.Y, .R, .B), 2 second component (.X, .I, .P)
	"event.add" : Events every N-th, >= 1 (in Selection mode N)
	"event.align" : Events' temporal arrangement (0 auto, 1 each in correct time, 2 trigger time first, 3 trigger time last, 4 trigger time of first displayed, 5 trigger time of last displayed, 6 trigger time difference to the first, 7 trigger time difference to last, 8 trigger time difference to first displayed, 9 trigger time difference to last displayed)
	"event.count" : Events count (in Selection mode N or N last)
	"event.index" : Events start index, >= 1 (in Selection mode 1, N, N-th from back)
	"event.select" : Events Selection mode (0 auto, 1 all, 2 last, 3 N last, 4 one, 5 N, 6 N-th from back). Upon activating Selection mode, the other properties for the event selection are reset. The Selection mode must be activated before other properties define the event selection in more detail.
	"function" : Purpose of the data element: 0 Standard, 1 color information for preceding line (color palette), 2 Boxplot Quartile, 3 Boxplot Whisker
	"instrument.fill" : Instrument: Fill (0: auto, 1: from below, 2: from above, 3: from the middle, 4: from y=0, 5 min/max span, percentage)
	"instrument.color.base" : Instrument: Main bar color; for format see rgb(); -1: auto
	"instrument.color.scheme" : Instrument: Color scheme (0: auto, 1: one color, 2: three colors, 3: three colors together, 4: two colors, keep overrun)
	"instrument.color.lower" : Instrument: Color below lower limit; for format, see rgb()
	"instrument.color.number" : Instrument: Color of the numerical value; for format, see rgb(); -1: auto
	"instrument.color.text" : Instrument: Color of text; for format see rgb(); -1: auto
	"instrument.color.upper" : Instrument: Color above upper limit; for format, see rgb()
	"instrument.limit.lower" : Instrument: Lower limit, minimum
	"instrument.limit.upper" : Instrument: Upper limit, maximum
	"instrument.slavepointer.color" : Instrument: Slave pointer color; for format, see rgb()
	"instrument.slavepointer.show" : Instrument: Display slave pointer? (0: no, 1: yes)
	"instrument.title.auto" : Instrument: Automatic title? (0: no, 1: yes)
	"instrument.unit.show" : Instrument: Show unit? (0: no, 1: yes)
	"instrument.width" : Instrument: Width in percent (0 to 100)
	"instrument.100.color" : Instrument: 100% values, line color; for format, see rgb()
	"instrument.100.line" : Instrument: 100% values, line? (0: no, 1: yes)
	"instrument.100.+" : Instrument: 100% values, physical value at +100%
	"instrument.100.-" : Instrument: 100% values, physical value at -100%
	"numerical.format" : Numerical representation: Format Option (1: fixed point; 2: floating point; 3: 1-Byte hex; 4: 2-Byte hex; 5: 4-Byte hex; 6: absolute time Auto; 7: absolute time with freely defined format; 8: relative time Auto; 9: rel. time with freely defined format). With Instruments 1 and 2.
	"numerical.places.left" : Numerical representation: Places left of decimal point 1 through 15
	"numerical.places.right" : Numerical representation: Decimal places: 0 through 15. Also with instruments
	"numerical.prefix" : Numerical representation: Unit prefix (100: auto, -12, -9, -6, -3, 0, 3, 6, 9). Also with instruments. auto not always available
	"period.add" : Periods every N-th, >= 1 (with Selection mode, N)

	"period.align" : Periods temporal arrangement (0 auto, 1 x0 to zero, 2 x0 the first period, 3 x0 the last period, 4 each period retains individual x0)
	"period.count" : Periods count (with Selection mode N or N last)
	"period.index" : Periods start index, >= 1 (with Selection mode 1, N, N-th from back)
	"period.length" : Length of one period, >= 1. Only effective if a valid Selection mode for comparison of periods is selected.
	"period.select" : Period Selection mode (7 no comparison of periods, 0 auto, 1 all, 2 last, 3 N last, 4 one, 5 N, 6 N-th from back). Upon activation of Selection mode the start index, count and N-th period are reset. Thus, the Selection mode must be set before further properties define the period selection in greater detail.
	"segment.add" : Segments every N-th, >= 1 (with Selection mode N)
	"segment.align" : Segments temporal arrangement (0 auto, 1 x-coordinate retained, 2 add z to x-coordinate)
	"segment.count" : Segments count (with Selection mode N or N last)
	"segment.index" : Segments start index, >= 1 (with Selection mode 1, N, N-th from back)
	"segment.select" : Segments Selection mode (0 auto, 1 all, 2 last, 3 N last, 4 one, 5 N, 6 N-th from back). Upon activating Selection mode, the properties start index, count and N-th segment are reset. Thus, Selection mode must be reset before further properties define the segment selection in more detail.
Value	
Value	The property's value

Description:

Examples:

Get the component displayed

```
CwSelectByIndex("data", 1)
cmp = CwDataGet("component")
```

See also:

CwDataSet, CwDataGetText

CwDataGetText

Scope: Curve Windows

Get text property of a data element

Declaration:

```
CwDataGetText ( Property ) -> Value
```

Parameter:

Property	Get which property?
	"channelname" : Channel name, e.g. "channel" or for a channel in a group, "group:channel"
	"instrument.title" : Instrument: Title
	"numerical.format.text" : Numerical representation: Format text for associated format option, e.g. <hh:mm:ss.s> for absoluter time
Value	
Value	The property's value

Description:

Examples:

Get the name of the channel displayed

```
CwSelectByIndex("data", 1)
name = CwDataGetText("channelname")
```

See also:

CwDataSet, CwDataGet

CwDataSet

Scope: Curve Windows

Set property of a data element

Declaration:

`CwDataSet ( Property, Value )`

Parameter:

Property	Which property is to be set?
	"component" : Channel component: 0 entire channel, 1 first component (.Y, .R, .B), 2 second component (.X, .I, .P)
	"event.add" : Events every N-th, >= 1 (in Selection mode N)
	"event.align" : Events' temporal arrangement (0 auto, 1 each in correct time, 2 trigger time first, 3 trigger time last, 4 trigger time of first displayed, 5 trigger time of last displayed, 6 trigger time difference to the first, 7 trigger time difference to last, 8 trigger time difference to first displayed, 9 trigger time difference to last displayed)
	"event.count" : Events count (in Selection mode N or N last)
	"event.index" : Events start index, >= 1 (in Selection mode 1, N, N-th from back)
	"event.select" : Events Selection mode (0 auto, 1 all, 2 last, 3 N last, 4 one, 5 N, 6 N-th from back). Upon activating Selection mode, the start index, count and N-th event are reset. The Selection mode must also be activated before further properties define the event selection in more detail.
	"function" : Bedeutung des Datenelementes (0 Standard, 1 Farbinformation zum Vorgänger (Farbpalette), 2 Boxplot Quartil, 3 Boxplot Whisker, 4 Bubbles)
	"instrument.fill" : Instrument: Fill (0: auto, 1: from below, 2: from above, 3: from the middle, 4: from y=0, 5 min/max span, percentage)
	"instrument.color.base" : Instrument: Main bar color; for format see rgb(); -1: auto
	"instrument.color.scheme" : Instrument: Color scheme (0: auto, 1: one color, 2: three colors, 3: three colors together, 4: two colors, keep overrun)
	"instrument.color.lower" : Instrument: Color below lower limit; for format, see rgb()
	"instrument.color.number" : Instrument: Color of the numerical value; for format, see rgb(); -1: auto
	"instrument.color.text" : Instrument: Color of text; for format see rgb(); -1: auto
	"instrument.color.upper" : Instrument: Color above upper limit; for format, see rgb()
	"instrument.limit.lower" : Instrument: Lower limit, minimum
	"instrument.limit.upper" : Instrument: Upper limit, maximum
	"instrument.slavepointer.color" : Instrument: Slave pointer color; for format, see rgb()
	"instrument.slavepointer.show" : Instrument: Display slave pointer? (0: no, 1: yes)
	"instrument.title" : Instrument: Title
	"instrument.title.auto" : Instrument: Automatic title? (0: no, 1: yes)
	"instrument.unit.show" : Instrument: Show unit? (0: no, 1: yes)
	"instrument.width" : Instrument: Width in percent (0 to 100)
	"instrument.100.color" : Instrument: 100% values, line color; for format, see rgb()
	"instrument.100.line" : Instrument: 100% values, line? (0: no, 1: yes)
	"instrument.100.+" : Instrument: 100% values, physical value at +100%
	"instrument.100.-" : Instrument: 100% values, physical value at -100%
	"numerical.calc.type" : Numerical representation: Calculation (1: max, 2: min, 3: mean). Also with instruments
	"numerical.calc.samples" : Numerical representation: Calculation with this many values. Also with instruments
	"numerical.format" : Numerical representation: Format Option (1: fixed point; 2: floating point; 3: 1-Byte hex; 4: 2-Byte hex; 5: 4-Byte hex; 6: absolute time Auto; 7: absolute time with freely defined format; 8: relative time Auto; 9: rel. time with freely defined format). With Instruments 1 and 2.
	"numerical.format.text" : Numerical representation: Format text for associated format option, e.g. <hh:mm:ss.s> for absoluter time

	"numerical.places.left" : Numerical representation: Places left of decimal point 1 through 15
	"numerical.places.right" : Numerical representation: Decimal places: 0 through 15. Also with instruments
	"numerical.prefix" : Numerical representation: Unit prefix (100: auto, -12, -9, -6, -3, 0, 3, 6, 9). Also with instruments. auto not always available
	"period.add" : Periods every N-th, >= 1 (with Selection mode, N)
	"period.align" : Periods temporal arrangement (0 auto, 1 x0 to zero, 2 x0 the first period, 3 x0 the last period, 4 each period retains individual x0)
	"period.count" : Periods count (with Selection mode N or N last)
	"period.index" : Periods start index, >= 1 (with Selection mode 1, N, N-th from back)
	"period.length" : Length of one period, >= 1. Only effective if a valid Selection mode for comparison of periods is selected.
	"period.select" : Period Selection mode (7 no comparison of periods, 0 auto, 1 all, 2 last, 3 N last, 4 one, 5 N, 6 N-th from back). Upon activation of Selection mode the start index, count and N-th period are reset. Thus, the Selection mode must be set before further properties define the period selection in greater detail.
	"segment.add" : Segments every N-th, >= 1 (with Selection mode N)
	"segment.align" : Segments temporal arrangement (0 auto, 1 x-coordinate retained, 2 add z to x-coordinate)
	"segment.count" : Segments count (with Selection mode N or N last)
	"segment.index" : Segments start index, >= 1 (with Selection mode 1, N, N-th from back)
	"segment.select" : Segments Selection mode (0 auto, 1 all, 2 last, 3 N last, 4 one, 5 N, 6 N-th from back). Upon activating Selection mode, the properties start index, count and N-th segment are reset. Thus, Selection mode must be reset before further properties define the segment selection in more detail.
Value	In which way should this property be set?

Description:

Examples:

Change the component of the channel displayed

```
CwSelectByIndex("data", 1)
CwDataSet("component", 1)
```

See also:

CwDataGet, CwDataGetText

CwDeleteElement

Scope: Curve Windows

Deletes the selected element

Declaration:

```
CwDeleteElement ( Element sort )
```

Parameter:

Element sort	What sort of element is to be deleted?
	"axis" : y-axis
	"cosys" : Coordinate system
	"line" : Line
	"marker" : Marker

Description:

If other elements are also included, they are also deleted. Thus for example, when deleting an axis, all the lines it contains and all the data elements they include are also deleted.

The channels (or variables or data) are not deleted.

Some elements such as a last coordinate system or also the x-axis can not be deleted.

Examples:

Select and delete an axis

```
CwSelectByIndex("y-axis", 2)
CwDeleteElement("axis")
```

See also:

CwSelectByIndex, CwNewElement

CwDisplayGet

Scope: Curve Windows

Get curve window property

Declaration:

`CwDisplayGet ( Property ) -> Value`

Parameter:

Property	Get which property?
	"displaymode" : Display type: 1 default, 2 Stacked, 3 Waterfall, 4 Color map, 5 Last value as number, 6 Bar meter, 7 Table, 8 3D, 9 Polar diagram
	"child" : Is the curve window an embedded window? (0 no, free-floating; 1 yes, embedded in the dialog or Panel)
	"color palette" : Color palette with default display: 0 no, 1 yes
	"colors.printer.indiv" : Individualized colors for this window, for printer (0 no, use global colors, 1 yes)
	"colors.printer.pattern" : Other line types (0 no, 1 yes). Only valid with individualized colors for this window for the printer and if the line type is set to "auto".
	"colors.screen.indiv" : Individualized colors for this window, on screen (0 no, use global colors, 1 yes)
	"colors.screen.pattern" : Other line types (0 no, 1 yes). Only valid with individualized colors for this window for the screen and if the line type is set to "auto".
	"cosys.count" : Amount of coordinate systems
	"cosys.max" : Coordinate system maximized (0 no, 1 yes)
	"data.count" : Total number of data elements
	"db.reference" : Reference value for dB calculations
	"grid" : Grid (0 no, 1 yes)
	"header.coordinate.system" : Header or footer; number of the coordinate system starting at 1 (0: auto). Access to the currently selected header.
	"header.count" : Number of texts which are to be displayed as the header, footer or title.
	"header.position" : Header or footer, title; position (0: auto; 1: top left; 2: top center; 3: top right; 4: bottom left; 5: bottom center; 6: bottom right; 7: left center; 8: center; 9: right center; 10: top left of coordinate system; 11: top center of coordinate system; 12: top right of coordinate system; 13: bottom left of coordinate system; 14: bottom center of coordinate system; 15: bottom right of coordinate system; 16: left center of coordinate system; 17: center of coordinate system; 18: right center of coordinate system). Access to currenty selected header
	"header.shiftx" : Header or footer, title; x-offset of text reference point, in mm, positive rightwards, negative leftwards. Access to the currently selected header
	"header.shifty" : Header or footer, title; y-offset of text reference point, in mm, positive upward, negative downward. Access to the currently selected header
	"header.text.angle" : Header or footer, title; text incline in degrees (-90 .. +90). How strongly is the text inclined from horizontal? Access to the currently selected header
	"header.text.color" : Header or footer, title; Text color; for format see rgb(), -1: automatic. Access to the currently selected header
	"header.text.size" : Header or footer, title; size of font in pt, e.g. 8 (0: auto). Access to the currently selected header
	"header.text.style" : Header or footer or title; font style (0: auto; 1: default; 2: bold; 3: cursive; 4: bold and cursive; 5: underlined; 6: bold and underlined; 7: cursive and underlined; 8: bold and cursive and underlined). Access to currently selected header
	"instrument.numval.show" : Instrument: Display numerical value (0: no, 1: left aligned, 2: centered, 3: right aligned)
	"instrument.title.show" : Instrument: Display title? (0: no, 1: left aligned, 2: centered, 3: right aligned)
	"instrument.axis.show" : Instrument: Display axis: (0: no, 1: yes)
	"instrument.tooltip" : Instrument: Display tooltip? (0: no, 1: name)
	"instrument.slavepointer" : Instrument: Slave pointer (1: auto, 2: line)
	"instrument.frame" : Instrument: Frame (1: auto, 2: together)
	"labels" : Labeling (0 no, 1 yes)

	"lastvalue.columns" : Last value as number: Columns (0: no, 1: yes)
	"lastvalue.font.size" : Last value as number: Font size, in pt, e.g. 8 (0: auto)
	"lastvalue.names" : Last value as number: Names? (0: no, 1: yes, 2: comment, 3: name and comment)
	"lastvalue.right" : Last value as number: Rightaligned? (0: no, 1: yes)
	"lastvalue.=" : Last value as number: Equals sign? (0: no, 1: yes)
	"legend.border" : Legend border (0 no, 1 yes)
	"legend.content" : Legend Text (0 channel name, 1 channel name (without group name), 2 channel comment, 3 channel name and comment, 4 channel name (without group name) and comment, 5 channel name without measurement, 6 channel name with index of selected measurement)
	"legend.curvecol" : Legend text in curve color (0 no, 1 yes)
	"legend.display" : Show Legends (0 auto, 1 always, 2 never, 3 if more than 1 curve is present)
	"legend.distx" : Legend horizontal distance from edge [mm] for movable legends
	"legend.disty" : Legend vertical distance from corner [mm] for movable legends
	"legend.font.size" : Legend font size, in pt, e.g. 8 (0: auto)
	"legend.font.style" : Legend font style (0: auto, 1: default, 2: bold, 3: italic, 4: bold and italic; 5: underlined; 6: bold and underlined; 7: italic and underlined; 8: bold and italic and underlined)
	"legend.lines" : Legend line probe (0 no, 1 yes)
	"legend.location" : Legend location (0 top, 1 top of each coord. system; 2 left, 3 left of each coord. system; 4 relative to top left corner; 5 relative to top right corner; 6 relative to bottom left corner; 7 relative to bottom right corner; 8 relative to right edge; 9 relative to left edge; 10 relative to top edge; 11 relative to bottom edge; 12 relative to center; 13 relative to top left corner, multiple; 14 relative to top right corner, multiple; 15 relative to bottom left corner, multiple; 16 relative to bottom right corner, multiple; 17 relative to right edge; 18 relative to left edge, multiple; 19 relative to top edge, multiple; 20 relative to bottom edge, multiple; 21 relative to center, multiple)
	"legend.numerical.display" : Legend with numerical value (0 no, 1 last value as number)
	"legend.numerical.maxdigits" : Legend numerical value max. digits (0..15)
	"legend.numerical.sep" : Legend numerical value with separator (0 no, 1 :, 2 =)
	"legend.numerical.unit" : Legend numerical value with unit (0 no, 1 yes)
	"legend.rowcol" : Legend rows/columns (0 rows, columns automatic; 1 fixed row count; 2 fixed column count)
	"legend.rows" : Legend rows/column count, if fixed number of rows/columns
	"legend.space.bottom" : Legend: Additional space at bottom margin [mm]
	"legend.space.left" : Legend: Additional space at left margin [mm]
	"legend.space.right" : Legend: Additional space at right margin [mm]
	"legend.space.top" : Legend: Additional space at top margin [mm]
	"legend.transparent" : Legend transparent (0 no, 1 yes)
	"line.count" : Total number of lines
	"link.color" : Link: Color of the marking line; for the format, see rgb(), -1 for automatic
	"link.coordinate" : Link: Control via other coordinate (0: auto, 1: x and y interchanged, 2: keep original)
	"link.edge.mouse" : Link: Adjustment of the axes upon moving the mouse to the window edge (0: auto, 1: no, 2: magnify, 3: move)
	"link.edge.position" : Link: Adjustment of the axes when the marking reaches the edge (0: auto, 1: no, 2: magnify, 3: move)
	"link.follow" : Link: This window follows (0: auto, 1: axis follows, 2: line follows)
	"link.influence" : Link: What does the link affect? (0: auto, 1: X-axis, 2: parameters of an XY-display, 3: X-, Y-axes (color map), 4: Y-axis, 5: cross-section)
	"link.linestyle" : Link: Marker line type (0: auto, 1: solid, 2: dotted, 3: dashed, 4..13 other combinations)
	"link.scale" : Link: Upon changing the scaling (0: auto, 1: line follows, 2: line stays at screen position)
	"link.shape" : Link: Graphical shape of the marking (0: auto, 1: line)
	"link.t" : XY-plot's t-parameter at link position; curve window must be linked accordingly
	"link.width" : Link: Thickness of line marking in mm (0: auto)

	"link.x" : X-coordinate at link position; window must be linked accordingly
	"link.y" : Y-coordinate at link position; window must be linked accordingly
	"map.backgnd" : Map display: Background (0 Auto, 1 Map, 2 Background picture , 3 Map from Internet).
	"map.draw" : Map display: Draw background picture (0 only on screen, 1 on printer also)
	"map.filled" : Map display: Which part of the window is to be covered by the picture? (0 only coordinate system, 1 entire window)
	"map.fit" : Map display: Fit background picture (0 auto, 1 fit horizontally, 2 fit vertically, 3 keep size)
	"map.lat1" : Map display: Latitude point 1 in degrees
	"map.lat2" : Map display: Latitude point 2 in degrees
	"map.lon1" : Map display: Longitude point 1 in degrees
	"map.lon2" : Map display: Longitude of point 2 in degrees
	"map.scale.adjust" : Map: Adapt scale. (0 no, 1 x-, y-axes are scaled in accordance with Internet maps)
	"map.x1" : Map display: Relative x-coordinate of point 1 (0.0 left border, 1.0 right border of picture)
	"map.x2" : Map display: Relative x-coordinate of point 2 (0.0 left border, 1.0 right border of picture)
	"map.y1" : Map display: Relative y-coordinate of point 2 (0.0 bottom border, 1.0 top border of picture)
	"map.y2" : Map display: Relative y-coordinate of point 1 (0.0 bottom border, 1.0 top border of picture)
	"marker.count" : Marker count
	"measure.free" : While the measurement window is open, measurement cursors can be moved freely. (0: No; 1: Yes)
	"measure.exist" : Does the measurement value window with measurement cursors exist (2 yes, 1 yes but hidden, 0 no)?
	"measure.p.left" : With the measurement value window open, the cursor parameter (in XY-representation) linked with the left mouse button
	"measure.p.right" : With the measurement value window open, the cursor parameter (in XY-representation) linked with the right mouse button
	"measure.x.left" : With the measurement value window open, the cursor's x-position linked with the left mouse button
	"measure.x.max" : With the measurement value window open, the x-position of the current right cursor (higher x, higher parameter)
	"measure.x.min" : With the measurement value window open, the x-position of the current left cursor (lower x, lower parameter)
	"measure.x.right" : With the measurement value window open, the cursor's x-position linked with the right mouse button
	"measure.y.left" : With the measurement value window open, the y-value of the cursor linked with the left mouse button
	"measure.y.right" : With the measurement value window open, the y-value of the cursor linked with the right mouse button
	"measure.z.left" : With the measurement value window open, the z-value of the cursor linked with the left mouse button
	"measure.z.right" : With the measurement value window open, the z-value of the cursor linked with the right mouse button
	"number.trimm" : Truncated numbers (0 no, 1 yes)
	"opt.on.delete" : Optimize upon deleting data sets (0 no, 1 yes)
	"overview.exist" : Does the overview window exist (1 yes, 0 no)?
	"scroll" : Scroll mode (0 no, 1 scroll, 2 stretch, 3 oscilloscope, 4 fill)
	"showtrigger" : Show trigger line (0 no, 1 yes)
	"splitmode.active" : Active view in Split mode (1 to number of views). 0, if no split mode. E.g. changes of the x-axis value range affect the active view.
	"splitmode.count" : Number of views in split mode. 1 for one single view without split mode
	"splitmode.width" : Split mode width of the active view in percent (0 to 100) of total width
	"suppress.lines.opt" : Optimization in conjunction with hidden lines (0: No, axes and coordinate systems remain visible; 1: Yes, axes and coordinate systems are made as small as possible or even invisible when all associated lines are hidden.)
	"win.client.dx" : With free-floating curve windows, width on the screen of the curve window's client area, stated in pixels
	"win.client.dy" : With free-floating curve windows, height on the screen of the curve window's client area, stated in pixels
	"win.client.x" : Distance between left edge of the client area to the left edge of the curve window on the screen with free-floating curve windows, stated in pixels

	"win.client.y" : Distance between top edge of the client area to the top edge of the curve window on the screen with free-floating curve windows, stated in pixels
	"win.dx" : Width of the curve window on the screen for floating curve windows, stated in pixels
	"win.dy" : Height of the curve window on the screen for floating curve windows, stated in pixels
	"win.x" : Position of the left edge of the curve window on the screen for floating curve windows, stated in pixels
	"win.y" : Position of the top edge of the curve window on the screen for floating curve windows, stated in pixels
	"y-axis.count" : Total number of y-axes
	"3D.angle1" : 3D: first specified angle for defining the perspective, in degrees (length or angle of z-axis)
	"3D.angle2" : 3D: second specified angle for defining the perspective, in degrees (width)
	"3D.angle3" : 3D: third specified angle for defining the perspective, in degrees (rotation)
	"3D.color.fillsymbol" : Color map: color of symbol filling; format see rgb()
	"3D.color.fix1" : Color palette: 1st fixed color; for format see rgb()
	"3D.color.fix2" : Color palette: 2nd fixed color; fof format see rgb()
	"3D.color.fix3" : Color palette: 3rd fixed color; for format see rgb()
	"3D.color.fix4" : Color palette: 4th fixed color; for format see rgb()
	"3D.color.isoback" : Color map: color of ISO line background; for format, see rgb(); -1 for automatic, -2 for transparent
	"3D.color.isoborder" : Color map: color of ISO line edge; for format see rgb(); -1 for automatic, -2 for transparent
	"3D.color.isolines" : Color map and 3D: color of ISO lines; for format see rgb()
	"3D.color.isotext" : Color map: color of ISO line text; for format, see rgb()
	"3D.color.symbolborder" : Color map: ColorSymbolEdge; for format, see rgb(); -1 for automatic, -2 for transparent
	"3D.color bar.width" : Color palette: Color probe width in mm; 0 for automatic
	"3D.color legend" : 3D: color legend shown (0: auto, 1: no, 2: color legend along vertical y-axis, 3: separate color legend at right, 4: separate color legend at left)
	"3D.color pattern" : Color palette: color scheme (0 auto, 1 two fixed colors, 2 three fixed colors, 3 four fixed colors, 4 spectral, 5 b/w compatible, 6 b/w, 7 black blue green yellow red, 8 blue cyan green yellow red, 9 blue green yellow red, 10 black blue pink red yellow white, 11 green cyan blue pink red, 12 black blue cyan green yellow red, 13 black blue red yellow white, 14 black blue cyan white, 15 white yellow red pink blue black, 16 white cyan blue black, 17 white yellow red blue black, 18 four fixed colors + first white, 19 four fixed colors + white in the middle, 20 four fixed colors + last white, 21 four fixed colors + white in front, black behind, 22 four fixed colors + black front, white behind)
	"3D.colors.number" : Color palette: number of colors for gradiated colors (2..1000)
	"3D.font.iso.size" : Color map: font size for labeling of ISO lines, in pts, e.g. 8.
	"3D.grid" : 3D: grid options (0 auto; 1 volume and edge surfaces; 2 volume; 3 volume and edge surfaces, only main grid; 4 volume, only main grid; 5 also at measurement points; 6 only at measurement points; 7 also at measurement points, behind; 8 dotted behind; 9 dotted behind, with edge surfaces; 10 edge surfaces)
	"3D.interpolation" : Color map: interpolation (0 auto, 1 linear, 2 constant)
	"3D.iso.exponent" : Color map: exponent for the labeling of ISO lines (-12, -9, ...9)
	"3D.iso.format" : Color map: ISO text format (0 auto, 1 fixed point, 2 floating point)
	"3D.iso.text" : Color map: labeling of the ISO lines (0 auto, 1 no, 2 yes, 3 as horizontal as possible, 4 as horizontal as possible and unobstructed, 5 as unobstructed as possible, 6 at discrete x-positions, 7 at discrete x-positions and close, 8 at discrete y-positions, 9 at discrete y-positions and close, 10 multiple along long lines , 11 frequent)
	"3D.isolines" : Color map and 3D: ISO lines desired (0 no, 1 yes)
	"3D.lowerlimit" : Color map: Lower limit. Only values above are taken into account.
	"3D.lowerlimit.use" : Color map: observe lower limit (0 no, 1 yes)
	"3D.perspective" : 3D: how the perspective is defined (0 auto, 1 shear with z-axis angle, 2 longhitude and latitude, 3 with rotation)
	"3D.places.right" : Color map: decimal places: 0..15
	"3D.represent" : Color map and 3D: Representation (0 auto, 1 color map with continuous color transitions, 2 color map with gradated colors, 3 symbols with size according to amplitude, 4 symbols with filling like amplitude)
	"3D.sym.mult" : Color map: multiplier for maximum symbol size. Factor by which the symbol size may be larger than the font size

	"3D.symbol.fixed" : Color map: fixed symbol (0 square from middle, 1 square from bottom, 2 square from lower left, 3 square from outside, 4 circle)
	"3D.symbol.var" : Color map: variable symbol (0 circle, 1 square, 2 diamond, 3 filled circle, 4 filled square, 5 filled diamond)
	"3D.z.coordinate.dz" : Color map, 3D: dz with data's range of z-coordinates fixed
	"3D.z.coordinate.mode" : Color map, 3D: Definition of data's z-coordinate range. Segmented data come with their own z-coordinates. With other data formats, z-coordinates may have to be added. (1: Fixed values 0, 1, 2, 3, ..; 2: Fixed values of z0, dz and z-unit, 3: auto, z-coordinates of data are used)
	"3D.z.coordinate.unit" : Color map, 3D: Unit with data's range of z-coordinates fixed
	"3D.z.coordinate.z0" : Color map, 3D: z0 with data's range of z-coordinates fixed
	"polar.skydirection" : Polardiagramm, Anzeige der Himmelsrichtungen festlegen. 0 keine, 4, 8, 16 Himmelsrichtungen
	"polar.displayangles" : Polardiagramm, Anzeige von Winkeln am Kreis, 0 keine, 1: Winkel in Grad [°]
	"polar.spin" : Polardiagramm, Drehrichtung des Winkels, 0 links drehend (mathematisch positiv, 1 rechts drehend (mathematisch negativ)
	"polar.anglezeropos" : Polardiagramm, 0°-Winkelposition, 0 rechts (0°), 1 oben (90°), 2 links (180°), 3 unten (270°), 4 frei definiert
	"polar.angleoffset" : Polardiagramm, Winkeloffset der 0° Position, nur wenn 0°-Winkelposition frei definiert (4) wurde
	"polar.axispos" : Polardiagramm, Position der Y-Achse, 0 innen oben, 1 links oben, 2 rechts oben, 3 innen, 4 links, 5 rechts
	"polar.xasangle" : Polardiagramm, Interpretation der X-Daten als Winkel, 0 kein Winkel, 1 Winkel in °, 2 Winkel in Radiant
Value	
Value	The property's value

Description:

Examples:

Get measurement cursor's x-coordinate

```
x = CwDisplayGet("measure.x.max")
```

See also:

CwDisplaySet, CwDisplayGetText

CwDisplayGetText

Scope: Curve Windows

Get general curve window text property

Declaration:

```
CwDisplayGetText ( Property ) -> Value
```

Parameter:

Property	Get which property?
	"header.text" : Header or footer, title; displayed text. Access to the currently selected header
	"title" : Title bar for free-floating curve window, "<auto>" for automatic determination
Value	
Value	The property's value

Description:

Examples:

Get the curve window's title bar

```
caption = CwDisplayGetText("title")
```

See also:

CwDisplaySet, CwDisplayGet

CwDisplaySet

Scope: Curve Windows

Set general curve window properties

Declaration:

```
CwDisplaySet ( Property, Value )
```

Parameter:

Property	Which property is to be set?
	"displaymode" : Display type: 1 default, 2 Stacked, 3 Waterfall, 4 Color map, 5 Last value as number, 6 Bar meter, 7 Table, 8 3D, 9 Polar diagram
	"color palette" : Color palette with default display: 0 no, 1 yes
	"colors.printer.indiv" : Individualized colors for this window, for printer (0 no, use global colors, 1 yes)
	"colors.printer.pattern" : Other line types (0 no, 1 yes). Only valid with individualized colors for this window for the printer and if the line type is set to "auto".
	"colors.screen.indiv" : Individualized colors for this window, on screen (0 no, use global colors, 1 yes)
	"colors.screen.pattern" : Other line types (0 no, 1 yes). Only valid with individualized colors for this window for the screen and if the line type is set to "auto".
	"cosys.max" : Coordinate system maximized (0 no, 1 yes)
	"db.reference" : Reference value for dB calculations
	"grid" : Grid (0 no, 1 yes)
	"header.coordinate.system" : Header or footer; number of the coordinate system starting at 1 (0: auto). Access to the currently selected header.
	"header.count" : Number of texts which are to be displayed as the header, footer or title.
	"header.position" : Header or footer, title; position (0: auto; 1: top left; 2: top center; 3: top right; 4: bottom left; 5: bottom center; 6: bottom right; 7: left center; 8: center; 9: right center; 10: top left of coordinate system; 11: top center of coordinate system; 12: top right of coordinate system; 13: bottom left of coordinate system; 14: bottom center of coordinate system; 15: bottom right of coordinate system; 16: left center of coordinate system; 17: center of coordinate system; 18: right center of coordinate system). Access to curenty selected header
	"header.shiftx" : Header or footer, title; x-offset of text reference point, in mm, positive rightwards, negative leftwards. Access to the currently selected header
	"header.shifty" : Header or footer, title; y-offset of text reference point, in mm, positive upward, negative downward. Access to the currently selected header
	"header.text" : Header or footer, title; displayed text. Access to the currently selected header
	"header.text.angle" : Header or footer, title; text incline in degrees (-90 .. +90). How strongly is the text inclined from horizontal? Access to the currently selected header
	"header.text.color" : Header or footer, title; Text color; for format see rgb(), -1: automatic. Access to the currently selected header
	"header.text.size" : Header or footer, title; size of font in pt, e.g. 8 (0: auto). Access to the currently selected header
	"header.text.style" : Header or footer or title; font style (0: auto; 1: default; 2: bold; 3: cursive; 4: bold and cursive; 5: underlined; 6: bold and underlined; 7: cursive and underlined; 8: bold and cursive and underlined). Access to currently selected header
	"history.size" : Maximum history memory in MBytes. -1 for automatic
	"instrument.numval.show" : Instrument: Display numerical value (0: no, 1: left aligned, 2: centered, 3: right aligned)
	"instrument.title.show" : Instrument: Display title? (0: no, 1: left aligned, 2: centered, 3: right aligned)
	"instrument.axis.show" : Instrument: Display axis: (0: no, 1: yes)
	"instrument.tooltip" : Instrument: Display tooltip? (0: no, 1: name)
	"instrument.slavepointer" : Instrument: Slave pointer (1: auto, 2: line)
	"instrument.frame" : Instrument: Frame (1: auto, 2: together)
	"labels" : Labeling (0 no, 1 yes)
	"lastvalue.columns" : Last value as number: Columns (0: no, 1: yes)

	"lastvalue.font.size" : Last value as number: Font size, in pt, e.g. 8 (0: auto)
	"lastvalue.names" : Last value as number: Names? (0: no, 1: yes, 2: comment, 3: name and comment)
	"lastvalue.right" : Last value as number: Rightaligned? (0: no, 1: yes)
	"lastvalue.=" : Last value as number: Equals sign? (0: no, 1: yes)
	"legend.border" : Legend border (0 no, 1 yes)
	"legend.content" : Legende Textinhalt (0 Kanalname, 1 Kanalname (ohne Gruppenname), 2 Kommentar des Kanals, 3 Kanalname und Kommentar, 4 Kanalname ohne Gruppenname und Kommentar, 5 Kanalname ohne Messung, 6 Kanalname mit Nummer der selekt. Messung, 7 Kanalname ohne Gruppe und Messung, 8 Kanalanme mit Kommentar ohne Gruppe und Messung)
	"legend.curvecol" : Legend text in curve color (0 no, 1 yes)
	"legend.display" : Show Legends (0 auto, 1 always, 2 never, 3 if more than 1 curve is present)
	"legend.distx" : Legend horizontal distance from edge [mm] for movable legends
	"legend.disty" : Legend vertical distance from corner [mm] for movable legends
	"legend.font.size" : Legend font size, in pt, e.g. 8 (0: auto)
	"legend.font.style" : Legend font style (0: auto, 1: default, 2: bold, 3: italic, 4: bold and italic; 5: underlined; 6: bold and underlined; 7: italic and underlined; 8: bold and italic and underlined)
	"legend.lines" : Legend line probe (0 no, 1 yes)
	"legend.location" : Legend location (0 top, 1 top of each coord. system; 2 left, 3 left of each coord. system; 4 relative to top left corner; 5 relative to top right corner; 6 relative to bottom left corner; 7 relative to bottom right corner; 8 relative to right edge; 9 relative to left edge; 10 relative to top edge; 11 relative to bottom edge; 12 relative to center; 13 relative to top left corner, multiple; 14 relative to top right corner, multiple; 15 relative to bottom left corner, multiple; 16 relative to bottom right corner, multiple; 17 relative to right edge; 18 relative to left edge, multiple; 19 relative to top edge, multiple; 20 relative to bottom edge, multiple; 21 relative to center, multiple)
	"legend.numerical.display" : Legend with numerical value (0 no, 1 last value as number)
	"legend.numerical.maxdigits" : Legend numerical value max. digits (0..15)
	"legend.numerical.sep" : Legend numerical value with separator (0 no, 1 :, 2 =)
	"legend.numerical.unit" : Legend numerical value with unit (0 no, 1 yes)
	"legend.rowcol" : Legend rows/columns (0 rows, columns automatic; 1 fixed row count; 2 fixed column count)
	"legend.rows" : Legend rows/column count, if fixed number of rows/columns
	"legend.space.bottom" : Legend: Additional space at bottom margin [mm]
	"legend.space.left" : Legend: Additional space at left margin [mm]
	"legend.space.right" : Legend: Additional space at right margin [mm]
	"legend.space.top" : Legend: Additional space at top margin [mm]
	"legend.transparent" : Legend transparent (0 no, 1 yes)
	"link.color" : Link: Color of the marking line; for the format, see rgb(), -1 for automatic
	"link.coordinate" : Link: Control via other coordinate (0: auto, 1: x and y interchanged, 2: keep original)
	"link.edge.mouse" : Link: Adjustment of the axes upon moving the mouse to the window edge (0: auto, 1: no, 2: magnify, 3: move)
	"link.edge.position" : Link: Adjustment of the axes when the marking reaches the edge (0: auto, 1: no, 2: magnify, 3: move)
	"link.follow" : Link: This window follows (0: auto, 1: axis follows, 2: line follows)
	"link.influence" : Link: What does the link affect? (0: auto, 1: X-axis, 2: parameters of an XY-display, 3: X-, Y-axes (color map), 4: Y-axis, 5: cross-section)
	"link.linestyle" : Link: Marker line type (0: auto, 1: solid, 2: dotted, 3: dashed, 4..13 other combinations)
	"link.scale" : Link: Upon changing the scaling (0: auto, 1: line follows, 2: line stays at screen position)
	"link.shape" : Link: Graphical shape of the marking (0: auto, 1: line)
	"link.width" : Link: Thickness of line marking in mm (0: auto)
	"map.backgnd" : Map display: Background (0 Auto, 1 Map, 2 Background picture, 3 Map from Internet). When "auto" is used, all "Map" display settings are discarded.
	"map.draw" : Map display: Draw background picture (0 only on screen, 1 on printer also)

	"map.filename" : Obsolete, please use CwActoinP! Map display: Name (including path and extension) of the file from which the picture is imported. The filename will only be used in an immediately subsequent call of CwAction ("map.load").
	"map.filled" : Map display: Which part of the window is to be covered by the picture? (0 only coordinate system, 1 entire window)
	"map.fit" : Map display: Fit background picture (0 auto, 1 fit horizontally, 2 fit vertically, 3 keep size)
	"map.lat1" : Map display: Latitude point 1 in degrees
	"map.lat2" : Map display: Latitude point 2 in degrees
	"map.lon1" : Map display: Longitude point 1 in degrees
	"map.lon2" : Map display: Longitude of point 2 in degrees
	"map.scale.adjust" : Map: Adapt scale. (0 no, 1 x-, y-axes are scaled in accordance with Internet maps)
	"map.x1" : Map display: Relative x-coordinate of point 1 (0.0 left border, 1.0 right border of picture)
	"map.x2" : Map display: Relative x-coordinate of point 2 (0.0 left border, 1.0 right border of picture)
	"map.y1" : Map display: Relative y-coordinate of point 2 (0.0 bottom border, 1.0 top border of picture)
	"map.y2" : Map display: Relative y-coordinate of point 1 (0.0 bottom border, 1.0 top border of picture)
	"measure.free" : While the measurement window is open, measurement cursors can be moved freely. (0: No; 1: Yes)
	"measure.p.left" : With the measurement value window open, the cursor parameter (in XY-representation) linked with the left mouse button
	"measure.p.right" : With the measurement value window open, the cursor parameter (in XY-representation) linked with the right mouse button
	"measure.x.left" : When the measurement value window is open, the x-position of the left mouse button's cursor. Not for XY-displays.
	"measure.x.right" : When the measurement value window is open, the x-position of the right mouse button's cursor. Not for XY-displays.
	"measure.y.left" : When the measurement value window is open, the y-value of the left mouse button's cursor. Only for display modes in which the cursor's y-coordinate can be freely moved, e.g. Color Map.
	"measure.y.right" : When the measurement value window is open, the y-value of the right mouse button's cursor. Only for display modes in which the cursor's y-coordinate can be freely moved, e.g. Color Map.
	"name" : Title of the curve window. Only for special applications in which a free-floating curve window needs a title for identifcation purposes, for instance for RgCurveSet().
	"number.trimm" : Truncated numbers (0 no, 1 yes)
	"opt.on.delete" : Optimize upon deleting data sets (0 no, 1 yes)
	"scroll" : Scroll mode (0 no, 1 scroll, 2 stretch, 3 oscilloscope, 4 fill)
	"shift.in.cfg" : Save the values for time shift and amplitde shift in the configuration (ccv) ? (0 no, 1 yes)
	"showtrigger" : Show trigger line (0 no, 1 yes)
	"splitmode.active" : Active view in Split mode (1 to number of views). 0, if no split mode. E.g. changes of the x-axis value range affect the active view.
	"splitmode.count" : Number of views in split mode. 1 for one single view without split mode
	"splitmode.width" : Split mode width of the active view in percent (0 to 100) of total width
	"suppress.lines.opt" : Optimization in conjunction with hidden lines (0: No, axes and coordinate systems remain visible; 1: Yes, axes and coordinate systems are made as small as possible or even invisible when all associated lines are hidden.)
	"title" : Title bar for free-floating curve window, "<auto>" for automatic determination
	"toolbar.on" : The toolbar with free-floating curve windows for special applications (0 turned off completely, context menu like with embedded windows. 1: on, default)
	"win.dx" : Width of the curve window on the screen for floating curve windows, stated in pixels
	"win.dy" : Height of the curve window on the screen for floating curve windows, stated in pixels
	"win.enable.cursors" : Determines whether the user can drag the cursors to relocate them: measurement cursors, link lines, etc. (0: No; 1: Yes). Has no effect on embedded curve windows in the Design mode.
	"win.x" : Position of the left edge of the curve window on the screen for floating curve windows, stated in pixels
	"win.y" : Position of the top edge of the curve window on the screen for floating curve windows, stated in pixels
	"3D.angle1" : 3D: first specified angle for defining the perspective, in degrees (length or angle of z-axis)

	"3D.angle2" : 3D: second specified angle for defining the perspective, in degrees (width)
	"3D.angle3" : 3D: third specified angle for defining the perspective, in degrees (rotation)
	"3D.color.fillsymbol" : Color map: color of symbol filling; format see rgb()
	"3D.color.fix1" : Color palette: 1st fixed color; for format see rgb()
	"3D.color.fix2" : Color palette: 2nd fixed color; fof format see rgb()
	"3D.color.fix3" : Color palette: 3rd fixed color; for format see rgb()
	"3D.color.fix4" : Color palette: 4th fixed color; for format see rgb()
	"3D.color.isoback" : Color map: color of ISO line background; for format, see rgb(); -1 for automatic, -2 for transparent
	"3D.color.isoborder" : Color map: color of ISO line edge; for format see rgb(); -1 for automatic, -2 for transparent
	"3D.color.isolines" : Color map and 3D: color of ISO lines; for format see rgb()
	"3D.color.isotext" : Color map: color of ISO line text; for format, see rgb()
	"3D.color.symbolborder" : Color map: ColorSymbolEdge; for format, see rgb(); -1 for automatic, -2 for transparent
	"3D.color bar.width" : Color palette: Color probe width in mm; 0 for automatic
	"3D.color legend" : 3D: color legend shown (0: auto, 1: no, 2: color legend along vertical y-axis, 3: separate color legend at right, 4: separate color legend at left)
	"3D.color pattern" : Color palette: color scheme (0: auto; 1: two fixed colors; 2: three fixed colors, 3: four fixed colors; 4: spectral; 5: b/w compatible; 6: b/w; 7: black blue green yellow red; 8: blue cyan green yellow red; 9: blue green yellow red; 10: black blue pink red yellow white; 11: green cyan blue pink red; 12: black blue cyan green yellow red; 13: black blue red yellow white; 14: black blue cyan white; 15: white yellow red pink blue black; 16: white cyan blue black; 17: white yellow red blue black; 18: four fixed colors + first white; 19: four fixed colors + white in the middle; 20: four fixed colors + last white; 21: four fixed colors + white in front, black behind; 22: four fixed colors + black front, white behind). Only in effect if 3D.represent is set to 1 or 2.
	"3D.colors.number" : Color palette: number of colors for gradiated colors (2..1000)
	"3D.font.iso.size" : Color map: font size for labeling of ISO lines, in pts, e.g. 8.
	"3D.grid" : 3D: grid options (0 auto; 1 volume and edge surfaces; 2 volume; 3 volume and edge surfaces, only main grid; 4 volume, only main grid; 5 also at measurement points; 6 only at measurement points; 7 also at measurement points, behind; 8 dotted behind; 9 dotted behind, with edge surfaces; 10 edge surfaces)
	"3D.interpolation" : Color map: interpolation (0 auto, 1 linear, 2 constant)
	"3D.iso.exponent" : Color map: exponent for the labeling of ISO lines (-12, -9, ...9)
	"3D.iso.format" : Color map: ISO text format (0 auto, 1 fixed point, 2 floating point)
	"3D.iso.text" : Color map: labeling of the ISO lines (0 auto, 1 no, 2 yes, 3 as horizontal as possible, 4 as horizontal as possible and unobstructed, 5 as unobstructed as possible, 6 at discrete x-positions, 7 at discrete x-positions and close, 8 at discrete y-positions, 9 at discrete y-positions and close, 10 multiple along long lines , 11 frequent)
	"3D.isolines" : Color map and 3D: ISO lines desired (0 no, 1 yes)
	"3D.lowerlimit" : Color map: Lower limit. Only values above are taken into account.
	"3D.lowerlimit.use" : Color map: observe lower limit (0 no, 1 yes)
	"3D.perspective" : 3D: how the perspective is defined (0 auto, 1 shear with z-axis angle, 2 longhitude and latitude, 3 with rotation)
	"3D.places.right" : Color map: decimal places: 0..15
	"3D.represent" : Color map and 3D: Representation (0 auto, 1 color map with continuous color transitions, 2 color map with gradated colors, 3 symbols with size according to amplitude, 4 symbols with filling like amplitude)
	"3D.sym.mult" : Color map: multiplier for maximum symbol size. Factor by which the symbol size may be larger than the font size
	"3D.symbol.fixed" : Color map: fixed symbol (0 square from middle, 1 square from bottom, 2 square from lower left, 3 square from outside, 4 circle)
	"3D.symbol.var" : Color map: variable symbol (0 circle, 1 square, 2 diamond, 3 filled circle, 4 filled square, 5 filled diamond)
	"3D.z.coordinate.dz" : Color map, 3D: dz with fixed range of z-coordinates of data. Only applies if "3D.z.coordinate.mode" has been set to 2.
	"3D.z.coordinate.mode" : Color map, 3D: Definition of data's z-coordinate range. Segmented data come with their own z-coordinates. With other data formats, z-coordinates may have to be added. (1: Fixed values 0, 1, 2, 3, ..; 2: Fixed values of z0, dz and z-unit, 3: auto, z-coordinates of data are used)
	"3D.z.coordinate.unit" : Color map, 3D: Unit with data's range of z-coordinates fixed. Only applies if "3D.z.coordinate.mode" has been set to 2.

	"3D.z.coordinate.z0" : Color map, 3D: z0 with data's range of z-coordinates fixed. Only applies if "3D.z.coordinate.mode" has been set to 2.
	"polar.skydirection" : Polardiagramm, Anzeige der Himmelsrichtungen festlegen. 0 keine, 4, 8, 16 Himmelsrichtungen
	"polar.displayangles" : Polardiagramm, Anzeige von Winkeln am Kreis, 0 keine, 1: Winkel in Grad [°]
	"polar.spin" : Polardiagramm, Drehrichtung des Winkels, 0 links drehend (mathematisch positiv, 1 rechts drehend (mathematisch negativ)
	"polar.anglezeropos" : Polardiagramm, Winkeloffset, 0 rechts (0°) , 1 oben (90°), 2 links (180°), 3 unten (270°), 4 frei definiert
	"polar.angleoffset" : Polardiagramm, 0° Winkeloffset
	"polar.axispos" : Polardiagramm, Position der Y-Achse, 0 innen oben, 1 links oben, 2 rechts oben, 3 innen, 4 links, 5 rechts
	"polar.xasangle" : Polardiagramm, Interpretation der X-Daten , 0 kein Winkel, 1 Winkel in °, 2 Winkel als Radiant
Value	In which way should this property be set?

Description:

Examples:

Set the display style to Standard

```
CwSelectWindow("curve1")
CwDisplaySet("displaymode", 1)
```

3D color palette

```
CwSelectWindow("curve1")
CwDisplaySet("displaymode", 8)
CwDisplaySet("3D.color pattern", 1)
CwDisplaySet("3D.color.fix1", rgb(0,255,0))
CwDisplaySet("3D.color.fix1", rgb(0,0,255))
```

Color palette in default display mode

```
CwNewWindow( "1", "show")
CwDisplaySet("displaymode", 1)
CwDisplaySet("color palette", 1)
CwDisplaySet("3D.color pattern",4)
CwDisplaySet("3D.colors.number",10)
CwNewChannel("append last axis", sintest1)
CwNewChannel("append last axis", sintest2)
CwSelectByChannel( "data", sintest2)
CwDataSet("function", 1)
CwSelectByIndex( "z-axis", 1 )
CwAxisSet( "visible", 0)
```

Header or footer or title: First, set the count. Then select and modify the haader.

```
CwDisplaySet("header.count", 1)
CwSelectByIndex("header", 1)
CwDisplaySet("header.text", "Title")
CwDisplaySet("header.text.size", 12)
CwDisplaySet("legend.space.top", 10)
```

See also:

CwDisplayGet, CwDisplayGetText

CwGlobalGet

Scope: Curve Windows

Get global property

Declaration:

```
CwGlobalGet ( Property, Parameter ) -> Value
```

Parameter:

Property	Get which property?
	"caption.option" : Caption of a free-floating curve window. (0 auto, 1 filename of the loaded ccv file)
	"colors.printer.pattern" : Other line types (0 no, 1 yes). Applies to all curve windows for the printer, unless set for individualized colors.
	"colors.screen.pattern" : Other line types (0 no, 1 yes). Applies to all curve windows for the screen, unless set for individualized colors.
	"double click.empty" : Response to double-clicking on empty areas (0: no action, 1: switch selection mode On/Off)
	"export.dpi" : Graphics export resolution in dpi (150, 300, 600, 1200)
	"export.orientation" : PDF export, orientation (0: auto, 1: portrait, 2: landscape)
	"export.pdf.append" : PDF export, attaching pages to existing PDF file (0: no, 1: yes). Only takes effect with manual export.
	"export.pdf.method" : For exporting the curve window's graph to a PDF file, use one of the following methods (0: auto, 1 Bitmap, 2 Vector graphic preferred, best quality, by means of XPS Document Writer)
	"graphics.type" : Graphics type for copying to the clipboard, plus printout and export (0: auto, vector graphic, 1: bitmap pixel graphic, 2: exact screen display)
	"measure.cursor.change" : How the measurement cursor responds to axis change (0: measurement cursor remains at its pixel position, 1: measurement cursor remains at its coordinate)
	"measure.cursor.hori" : Enable horizontal measurement cursor? (0: No; 1: Yes)
	"sample.index" : Index of selected sample. Beginning with 1 for first sample. If data have segments or events, then counting starts at the very first sample.
	"sample.x" : X-component (2nd component/phase, or real part) of selected sample
	"sample.y" : Y-component (1st component/magnitude, or imaginary part) of selected sample
	"sample.z" : Z-coordinate of selected sample. E.g. coordinate of color-determining channel
	"y-axis.navi.x" : How the y-axis changes when navigating in x-direction (0: fix y-axes, 1 y-axes stay automatic)
Parameter	Parameter, depending on property; otherwise 0
Value	
Value	The property's value

Description:

Examples:

A sample is selected in a curve window and afterwards its y value is retrieved via a kit function.

```
y = CwGlobalGet("sample.y", 0)
```

See also:

CwGlobalSet, CwGlobalGetText

CwGlobalGetText

Scope: Curve Windows

Get global text property

Declaration:

```
CwGlobalGetText ( Property, Parameter ) -> Value
```

Parameter:

Property	Get which property?
	"sample.x.name" : For xy plot, the channel name of the x component
	"sample.y.name" : Channel name of selected sample; with xy plots, channel name of the y component
	"sample.z.name" : Name of the channel which determines the color of the selected sample. For a xyz plot, the channel name of z component
Parameter	Parameter, depending on property; otherwise 0
Value	
Value	The property's value

Description:

Examples:

A sample is selected in a curve window and subsequently, a kit function gets the channel name.

```
name = CwGlobalGetText("sample.y.name", 0)
```

See also:

CwGlobalSet, CwGlobalGet

CwGlobalSet

Scope: Curve Windows

Set general global property for all curve windows

Declaration:

```
CwGlobalSet ( Property, Value )
```

Parameter:

Property	Which property is to be set?
	"caption.option" : Caption of a free-floating curve window. (0 auto, 1 filename of the loaded ccv file)
	"close all" : Closes al free-floating curve windows (sets value = 0)
	"colors.printer.pattern" : Other line types (0 no, 1 yes). Applies to all curve windows for the printer, unless set for individualized colors.
	"colors.screen.pattern" : Other line types (0 no, 1 yes). Applies to all curve windows for the screen, unless set for individualized colors.
	"dir.ccv" : Sets the default directory for ccv files. It is valid during the current session. The directory can only be set when working with projects.
	"double click.empty" : Response to double-clicking on empty areas (0: no action, 1: switch selection mode On/Off)
	"export.dpi" : Graphics export resolution in dpi (150, 300, 600, 1200)
	"export.orientation" : PDF export, orientation (0: auto, 1: portrait, 2: landscape)
	"export.pdf.append" : PDF export, attaching pages to existing PDF file (0: no, 1: yes). Only takes effect with manual export.
	"export.pdf.method" : For exporting the curve window's graph to a PDF file, use one of the following methods (0: auto, 1 Bitmap, 2 Vector graphic preferred, best quality, by means of XPS Document Writer)
	"font.name" : Name of the default font for captions in curve windows, e.g. "Arial".
	"font.size" : Size of default font for captions in curve windows, in pts, e.g. 8.
	"graphics.type" : Graphics type for copying to the clipboard, plus printout and export (0: auto, vector graphic, 1: bitmap pixel graphic, 2: exact screen display)
	"infront.of.main" : Works like the FAMOS option "Never hidden by main window": 0: Main window can also hide the curve window; 1: The curve window is never hidden by the main window. Applies to free-floating curve windows created using this Kit. Unless the funciton is called, the value 0 applies.
	"load.show" : How is a curve window to be displayed after a configuration is loaded (0: always hidden, 1: automatic as recorded in the CCV file). Applies to free-floating curve window loaded using this Kit. Unless the function is called, the value 1 applies.
	"measure.cursor.change" : How the measurement cursor responds to axis change (0: measurement cursor remains at its pixel position, 1: measurement cursor remains at its coordinate)
	"measure.cursor.hori" : Enable horizontal measurement cursor? (0: No; 1: Yes)
	"y-axis.navi.x" : How the y-axis changes when navigating in x-direction (0: fix y-axes, 1 y-axes stay automatic)
Value	In which way should this property be set?

Description:

Examples:

Set font size for curve window

```
CwGlobalSet("font.size", 10) ; font size to 12 pt
```

Settings for pdf export

```
CwGlobalSet("graphics.type", 0)
CwGlobalSet("export.dpi", 300)
CwGlobalSet("export.pdf.append", 0)
CwGlobalSet("export.orientation", 1)
CwGlobalSet("export.pdf.method", 2)
CwPrintSet("layout", 0)
CwPrintSet("width", 180)
CwPrintSet("height", 260)
```

See also:

CwGlobalGet, CwGlobalGetText, CwPrintSet

CwIsWindow

Scope: Curve Windows

Determines whether the specified curve window exists.

Declaration:

`CwIsWindow ( Identification ) -> Exist`

Parameter:

Identification	This data set identifies the curve window. With CwSelectMode, the system determined previously how identification is performed.
Exist	
Exist	Exist (=0, if not present; <> 0, if present)

Description:

Examples:

```
data=ramp(0,1,10)
if CwIsWindow(data) <> 0
   ; e.g select the curve window and work with it
end
```

See also:

CwSelectMode

CwLineGet

Scope: Curve Windows

Get line property

Declaration:

`CwLineGet ( Property ) -> Value`

Parameter:

Property	Get which property?
	"type" : Line type: 0 none, 1 lines, 2 steps, 3 dots, 4 vert. line, 5 bars, 7 3D bars, 8 interpolation
	"color.printer" : Line color on the printer; for format see rgb() and -1 for automatic
	"color.screen" : Color of the line on the screen; for format, see rgb().
	"area.border" : Boundary for display of area below curves which are not stair steps or straight lines (0: auto; 1: steps; 2: linear)
	"area.color" : This color is used for displaying areas below curves. For the format, see rgb() and -1 for automatic
	"area.color2" : This is the 2nd color in a color gradient display. For the format, see rgb() and -1 for automatic
	"area.fill" : Color fill for area below the curve (0: none; 1: up to y= zero; 2: to bottom; 3: inside)
	"area.gradient" : Color gradients for the display of surfaces below curves (0: none; 1: from top to bottom; 2: from bottom to top; 3: from left to right; 4: from right to left)
	"auxiliary" : Line is an auxiliary line (0: No; 1: Yes)
	"bar.begin.y" : Bar beginning in y-direction (0: auto, 1: begin at 0, 2: begin at bottom surface, 3: begin below bottom surface)
	"bar.color.type" : Bar color scheme (0: auto, 1: with color gradient, 2: single-color)
	"bar.place.x" : Placement of bar in x-direction (0: auto, 1: between one measurement value and the next, 2: centered around measurement value, 3: aligned to measurement value)
	"bar.place.z" : Placement of bar in z-direction (0: auto, 1: between one measurement value and the next, 2: centered around measurement value, 3: aligned to measurement value)
	"bar.width.x" : Bar width in x-direction, stated in percent: 1 to 100, 0: auto
	"bar.width.z" : Bar width in z-direction, stated in percent: 1 to 100, 0: auto
	"color2" : Color 2, e.g. with 3D the color of the surface. For the format, see rgb() and -1 for automatic.
	"count.data" : Amount of data elements contained in this line
	"cross section.option" : Calculate a cross section of the data? (0: no, 1: x = constant, 2: y = constant)
	"cross section.value" : The value at which the cross section is to be taken
	"effect" : Display the data (in a color map) as a real line, for instance (0: auto; 1: overlapping; 2: upper limit; 3: lower limit; 4: left limit; 5: right limit; 6: outer limit; 7: inner limit; 8: RGB)
	"label.color" : Color of the labeling; for format, see rgb(), (-1: auto)
	"label.font.size" : Font of the line labeling, in pt, e.g. 8
	"label.format" : Numerical format of the labeling (0: auto, 1: fixed point, 2: floating point)
	"label.option" : Should the measurement points be labeled with their numerical values. (0: no, 1: yes)
	"label.placement" : Position of the labeling (0: auto, 1: right, 2: top right, 3: bottom right, 4: left, 5: top left, 6: bottom left, 7: middle, 8: top middle, 9: bottom middle)
	"label.precision" : Precision of the numerical values of the labeling. For fixed-point and floating point, the decimal places 0..14; else number of valid digits: 1..14
	"label.selection" : Value selection for the labeling (0: auto, 1: y, 2: x, 3: parameter, 4: magnitude, 5: phase)
	"legend.append" : Show name extension in legend for line (0 auto, 1 yes, 2 no)
	"legend.channel" : Show channel name in legend for the line (0 auto, 1 yes, 2 no)
	"legend.comment" : Show comment in legend for lines (0 auto, 1 yes, 2 no)
	"legend.group" : Show group names in legend for line (0 auto, 1 yes, 2 no)
	"legend.measurement" : Show measurement in legend for line (0 auto, 1 yes, 2 no, 3 number of selected measurement)

	"legend.numerical.format" : Format of the numerical values in the line legend (0: auto, 1: fixed point, 2: floating point)
	"legend.numerical.precision" : Precision of the numerical values in the line's legend. For fixed- and floating point, the decimal places 0..14; else number of valid digits 1..14
	"legend.numerical.option" : Allow numerical values in line legend (0: auto, 1: no)
	"legend.sample" : Show line probe in legend for lines (0 auto, 1 yes, 2 no)
	"legend.show" : Show legend for line (0: auto, 1: yes, 2: no, 3: text with placeholders)
	"linestyle.printer" : Line type on the printer (0 auto, 1 solid, 2 dotted, 3 dashed, 4..13 other combinations)
	"linestyle.screen" : Line type on the monitor (0 auto, 1 solid, 2 dotted, 3 dashed, 4..13 other combinations)
	"scale.type" : Scaling of line (0: auto; 1: from 0 to 1 relative; 2: from 0 to 1 with offset; 3: mm relative; 4: mm with offset; 5: stretch image; 6: adjust image horizontally; 7: adjust image vertically; 8: show complete image; 9: keep image size)
	"shift.x" : Time shift in physical units. For linear, an offset; for logarithmic, a factor.
	"shift.y" : Amplitude shift in physical units. For linear, an offset; for logarithmic, a factor.
	"suppress" : Hide line (0: No = default; 1: Yes)
	"symbol" : Symbol: 0 none, 1 square, 2 circle, 3-4 triangles, 5 diamond, 6 pulse, 7 X, 8 fat dots, 9 horz. line, 10-11 triangles, 12-13 X, 14 filled diamond, 15 square, 16-19 triangles, 20-21 squares, 22-23 X, 24 minus, 25 empty
	"symbol.count" : Number of symbols, interpreted according to symbol count option
	"symbol.count.opt" : symbol count option: 1: at every sample (symbol count=0); 2: fixed number of symbols to be drawn across the width of the coordinate system (symbol count>0).
	"symbolsize.printer" : Symbol size on printer in mm (-1 auto)
	"symbolsize.screen" : Symbol size on monitor in mm (-1 auto)
	"uncertainty.show" : Display uncertainty (0 no, 1 colored area, 2 line). Please refer to property: Uncertainty.
	"uncertainty.color" : This color used for uncertainty display; for format see rgb() and -1 for automatic
	"uncertainty.selection" : Selection for the measurement uncertainty (0: auto, 1: standard measurement uncertainty, 2: expanded measurement uncertainty). See the properties for Uncertainty and Expanded uncertainty. With "auto", the standard measurement uncertainty is given preference.
	"width.printer" : Line thickness on the printer in mm (-1 auto)
	"width.screen" : Line thickness on the monitor in mm (-1 auto)
	"3D.surface" : 3D: selection of the surface (0 auto, 1 filled wire grid, 2 filled, 3 wire grid, 4 points, 5 wire grid in color palette, 6 Space curve, 7 Space curve with color tones)
Value	
Value	The property's value

Description:

Examples:

Query of which symbol is currently used to represent the line

```
CwSelectByIndex("line", 1)
symbol = CwLineGet("symbol")
```

See also:

CwAxisLineSet

CwLineSet

Scope: Curve Windows

Set line property

Declaration:

```
CwLineSet ( Property, Value )
```

Parameter:

Property	Which property is to be set?
	"type" : Line type: 0 none, 1 lines, 2 steps, 3 dots, 4 vert. line, 5 bars, 7 3D bars, 8 interpolation
	"color.printer" : Line color on the printer; for format see rgb() and -1 for automatic
	"color.screen" : Line color on the monitor; for format see rgb() and -1 for automatic
	"area.border" : Boundary for display of area below curves which are not stair steps or straight lines (0: auto; 1: steps; 2: linear)
	"area.color" : This color is used for displaying areas below curves. For the format, see rgb() and -1 for automatic
	"area.color2" : This is the 2nd color in a color gradient display. For the format, see rgb() and -1 for automatic
	"area.fill" : Color fill for area below the curve (0: none; 1: up to y= zero; 2: to bottom; 3: inside)
	"area.gradient" : Color gradients for the display of surfaces below curves (0: none; 1: from top to bottom; 2: from bottom to top; 3: from left to right; 4: from right to left)
	"auxiliary" : Line is an auxiliary line (0: No; 1: Yes)
	"bar.begin.y" : Bar beginning in y-direction (0: auto, 1: begin at 0, 2: begin at bottom surface, 3: begin below bottom surface)
	"bar.color.type" : Bar color scheme (0: auto, 1: with color gradient, 2: single-color)
	"bar.place.x" : Placement of bar in x-direction (0: auto, 1: between one measurement value and the next, 2: centered around measurement value, 3: aligned to measurement value)
	"bar.place.z" : Placement of bar in z-direction (0: auto, 1: between one measurement value and the next, 2: centered around measurement value, 3: aligned to measurement value)
	"bar.width.x" : Bar width in x-direction, stated in percent: 1 to 100, 0: auto
	"bar.width.z" : Bar width in z-direction, stated in percent: 1 to 100, 0: auto
	"color2" : Color 2, e.g. with 3D the color of the surface. For the format, see rgb() and -1 for automatic.
	"cross section.option" : Calculate a cross section of the data? (0: no, 1: x = constant, 2: y = constant)
	"cross section.value" : The value at which the cross section is to be taken
	"effect" : Display the data (in a color map) as a real line, for instance (0: auto; 1: overlapping; 2: upper limit; 3: lower limit; 4: left limit; 5: right limit; 6: outer limit; 7: inner limit; 8: RGB)
	"label.color" : Color of the labeling; for format, see rgb(), (-1: auto)
	"label.font.size" : Font of the line labeling, in pt, e.g. 8
	"label.format" : Numerical format of the labeling (0: auto, 1: fixed point, 2: floating point)
	"label.option" : Should the measurement points be labeled with their numerical values. (0: no, 1: yes)
	"label.placement" : Position of the labeling (0: auto, 1: right, 2: top right, 3: bottom right, 4: left, 5: top left, 6: bottom left, 7: middle, 8: top middle, 9: bottom middle)
	"label.precision" : Precision of the numerical values of the labeling. For fixed-point and floating point, the decimal places 0..14; else number of valid digits: 1..14
	"label.selection" : Value selection for the labeling (0: auto, 1: y, 2: x, 3: parameter, 4: magnitude, 5: phase)
	"legend.append" : Show name extension in legend for line (0 auto, 1 yes, 2 no)
	"legend.channel" : Show channel name in legend for the line (0 auto, 1 yes, 2 no)
	"legend.comment" : Show comment in legend for lines (0 auto, 1 yes, 2 no)
	"legend.group" : Show group names in legend for line (0 auto, 1 yes, 2 no)
	"legend.measurement" : Show measurement in legend for line (0 auto, 1 yes, 2 no, 3 number of selected measurement)
	"legend.numerical.format" : Format of the numerical values in the line legend (0: auto, 1: fixed point, 2: floating point)

	"legend.numerical.precision" : Precision of the numerical values in the line's legend. For fixed- and floating point, the decimal places 0..14; else number of valid digits 1..14
	"legend.numerical.option" : Allow numerical values in line legend (0: auto, 1: no)
	"legend.sample" : Show line probe in legend for lines (0 auto, 1 yes, 2 no)
	"legend.show" : Show legend for line (0: auto, 1: yes, 2: no, 3: text with placeholders)
	"legend.text" : Text displayed as the legend. The text may contain placeholders.
	"linestyle.printer" : Line type on the printer (0 auto, 1 solid, 2 dotted, 3 dashed, 4..13 other combinations)
	"linestyle.screen" : Line type on the monitor (0 auto, 1 solid, 2 dotted, 3 dashed, 4..13 other combinations)
	"measure.cursor.set" : The measurement cursor is set to this line (1 linked with the left mouse button, 2 linked with the right mouse button).
	"scale.type" : Scaling of line (0: auto; 1: from 0 to 1 relative; 2: from 0 to 1 with offset; 3: mm relative; 4: mm with offset; 5: stretch image; 6: adjust image horizontally; 7: adjust image vertically; 8: show complete image; 9: keep image size)
	"shift.x" : Time shift in physical units. For linear, an offset; for logarithmic, a factor.
	"shift.y" : Amplitude shift in physical units. For linear, an offset; for logarithmic, a factor.
	"suppress" : Hide line (0: No = default; 1: Yes)
	"symbol" : Symbol: 0 none, 1 square, 2 circle, 3-4 triangles, 5 diamond, 6 pulse, 7 X, 8 fat dots, 9 horz. line, 10-11 triangles, 12-13 X, 14 filled diamond, 15 square, 16-19 triangles, 20-21 squares, 22-23 X, 24 minus, 25 empty
	"symbol.count" : Number of symbols, interpreted according to symbol count option
	"symbol.count.opt" : symbol count option: 1: at every sample (symbol count=0); 2: fixed number of symbols to be drawn across the width of the coordinate system (symbol count>0).
	"symbolsize.printer" : Symbol size on printer in mm (-1 auto)
	"symbolsize.screen" : Symbol size on monitor in mm (-1 auto)
	"uncertainty.color" : This color used for uncertainty display; for format see rgb() and -1 for automatic
	"uncertainty.show" : Display uncertainty (0 no, 1 colored area, 2 line). Please refer to property: Uncertainty.
	"uncertainty.selection" : Selection for the measurement uncertainty (0: auto, 1: standard measurement uncertainty, 2: expanded measurement uncertainty). See the properties for Uncertainty and Expanded uncertainty. With "auto", the standard measurement uncertainty is given preference.
	"width.printer" : Line thickness on the printer in mm (-1 auto)
	"width.printer.pt" : Line thickness on the printer in pt (-1: auto)
	"width.screen" : Line thickness on the monitor in mm (-1 auto)
	"width.screen.pt" : Line thickness on the monitor, in pt (-1: auto)
	"3D.surface" : 3D: selection of the surface (0 auto, 1 filled wire grid, 2 filled, 3 wire grid, 4 points, 5 wire grid in color palette, 6 Space curve, 7 Space curve with color tones)
Value	In which way should this property be set?

Description:

Examples:

Adjust stair steps

```
CwSelectByIndex("line", 1)
CwLineSet("style", 2)
```

See also:

CwAxisLineGet

CwLoadCCV

Scope: Curve Windows

Opens the curve configuration from a *.ccv-file.

Declaration:

```
CwLoadCCV ( Identification, Filename ) -> Error text
```

Parameter:

Identification	This data set identifies the curve window. With CwSelectMode, the system determined previously how identification is performed.
Filename	From which file should the configuration be opened?
Error text	
Error text	Error text at fault condition; or if operation successful, an empty string. (optional)

Description:

The function regenerates the curve window, unless a curve window with the specified identity is already present.

The function also immediately selects the curve window.

CwSelectWindow() need not be called afterwards. That will be necessary for later selection of this curve window once other curve windows have been selected.

If the identification is a text, this applies: If a curve window's title is present in the opened Panel or dialog, it will be addressed. If it is not present, but a free-floating curve window with that identification exists, that one will be addressed. Otherwise a new free-floating curve window will be created.

If the identification is a data set, a free-floating curve window will be addressed or created.

The configuration of the curve window contains all attributes of the display, but not the displayed measured data themselves.

The behavior of the function can be influenced by a previous call to the function CwGlobalSet.

Examples:

Loads a CCV file

```
data = ramp(0,1,10)
CwLoadCCV(data, "c:\imc\ccv\1.ccv")
```

Loading the CCV file for a curve window embedded in the Panel. The CCV file is located in the project directory.

```
CwLoadCCV("curve1", "1.ccv")
```

If multiple curve windows in the Panel have the same title, then the page name ("page1" in this case) separated with a dot must be prefixed.

```
CwLoadCCV("page1.curve1", "1.ccv")
```

Loading the CCV file for a curve window embedded in the dialog or Panel

```
CwLoadCCV("curve1", "c:\imc\ccv\1.ccv")
```

Open from the CCV folder or from the project folder

```
CwLoadCCV(data, "1.ccv")
```

Loads an error query

```
errortext = CwLoadCCV(data, "1.ccv")
if errortext <> ""
   ; ...
end
```

Loading of a CCV file; identification by means of a variable

```
CwLoadCCV(data, "1.ccv")
...
CwSelectWindow(data)
```

Loading of a CCV file; identification using text

```
CwLoadCCV("t", "c:\imc\ccv\1.ccv")
...
CwSelectWindow("t")
```

Loading of a CCV file; creating a free-floating curve window without identification

Afterwards the curve window is selected. However, it will generally not be possible to select it again after a different curve window has been selected.

```
CwLoadCCV(0, "1.ccv")
```

See also:

CwSaveCCV

CwLoadSettings

Scope: Curve Windows

This function loads a global setting from the file specified in TxFileName into the Curve Manager.

Declaration:

```
CwLoadSettings ( TxFileName, SvSetting, SvParameter )
```

Parameter:

TxFileName	TxFileName
SvSetting	SvSetting
	1 : All curve windows are immediately displayed using the new color scheme. This includes all of the settings in the color dialog.
	2 : For all subsequent printing procedures, for transfer to the Report Generator, copying to the Clipboard, and graphics export, the new colors will be applied.
	3 : All settings belonging to the dialog <Clipboard settings> are applicable effective immediately. For all subsequent printing procedures, and transfer to the Report Generator, copying to the Clipboard and graphics export, these settings will be applied. However, any reports for which the current settings are not to be applied upon transferring are unaffected.
	4 : Some Settings made in the dialog <curve window presettings...> are loaded. These settings will be used for all curve windows. The font used for screen display is one of these settings.
SvParameter	SvParameter

Description:

The file is usually expected in the CCV-directory (imc FAMOS). Some of the options affect all open windows. When not otherwise specified, set SvParameter = 0.

The function does not produce an error message when the file is incomplete or does not even exist.

Examples:

The file c:\imc\set\colors.set was previously saved.

```
CwLoadSettings ( "..\set\colors.set", 1, 0 )
```

See also:

CwGlobalSet

CwMarkerGet

Scope: Curve Windows

Get marker property

Declaration:

`CwMarkerGet ( Property ) -> Value`

Parameter:

Property	Get which property?
	"x.type" : Type of x-coordinate (1 physical unit, 2 percent of axis length)
	"x" : x-coordinate whose meaning is determined by "Type of x-coordinate"
	"y.type" : Type of y-coordinate (1 physical unit, 2 percent of axis length)
	"y" : y-coordinate whose meaning is determined by "Type of x-coordinate"
	"angle" : Angle of conneting line in degrees (-360 .. +360). Not valid for "Type of line length" = "XY percentual".
	"arrow" : Arrow type (0 none, 1 broad, 2 narrow, 3 broad filled, 4 narrow filled, 5 large, 6 large filled, 7 circle, 8 star, 9 line, 10 point)
	"arrow.size" : Size of arrow in mm (0.5 .. 10.0), 0 for auto
	"border" : Frame for the text (0 none, 1 simple, 2 with tip, 3 double)
	"calculation" : For an order line, the calculation (0: order line in the RPM-spectrum; 1: RPM over frequency; 2: frequency over RPM; 3: rotational frequency over frequency; 4: frequency over rotational frequency; 5: hyperbola in the order spectrum; 6: RPM and order; 7: rotation frequency and order)
	"color.background" : Background color; for format, see rgb(); -1 for automatic, -2 for transparent
	"color.text" : Color of text, format see rgb() and -1 for automatic
	"dimline.arrow.position" : Dimension line: Arrow position (0 auto, 1 outer)
	"dimline.dimension line.extend" : Dimension line: Extend dimension line in mm, >=0
	"dimline.distance" : Dimension line: Distance of the dimension line from the marker: In mm from one of the two reference markers. Or in relative terms as als fraction of the axis length, e.g. 0.1 for 10% of the axis length either to the right edge or the bottom. Or fixed: 0..1 for the region either left of or above the coordinate system, 1..2 within, 2..3 right of or below the coordinate system.
	"dimline.distance type" : Dimension line: How is the distance specified? (0: fixed, 1: relative from 1st reference marker, 2: in mm from 1st reference marker, 3: relative to 2nd reference marker, 4: in mm from 2nd reference marker)
	"dimline.projection line.distance" : Dimension line: Distance of projection line in mm, >= 0
	"dimline.projection line.extend" : Dimension line: Extend projection line in mm, >=0
	"extension" : For horizontal/vertical line, the extension (0 all coordinate systems, 1 from the marker to left/bottom, 2 from the marker until free position, 3 between 2 free positions, 4 across one coordinate system)
	"font.size" : Font size for marker text, in pts, e.g. 8.
	"harmonic.type" : If the marker is a harmonics cursor, the type (0: not a harmonics cursor, 1: harmonics of a fundamental oscillation, 2: Offset first oscillation, subsequent are periodic, 3: one or multiple harmonics with side bands, 4 two lines whose distance is a factor)
	"harmonic.index" : If the marker is a harmonics cursor, an index for distinguishing the individual cursors
	"index" : Index of the marker. The index to the list of markers starts with 1.
	"line.end" : For order lines, the end of the line in percent (0 .. 100)
	"line.index" : Index of the assigned line (line element, not connecting line!), >= 1 for valid line, else 0
	"line.start" : For order lines, the start of the line in percent (0 .. 100)
	"line.text.pos" : For order lines, the position of the labeling along the line in percent
	"linelength" : Length of line between marker point and marker text. The meaning of the length is determined by "Type of x-line length". For "Type of line length" = "XY percentual", the length of the line in the x-direction is given here.
	"linelength.type" : Type of line length (1 x-units, 2 % of x-axis, 3 y-units, 4 % of y-axis, 5 % of text height, 6 XY percentual: % x-axis and LineLength2 % of y-axis)
	"linelength2" : Length of line in y-direction, only for "Type of line length" = "XY percentual".
	"linestyle" : Line type (0 auto, 1 solid, 2 dotted, 3 dashed, 4..13 other combinations)

	"linewidth" : Line thickness in mm (0 auto)
	"multiple" : For order lines, the parameter specified can only be a multiple of this value. If it is zero, there is no restriction.
	"parameter" : The parameter for order lines; the type depends on the calculation. For an order line, the order; the frequency for a hyperbola.
	"pinnedtext" : Pinned text. In what direction should the text box extend. Typically 0. (0=auto, 1 top right, 2 top left, 3 bottom right, 4 bottom left)
	"ref.1" : Index of the 1st reference marker of a dimension line. The index to the list of markers starts with 1 and selects a standard marker.
	"ref.2" : Index of the 2nd reference marker of a dimension line. The index to the list of markers starts with 1 and selects a standard marker.
	"text.orientation" : Orientation of the text in degrees (-90 .. +90). What is the angle of the text from horizontal?
	"type" : Marker type (0 standard, 1 vertical line, 2 horizontal line, 3 text, 4 vertical dimension line, 5 horizontal dimension line)
	"value.abs" : Dimension line: Use absolute value of the difference for labeling (0: no, 1: yes)
	"within.cosys" : Is the marker only drawn inside of the coordindate system (0 no, 1 yes)
Value	
Value	The property's value

Description:

Examples:

```
mini = CwMarkerGet("min")
```

See also:

CwMarkerSet, CwMarkerGetText

CwMarkerGetText

Scope: Curve Windows

Get text property of a marker

Declaration:

```
CwMarkerGetText ( Property ) -> Value
```

Parameter:

Property	Get which property?
	"text" : Text
	"text.placeholders" : Text with placeholders
Value	
Value	The property's value

Description:

Examples:

Get a marker's text

```
CwSelectByIndex("marker", 1)
text = CwMarkerGet("text")
```

See also:

CwMarkerSet, CwMarkerGet

CwMarkerSet

Scope: Curve Windows

Set marker property

Declaration:

```
CwMarkerSet ( Property, Value )
```

Parameter:

Property	Which property is to be set?
	"text" : Text
	"x.type" : Type of x-coordinate (1 physical unit, 2 percent of axis length)
	"x" : x-coordinate whose meaning is determined by "Type of x-coordinate"
	"y.type" : Type of y-coordinate (1 physical unit, 2 percent of axis length)
	"y" : y-coordinate whose meaning is determined by "Type of x-coordinate"
	"angle" : Angle of conneting line in degrees (-360 .. +360). Not valid for "Type of line length" = "XY percentual".
	"arrow" : Arrow type (0 none, 1 broad, 2 narrow, 3 broad filled, 4 narrow filled, 5 large, 6 large filled, 7 circle, 8 star, 9 line, 10 point)
	"arrow.size" : Size of arrow in mm (0.5 .. 10.0), 0 for auto
	"border" : Frame for the text (0 none, 1 simple, 2 with tip, 3 double)
	"calculation" : For an order line, the calculation (0: order line in the RPM-spectrum; 1: RPM over frequency; 2: frequency over RPM; 3: rotational frequency over frequency; 4: frequency over rotational frequency; 5: hyperbola in the order spectrum; 6: RPM and order; 7: rotation frequency and order)
	"color.background" : Background color; for format, see rgb(); -1 for automatic, -2 for transparent
	"color.text" : Color of text, format see rgb() and -1 for automatic
	"dimline.arrow.position" : Dimension line: Arrow position (0 auto, 1 outer)
	"dimline.dimension line.extend" : Dimension line: Extend dimension line in mm, >=0
	"dimline.distance" : Dimension line: Distance of the dimension line from the marker: In mm from one of the two reference markers. Or in relative terms as als fraction of the axis length, e.g. 0.1 for 10% of the axis length either to the right edge or the bottom. Or fixed: 0..1 for the region either left of or above the coordinate system, 1..2 within, 2..3 right of or below the coordinate system.
	"dimline.distance type" : Dimension line: How is the distance specified? (0: fixed, 1: relative from 1st reference marker, 2: in mm from 1st reference marker, 3: relative to 2nd reference marker, 4: in mm from 2nd reference marker)
	"dimline.projection line.distance" : Dimension line: Distance of projection line in mm, >= 0
	"dimline.projection line.extend" : Dimension line: Extend projection line in mm, >=0
	"extension" : For horizontal/vertical line, the extension (0 all coordinate systems, 1 from the marker to left/bottom, 2 from the marker until free position, 3 between 2 free positions, 4 across one coordinate system)
	"font.size" : Font size for marker text, in pts, e.g. 8.
	"line.end" : For order lines, the end of the line in percent (0 .. 100)
	"line.selected" : The marker is assigned to the selected line (line element, not connecting line!). Value = 1.
	"line.start" : For order lines, the start of the line in percent (0 .. 100)
	"line.text.pos" : For order lines, the position of the labeling along the line in percent
	"linelength" : Length of line between marker point and marker text. The meaning of the length is determined by "Type of x-line length". For "Type of line length" = "XY percentual", the length of the line in the x-direction is given here.
	"linelength.type" : Type of line length (1 x-units, 2 % of x-axis, 3 y-units, 4 % of y-axis, 5 % of text height, 6 XY percentual: % x-axis and LineLength2 % of y-axis)
	"linelength2" : Length of line in y-direction, only for "Type of line length" = "XY percentual".
	"linestyle" : Line type (0 auto, 1 solid, 2 dotted, 3 dashed, 4..13 other combinations)
	"linewidth" : Line thickness in mm (0 auto)
	"multiple" : For order lines, the parameter specified can only be a multiple of this value. If it is zero, there is no restriction.

	"parameter" : The parameter for order lines; the type depends on the calculation. For an order line, the order; the frequency for a hyperbola.
	"pinnedtext" : Pinned text. In what direction should the text box extend. Typically 0. (0=auto, 1 top right, 2 top left, 3 bottom right, 4 bottom left)
	"ref.1" : Index of the 1st reference marker of a dimension line. The index to the list of markers starts with 1 and selects a standard marker.
	"ref.2" : Index of the 2nd reference marker of a dimension line. The index to the list of markers starts with 1 and selects a standard marker.
	"text.orientation" : Orientation of the text in degrees (-90 .. +90). What is the angle of the text from horizontal?
	"within.cosys" : Is the marker only drawn inside of the coordindate system (0 no, 1 yes)
	"value.abs" : Dimension line: Use absolute value of the difference for labeling (0: no, 1: yes)
Value	In which way should this property be set?

Description:

Examples:

Set a marker's text

```
CwSelectByIndex("marker", 1)
CwMarkerSet("text", "Max!")
```

Set a marker's text with placeholders

```
CwSelectByIndex("marker", 1)
CwMarkerSet("<auto> Left!", "Max!")
```

See also:

CwMarkerGet, CwMarkerGetText, CwNewElement

CwNewChannel

Scope: Curve Windows

Displays a channel in the curve window

Declaration:

`CwNewChannel ( Position, Channel )`

Parameter:

Position	At which location should the new data be inserted?
	"append last axis" : The new data are appended at the last y-axis present with a new line. The new line is selected.
	"append line" : A new line, to which the new data are assigned, is appended behind the selected line. The new line is selected.
	"append new axis" : The new data are appended to the last coordinate system with a new line and a new y-axis. The new line and new axis are selected.
	"append new cosys" : The new data are inserted into a newly appended coordinate system with a new line and a new y-axis. The new coordinate system, new line and new axis are selected.
	"append to cosys" : The new data are inserted as a new line in a new y-axis. The new y-axis will be inserted as the last y-axis of the previously selected coordinate system. The new line and new axis are selected.
	"first axis" : The new data are inserted as a new line with its own axis. The new line, along with its axis and coordinate system, is selected.
	"first line" : The new data are inserted as a new line in the first position. The new line, along with its axis and coordinate system, is selected.
	"insert axis" : The new data are inserted as a new line in a new y-axis. The new y-axis will be inserted behind the previously selected y-axis. The new line and new axis are selected.
	"insert cosys" : The new data are inserted as a new line in a new y-axis and a new coordinate system. The new coordinate system will be inserted behind the previously selected coordinate system. The new coordinate system, new line and new axis are selected.
	"insert first axis" : The new data are inserted as a new line with a new y-axis. The new y-axis is inserted as the first y-axis in the previously selected coordinate system. The new line and new axis are selected.
	"insert first cosys" : The new data are inserted into a newly added coordinate system with a new line and a new y-axis. The new coordinate system will be the first coordinate system. The new coordinate system, new line and new axis are selected.
	"insert first line" : The new data are inserted as a new line, which will be the first for the previously selected axis. The new line will be selected.
	"insert line" : A new line, to which the new data are assigned, is inserted before the line selected. The new line will be selected.
	"replace data" : The selected data element receives the new channel.
	"replace line" : The data on the selected line are replaced with the new data.
Channel	Data set to be displayed in the curve window

Description:

Depending on the display type, no new axes or coordinate systems may be created.

Examples:

Add channels to a curve window

```
CwSelectWindow("curve1")
CwNewChannel("append last axis", channel1)
CwNewChannel("append last axis", channel2)
```

Add a channel's magnitude

```
CwSelectWindow("curve1")
CwNewChannel("append new axis", Spectrum.m)
```

Replace one displayed channel with another

```
CwSelectWindow("curve1")
CwSelectByIndex("line", 1)
CwNewChannel("replace line", channel2)
```

See also:

CwNewChannel_xy, CwNewChannel_xyz

CwNewChannel_xy

Scope: Curve Windows

Two channels are displayed as an xy-display in the curve window.

Declaration:

`CwNewChannel_xy ( Position, Channel 1, Channel 2, Option )`

Parameter:

Position	At which location should the new data be inserted?
	"append last axis" : The new data are appended at the last y-axis present with a new line. The new line is selected.
	"append line" : A new line, to which the new data are assigned, is appended behind the selected line. The new line is selected.
	"append new axis" : The new data are appended to the last coordinate system with a new line and a new y-axis. The new line and new axis are selected.
	"append new cosys" : The new data are inserted into a newly appended coordinate system with a new line and a new y-axis. The new coordinate system, new line and new axis are selected.
	"append to cosys" : The new data are inserted as a new line in a new y-axis. The new y-axis will be inserted as the last y-axis of the previously selected coordinate system. The new line and new axis are selected.
	"first axis" : The new data are inserted as a new line with its own axis. The new line, along with its axis and coordinate system, is selected.
	"first line" : The new data are inserted as a new line in the first position. The new line, along with its axis and coordinate system, is selected.
	"insert axis" : The new data are inserted as a new line in a new y-axis. The new y-axis will be inserted behind the previously selected y-axis. The new line and new axis are selected.
	"insert cosys" : The new data are inserted as a new line in a new y-axis and a new coordinate system. The new coordinate system will be inserted behind the previously selected coordinate system. The new coordinate system, new line and new axis are selected.
	"insert first axis" : The new data are inserted as a new line with a new y-axis. The new y-axis is inserted as the first y-axis in the previously selected coordinate system. The new line and new axis are selected.
	"insert first cosys" : The new data are inserted into a newly added coordinate system with a new line and a new y-axis. The new coordinate system will be the first coordinate system. The new coordinate system, new line and new axis are selected.
	"insert first line" : The new data are inserted as a new line, which will be the first for the previously selected axis. The new line will be selected.
	"insert line" : A new line, to which the new data are assigned, is inserted before the line selected. The new line will be selected.
	"replace line" : The data on the selected line are replaced with the new data.
Channel 1	1st channel to be displayed in the curve window
Channel 2	2nd channel to be displayed in the curve window
Option	Option
	"yx" : 1st channel becomes y-component, 2nd channel x-component.
	"xy" : 1st channel becomes the x-component, 2nd channel the y-component.

Description:

Depending on the display type, no new axes or coordinate systems may be created.

Examples:

Add channels to a curve window

```
CwSelectWindow("curve1")
CwNewChannel_xy("append last axis", level, speed, "yx")
CwNewChannel_xy("append last axis", temperature, speed, "yx")
```

Replace a displayed channel with a new xy representation

```
CwSelectWindow("curve1")
CwSelectByIndex("line", 1)
CwNewChannel_xy("replace line",  level, speed, "yx")
```

See also:

CwNewChannel, CwNewChannel_xyz

CwNewChannel_xyz

Scope: Curve Windows

Three channels are displayed in xyz-representation in the curve window.

Declaration:

```
CwNewChannel_xyz ( Position, Channel 1, Channel 2, Channel 3, Option )
```

Parameter:

Position	At which location should the new data be inserted?
	"append last axis" : The new data are appended at the last y-axis present with a new line. The new line is selected.
	"append line" : A new line, to which the new data are assigned, is appended behind the selected line. The new line is selected.
	"append new axis" : The new data are appended to the last coordinate system with a new line and a new y-axis. The new line and new axis are selected.
	"append new cosys" : The new data are inserted into a newly appended coordinate system with a new line and a new y-axis. The new coordinate system, new line and new axis are selected.
	"append to cosys" : The new data are inserted as a new line in a new y-axis. The new y-axis will be inserted as the last y-axis of the previously selected coordinate system. The new line and new axis are selected.
	"first axis" : The new data are inserted as a new line with its own axis. The new line, along with its axis and coordinate system, is selected.
	"first line" : The new data are inserted as a new line in the first position. The new line, along with its axis and coordinate system, is selected.
	"insert axis" : The new data are inserted as a new line in a new y-axis. The new y-axis will be inserted behind the previously selected y-axis. The new line and new axis are selected.
	"insert cosys" : The new data are inserted as a new line in a new y-axis and a new coordinate system. The new coordinate system will be inserted behind the previously selected coordinate system. The new coordinate system, new line and new axis are selected.
	"insert first axis" : The new data are inserted as a new line with a new y-axis. The new y-axis is inserted as the first y-axis in the previously selected coordinate system. The new line and new axis are selected.
	"insert first cosys" : The new data are inserted into a newly added coordinate system with a new line and a new y-axis. The new coordinate system will be the first coordinate system. The new coordinate system, new line and new axis are selected.
	"insert first line" : The new data are inserted as a new line, which will be the first for the previously selected axis. The new line will be selected.
	"insert line" : A new line, to which the new data are assigned, is inserted before the line selected. The new line will be selected.
	"replace line" : The data on the selected line are replaced with the new data.
Channel 1	1st channel to be displayed in the curve window
Channel 2	2nd channel to be displayed in the curve window
Channel 3	3rd channel, to be displayed in the curve window
Option	
	"yxz" : 1st channel becomes the y-component, 2nd channel the x-component, 3rd channel the z-component.
	"xyz" : 1st channel becomes the x-component, 2nd channel the y-component, 3rd channel the z-component.

Description:

Depending on the display type, no new axes or coordinate systems may be created.

Not all display styles support this kind of superposition

Examples:

Add channels to a curve window

```
CwSelectWindow("curve1")
CwNewChannel_xyz("append last axis", level, speed, torque, "yxz")
```

Replace a displayed channel with a new superposition of 3 channels

```
CwSelectWindow("curve1")
CwSelectByIndex("line", 1)
CwNewChannel_xyz("append last axis", level, speed, torque, "yxz")
```

See also:

CwNewChannel, CwNewChannel_xy

CwNewElement

Scope: Curve Windows

A new element is created in the curve window.

Declaration:

```
CwNewElement ( Element sort )
```

Parameter:

Element sort	Which sort of element is to be created at what position?
	"marker" : A new marker is appended to the marker list at the end.
	"marker.abs" : A new marker is added to the back of the list of markers. Its position is determined by x- and y-coordinates.
	"marker.user" : A new marker is added to the back of the list of markers. The current curve window pre-settings are applied.
	"marker.harmonic.harmonic" : Generates all markers belonging to a harmonics cursor for a fundamental oscillation with harmonics.
	"marker.harmonic.harmonic.user" : Generates all markers belonging to a harmonics cursor for a fundamental oscillation with harmonics. The curve window's current pre-settings are applied.
	"marker.harmonic.offset" : Generates all markers belonging to a harmonics cursor for periodic processes with arbitrary start.
	"marker.harmonic.offset.user" : Generates all markers belonging to a harmonics cursor for periodic processes with arbitrary start. The curve window's current presettings are applied.
	"marker.harmonic.ratio" : Generates all markers belonging to a harmonic cursor for 2 oscillations in a fixed ratio.
	"marker.harmonic.ratio.user" : Generates all markers belonging to a harmonic cursor for 2 oscillations in a fixed ratio. The curve window's current presettings are applied.
	"marker.harmonic.sideband" : Generates all markers belonging to a harmonics cursor for fundamental oscillation with side bands.
	"marker.harmonic.sideband.user" : Generates all markers belonging to a harmonics cursor for fundamental oscillation with side bands. The curve window's current presettings are applied.
	"marker.hori.dimline" : A new marker in the form of a horizontal dimension line is added to the back of the list of markers.
	"marker.hori.dimline.user" : A new marker in the form of a horizontal dimension line is added to the back of the list of markers. The current curve window pre-settings are applied.
	"marker.hori.line" : A new marker in the form of a horizontal line is added to the back of the list of markers.
	"marker.hori.line.user" : A new marker in the form of a horizontal line is added to the back of the list of markers. The current curve window pre-settings are applied.
	"marker.text" : A new marker in the form of a text is added to the back of the list of markers.
	"marker.text.user" : A new marker in the form of a text is added to the back of the list of markers. The current curve window pre-settings are applied.
	"marker.vert.dimline" : A new marker in the form of a vertical dimension line is added to the back of the list of markers..
	"marker.vert.dimline.user" : A new marker in the form of a vertical dimension line is added to the back of the list of markers. The current curve window pre-settings are applied.
	"marker.vert.line" : A new marker in the form of a vertical line is added to the back of the list of markers.
	"marker.vert.line.user" : A new marker in the form of a vertical line is added to the back of the list of markers. The current curve window pre-settings are applied.
	"marker.order" : A new marker in the form of an order line is added to the end of the list of markers.
	"marker.order.user" : A new marker in the form of an order line is added to the end of the list of markers. The current curve window pre-settings are applied.

Description:

The newly created element is also selected right away.

Examples:

Add a new marker to the curve window and parameterize it

```
CwNewElement("marker")
CwMarkerSet("text", "Max!")
```

See also:

CwSelectByIndex, CwMarkerSet

CwNewWindow

Scope: Curve Windows

Generates an empty curve window

Declaration:

`CwNewWindow ( Identification, ShowOption )`

Parameter:

Identification	This data set identifies the curve window. This means that the curve window will be identifiable later.
ShowOption	Show the curve window immediately?
	"show" : Show
	"hide" : Don't show

Description:

A free-floating empty curve window is created. Only if no curve window with the specified identification already exists.

The function also immediately selects the curve window.

If the window is hidden when created, it must be made visible for the purpose of certain operations.

If a curve window having the specified identification already exists, the parameter ShowOption will not be applied: The window will remain unchanged.

Examples:

Display an empty curve window, then a channel in it

```
data=ramp(0,1,10)
CwNewWindow(data, "show")
CwNewChannel("append new axis",data)
```

See also:

CwLoadCCV

CwPosition

Scope: Curve Windows

Adjustment of the position and size of the selected curve window

Declaration:

```
CwPosition ( X, Y, dX, dY )
```

Parameter:

X	X
Y	Y
dX	dX
dY	dY

Description:

SvX, SvY: upper left corner of the window

SvdX, SvWdY: size of the window

Examples:

```
CwSelectWindow("curve1")
CwPosition ( 0, 0, 640, 480 )
```

See also:

CwLoadCCV

CwPrintSet

Scope: Curve Windows

Sets properties for printing a curve window

Declaration:

`CwPrintSet ( Property, Value )`

Parameter:

Property	Which property is to be set?
	"individual" : Individualized settings: (0 no, 1 yes)
	"layout" : Layout: (0: coordinate system size; 1: proportions as on screen; 2: total size)
	"cosys.open" : Coordinate system open: (0: no; 1: yes)
	"signature.show" : Show caption (0: no; 1: yes)
	"signature" : Caption
	"measure.show" : Show measurement crosshairs and measurement values (0: no; 1: yes)
	"width" : Width in mm. Depending on the layout, either for the coordinate system, or the total
	"height" : Height in mm. Depending on the layout, either for the coordinate system, or the total
	"zlength" : Length of 3D z-axis; stated in mm
	"symbol.size" : Symbol diameter; stated in mm
	"ticks.in" : Main inner tick size; stated in mm
	"ticks.out" : Main outer tick size; stated in mm
	"small ticks.in" : Small inner tick size; stated in mm
	"small ticks.out" : Small outer tick size; stated in mm
	"xscale" : Height of x-scale stated in mm; 0 auto
	"yscale" : Width of y-scale, stated in mm; 0 auto
	"linewidth.cosys" : Line width of coordinate system. <0 for width stated in pixels. For example -3 for 3 pixels.
	"linewidth.grid" : Line width of main grid. <0 for width stated in pixels. For example -3 for 3 pixels.
	"linewidth.sec grid" : Line width of secondary grid. <0 for width stated in pixels. For example -3 for 3 pixels.
	"linewidth.cursor" : Cursor line width. <0 for width stated in pixels. For example -3 for 3 pixels.
	"linewidth.curve" : Line width of curves. <0 for width stated in pixels. For example -3 for 3 pixels.
	"linestyle.cosys" : Line style of coordinate system (0: solid; 1: dotted; 2: fewer dots; 3: dashed; 4: fewer dashes; 5: dash-dot)
	"linestyle.grid" : Line style of main grid (0: solid; 1: dotted; 2: fewer dots; 3: dashed; 4: fewer dashes; 5: dash-dot)
	"linestyle.sec grid" : Line style of secondary grid (0: solid; 1: dotted; 2: fewer dots; 3: dashed; 4: fewer dashes; 5: dash-dot)
	"linestyle.cursor" : Cursor ine style (0: solid; 1: dotted; 2: fewer dots; 3: dashed; 4: fewer dashes; 5: dash-dot)
	"angle 3D" : Angle of 3D z-axis; stated in degrees
	"font.name" : Font name, for example "Arial".
	"font.size" : Font size, stated in pt, e.g. 8
	"font.style" : Font style (0: auto; 1: default; 2: bold; 3: italic; 4: bold and italic; 5: underlined; 6: bold and underlined; 7: italic and underlined; 8: bold and italic and underlined)
	"small font.name" : Small font name, for example "Arial".
	"small font.size" : Small font size, stated in pt, e.g. 8
	"small font.style" : Small font style (0: auto; 1: default; 2: bold; 3: italic; 4: bold and italic; 5: underlined; 6: bold and underlined; 7: italic and underlined; 8: bold and italic and underlined)
Value	In which way should this property be set?

Description:

If settings are indiviudalized (see "individualized") to a curve window, then only that curve window will be modified. Otherwise global change.

These settings are used for printing, exporting graphics, and when copying to the Clipboard and to the Report Generator.

Examples:

Sets the total size for the protrait format. Only for the current curve window

```
CwPrintSet("individual", 1)
CwPrintSet("layout", 2)
CwPrintSet("width", 180)
CwPrintSet("height", 260)
```

Preparation for pdf export

```
CwGlobalSet("graphics.type", 0)
CwGlobalSet("export.dpi", 300)
CwGlobalSet("export.pdf.append", 0)
CwGlobalSet("export.orientation", 1)
CwGlobalSet("export.pdf.method", 2)
CwPrintSet("layout", 2)
CwPrintSet("font.size", 10)
CwPrintSet("font.style", 0)
CwPrintSet("font.name", "Arial")
```

See also:

CwDisplaySet, CwGlobalSet

CwReplace

Scope: Curve Windows

In a curve window, a channel designated "OldDesignation" is displayed. This channel is now to be replaced with ChannelReplacement in this curve window. Next, ChannelReplacement is to be displayed instead of the channel with the name OldDesignation.

Declaration:

```
CwReplace ( ChannelReplacement, OldDesignation ) -> NumberReplaced
```

Parameter:

ChannelReplacement	Data set to be displayed in the curve window
OldDesignation	Designation of the channel displayed
NumberReplaced	
NumberReplaced	Returns the number of channels replaced. (optional)

Description:

If the same channel occurs repeatedly, it is also replaced repeatedly. If the channel isn't present at all, the function has no effect.

Channel names which are allowed take the form "channel", or "group:channel" for channels in groups.

The function is used when after loading a curve window configuration, other chanels are to be displayed in this configuration than the ones for which the configuration was saved.

Upon first replacement, the data element, the associated line, axis and the coordinate system are selected.

The parameter OldDesignation is specified without .X or .Y

The parameter ChannelReplacement can contain .X or .Y, if from this moment forward only exactly that component is selected for display.

Examples:

Loads a configuration which was saved when the channels had different names (then: CH1 and CH2). The channels, which are now called temperature and pressure, are to be displayed.

```
CwLoadCCV("curve1", "1.ccv")
CwReplace(temperature, "CH1")
CwReplace(pressure, "CH2")
```

Channel S ist replaced by the magnitude of a dataset. The component .M will be selected for display.

```
CwReplace(Spectrum.M, "S")
```

See also:

CwNewChannel

CwSaveCCV

Scope: Curve Windows

Saves the configuration of the selected curve window in a *.ccv-file.

Declaration:

```
CwSaveCCV ( Filename ) -> Error text
```

Parameter:

Filename	To which file should the configuration be saved?
Error text	
Error text	Error text at fault condition; or if operation successful, an empty string. (optional)

Description:

The configuration of the curve window contains all attributes of the display, but not the displayed measured data themselves.

Examples:

Saves to a CCV file

```
CwLoadCCV(data, "1.ccv")
CwDisplaySet("displaymode", 2)
CwSaveCCV("2.ccv")
```

Saves to a CCV file

```
CwSelectWindow(data)
CwSaveCCV("2.ccv")
```

See also:

CwLoadCCV

CwSelectByChannel

Scope: Curve Windows

Selects an element (e.g. axis) of the selected curve window by means of a channel.

Declaration:

```
CwSelectByChannel ( Element sort, Channel )
```

Parameter:

Element sort	What sort of element is to be selected?
	"axis" : y-axis to the channel
	"cosys" : Coordinate system for the channel
	"data" : Data element to the channel
	"line" : Line to the channel
Channel	Data set displayed in the curve window. The variable itself or its name.

Description:

If the data set is displayed multiple times, the first element found is determined.

If the element does not exist, no more of this sort is selected.

If the channel is specified by its name, then these are allowed: "channel" or "group:channel"

This technique also works if the channel does not even exist at the moment, but its name appears in the curve window.

Examples:

Select the axis which represents the variable data

```
CwSelectWindow("curve1")
CwSelectByChannel("axis", data)
CwAxisSet("min", -10)
CwAxisSet("max", 10)
```

Selects the line representing the channel having the name "ch1".

```
CwSelectWindow("curve1")
CwSelectByChannel("line", "ch1")
CwLineSet("symbol", 2)
```

See also:

CwSelectByIndex

CwSelectByIndex

Scope: Curve Windows

Selects an element (e.g. axis) within a selected curve window

Declaration:

```
CwSelectByIndex ( Element sort, Index )
```

Parameter:

Element sort	What sort of element is to be selected?
	"axis from data" : y-axis for the selected data element (Index = 1)
	"axis from line" : y-axis for the selected line (index = 1)
	"x-axis" : x-axis (Index = 1)
	"y-axis" : y-axis
	"y-axis in cosys" : y-axis in the selected coordinate system
	"z-axis" : z-axis or angle-axis (index = 1)
	"3D color-axis" : Axis with 3D color legend (Index = 1)
	"cosys" : Coordinate system
	"cosys from axis" : Coordinate system for the selected y-axis (Index = 1)
	"cosys from data" : Coordinate system for the selecte data element (Index = 1)
	"cosys from line" : Coordinate system for the selected line (Index = 1)
	"data" : Data element
	"data in axis" : Data element in the selected y-axis
	"data in cosys" : Data element in the selected coordinate system
	"data in line" : Data element in the selected line
	"line" : Line
	"line from data" : Line to the selected data element (Index = 1)
	"line from marker" : Line to the selected marker (index = 1)
	"line in axis" : Line in the selected y-axis
	"line in cosys" : Line in the selected coordinate system
	"line.measure" : Line to the measurement cursor (1 left, 2 right)
	"marker" : Marker
	"usertick" : User ticks, Axis
	"header" : Header or footer or title
Index	Index; the "this-many-th" element of this sort. Beginning with 1.

Description:

If the element does not exist, no more of this sort is selected.

A coordinate system, an axis, a line, a data element and a marker can all be selected simultaneously. Only one element of a sort can be selected.

If the user alters the curve window by means of a menu and dialogs, it can affect the selection. In particular, the selection can disappear.

Examples:

Select and parameterize the 1st y-axis

```
CwSelectWindow("curve1")
CwSelectByIndex("y-axis", 1)
CwAxisSet("min", -10)
CwAxisSet("max", 10)
```

See also:

CwSelectByChannel

CwSelectMode

Scope: Curve Windows

Determines how the curve window is later identified

Declaration:

`CwSelectMode ( Identification type )`

Parameter:

Identification type	How is the curve window to be identified?
	"auto" : Automatic identification
	"caption" : The caption of the window is used for its identification. Only for floating curve windows.
	"newest" : Last curve window created
	"title" : The curve window's title should identify the curve window. For curve windows in dialogs and in the Panel.
	"variable" : One variable is the reference data set

Description:

If the function has not been called yet, the system chooses "auto".

With automatic identification, the reference data set is first verified. If no curve window is found in the process, the title is verified.

With automatic identification, when a curve window is created, if a string variable is the identification, then the content of this string variable is interpreted as the title.

The function CwSelectMode itself does not determine the identification. It only determines how other functions will perform identification later, e.g. CwSelectWindow or CwNewWindow.

The identification type "newest" is only useful for applications in which no identification is known, for example for selecting a duplicate window which has just been created, or a manually opened window. Then, as the parameter for the subsequent function CwSelectWindow(), an empty text would be adequate: CwSelectWindow("")

The identification method "caption" is only suitable for special applications, because the caption is changed by many operations.

When restarting imc FAMOS or restarting a sequence which does not belong to a dialog or panel, the selection mode will be set to automatic.

If a sequence sets the selection mode explicitly, it is recommended to set it back to automatic before exiting the sequence.

Examples:

For identifying curve windows, only the technique with the reference data set should be used.

```
CwSelectMode("variable")
CwSelectWindow(data)
```

For identifying curve windows, only the technique with the title should be used.

```
CwSelectMode("title")
CwSelectWindow("curve1")
```

Produces the automatic mode

```
CwSelectMode("auto")
CwSelectWindow("curve1")
; ... Working with the curve window
CwSelectWindow(data)
; ... Working with the curve window
```

Identifies a duplicate window which had just been created:

```
CwAction("win.twin")
CwSelectMode("newest")
CwSelectWindow("")
```

See also:

CwSelectWindow

CwSelectWindow

Scope: Curve Windows

Selects a curve window on the basis of a variable used as reference, or on the basis of its title.

Declaration:

```
CwSelectWindow ( Identification ) -> Exist
```

Parameter:

Identification	This data set identifies the curve window. With CwSelectMode, the system determined previously how identification is performed.
Exist	
Exist	Returns 1 if the curve window exists; else 0 (optional)

Description:

If the window does not exist, none is created or selected. If creation of one is desired, use the function CwNewWindow.

If the curve window could have been closed in the meantime, the return value should be queried.

If the curve window does not exist, an error message is displayed, but the sequence is not cancelled.

Examples:

The variable data is displayed as a curve window

```
CwSelectWindow(data)
CwDisplaySet("displaymode", 1)
CwAction("unzoom")
```

In a dialog or Panel there is a curve window with the title "curve1"

```
CwSelectWindow("curve1")
CwAction("unzoom")
```

Next, an additional curve window with the title "curve2" is to be edited:

```
CwSelectWindow("curve2")
CwDisplaySet("displaymode", 2)
```

Selection of a curve window "curve1" in the Panel.

```
CwSelectWindow("curve1")
```

If multiple curve windows in the Panel have the same title, then the page name ("page1" in this case) separated with a dot must be prefixed.

```
CwSelectWindow("page1.curve1")
```

Transfer to the Report Generator

```
CwSelectWindow(data)
CwDisplaySet("name", "nn")
CvPosi( "nn", 0, 0, 600, 400 )
RgCurveSet("r1", "nn", 0)
```

Alternatively for free-floating curve windows:

```
CwSelectWindow(data)
CwDisplaySet("displaymode", 2)
CvPosi( data, 0, 0, 600, 400 )
RgCurveSet("r1", data, 0)
```

Alternatively for embedded curve windows:

```
CwSelectWindow("curve2")
CwDisplaySet("displaymode", 2)
RgCurveSet("r1", "curve2", 0)
```

The success of the function can be verified (alternatively: CwIsWindow):

```
if CwSelectWindow("curve1") = 0
   error handling...
end
```

Before selecting a curve window, the selection mode is set to a well defined value.

```
CwSelectMode("auto")
CwSelectWindow("curve1")
```

See also:

CwSelectMode

CwSequenceEnable

Scope: Curve Windows

Specifies the situations in which FAMOS calls an event sequence for a curve window. The situation consists of a specified location (e.g. over the coordinate system) and a mouse operation (e.g. left mouse button clicked).

Declaration:

```
CwSequenceEnable ( Location, MouseLeft [, MouseRight] [, MouseMove] )
```

Parameter:

Location	Location specification. Where is the mouse?
	"cosys" : Within the coordinate system; MouseLeft=("no", "click", "drag", "2"); MouseRight=("no", "click"); MouseMove=("no", "move")
	"area" : Area outside of the coordinate system; MouseLeft=("no", "click", "drag", "2"); MouseRight=("no", "click"); MouseMove= ("no", "move")
	"measure" : Measurement cursor; MouseLeft=("no", "click", "drag"); MouseRight=("no", "click"); MouseMove=("no").
	"link" : Link line; MouseLeft=("no", "click", "drag"); MouseRight=("no"); MouseMove=("no")
	"all" : Reset all: the entries for all locations are deleted. MouseLeft=("no"); MouseRight=("no"); MouseMove=("no")
MouseLeft	Operation of left mouse button. For which kind of user operation of the left mouse button at the specified location is a sequence to run.
	"no" : No; don't use. Do not run any sequence.
	"click" : Click on left mouse button. The sequence starts when the mouse button is released.
	"drag" : Drag. The dragging motion performed with the left mouse button. Press the left mouse button down and move the mouse simultaneously, then release the button. The sequence starts when the button is pressed down. Typically, it runs until the button is released.
	"2" : Left button double-clicked (clicked twice)
MouseRight	Operation of right mouse button. For which kind of user operation of the right mouse button at the specified location is a sequence to run. (optional , Default value: "no")
	"no" : No; don't use. Do not run any sequence.
	"click" : Click of right mouse button. The sequence is started when the mouse button is released.
MouseMove	A sequence runs when the mouse is moved at the specified location. (optional , Default value: "no")
	"no" : No; don't use. Do not run any sequence.
	"move" : The user moves (slides) the mouse. No mouse button is pressed.

Description:

In the FAMOS Panel, it is possible to save event sequences for a curve window: Thus for instance, it is possible to configure such an event sequence to run in response to clicking the mouse over the curve window. In the event sequence, it is possible to evaluate the mouse pointer's position and to respond accordingly.

The function CwSequenceEnable() is called before the potential initiation of event sequences for the curve window. This thus governs/specifies, in what kind of situations an event sequence can even be triggered.

The curve window must be selected.

The function can be repeatedly called for a variety of situations. For each situation it can be specified for what action by the user a sequence is called.

If the right mouse button is designated for triggering a sequence, then right-clicking the mouse over the curve window does not open a context menu.

If the left mouse button is designated for triggering a sequence, then left-clicking the mouse over the curve window does not perform the action which it normally would perform.

During measurement, the combination of "drag" plus right-clicking starts the sequence upon first pressing one of the two buttons. Typically, the sequence runs as long as at least one of the two buttons is pressed down.

The function should not be called within an event sequence pertaining to the curve window. Exception: At the very end, with the parameters ("all", "no").

Examples:

Specifies that when initializing a Panel, double-clicking over the area of a coordinate system starts a sequence.

```
CwSequenceEnable("all", "no")
CwSequenceEnable("cosys", "2", "no", "no")
```

Specifies that when initializing a Panel, a single click over the area of a coordinate system starts a sequence.

Furthermore, it is expressly specified that no sequence is to start in response to clicking on any other loactions.

```
CwSequenceEnable("cosys", "click", "no", "no")
CwSequenceEnable("area", "no", "no", "no")
CwSequenceEnable("measure", "no", "no", "no")
CwSequenceEnable("link", "no", "no", "no")
```

The curve window's default operation style is restored. No sequence is to run.

```
CwSequenceEnable("all", "no")
```

Specifies that when initializing a Panel, any click anywhere or right-clicking within the coordinate system starts a sequence.

In all these situations, the same event sequence is started. Within the sequence, you can query the current situation using CwSequenceState().

```
CwSequenceEnable("all", "no")
CwSequenceEnable("cosys", "click", "click", "no")
CwSequenceEnable("area", "click", "no", "no")
```

Specifies that when initializing a Panel, a sequence starts after moving the measurement cursor.

The sequence runs upon releasing the mouse button, not already during the cursor motion.

```
CwSequenceEnable("all", "no")
CwSequenceEnable("measure", "click", "no", "no")
```

Specifies that when initializing a Panel, a sequence runs after the link line is moved.

The sequence runs upon releasing the mouse button, not already during the cursor motion.

```
CwSequenceEnable("all", "no")
CwSequenceEnable("link", "click", "no", "no")
```

See also:

CwSequenceState

CwSequenceState

Scope: Curve Windows

Queries the status and mouse position within a FAMOS event sequence pertaining to the curve window.

Declaration:

```
CwSequenceState ( Property ) -> State
```

Parameter:

Property	Get which property?
	"dragging" : drag still running (0: No; 1: Yes). Used for constructing loops.
	"cancel" : Drag operation cancelled (0: No; 1: Yes). A drag-motion may be interrupted by the ESC key or a Timeout, as examples.
	"val.x" : x-coordinate in units of the x-axis
	"val.y" : y-coordinate in units of the y-axis
	"operation" : The action performed (0: none; 1: left click; 2: right click; 3: drag; 4: move; 5: double-left click)
	"location" : The location at which the operation began; see CwSequenceEnable()'s parameter: Location. (1: coordinate system; 2: area; 3: measurement cursor; 4: link line)
	"key.shift" : Shift-key pressed (0: No; 1: Yes)
	"key.ctrl" : Ctrl-key pressed (0: No; 1: Yes)
	"mouse.left" : Left mouse button pressed (0: No; 1: Yes)
	"mouse.right" : Right mouse button pressed (0: No; 1: Yes)
	"outside" : Mouse position outside (0: No; 1: Yes). In cases of drag operations which make the mouse exit the actual object.
	"over.cosys" : Is the mouse over this coordinate system? (0: No; else coordinate system index >= 1)
	"over.line" : Has mouse passed over this line? (0: No; else Line index >= 1). This query can require considerable time. This query returns the current value, not the saved value which corresponds to the status.
	"over.marker" : Is the mouse over this marker (0: No; else Marker index >= 1)
	"over.x-axis" : Is the mouse over the x-axis (0: No; else x-axis index >= 1)
	"over.y-axis" : Is the mouse over the y-axis (0: No; else y-axis index >= 1)
	"pix.x" : x-pixel position. Starting from zero at the left edge of the curve window. Only in conjunction with coordinate system and area
	"pix.y" : y-pixel position. Starting from zero at the top edge of the curve window. Only in conjunction with coordinate system and area
	"related.cosys" : Is the coordinate system involved (0: No; else coordinate system index >= 1)
	"related.line" : Is the line involved? (0: No; else Line index >= 1). E.g. in case of measurement cursor travelling along line
	"related.marker" : Is the marker involved? (0: No; else Marker index >= 1)
	"related.x-axis" : Is the x-axis involved? (0: No; else x-axis index >= 1)
	"related.y-axis" : Is the y-axis involved (0: No; else y-axis index >= 1)
State	
State	The state queried

Description:

In the FAMOS Panel, it is possible to save event sequences for a curve window: Thus for instance, it is possible to configure such an event sequence to run in response to clicking the mouse over the curve window. In the event sequence, it is possible to evaluate the mouse pointer's position and to respond accordingly.

The function CwSequenceState() is called within an event sequence pertaining to the curve window.

The curve window must be selected.

At the beginning of an event sequence, the curve window is already selected.

Returns 0 if the corresponding situation does not pertain.

When a dragging motion is performed, then a loop uses the property "dragging" to query whether the dragging procedure is still in progress.

Within the loop, it is possible to query the current status. In the first iteration, the coordinates at the beginning of the motion are returned. In the last iteration, the coordinates at the end of the motion are returned. This can be applied in order to avoid needing to write multiple copies (before, within and after the loop) of sequence command lines which depend on the coordinates.

Only after a repeat call of the property "dragging" is it possible to return new and different coordinates. This can be applied in order to query consistent and matching values within the loop.

When a drag motion is performed, the property "dragging" returns 1 at least once, to that any pertinent loop will run at least once.

If a drag-motion is cancelled (e.g. ESC), the property "dragging" may return a 0 starting at the 2nd call.

While the event sequence is being performed, FAMOS assumes that no new mouse click occurs. For this reason, an event sequence needs to be processed as quickly as possible. When operating the curve window, the user may need to briefly hold the mouse still before clicking in order to delay until any still running event sequence is finished.

Within a loop having the condition "dragging", CwUpdateEnable(0) is not permitted to be constantly active. The reason is that any movements or clicks of the mouse, which would change the status, would be ignored.

A query of "dragging" returns 0 after approx. 5s without any movement or clicks of the mouse. This is interpreted as a Cancel. A "cancel" query will return 1 in this case.

Processing an event sequence can take much more time if information is written to the FAMOS output window. This would include such things as warnings or other messages. See, for example, the material on calling SetOption("Func.NoInfoMessages", "Yes").

Examples:

Event sequence upon clicking on the coordinate system: The click's coordinates are queried.

```
x = CwSequenceState("val.x")
y = CwSequenceState("val.y")
```

This and other states can be queried and evaluated.

At an appropriate place in the process, e.g. upon initialization of the Panel, clicking on the curve window was allowed:

```
CwSequenceEnable("all", "no")
CwSequenceEnable("cosys", "click", "no", "no")
```

Event sequence upon drag-motion over the coordinate system: A query is run in a loop of whether the drag-motion is still in progress.

```
x = CwSequenceState("val.x")
; here you enter what is to happen when the mouse button is pressed down.
while CwSequenceState("dragging") <> 0
   x = CwSequenceState("val.x")
   y = CwSequenceState("val.y")
   ; here you enter what is to happen during the drag-motion.
end
x = CwSequenceState("val.x")
; here you enter what is to happen upon releasing the mouse button.
```

At an appropriate place in the process, e.g. upon initialization of the Panel, dragging in the curve window was allowed:

```
CwSequenceEnable("all", "no")
CwSequenceEnable("cosys", "drag", "no", "no")
```

Abbreviated event sequence upon drag-motion

This makes use of the fact that all coordinates are enumerated in the loop during the motion, particularly at the beginning and at the end.

Thus if no special treatment is needed at the beginning or end of the drag-motion, the entire processing can be performed within the loop.

```
while CwSequenceState("dragging") <> 0
   x = CwSequenceState("val.x")
   ; here you enter what is to happen during the drag-motion.
end
```

An event sequence is used for multiple situations, e.g. right-click and left-click.

The situation must be queried with either CwSequenceState("operation") or CwSequenceState("location") depending on the combinations possible in CwSequenceEnable().

In more complex applications, Switch Case constructs are used.

```
x = CwSequenceState("val.x")
if CwSequenceState("operation") = 1
   y = CwSequenceState("val.y")
   ; here you enter what is to happen in response to left-clicking.
else
   ; here you enter what is to happen in response to right-clicking.
end
```

At an appropriate place in the process, e.g. upon initialization of the Panel, clicking on the curve window was allowed:

```
CwSequenceEnable("all", "no")
CwSequenceEnable("cosys", "click", "click", "no")
```

Event sequence upon releasing the left measurement cursor. A position is to be specified by using the measurement cursor.

```
x1 = CwSequenceState("val.x")
x2 = CwDisplayGet("measure.x.left")
```

x1 is the position exactly where the mouse button is released.

x2 is the position of the measurement cursor at the current point in time. Usually a (brief) time elapses between release of the button and when the sequence starts and runs. During that time, the mouse position may already have changed.

At an appropriate place in the process, e.g. upon initialization of the Panel, the measurement cursor was allowed to trigger the event sequence:

```
CwSequenceEnable("all", "no")
CwSequenceEnable("measure", "click", "no", "no")
```

Drag the measurement cursors with either the left or right mouse button, or both.

```
while CwSequenceState("dragging") <> 0
   x = CwSequenceState("val.x")
   LeftMouseButtonDown = CwSequenceState("mouse.left")
   RightMouseButtonDown = CwSequenceState("mouse.right")
   xLeft = CwDisplayGet("measure.x.left")
   xRight = CwDisplayGet("measure.x.right")
   ; here you enter what is to happen during the drag-motion.
end
```

At an appropriate place in the process, e.g. upon initialization of the Panel, this support was activated:

```
CwSequenceEnable("all", "no")
CwSequenceEnable("measure", "drag", "click", "no")
```

Was a dragging operation successful? Another important operation is intended to be performed at the end of the dragging operation. But it must not be performed if the user cancels the drag or it is cancelled for any other reason.

```
while CwSequenceState("dragging") <> 0
end
if CwSequenceState("cancel") = 0
   ; here, the end result was reached successfully. Not cancelled!
   ; here, run the important operation
end
```

See also:

CwSequenceEnable

CwUpdateEnable

Prevents refreshing of the curve window

Declaration:

```
CwUpdateEnable ( SvUpdate )
```

Parameter:

SvUpdate	Refresh on/off
	0 : Disables refreshing of the curve window
	1 : Allow refreshing of the curve window (again)

Description:

If [SVUpdate] is set to a nonzero value, then during running of a sequence, WM_PAINT- and other messages are allowed and imc FAMOS remains operable.

Otherwise not! This makes it possible, for example to prevent repeated refreshing of a curve window when redesigning it.

Caution:

[SvUpdate] should only be set to 0 before a group of functions for the purpose of configuring a curve window, and should be reset to 1 immediately afterward. Severe malfunctioning can result from incorrect use of this function!!!

Examples:

A curve window's axes are parameterized. The curve window is redrawn only after the last instruction.

```
CwUpdateEnable(0)
CwSelectWindow("curve1")
CwSelectByIndex("y-axis", 1)
CwAxisSet("range", 4)
CwAxisSet("min", -10)
CwAxisSet("max", 10)
CwSelectByIndex("x-axis", 1)
CwAxisSet("scale", 4)
CwAxisSet("range", 1)
CwUpdateEnable(1)
```

DataFormat?

Determines the data format of a variable.

Declaration:

```
DataFormat? ( Variable ) -> SvFormatCode
```

Parameter:

Variable	Variable whose data format is to be determined
SvFormatCode	
SvFormatCode	Format
	0 : 4 Byte real (float)
	1 : 8 Byte real (double)
	2 : 1 Byte integer
	3 : 2 Byte integer
	4 : 4 Byte integer
	5 : 1 Byte unsigned integer
	6 : 2 Byte unsigned integer
	7 : 4 Byte unsigned integer
	8 : Digital
	9 : 2 Byte integer differences
	10 : 6 Byte unsigned integer
	11 : Time-stamped ASCII
	12 : 8 Byte integer
	13 : 8 Byte unsigned integer
	-1 : Unknown data format
	-2 : Data group
	-3 : Text
	-4 : Text array

Description:

This function determines which data format is used for a variable.

The data format specifies how the individual values are saved in memory/the data carrier. The memory requirements for a data set, the value range and the achievable precision are determined by the data format.

When an XY-data set is transferred to this function, the data format of the y-components is returned. If a complex data set is transferred, the data format of the magnitude or real part is determined. Component characteristics can be used to make an inquiry of other components.

```
DFormPhase = DataFormat?(MagnitudePhase.P)
DFormImag = DataFormat?(RealImag.I)
DFormX = DataFormat?(XYdata.X)
```

For the purpose of verifing a variable's data type, the new funtion VerifyVar() is more appropriate in general.

Examples:

```
Format = DataFormat?(MyData)
IF Format <> 0
   SetDataFormat(MyData, 1, 0, 0)
END
```

If the data set is not already in the real, 4-bytes format, it will be converted to this format.

See also:

SetDataFormat, VerifyVar, GetScale

dB

Conversion to decibels; i.e. 20 * log...

Declaration:

```
dB ( InputData ) -> Transformed
```

Parameter:

InputData	Data to be expressed in dB.
Transformed	
Transformed	Resulting data set

Description:

The data passed as the parameter are expressed in decibels. Complex parameters are transformed to type Dp, i.e. polar coordinate representation with magnitude in dB. Decibel means twenty times the log (base ten) of the specified value and is usually abbreviated as dB. Decibel (dB) is not a unit, but rather an indication of the way a value is to be calculated.

Calculation in dB is standard for transfer functions and sound measurements. A quantity should be calculated in dB when values of very large and very small magnitudes are to be displayed with equal relative accuracy.

- An absolute value calculation always precedes the actual conversion of real numbers into dB.
- With normal or XY-data sets the x-coordinate(s) of the parameter and the result are the same.
- Complex data sets of the type Dp cannot be processed using this function.
- The parameter may be structured (events/segments).
- The dB calculation sets the y-unit to dB, a common notation for axis labels in graphs. When combined with other units, dB is always discarded, since it is not an actual unit.
- When this function is used for complex data sets displayed as a polar plot, the polar plot display disappears.

Examples:

```
NDdecibel = dB(NDdata)
; This formula is equivalent to the formula:
NDdecibel = 20 * log(Abs(NDdata))
; only the unit is different.
```

Calculation of a spectrum in dB:

```
DPspectrum = dB(FFT(NDdata))
```

Calculations in dB can result in very large negative values, which are a hindrance for further processing. The Clip function should be used to limit the dB number to reasonable minimum values, e.g. -100dB. The upper limit of 1000 should be set to a sufficiently large value:

```
NDdecibel = Clip(dB(NDdata), 1000, -100)
```

See also:

idB, log, Clip

DbBeginTransaction

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Beginning of a transaction

Declaration:

```
DbBeginTransaction ( ConnectID ) -> ErrorCode
```

Parameter:

ConnectID	Connection identifier
ErrorCode	
ErrorCode	Result: Error number for error number, else zero.
	= 0 : No error
	< 0 : Error number

Description:

A transaction is begun with this function. The function can be used when multiple tasks in the sequence are to be combined with each other so that they can be executed as a single processing unit.

This function is not to be used when only a single call of DbSql(), DbInsert(), DbUpdate() or DbUpdate1() occurs. These functions have an internal transaction control.

If the function DbBeginTransaction() is called, the internal transaction control is rescinded. The internal transaction control only goes back into effect again after calling DbEndTransaction().

This function must always be called as a pair along with the function DbEndTransaction().

Examples:

The group grpInsert1 is inserted into the table "User" and the FAMOS group grpInsert2 into the table "Group". Toward this end, a transaction is started. If both functions were completed without any error, the transaction is concluded with a Commit. In case of an error, a rollback is performed.

```
errorcode=DbBeginTransaction(ConnectID)
if errorcode < 0
   errortext=DbGetLastErrorText(ConnectID,1)
end
result1=DbInsert(ConnectID,"User",grpInsert1)
result2=DbInsert(ConnectID,"Group",grpInsert2)
commit=1
if result1 < 0 or result2 < 0
   errortext=DbGetLastErrorText(ConnectID,1)
   commit=0
end
errorcode = DbEndtransaction(ConnectID,commit)
if errorcode < 0
   errortext=DbGetLastErrorText(ConnectID,1)
end
```

See also:

Transaktionen, DbEndTransaction, DbGetLastErrorText, DbGetLastErrorCode

DbClosePanel

Closes the active Panel
This function is obsolete; instead the newer function PnClose() should be used (as of V2022).

Declaration:

```
DbClosePanel ( SvOption )
```

Parameter:

SvOption	Options parameter or return value of a Panel dialog

Description:

Closes the active Panel or all Panels.

The meaning of the Options parameter depends on the call's context:

Call within an event sequence of a Panel-dialog which had been started by the function Dialog():

The Panel-dialog is closed and the parameter passed is used as the return value of the Dialog()-command. The function thus behaves analogously to the function DlgCloseDialog() for user-defined dialogs.

Else:

A value of 0 signifies that the active Panel is to be closed. A 1 closes all open panels.

The Panels will not be saved; any canges will be lost.

Examples:

A Panel file is opened. A variety of updates are performed, after which the Panel is printed and then closed again.

```
err = DbLoadPanel("d:\templates\result.panel", 0)
IF err <> 0
   BoxMessage("Error", GetLastError(), "!1")
ELSE
   ; various updates
   ; ...
   PnPrint(0)
   DbClosePanel(0)
END
```

A Panel 'InputValue.panel' consists of, among other things, an input box "input" for entering a positive numerical value, as well as 2 buttons 'OK' and 'Cancel'. The Dialog()-command returns the entered value, or -1 to cancel.

Event-sequence 'Button pressed' for the 'OK'-button:

```
value = PnGetValue("input")
DbClosePanel(value)
```

Event-sequence 'Button pressed' for the 'Cancle'-button

```
DbClosePanel(-1)
```

Event-sequence 'Close' (user utilizes system menu to close):

```
; Same behavior as for the 'Cancel'-button
DbClosePanel(-1)
```

Calling the dialog:

```
value = Dialog("InputValue.panel", "", 0)
IF value < 0
   EXITSEQUENCE 0
END
;Continue with sequence...
...
```

See also:

PnClose, PnLoad, DbLoadPanel, DbShow, Dialog

DbConnect

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Sets up a connection to the specified database system.

Declaration:

```
DbConnect ( ServerType, TxServerName, TxDatabaseName, TxUserName, TxPassword, TxExtConnectionString ) ->
ConnectID
```

Parameter:

ServerType	The database system's type
	1 : Access to Microsoft SQL Server Compact Edition 4.0. by means of ADO.Net Provider
	2 : Access to Microsoft SQL Server (2005, 2008) by means of ADO.Net Provider
	3 : Access to an Oracle database system (10g, 11g, 12c) by means of Oracle Data Provider for .NET.
	4 : Access to a MySQL Server (5.5, 5.6) by means of MySQL Connector/NET
	5 : Access to a server via ODBC
	6 : Access to an Oracle database system (10g, 11g, 12c) using the Oracle Data Provider for .NET, Managed Driver
TxServerName	Name of the server
TxDatabaseName	Name of the database
TxUserName	The user name for logging on to the server
TxPassword	The password for the specified user name
TxExtConnectionString	Extension for the standard connection string
ConnectID	
ConnectID	Result: Connection identifier or error number
	> 0 : Connection identificator
	< 0 : Error number

Description:

A connection string is assembled from the parameters of the function. A connection object is created and a connection test is performed. After a successful test, a valid connection identifier is returned. In order to determine an error, the functions DbGetLastErrorCode() and / or DbGetLastErrorText() must then be used.

Multithreading: All functions of the Database kit may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

A connection to an Oracle database server is established by using the file tnsnames.ora.

```
DbInitialize()
ConnectId = DbConnect(3,"ORCL","","MyUsername"," MyPassword","")
if ConnectId < 0
   errortext=DbGetLastErrorText(0,1)
   exitsequence 1
end
```

A connection to an MySQL server is established. It operates via the TCP-port 3307. The port-number is specified as the extension of the connection string.

```
DbInitialize()
ConnectId=DbConnect(4,"localhost","Sample_DB","MyUsername","MyPassword ", "port=3307;")
if ConnectId < 0
   errortext=DbGetLastErrorText(0,1)
   exitsequence 1
end
```

A connection is established to an MS SQL server by means of Windows authentication.

```
DbInitialize()
ConnectId=DbConnect(2,"MyPC\SQL2008EXPRESS","SampleDB","","","")
if ConnectId < 0
```

```
   errortext=DbGetLastErrorText(0,1)
   exitsequence 1
end
```

Establish a connection to a MySQL server via ODBC. A system data source called "DSN_MySql" is set up for this in the ODBC manager.

```
DbInitialize()
ConnectId=DbConnect(5,"DSN_MySql","","MyUsername","MyPassword ","")
if ConnectId < 0
   errortext=DbGetLastErrorText(0,1)
   exitsequence 1
end
```

See the user's manual for more examples.

See also:

Datenbankverbindung, DbDisconnect, DbGetLastErrorText, DbGetLastErrorCode

DbDisconnect

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Disconnects a connection to a database.

Declaration:

```
DbDisconnect ( ConnectID ) -> ErrorCode
```

Parameter:

ConnectID	Connection identifier
ErrorCode	
ErrorCode	Result: Error number for error number, else zero.
	= 0 : No error
	< 0 : Error number

Description:

Disconnections the connection to the database.

The functions DbConnect() and DbDisconnect() should always be called in pairs.

Examples:

A connection to an Oracle database server is established. Next, the database system is accessed. At the end, the connection is disconnected.

```
ConnectId = DbConnect(3,"ORCL","","MyUsername"," MyPassword","")
   :
   :
errorcode = DbDisconnect(ConnectId)
```

See also:

Datenbankverbindung, DbConnect, DbGetLastErrorText, DbGetLastErrorCode

DbEndTransaction

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

A transaction is codified, or reset.

Declaration:

```
DbEndTransaction ( ConnectID, Commit ) -> ErrorCode
```

Parameter:

ConnectID	Connection identifier
Commit	Saves or reverses the transaction steps.
	0 : All steps of the transaction are reversed (Rollback).
	<> 0 : Processing steps are permanently saved (Commit).
ErrorCode	
ErrorCode	Result: Error number for error number, else zero.
	= 0 : No error
	< 0 : Error number

Description:

This function must always be called in a pair along with the function DbBeginTransaction().

Examples:

See also:

Transaktionen, DbBeginTransaction, DbGetLastErrorText, DbGetLastErrorCode

DbGetLastErrorCode

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Determines the error number of the las Database Kit error to occur.

Declaration:

`DbGetLastErrorCode ( ConnectID, ClearError ) -> ErrorCode`

Parameter:

ConnectID	Connection identifier
ClearError	Specifies whether the last error is to be deleted.
	1 : Delete error
	0 : Don't delete error
ErrorCode	
ErrorCode	Result: The error number of the last Database Kit error to occur

Description:

With this function, the error number of the last error to occur can be read.

If a 1 is specified for the parameter "ClearError", then the error memory is subsequently cleared.

Each connection has its own error memory. For this reason, the parameter "ConnectId" must be specified.

In case of an error, most Kit-functions return the error number. Error numbers are always negative numbers.

In the user's manual, the errors are described in the section Error Codes.

Examples:

See also:

Verschiedenes, DbGetLastErrorText

DbGetLastErrorText

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Gets the error text of the last Database-Kit error to occur.

Declaration:

```
DbGetLastErrorText ( ConnectID, ClearError ) -> TxError
```

Parameter:

ConnectID	Connection identifier
ClearError	Specifies whether the last error is to be deleted.
	1 : Delete error
	0 : Don't delete error
TxError	
TxError	Error text

Description:

With this functon, it is possible to read the text of the last error to occur.

If a 1 is specified for the parameter "ClearError", then the error memory is subsequently cleared.

Each connection has its own error memory. For this reason, the parameter "ConnectId" must be specified.

If the function DbConnect() fails, then when calling DbGetLastError(), the parameter ConnectId =0 must be specified.

In the user's manual, the errors are described in the section Error Codes.

Examples:

See also:

Verschiedenes, DbGetLastErrorCode

DbInitialize

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Initialization of the settings in the Kit. Any existing database connections are disconnected.

Declaration:

```
DbInitialize ( ) -> KitVersion
```

Parameter:

KitVersion	
KitVersion	Result: Number of the kit version

Description:

The function initializes the Database-Kit. A defined initial condition of the kit is established. Any transactions are concluded with a rollback. All connection objects and the internal error memory will be deleted.

It is recommended to call this function at the start of every sequence.

Multithreading: All functions of the Database kit may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

DbInsert

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Execution of an INSERT-instruction over multiple database rows

Declaration:

```
DbInsert ( ConnectID, TxTableName, GrpInsertValues ) -> Result
```

Parameter:

ConnectID	Connection identifier
TxTableName	Name of the database table
GrpInsertValues	The group contains text boxes and data sets to be inserted into the database table.
Result	
Result	Result: An error number in case of error, else the number of database rows inserted.
	>= 0 : Number of database rows inserted
	< 0 : Error number

Description:

Using this function, it is possible to insert data from FAMOS-objects into a database table. The names of the data sets and text boxes in the group need to be present in the database table as columns. All elements contained in the group are inserted into the table. The group's data sets and text boxes must be of the same size. The size determines the number of table rows inserted.

If a Blob-column is incuded, then the only permitted size of the group elements is 1. The name of the data set corresponding to the Blob-column is inserted into the table's one box. If the group elements' size is > 1, then only the data having the first index are inserted.

In case an error occurs, the error text can be determined using the function DbGetLastErrorText().

Examples:

A new row with a Blob-column is created.

```
; --- Important! Only one row may be loaded ------------
grpResult = DbSelect(ConnectID,"Select * from Measurement WHERE Id = 1","")
errorcode = DbGetLastErrorCode(ConnectID,0)
if errorcode < 0
   errortext = DbGetLastErrorText(ConnectID,1)
end
; --- Determining new ID value for insertion ------------------
grpMax = DbSelect(ConnectID,"Select Max(Id) AS MaxID from Measurement ","")
maxid = grpMax:MaxID[1]
grpResult:ID[1]=maxid+1
; --- Loading channel Sintest1 --------------------------------------
FileLoad("Sintest1.dat","",0)
grpResult:CHANNEL=sintest1
; --- Saving triggering time as a date ---------------------------
grpResult:Date[1]=Time?(grpResult:CHANNEL)
grpResult:Name[1] ="Sintest1"
grpResult:Maximum[1]=Max(sintest1)
; --- Inserting a channel as a Blob into the table --------------
; --- Important! The group elements may only have the size 1 -
Result = DbInsert(ConnectID,"Measurement",grpResult)
if result < 0
   errortext = DbGetLastErrorText(ConnectID,1)
end
```

See also:

Datenzugriff, DbSelect, DbUpdate1, DbUpdate, DbBeginTransaction, DbEndTransaction, DbGetLastErrorText, DbGetLastErrorCode

DbLoadPanel

A Panel-file is loaded and displayed.
The function is obsolete; instead the newer function PnLoad() should be used (as of V2022).

Declaration:

```
DbLoadPanel ( TxFilename, Zero ) -> Success
```

Parameter:

TxFilename	Name of the Panel file to be opened
Zero	Reserved, always set to 0
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

Loads the Panel-file specified.

If the specified filename has no name extension, then the system assumes ".panel".

If no complete path is specified with the filename, the system searches for the file in this sequence of folders:

- Project folder: When a project is active, the search is conducted initially in the current project's folder.
- Default folder for Panel-files: FAMOS presettings for Panels/dialogs/sequences

To start a Panel in Dialog-mode, use the command Dialog().

Examples:

A Panel file is opened. A variety of updates are performed, after which the Panel is printed and then closed again.

```
err = DbLoadPanel("d:\templates\result.panel", 0)
IF err <> 0
   BoxMessage("Error", GetLastError(), "!1")
ELSE
   ; various updates
   ; ...
   PnPrint(0)
   PnClose(0)
END
```

See also:

PnLoad, PnClose, DbClosePanel

DbOption

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Setting of optional parameters

Declaration:

```
DbOption ( ConnectID, TxParametername, TxParameterValue ) -> ErrorCode
```

Parameter:

ConnectID	Connection identifier
TxParametername	Name of the stipulated optional parameter
	TimeStampMap : If the parameter value is "yes", then Date/Time-columns are converted to a text-array; for "no", to a normal data set. Default value = "no".
	NumericAsDouble : If the parameter value is "yes", each numerical column in a data set is converted to a data set in the 8-Byte Real data format; for "no", the data set as the appropriate data format (see user's manual setion Conversion of Data). Default value = "no".
	CommandTimeOut : By means of this parameter, the time to wait is specified, in seconds, until the attempt to execute a command is concluded and an error generated. The default value is 30.
	BlobAsChannel : If the parameter value is "yes", the content of the blob field is interpreted as a channel, if "no" is used as a binary object (e.g. image). Default value = "yes".
TxParameterValue	Parameter value
ErrorCode	
ErrorCode	Result: Error number for error number, else zero.
	= 0 : No error
	< 0 : Error number

Description:

With this function, optional parameters for each connection can be set.

The name of this optional parameter must match a stipulated name.

Examples:

See also:

Verschiedenes, DbSelect

DbSelect

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Execution of a SELECT instruction. The selected data are joined together and returned as a FAMOS group.

Declaration:

```
DbSelect ( ConnectID, TxSqlStatement, TxTimeColumn ) -> GrpResult
```

Parameter:

ConnectID	Connection identifier
TxSqlStatement	SQL SELECT instruction. This instruction is passed to the database unchanged.
TxTimeColumn	If not "", the content of this column is used to assign a time track to the numerical group elements.
GrpResult	
GrpResult	Result: This group is filled with text boxes and data sets which correspond to the selected column names. In case of error, the group is empty.

Description:

The function executes the SELECT instruction, converts the columns to data sets/text boxes.

These are joined together in and returned as a FAMOS group.

Each column results in a group element (data set or text box) of the same name.

The number of rows imported determines the size of the group elements.

The user's manual contains an overview of how the data types of the columns are converted to FAMOS objects.

If contains one or more Blob-columns, the result may only return one row.

The values of the Blob-column boxes result in one data set each. If the query returns multiple result rows, the Blob-columns are not converted.

If the parameter TxTimeColumn is set and it corresponds to a numerical column, then all other numerical columns are converted to XY-data sets, with the time column as a X-component.

The optional parameter TimeStampMap = yes determines that all columns of the type Date/Time are converted to a text box. By default, these columns are converted to a normal data set in the imc time format.

The optional parameter NumericAsDouble = yes determines that all numerical columns in data sets are converted with the format 8 Byte Real (Double).

To determine any error occurring, the functions DbGetLastErrorCode() and/or DbGetLastErrorText() are to be used subsequently.

Examples:

With this query, all columns of the table measurement are to be imported.

```
grpResult = DbSelect(ConnectID,"Select * from Measurement WHERE Id > 0","")
errorcode=DbGetLastErrorCode(ConnectID,0)
if errorcode < 0
   errortext=DbGetLastErrorText(ConnectID,1)
end
```

See also:

Datenzugriff, DbOption, DbGetLastErrorText, DbGetLastErrorCode

DbSetActivePanel

Scope: Panels

Sets the active Panel

Declaration:

`DbSetActivePanel ( TxTitle )`

Parameter:

TxTitle	Title of the Panel to be activated

Description:

This function is only helpful if multiple Panels are displayed simultaneously. The kit's functions always apply to the currently active (i.e. visible) Panel.

Examples:

While multiple Panels are open, the user is prompted whether to print the last Panel opened. Since the user may have manually activated a different Panel in the meantime, the system must ensure that the subsequent print command is actuall applied to the desired Panel.

Multithreading: The functions for Panel remote control can be called anywhere and have a global effect. The Panel selected here is therefore valid for all execution threads.

```
err = PnLoad("Dontcare.panel")
;...
err = PnLoad("Report.panel")
;...
IF err = 0
   ok = BoxMessage("Report","Print ?","?2")
   IF ok = 1
      DbSetActivePanel("Report")
      PnPrint(0)
   END
END
```

See also:

DbShow, PnLoad

DbSetPanelWindow

Scope: Panels

Sets the position of the active Panel on the screen.

Declaration:

`DbSetPanelWindow ( Status, left, top, wide, high )`

Parameter:

Status	Window status
	0 : Minimized (as icon)
	1 : Normal (floating)
	2 : Maximized (full screen)
	3 : Docked in main window
left	X-coodinate of left upper corner
top	Y-coodinate of left upper corner
wide	Window width
high	Window height

Description:

With this function it is possible to control the position and size of the active Panel window on the screen.

The specified window coordinates determine the window's default position in free-floating mode. The entry is in screen pixels.

If [wide] and [high] are both 0, thecurrent size remains intact; only the position is corrected accordingly. If additionally [left] and [top] are 0, the default position remains intact.

Examples:

A Panel-file is loaded. The Panel is undocked from the main window and displayed as a free floating window of size 600x400 in the right upper corner of the primary monitor.

```
err = PnLoad("d:\templates\result.panel")
IF err = 0
   right = GetSystemInfo("Screen.PrimaryWorkArea", "left") + GetSystemInfo("Screen.PrimaryWorkArea", "width")
   DbSetPanelWindow(1, right-600, 0, 600, 400)
END
```

See also:

DbShow, PnClose

DbShow

Scope: Panels

Controls the visibility of all open Panel-windows

Declaration:

```
DbShow ( Task )
```

Parameter:

Task	Command parameter
	0 : Close all Panels
	1 : Display as free window
	2 : Display as icons
	3 : Display in full screen

Description:

With this function, the display status of all open Panel-windows can be controlled.

Upon exiting (Task = 0), all changes to the Panels which were not yet saved are lost.

As of FAMOS 7.0, this function is retained for reasons of compatibility with older FAMOS-sequences; in newly created sequences, the function DbSetPanelWindow() should be used, by means of which the visibility of each Panel can be controlled separately.

See also:

PnLoad, PnClose, DbSetPanelWindow

DbSql

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Execution of an SQL instruction without returning of data, i.e. INSERT, UPDATE or DELETE can be executed.

Declaration:

```
DbSql ( ConnectID, TxSqlStatement ) -> Result
```

Parameter:

ConnectID	Connection identifier
TxSqlStatement	SQL instruction. This instruction is passed to the database system unchanged.
Result	
Result	Result: An error number in case of error, else the number of data rows affected.
	>= 0 : Number of data rows affected
	< 0 : Error number

Description:

Mit dieser Funktion können SQL- Anweisungen ausgeführt werden, die keine Ergebnisse zurückliefern (UPDATE, INSERT, DELETE). Der Rückgabewert entspricht der Anzahl der betroffenen Zeilen. Es lassen sich aber auch DDL Kommandos wie z.B. "ALTER TABLE Messung ADD MyCol INTEGER" ausführen. Bei einem DDL- Kommando ist der Rückgabewert bei erfolgreicher Ausführung 0.
In case an error occurs, the error text can be determined using the function DbGetLastErrorText().

Examples:

All values in the column "Status" of the table "Measurement" are set to "Ready".

```
result =DbSql(ConnectId,"Update measurement to Status ='Ready'")
if result < 0
   errortext=DbGetLastErrorText(ConnectId,1)
end
```

The column "MyCol" having the Integer data type is added to the table "Measurement".

```
result =DbSql(ConnectId,"ALTER TABLE Messung ADD MyCol Integer")
if result < 0
   errortext=DbGetLastErrorText(ConnectId,1)
end
```

See also:

Datenzugriff, DbBeginTransaction, DbEndTransaction, DbInsert, DbUpdate1, DbUpdate, DbGetLastErrorText, DbGetLastErrorCode

DbUpdate

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Repeated execution of an UPDATE instruction

Declaration:

```
DbUpdate ( ConnectID, TxSqlStatement, GrpUpdateValues ) -> Result
```

Parameter:

ConnectID	Connection identifier
TxSqlStatement	SQL UPDATE instruction
GrpUpdateValues	Contains data for updating and for the WHERE condition.
Result	
Result	Result: An error number in case of error; else the number of updated database rows
	>= 0 : Number of updated database rows
	< 0 : Error number

Description:

The function executes the UPDATE statement n times according to the size of the group elements transferred. An index loops from 1 to the size of the group elements. The values of the group elements addressed by the index replace the placeholders in the UPDATE statement. The update instruction is executed.

The UPDATE statement must be specified in full as a TxSqlStatement. The question mark **?+Group element name** is to be used as a placeholder for the values to be replaced from the group elements. The placeholders in the SET part and in the WHERE condition are replaced by the data from the GrpUpdateValues group. If there is a blob column in the UPDATE statement, only the 1st row in the database table is updated.

In case an error occurs, the error text can be determined using the function DbGetLastErrorText().

Examples:

The columns ID, NAME and DATE are imported. The group elements have a size of 13. Now, the individual names will be changed. Subsequently, the changed group updates the database table.

The UPDATE-instruction is executed 13 times. Upon each UPDATE, the placeholders are replaced with the corresponding data from the group elements. The function's result is 13.

```
grpResult =DbSelect(ConnectID,"Select ID, Name,Date from Measurement","")
; --- Constructing new name for the updating ---------------
for i= 1 to leng?(grpResult:ID)  Step 1
   measurement=grpResult:Name[i] +"_"+ TForm(grpResult:ID[i],"")
   grpResult:Name[i]=measurement
end
updatestatement="update Messung set Name=?Name where Id =?Id "
result=DbUpdate(ConnectId,updatestatement,grpResult)
if result < 0
   errortext=DbGetLastErrorText(ConnectID,1)
end
```

See also:

Datenzugriff, DbUpdate1, DbBeginTransaction, DbEndTransaction, DbGetLastErrorText, DbGetLastErrorCode

DbUpdate1

Scope: Database remote control

Required Extension-Kit: (imc Database Kit)

Execution of an UPDATE instruction

Declaration:

```
DbUpdate1 ( ConnectID, TxSqlStatement, GrpUpdateValues, SampleIndex ) -> Result
```

Parameter:

ConnectID	Connection identifier
TxSqlStatement	SQL UPDATE instruction
GrpUpdateValues	Contains data for updating and for the WHERE condition
SampleIndex	Updating of the database values (1...) only occurs with the values from group elements addressed via the SampleIndex.
Result	
Result	Result: An error number in case of error; else the number of updated database rows
	>= 0 : Number of updated database rows
	< 0 : Error number

Description:

Mit dieser Funktion wird eine UPDATE-Anweisung ausgeführt. Die UPDATE-Anweisung muss komplett als TxSqlStatement angegeben sein. The question mark **?+Group element name** is to be used as a placeholder for the values to be replaced from the group elements. The placeholders in the SET part and in the WHERE condition are replaced by the data from the GrpUpdateValues group.

Only the data from the group elements that are addressed by the SampleIndex are used for the update instruction. There can be more group elements in the group than are required for the update instruction. Only the group elements that are specified in the update instruction are used.

In case an error occurs, the error text can be determined using the function DbGetLastErrorText().

Examples:

The values of the column MAXIMUM should be be set to 1000 where the value of the column NAME ends with 1x.

After reading in the columns, the value is set from the maximum with SampleIndex = 1 to 1000. There is a placeholder in the SET part of the UPDATE statement. The WHERE condition "Name LIKE '% 1_'" filters 4 rows that are updated. The result of the function is 4.

```
grpResult =DbSelect(ConnectID,"Select Id,Name,Date,Maximum from Masurement", "")
grpResult:Maximum[1]=1000;
updatestatement="update Messung set Maximum=?Maximum where Name LIKE '%1_' "
result=DbUpdate1(ConnectId,updatestatement,grpResult,1)
if result < 0
   errortext=DbGetLastErrorText(ConnectID,1)
end
```

The value of the column MAXIMUM and of the row with the ID-value 11 is to be set to 0.

```
grpResult =DbSelect(ConnectID,"Select Id,Name,Date,Maximum from Measurement", "")
;The 11th value of the data set MAXIMUM will be set to 0.
grpResult:Maximum[11]=0;
updatestatement="update Messung set Maximum=?Maximum where Id =?Id"
;The UPDATE-instruction is executed with the data which are addressed by means of the SampleIndex 11.
result=DbUpdate1(ConnectId,updatestatement,grpResult,11)
if result < 0
   errortext=DbGetLastErrorText(ConnectID,1)
end
```

See also:

Datenzugriff, DbUpdate, DbBeginTransaction, DbEndTransaction, DbGetLastErrorText, DbGetLastErrorCode

DDEInq

Available in: Professional Edition and above

Requests a text string, single value or data set from another application via DDE.
DDE is an outdated technology and may not be supported by future versions of Windows and MS Office. For communication with EXCEL, the functions from the EXCEL function group, such as XLWbOpen () and XLSetValues ??(), should be used instead.

Declaration:

```
DDEInq ( TxApplication, TxTopic, TxElement, SvDataType ) -> Data
```

Parameter:

TxApplication	The name of the program from which data are to be requested
TxTopic	The topic of the conversation
TxElement	Element requested
SvDataType	Data type
	1 : Text
	2 : Single value
	3 : Data set
Data	
Data	Data read from application; type according to [SvDataType].

Description:

Requests a text string, single value or data set from another application via DDE.

DDE (Dynamic Data Exchange) is a communication mechanism under Windows for exchanging data or commands, which is supported bby many Windows applications.

Please refer to documentation of the target applications for information about available topics and the designations of the elements.

- Text strings are always requested in ASCII format
- If text strings are requested, the maximum string length is 255 characters.
- When a single value or a data set is requested, an inquiry for imc FAMOS-DDE format is performed first. If this format is not accepted, a second request is made for data in ASCII format.
- When data are read from another application in ASCII format, any separators may be located between the values. The numbers may be real numbers, written with a decimal point if desired. Decimal commas are not allowed.
- An error message is generated if the other application does not respond or provide any data.
- A complete DDE conversation is carried out every time, including initialization, query and end.
- When CwUpdateEnable(0) or CvUpdate(0) was called, no DDE-communication is possible!

Examples:

An existing Excel file with the table sheet "Table1" is opened.

The element in the 1st line, 2nd column is queried and returned as text.

```
Execute("Excel.exe","c:\temp\1.xlsx", "open", 0, 3)
TxCell = DDEInq("EXCEL","Table1","R1C2", 1)
```

Note: For the remote control of EXCEL, the functions of the Excel kit (XlWbOpen, XlSetValues, XlGetValues ...) are generally more suitable.

```
Dat1 = DDEInq("EXCEL","Tab1","R1C1:R100C1", 3)
```

Reads a maximum of 100 values from the first column of the Excel table "Sheet1".

See also:

DDESend, DDESet

DDESend

Available in: Professional Edition and above

A commando is sent to another application via DDE.
DDE is an outdated technology and may not be supported by future versions of Windows and MS Office. For communication with EXCEL, the functions from the EXCEL function group, such as XLWbOpen () and XLSetValues ??(), should be used instead.

Declaration:

```
DDESend ( TxApplication, TxTopic, TxCommand ) -> SvStatus
```

Parameter:

TxApplication	The name of the program to which the commands are to be sent
TxTopic	The topic of the conversation
TxCommand	The command itself which is to be sent
SvStatus	
SvStatus	0, if the command was executed -1, if the other application does not respond Else: REturn value from the other application

Description:

A command is sent to another application by means of DDE.

DDE (Dynamic Data Exchange) is a communication mechanism under Windows for exchanging data or commands, which is supported bby many Windows applications.

Please refer to documentation of the target application for information about available topics and the syntax of the command texts.

- The syntax of the command is determined by the receiving application.
- The return value indicates whether the other application received and executed the command.
- A complete DDE conversation is carried out every time, including initialization, query and end.
- If the other application is busy, the call repeats until the other application responds.
- When CwUpdateEnable(0) or CvUpdate(0) was called, no DDE-communication is possible!

Examples:

```
error = DDESend("EXCEL","SYSTEM","[NEW]")
IF error
   PAUSE Command not executed
END
```

A command is sent to EXCEL. This command causes EXCEL to open a new spreadsheet.

```
TxCommand = "[Open(""Table1.xls"")]"
eror = DDESend("EXCEL", "SYSTEM", TxCommand)
```

Excel is instructed to open the file "Table1.xls".

Note: For the remote control of EXCEL, the functions of the Excel kit (XlWbOpen, XlSetValues, XlGetValues ...) are generally more suitable.

See also:

DDEInq, DDESet

DDESepar

Available in: Professional Edition and above

The separator for transmission of ASCII texts by means of the function DDESet() is defined.

Declaration:

```
DDESepar ( TxChar )
```

Parameter:

TxChar	A text containing the separator character(s)

Description:

The function DDESet can transfer data sets in ASCII format. The individual numerical values must be separated from each other by some easily recognizable character or string. This character of string can be defined using either this function or SetOption().

The transmitted ASCII-text has this structure:

```
Number Separator Number Separator .. Number
```

- You can also set the separator plus additional formatting options using the dialog "Options"/"DDE" or the function SetOption().
- The separator string for numbers should normally not contain any numbers.
- Thge string may have a maximum of 4 characters.
  Multithreading: The function has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
DDESepar(SvToChar(0x09))
; or better:
DDESepar("~009")
```

A TAB sign is specified as a separator (ASCII code: 9hex). This is appropriate for transferring a row to Excel.

```
DDESepar(TAdd(SvToChar(0x0d), SvToChar(0x0a)))
; or better
DDESepar("~013~010")
```

A combination of return/line feed is selected as the separator. This is appropriate for transferring a column to Excel.

See also:

DDESet, DDEInq, SetOption

DDESet

Available in: Professional Edition and above

Transfers the contents of a variable to another application via DDE.
DDE is an outdated technology and may not be supported by future versions of Windows and MS Office. For communication with EXCEL, the functions from the EXCEL function group, such as XLWbOpen () and XLSetValues ??(), should be used instead.

Declaration:

```
DDESet ( TxApplication, TxTopic, TxElement, Data, SvDataType ) -> SvStatus
```

Parameter:

TxApplication	Name of the program to which the data are to be transferred
TxTopic	The topic of the conversation
TxElement	Name of the element (variable) in the receiving program to which the data are sent
Data	Data to be transferred: single value, equidistant data set or text string
SvDataType	Data type
	1 : Transfer data as text (ASCII-format)
	2 : Transfer data in the imc FAMOS format (also for data type=3)
SvStatus	
SvStatus	0, if data were transferred -1, if the other application does not respond Else: Value returned by the other application

Description:

Sets a text string, single value or data set from another application via DDE.

DDE (Dynamic Data Exchange) is a communication mechanism under Windows for exchanging data or commands, which is supported bby many Windows applications.

Please refer to documentation of the target applications for information about available topics and the designations of the elements.

- Use the DDESepar function 0r SetOption() to define a separator for transfer of normal data sets in ASCII format.
- When transferring numerical values in the ASCII-format, you can provide detailed specifications of the format using the dialog "Options"/"DDE".
- The return value indicates whether the other application received the data.
- A complete DDE conversation is carried out every time, including initialization, query and end.
- When CwUpdateEnable(0) or CvUpdate(0) was called, no DDE-communication is possible!

Examples:

An existing Excel file with the table sheet "Table1" is opened.

The cell in the 1st line, 2nd column is set to "Hello Excel".

```
Execute("Excel.exe", "c:\temp\Workbook1.xlsx", "open",0,3)
ret = DDESet("EXCEL", "Table1", "R1C2", "Hello Excel", 1)
```

Note: For the remote control of EXCEL, the functions of the Excel kit (XlWbOpen, XlSetValues, XlGetValues ...) are generally more suitable.

Enters the values in data set "wave" in the first column of the Excel table "Tab1". Data are converted to the ASCII format before they are transferred.

```
ret = DDESet("EXCEL", "Table1", "R2C1:R100C1", data, 1)
```

See also:

DDEInq, DDESepar, DDESend, SetOption

DEFAULT

Initializes the 'ELSE'-branch in a in a case differentiation (multiple branching) induced by the command SWITCH. The subsequent command block is run if the preceding CASE condition is met.

Declaration:

`DEFAULT`

Description

The end of the instruction belonging to this DEFAULT is determined by the END-command belonging to the SWITCH.

Examples:

A descriptive text is formulated for a value nomally lying within the range from 0 to 100. If it is ouside of this range, an error message is displayed.

```
SWITCH Round(value, 1)
CASE 0 TO 48
    Tx = "Lower half"
CASE 49,50,51
    Tx = "Center"
CASE 52 To 100
   Tx = "Upper half"
DEFAULT
   PAUSE Invalid Value
END
```

See also:

SWITCH, CASE, IF

DELETE

Deletes a variable

Declaration:

```
DELETE VariableName
```

Parameter:

VariableName	Name of the variable to be deleted

Description

The variable supplied as the parameter is deleted from the Variable list.

The wildcards character may be specified to delete a series of variables. The wildcard character '?' represents an exact character, '*' represents an undefined number of characters.

```
DELETE *
```

All variables are deleted.

```
DELETE ??
```

All variables whose names are exactly 2 characters long are deleted.

```
DELETE *1a*
```

All variables, in which the string '1a' is present (also at the beginning or end) are deleted.

```
DELETE a*:channel?
```

All channels with the name 'Channel', followed by any character, are deleted from those data groups whose names begin with an 'a'.

```
DELETE temp?:@Test_2022_12_01
```

All channels that belong to the "Test_2022_12_01" measurement and whose name is made up of "temp" and another character are deleted.

Examples:

A variable is created, saved and then deleted:

```
Var = 2
FileSave("result.dat", "", 0, Var)
DELETE Var
```

One channel in a data group is deleted.

```
DELETE Group:Channel1
```

See also:

SHOW, RENAME

DeqCalc

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: The actual calculation

Declaration:

```
DeqCalc ( ) -> Remainder count
```

Parameter:

Remainder count	
Remainder count	Remainder count; the remaining number of sampling points for the calculation

Description:

This function performs tha actual calculation. Previously, all integration variables, constants, the time basis and input signals had been specified.

Upon first calling this function, the initialization and testing are performed, since this generally could only happen once definition of the entire task is complete.

Upon the first call, the function begins the calculation. If the calculation is not completed by a certain amount of time, the function returns anyway.

This function is usually called within a loop until the calculation is completely finished. This makes it possible to cancel the operation.

The calculation can require a long time in total.

A single call of the function lasts approx. 0.5s, if the entire calculation is not yet completed. It is also possible for the time to be excceded if calculation of a subsequent sample point is very demanding.

When the calculation has been completed, the function returns 0.

Solving differential equations always begins with calling DeqInit(). After all necessary initialization has occurred, the actual calculation with DeqCalc() begins before the reuslts are retrieved by means of DeqResult().

Examples:

Calculation with checking of the remaining sampling points to be calculated.

```
local Remain = 1
while Remain > 0
   Remain = DeqCalc()
end
```

Simple loop

```
while DeqCalc() > 0
end
```

DeqConst

Available in: Professional Edition and above

Solution of a system of ordrninary differential equations: Speifies constants

Declaration:

```
DeqConst ( Value, Formula symbols )
```

Parameter:

Value	Value of the constant
Formula symbols	Formula symbols for these constants, e.g. "C1"

Description:

In the formulas with y'=, it is possible to enter constants directly as their numerical value. Or alternatively, as a symbol. The value of the constants is specified using this function and can be conveniently extracted from a FAMOS-variable.

The function is called once for each constant used in a formula.

Solving differential equations always begins with calling DeqInit(). Before the actual calculation with DeqCalc(), it may be necessary to call DeqConst().

Examples:

undamped pendulum

```
DeqInit(2, 0, 0)
local Length = 1.5
DeqConst(Length, "Length")
DeqConst(9.81, "g")
DeqX(ramp(0,0.01,1000))
DeqY("phi'", 3 )
DeqY("phi''=-(g/Length)*sin(phi)", 0 )
```

Alternative without a constant

```
DeqInit(2, 0, 0)
DeqX(ramp(0,0.01,1000))
DeqY("phi'", 3 )
DeqY("phi''=-(9.81/1.5)*sin(phi)", 0 )
```

DeqError

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Queries errors

Declaration:

```
DeqError ( ) -> Error
```

Parameter:

Error	
Error	Error (0: OK; 1: precision level not met; 2: infinte)

Description:

Solving differential equations always begins with calling DeqInit(). After the actual calculatoin with DeqCalc(), DeqError() is used to query any errors.

Examples:

Values approach infinity; Error=2

```
DeqInit(1, 0, 0)
DeqX(ramp(0,1,1000))
DeqY("y'=y", 1)
while DeqCalc() > 0
end
Error = DeqError()
```

Checking for value range violation and loss of precision

```
DeqInit(1, 0, 0)
DeqX(ramp(0,1,100))
DeqY("y'=-0.1*y", 1)
while DeqCalc() > 0
end
Error = DeqError()
Verify(Error=0)
```

DeqFinish

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Conclusion of the analysis; clearing of memory space

Declaration:

```
DeqFinish ( )
```

Parameter:

Description:

This function clears out any memory space which had been required in conjunction with the calculation.

It is recommended to call this function, but not mandatory.

At the beginning of any new calculation using DeqInit(), all memory space used in a previous analysis is cleared.

After calling this function, no further Deq*() functions can be called. It is only possible to begin an entirely new analysis by using DeqInit().

Solution of differential equations always begins with calling DeqInit(). Conclusion of the analysis is performed with DeqFinish().

Examples:

Decay function, clearing of memory space at the conclusion

```
DeqInit(1, 0, 0)
DeqY("y'=-0.1*y", 1, 1e-7, 1e-8)
DeqX(ramp(0,0.3, 300)))
while DeqCalc() > 0
end
Y = DeqResult("y")
DeqFinish()
```

DeqInit

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Initialization

Declaration:

```
DeqInit ( [Y-count] [, U-count] [, Procedure] )
```

Parameter:

Y-count	number of equations in the system (optional , Default value: 1)
U-count	number of input data sets (optional , Default value: 0)
Procedure	According to what procedure is the calculation conducted? (optional , Default value: 0)
	0 : RK 45: explicit Runge-Kutta 5th order, Dormand-Prince; embedded method
	1 : Mix of implicit and explicit Runge-Kutta. Section-by-section decision for the probably faster procedure.
	2 : RK 34, implicit Runge-Kutta 4th-order, GRK4T mtehod, a Rosenbrock-Wanner (ROW) method with Kaps-Rentrop coefficients
	3 : RK 4, classical Runge-Kutta 4th order; k1/6+k2/3+k3/3+k4/6

Description:

Solving the system of differential equations is performed over multiple function calls. All start with DeqInit().

Form of the differential equations

Each of the ordinary equation system's equations takes the form y'=f(x,y,u)

y is is the variable to be integrated. Each of the system's individual equations must be put into the form "y'=...".

u is the input data. In the equation, these are mostly time functions which are here specified by an existing time domain data set.

x is the time, the independent coordinate. All y values are functions of time, thus y=y(x)

As an alternative, M*y'=f(x,y,u) with a mass matrix is also possible. Here, y designates the vector containing the variables to be integrated; f is itself a vector.

Procedure

The procedure works with automatic stepwidth control, which however extends only up to a resolution increase factor of approx. 10000. If this is not sufficient, a loss of calculation precision is noted.

A smaller stepwidth requires more calculation resources.

Procedure=0 is usually the best choice.

Procedure=2, implied Runge-Kutta is generally only useful with stiff systems. This procedure does not support all options.

Procedure=3 (RK 4) uses Richardson extrapolation for stepwidth control (comparison of result at half stepwidth).

A call having a group as the parameter is not possible.

Solving differential equations always begins by calling DeqInit(). The example below illustrates the principles of the process of calling the functions.

If critical errors occur in the procedure, such as insufficient memory space, then the whole procedure will usually need to restart with DeqInit().

Examples:

Oscillating body, single degree of freedom (SDOF). The plot of force over time is given. Find the displacement, velocity and acceleration.

The equation is given as m*y'' + c*y' + k*y = F; where m is mass, y displacement, F force; c=2*damp*omega; k=omega^2;omega=f0*PI2

Transforming gives: v' = u - 2*damp*omega*v - omega^2*y ; and y'=v; with u = F/m

```
Force_m=1/(0.1+ramp(0,1e-3,1000))
DeqInit(2, 1, 0)
DeqInput(Force_m,"u")
DeqConst(20, "f0")
DeqConst(0.1, "damp")
DeqY("y'=v", 0, 1e-5, 1e-7, "m" )
DeqY("v' = u-2*damp*(f0*PI2)*v-((f0*PI2)^2)*y", 0.02, 1e-5, 1e-8, "m/s", 1 )
while DeqCalc() > 0
end
Displacement = DeqResult("y")
Speed = DeqResult("v")
Acceleration = DeqResult("v'")
```

Principles of the process

Solving differential equations always begins with a call of DeqInit(). Next, the equations' formulas, input data etc. are specfied. At least one call of DeqY() must be included. Additionally, at least one call of DeqInput() or DeqX() must occur. Next, calculation is performed using DeqCalc(). Next, the results are returned. Finally, error query with DeqError() and clearing of memory space with DeqFinish() are optional.

```
DeqInit()
DeqInput() or DeqX()
DeqY()
while DeqCalc() > 0
end
DeqResult()
DeqError()
DeqFinish()
```

DeqInput

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Specifies an input function

Declaration:

```
DeqInput ( Input signal [, Formula symbols] )
```

Parameter:

Input signal	Data set with the input signal U
Formula symbols	Formula symbol for this input signal, e.g. u1. If not specified, then u1, u2 etc. are assumed. (optional , Default value: "")

Description:

The input signal is considered to be interpolated in a stairstep shape.

In the function DeqInit(), one specifies how many input signals are used. For each input signal, the function DeqInput() is called once.

All input signals (as well as the data set for X) must have the same time basis, meaning the same start time, same sampling interval, and same length.

If DeqX() is not called, in other words there is no data set for X, then we have: The start-time, sampling interval and length determine the sampling points of the result data.

The sampling interval must be specified so that it is appropriate to the problem to be calculated. If it is too high, the algorithm will need too much time for stepwidth refinement.

The data set specified is equidistant. It may not contain any events or segments.

A call having a group as the parameter is not possible.

Solving differential equations always beings with a call of DeqInit(). Before the actual calculation using DeqCalc(), at least one call of DeqInput() or DeqX() must occur.

Examples:

SDOF. Specification of the force F. A FAMOS-variable already exists which specifies the plot of the force. In the formula, the force is to be designated F.

```
Force=ramp(0,1e-3,1000)
DeqInit(2, 1, 0)
DeqInput(Force, "F")
DeqY("y'", 0.0)
DeqY("y''=F/8-10*y'-20000*y", 1)
```

2 input signals

```
hx=ramp(0,1e-3,1000)
gx=ramp(0,1e-3,1000)*2
DeqInit(1, 2, 0)
DeqInput(hx, "h")
DeqInput(gx, "g")
DeqY("y'=y-0.1*(h-g)", 1.0)
```

DeqMassMatrix

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Specifies an element of the mass matrix

Declaration:

```
DeqMassMatrix ( Element, Row, Column )
```

Parameter:

Element	Numerical value (single value) or freely definable formula
Row	Row, from 1 to Y-count from DeqInit()
Column	Column, from 1 to Y-count from DeqInit()

Description:

When using a square mass matrix M, the system of ordninary differential equations takes the form:

M*y'=f(x,y,u)

M often signifies the mass matrix.

As soon as the function has been called once, a mass matrix is pre-initialized: With 1 on the diagonals, else 0. This function is called only for the few elements of the matrix which deviate from that.

The element can be specified as a known numerical value, e.g. 7.7.

Alternatively, the element can be specified as a formula. The formula can lead to a constant value. However, it may also contain the other quantities appearing in f(x,y,u). The term defining the element is specified in the formula.

The function is called once for each element of the matrix. But this is only necessary if the element differs from the corresponding element of the identity matrix.

Solving differential equations always begins with calling DeqInit(). Before the actual calculation with DeqCalc(), some calls of DeqMassMatrix() may be necessary.

Examples:

Crane trolley damped with swinging load

mass matrix:

[	1	0	0	0	]
[	0	1	0	0	]
[	0	0	mLd+mcr	Lng*mLd*cos(phi)	]
[	0	0	Lng*mLd*cos(phi)	JLd+mLd*Lng^2	]

```
DeqInit(4, 0, 0)
DeqConst(9.81, "g"); m /s^2
DeqConst(150, "mcr"); mass crab, kg
DeqConst(900, "mLd"); mass load, kg
DeqConst(2, "Lng"); distance between crab and load, m
DeqConst(120, "JLd"); kgm^2
DeqConst(200, "c") ; damper
DeqMassMatrix("mLd+mcr", 3, 3)
DeqMassMatrix("JLd+mLd*Lng^2", 4, 4)
DeqMassMatrix("Lng*mLd*cos(phi)", 3, 4)
DeqMassMatrix("Lng*mLd*cos(phi)", 4, 3)
DeqX(ramp(0,2e-3,6000), "t")
DeqY("x'=v", 0, 1e-5, 1e-7,"m", 0)
DeqY("phi'=omega", 0.5, 1e-5, 1e-7,"", 0)
DeqY("v'=mLd*Lng*omega^2*sin(phi)-v*c", 0, 1e-5, 1e-7,"m/s", 0)
DeqY("omega'=-mLd*g*Lng*sin(phi)", 0, 1e-5, 1e-7,"1/s", 0)
while DeqCalc() > 0
end
X = DeqResult("x")
phi= DeqResult("phi")
v = DeqResult("v")
omega = DeqResult("omega")
```

DeqResult

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Gets a result

Declaration:

```
DeqResult ( Name_Y ) -> Result
```

Parameter:

Name_Y	Name of the requested variable, for example "y"
Result	
Result	The calculated plot of the variable requested

Description:

All quantities which were introduced with DeqY() can be retrieved. For instance if "y'=..." was defined, then the result y can be retrieved.

Additional derivatives can be retrieved if their calculation had previously been ordered by means of DeqY() and the parameter "Derivatives": For instance, if "y'=..." was defined and "Derivatives"=1, then y' can be retrieved. If "Derivatives"=2, then y'' can also be retrieved.

Solving differential equations alwas begins with calling DeqInit(). After the actual calculation with DeqCalc(), the results are retrieved using DeqResult().

Examples:

Decay function

```
DeqInit(1, 0, 0)
DeqY("y'=-0.1*y", 1, 1e-7, 1e-8, "", 0)
DeqX(ramp(0,0.3, 300))
while DeqCalc() > 0
end
Y = DeqResult("y")
```

Decay function; derivatives desired

```
DeqInit(1, 0, 0)
DeqY("y'=-0.1*y", 1, 1e-7, 1e-8, "", 2)
DeqX(ramp(0,0.3, 300))
while DeqCalc() > 0
end
Y = DeqResult("y")
{Y'} = DeqResult("y'")
{Y''} = DeqResult("y''")
```

undamped pendulum

```
DeqInit(2, 0, 0)
DeqX(ramp(0,0.01,1000))
DeqY("phi'", 3 )
DeqY("phi''=-(9.81/1.5)*sin(phi)", 0 )
while DeqCalc() > 0
end
phi = DeqResult("phi")
{phi'} = DeqResult("phi'")
```

DeqX

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Specifiying the time basis

Declaration:

```
DeqX ( TimeBasis_Signal [, Formula symbol x] )
```

Parameter:

TimeBasis_Signal	Signal serving as the time basis
Formula symbol x	Formula symbol for x, the independent coordinate with respect to which the derivative is taken. Can also be redefined as "t", for example. (optional , Default value: "")

Description:

Here, a data set having the desired time base for the result is supplied.

The content of the signal specified is ignored.

The start-time, sampling interval, x-unit and length determine the sampling points of the result data.

All input signals (as well as the data set for X) must have the same time basis, meaning the same start time, same sampling interval, and same length.

If there are no input data U present, so that DeqInput() is not called, it is absolutely necessary to call DeqX().

The sampling interval must be specified so that it is appropriate to the problem to be calculated. If it is too high, the algorithm will need too much time for stepwidth refinement.

The data set specified is equidistant. It may not contain any events or segments.

A call having a group as the parameter is not possible.

If DeqX() is not called, then "x" is assumed as the formula symbol.

Even if DeqInput() is called, it is permitted to call DeqX(). This can be used to redefine the formula symbol. It is convenient to use one of the data sets passed to DeqInput() as the data set.

Solving differential equations always beings with a call of DeqInit(). Before the actual calculation using DeqCalc(), at least one call of DeqInput() or DeqX() must occur.

Examples:

Integration of the parabola whose derivative is given: y'= 3*t^2

```
tt = ramp(0,1e-3, 10000)
DeqInit(1, 0, 0)
DeqX(tt, "t")
DeqY("y'=0.03*t^2", 0)
```

DeqY

Available in: Professional Edition and above

Solution of a system of ordinary differential equations: Specifying a variable to be integrated

Declaration:

```
DeqY ( Formula [, initial value] [, Tolerance, relative] [, Tolerance, absolute] [, y-unit] [, Derivatives] [,
Prior record] )
```

Parameter:

Formula	Freely defineable formula. E.g. " y'=-0.1*y ". This is used to define one equation in the equation system.
initial value	initial value; for y, if the formula starts with y'= (optional , Default value: 0)
Tolerance, relative	Tolerance, relative; for y, if the formula starts with y'= (optional , Default value: 0)
Tolerance, absolute	Tolerance, absolute; for y, if the formula starts with y'= (optional , Default value: 0)
y-unit	y-unit for the calculated y (optional , Default value: "")
Derivatives	Besides y, should derivative such as y' and y'' also be offered as the calculation result? (optional , Default value: 0)
	0 : no
	1 : 1st derivative
	2 : 1st and 2nd derivative
Prior record	Prior record. Value before the process start. Only with delay differential equations (DDE), if this Y is also used in a fomula by means of delay(). (optional , Default value: 0)

Description:

Tolerances

The tolerances are applied at each individual step. But not in terms of the calculated value compared to the true value, which is not known. Rather, as the comparison between two calculated values.

In conjunction with stepwidth control and resolution increase of the stepwidth, tighter tolerances are used. However, the resulting deviation for the entire step can become larger.

If tolerances are too high, there is a special hazard of instability emerging. This is because grossly incorrect values are classified as accepted values.

If the absolute and relative tolerance are both = 0, the assumed relative tolerance = 1e-6 and the assumed absolute tolerance = 1e-20.

Caution: if either of the values absolute/relative tolerance becomes = 0: If the absolute tolerance = 0, then for a tiny magnitude the steps have extremely fine resolution. If the relative tolerance = 0, then for a gigantic magnitude the steps have extremely fine resolution. For this reason, both values should be specified to be as appropriate as possible.

Variables to be integrated

In the formula, the derivative of a quantity on the left side of the equals sign is generally specified. On the right side, the term according to which this derivative is calculated.

In cases with a differential calculation in which a higher derivative is defined, for instance "y1''=...", then a dummy variable is generally introduced: y1'=y2; This then leads to the formula "y2'=..." and an additional formula "y1'=y2". DeqY() is then called twice.

Alternatively, it is possible to do without the dummy variable: "y1''=..." is specified as a formula. "y1'" without any equals sign is specified as the second formula in a subsequent call of DeqY(). This is because for y1', it is also necessary to specify the initial values, tolerances etc.

Solving differential equtions always begins with calling DeqInit(). Before the actual calculation using DeqCalc(), there must be at least one call of DeqY().

Continuity

The formula should lead to a continuous function. The Runge-Kutta algorithm assumes this. In case of signal jumps between sampling points, the Runge-Kutta generally responds with (highly) increased resolution of the stepwidth.

An implicit Runge-Kutta requires the use of the Jacobi-matrix. In this case, all derivatives of the formula must also be continuous.

If any input variables are specified using DeqInput(), these are interpreted as steps. Since, however, the resulting discontinuities only occur at the sampling points, they do not cause problems for the Runge-Kutta procedure.

Formula content

Along with the arithmentical operators, the formula can also contain the functions sin, cos, tan, exp, asin, acos, atan, ln, log, sqrt.

All trigonometric functions work with radians. If a phase expressed in degrees is desired, the pre-defined constants PI and PI2 can be used.

Special functions

abs	With the absolute value abs(x), be aware of the discontinuous derivative at x=0.
sign	Sign function. sign(x) returns 1, 0, -1 for x>0, x=0, x<0. This function is discontinuous at x=0.
condi	Decision function. condi(y,a,b) returns a, if y does not equal 0; else b. Example: condi(x>0.1, x, 0). Continuity depends on the parameters.
delay	Delay function. delay(y,delaytime); y is one of the variables to be integrated. This variable is delayed by the fixed, specified time amount "delaytime". When this function is used, it leads to a delay differential equation (DDE).

Delay differential equations (DDE)

The delayed variables, in other words y(x-x0) instead of y(x), are denoted within a formula by means of the function delay().

The delayed plots are interpreted cubically between the sampling points measured.

The undelayed function starts with the value provided as the parameter "Initial Value". Since the delayed function also requires values from before the actual start, the value of the parameter "Prior Record" is applied. This defines the value before the start time.

If the delay time is very long, the procedure requires a correspondingly large data volume.

If the delay time is very short, it may be necessary to work with a shortened stepwidth.

Examples:

Decay function with initial value y=3.0

```
DeqInit(1, 0, 0)
DeqY("y'=-0.1*y", 3.0)
```

SDOF. The equation "y''=u-12*y'-400000*y" is given. y is the displacement. y'' is the acceleration, y'=v is introduced as the velocity.

```
DeqInit(2, 1, 0)
DeqY("y'=v", 0.0)
DeqY("v'=u-12*v-400000*y", 1)
```

SDOF. The quation "y''=u-12*y'-400000*y" is given. It is possible to skip the extra symbol y'=v. However, for y' and v'(=y'') an initial value must be specified. The defining equation is present only once, namely with the highest derivative.

```
DeqInit(2, 1, 0)
DeqY("y'", 0.0)
DeqY("y''=u-12*y'-400000*y", 1)
```

Decay function; derivatives desired

```
DeqInit(1, 0, 0)
DeqY("y'=-0.1*y", 1, 1e-7, 1e-8, "", 2)
DeqX(ramp(0,0.3, 300))
while DeqCalc() > 0
end
Y = DeqResult("y")
{Y'} = DeqResult("y'")
{Y''} = DeqResult("y''")
```

Retarded differential equation, Mackey-Glass. P upon start and previously at 0.1

```
P_ini=0.1
DeqInit(1, 0, 0)
N=10
Theta= 1
DeqConst(Theta^N, "theta")
DeqConst(22, "tau")
DeqConst(0.2, "beta")
DeqConst(0.1, "gamma")
DeqConst(N, "N")
DeqX(ramp(0,0.2,4000))
DeqY("P'=(beta*theta*delay(P,tau))/(theta+delay(P,tau)^N)-gamma*P", P_ini, 1e-6, 1e-6, "", 0, P_ini )
while DeqCalc() > 0
end
P = DeqResult("P")
```

DFilt

Digital filter

Declaration:

```
DFilt ( Data, Coefficients ) -> Filtrate
```

Parameter:

Data	Data set to be filtered. Permitted data types: [ND]
Coefficients	Data set containing the denominator and numerator coefficients
Filtrate	
Filtrate	Filtering results

Description:

This function filters a channel using a digital filter. The filter is specified using a coefficient data set; only causal filters can be calculated.

The coefficients can be expressed in either of two ways.

1. Rational Polynomial

The coefficients must meet the following pattern:

$$\frac{Y}{U} = \frac{b_0 + b_1 z^{-1} + \ ... \ + b_n z^{-n}}{a_0 + a_1 z^{-1} + \ ... \ + a_n z^{-n}} \quad ; \quad a_0 \neq 0$$

or expressed in the time domain:

$$y[t] = \frac{b_0}{a_0}u[t] + \frac{b_1}{a_0}u[t-1] + \frac{b_2}{a_0}u[t-2] + ... - \frac{a_1}{a_0}y[t-1] - \frac{a_2}{a_0}y[t-2]..$$

The coefficient data set must first contain all an followed by all bn. The coefficient data set must always contain the same number of numerator and denominator coefficients. Any unnecessary coefficients are specified as zero. The numerator coefficient must be specified first and the first value of the numerator coefficient. The first coefficient i the denominator a0 must not be equal to zero.

2. Biquad Form

The filter was desiged as a "series connection" of 2nd order filters.

Each of these filters is defined by a biquad-term of the form

$$\frac{Y}{U} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

or in the time domain as follows:

```
y[t] = b0u[t] + b1u[t-1] + b2u[t-2]+ a1y[t-1] + a2y[t-2]
```

The coefficient data set must then be specified in the following manner:

```
0 | a2 a1 b2 b1 b0 | a2 a1 b2 b1 b0 |...| a2 a1 b2 b1 b0 |
0 |_1. biquad______|_2. biquad______|...|_n. biquad______|
```

The preceding zero is the fixed identifier used to automatically identify the format.

This format of coefficient transfer is used by the imc Filter Design program.

Examples:

```
file = GetSystemInfo("Famos.Path.SampleData", "") + "\Damped_Harmonic_Oszillator.dat"
FileLoad(file, "", 0)
ParamT = [1, -0.9, 2, 0.1]
Filtrat = DFilt(Damped_Harmonic_Oszillator, ParamT)
```

The example file 'Damped_Harmonic_Oszillator.dat' is loaded. A variable 'ParamT' is filled with filter coefficients of a first-order low-pass filter. The data set is filtered with the low-pass filter defined in this way and placed on the 'filtrate' variable.

See also:

Smo, Smo3, FilterAnalog, FiltHP, FiltLP, FiltBP, FiltBS

DFTSpectrum

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

The DFT (discrete Fourier transformation) is applied to the time-based signal. For this purpose, the time signal's rms-spectrum is determined. The time signal's length need not be a power of two.

Declaration:

```
DFTSpectrum ( Time-based signal, WindowType ) -> Result
```

Parameter:

Time-based signal	The time plot of the signal from which the spectrum is to be computed
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Result	
Result	The spectrum determined is a complex data set with magnitude and phase. The magnitude of the individual frequency lines is stated as an RMS-value.

Description:

Examples:

The DFT of a time-based signal t is to be determined using rectangular windowing.

```
Spectrum = DFTSpectrum ( t, 0 )
```

See also:

FFT, ZoomSpectrumChirpZ

Dialog

Calls a user-defined dialog or a Panel in Dialog-mode.

Declaration:

```
Dialog ( TxFileName, TxParameter, SvOption ) -> SvReturn
```

Parameter:

TxFileName	Filename of the dialog-definitions file (*dlg) or of the Panel-file (*.panel).
TxParameter	The parameter is passed on to the event sequence "Init".
SvOption	Option-parameter. Always zero.
SvReturn	
SvReturn	Return value of the dialog/Panel. Corresponds to the value of the parameter passed to the function DlgCloseDialog() or PnClose().

Description:

The function starts a user-defined dialog or a Panel in Dialog-mode.

The function only returns once the dialog/panel window has been closed again by the user. In the meantime, the FAMOS-main window is disabled.

The function's return value equals the parameter passed to the function DlgCloseDialog() or PnClose().

For Panels started by means of this command, the option "Panel settings"/"Use as dialog-window" should be activated.

The 2nd parameter is passed on unchaged to the event-sequence "Initialization".

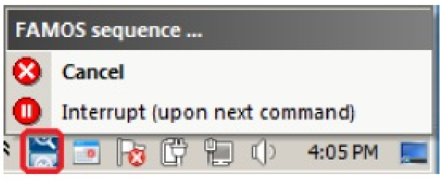
Unless a complete path is specified along with the filename, the system looks for the file in the following folders in succession:

- Project folder: If a project is open, the system looks for it in the current project folder.
- Current working folder: This is the folder from which the calling sequence was opened.
- Default folder for sequences, dialogs and Panels: Upon launching imc FAMOS, this is initially the folder specified under "Options"/ "Folders". It can be reset using either the command MDIR or the function SetOption().

Unless a file extension is specified, the filename is initially extended with '.dlg'. If no such file is present, the extension '.panel' is tried next.

When a sequence is active, the Windows taskbar's notification area contains an additional icon. Right-clicking the mouse calls a context menu which offers a command for interrupting the current run.

To interrupt, you can also use the keyboard combination "CTRL" + "Break".



On the creation and use of dialogs and Panels, please see the chapters 'User-defined dialogs' and 'Panels' in the imc FAMOS user's manual for more information.

Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

In a dialog called 'EditProperties.dlg', a variety of characteristic data set values is displayed, including the trigger time in a 'Calendar'- and a 'Time'-element. The data set's name is passed to the function Dialog(). When the dialog is closed using the 'OK' button, any changes made by the user to the trigger time are applied in the data set and the dialog is closed.

Sample call:

```
OK = Dialog("EditProperties", "Chan1", 0)
```

Event-sequence 'Dialog Initialization'

```
TxVarName = PA1
time = Time?(<TxVarName>)
DlgSetValue("time", time)
DlgSetValue("date", time)
```

Event-sequence 'Pressed' for the 'OK'-button

```
date = DlgGetValue("time")
```

```
time = DlgGetValue("date")
SetTime(<TxVarName>, TimeAdd(date, time))
DlgCloseDialog(1)
```

Event-sequence 'Pressed' for the 'Cancel'-button

```
DlgCloseDialog(0)
```

When the dialog is closed with the [OK]-button, the variable OK receives the value 1. On the other hand, if the [Cancel]-button is pressed, a 0 is returned.

A panel 'InputValue.panel' generally consists of an input box "input" for entering a positive numerical value, as well as 2 buttons: 'OK' and 'Cancel'. The Dialog()-command returns the value entered, or -1, if the intent is to cancel.

Event-sequence 'Button pressed' for the 'OK'-button:

```
value = PnGetValue("input")
PnClose(value)
```

Event-sequence 'Button pressed' for the 'Cancel'-button

```
PnClose(-1)
```

Event-sequence 'Close' (user utilizes the system menu to exit):

```
; Same behavior as 'Cancel'-button
PnClose(-1)
```

Call of the dialog:

```
value = Dialog("InputValue.panel", "", 0)
IF value < 0
   EXITSEQUENCE 0
END
;Further in the sequence...
...
```

See also:

SEQUENCE, BoxMessage, BoxValue?, BoxText?

Diff

Differentiation, derivatives

Declaration:

```
Diff ( Data ) -> Result
```

Parameter:

Data	Data set to be differentiated. Permitted data types: [ND],[XY]
Result	
Result	Result of differentiation

Description:

Generates the derivative of the data set passed. The derivative is formed according to a simple, effective algorithm:

The differences between adjacent values (delta-y) are computed and divided by the x-distance between them. For normal, real data sets this distance equals the sampling time (or delta-x).

Differentiation is thus the inverse of integration.

The derivative of a data set yields the slope at every point. Note that differentiation "roughens" the data set.

- Because a difference is formed when calculating two adjacent values, the length of the data set becomes one point shorter. Differentiation of an empty data set or a single value results in an empty data set.
- Special feature with XY-data: The last calculated value is appended again. This enables a meaningful display of the result in the curve window (step display) as well as ensures reversibility with the Int() function. The length of the input data and the result is therefore the same here.
- The unit of the differentiated data set is the quotient of the y-and x-unit of the specified data set.

Examples:

```
NDdiff = Diff(NDdata)
```

Simple calculation of the slope

```
NDdiff = Diff(Smo3(NDdata))
```

Often the data set should be smoothed before differentiation to reduce the influence of noise on the result. Noise strongly distorts the result of differentiation.

```
NDdiff = Red(Diff(IPol(NDdata, 3)), 3)
```

When data sets are not very noisy but have relatively large jumps in the function values, it is appropriate to apply spline-interpolation before performing differentiation. After differentiation, the data volume can be reduced to an appropriate information volume using the sampling function Red().

```
NDdata = Diff(Int(NDdata))
```

This formula does not change the data set.

```
NDyoff = Int(Diff(NDdata))
```

The calculated data set may differ by an offset in the y direction.

See also:

Int, MInt

DisplayY?

Queries the fixed scaling for curve window display

Declaration:

```
DisplayY? ( Data, SvChoice ) -> SvScaleValue
```

Parameter:

Data	Data set whose Y-scaling is to be determined
SvChoice	Selection of the parameter to be queried
	0 : Bottom scale value
	1 : Top scale value
SvScaleValue	
SvScaleValue	Bottom/top scale value

Description:

A fixed Y-axis scaline can also be assiged to a data set as an additional property. It is then used for the display of the data set in a curve window if the curve window's setting "Automatic Y-axis scaling" is active.

If a fixed scaling has been assignedd to the y-axis, this function returns the range boundaries.

If no fixed scaling has been assigned, then 0 is returned for both boundariy values. The data set will then be scaled according to its own domain (value range).

Examples:

```
yMin = DisplayY?(data, 0)
IF yMin > 0
   SetDisplayY(data, 0, DisplayY?(data, 1))
ELSE
   yMax = DisplayY?(data, 1)
   IF yMax < 0
      SetDisplayY(data, DisplayY?(data, 0),0)
   END
END
```

If no fixed scaling has been assigned to the data set and the zero-line is not included, either the bottom or top scale value is corrected to 0.

See also:

SetDisplayY, Color?

Div

Time-correct or x-correct division.

Declaration:

```
Div ( Dividend, Divisor, SvOption ) -> Quotient
```

Parameter:

Dividend	First parameter; Dividend; allowed types: [ND],[XY].
Divisor	Second parameter, Divisor; allowed types: [ND],[XY].
SvOption	Option
	0 : The trigger time of the two summands is ignored.
	1 : Time-correct superposition with regard to trigger-time
Quotient	
Quotient	Quotient; result of division [XY]

Description:

Two data sets undergo time-correct or x-correct division, meaning that one of the y-values for each shared x-value is divided by the other's.

The result is defined only within the x-range which is shared by both data sets. Within this range a resultvalue is determined for every point at which at least one of the data sets possesses a value. If no value exists for the other data set, one is determined by linear interpolation.

The x-tracks of both parameter data sets must be monotonous, i.e. the x-coordinates must increase continuously.

The division operator '/', by contrast, divides the values of each successive point of each data set, without regard to their absolute x-positions.

Examples:

Two channels are measured; one between 11:00 and 13:00, and the other between. 12:00 and 14:00.

```
Quot12_13h = Div(voltage11_13h, voltage12_14h, 1)
```

Time-correct division of the two data sets is performed, with regard given to the trigger time. The result is defined for the time period between 12:00 and 13:00 hours.

See also:

/(Division), Add, Sub, Mult, Append

DlgApplyData

Scope: User-defined dialogs

Checks validity of changes to any input boxes and/or applies in the linked variables.

Declaration:

`DlgApplyData ( ) -> Success`

Parameter:

Success	
Success	0 for error. 1 for success.

Description:

For all dialog elements with active data linkage, the current value is checked against the criteria set under Element Properties (see 'Validation'), and upon success, the value is adopted by any linked variable.

If the check fails, the used is informed by means of a message box. A typical applicaton of this function would be, for instance, the 'Pressed'-sequence of an 'Apply'-button, or the 'Close dialog'-sequence.

Examples:

When the 'OK'-button is pressed, all data entered are checked and any linked variables updated. Upon success, the dialog is closed.

--> Event sequence 'Button pressed' for the 'OK'-button

```
IF DlgApplyData()
   DlgCloseDialog(0)
END
```

See also:

DlgCloseDialog

DlgCloseDialog

Scope: User-defined dialogs

Closes the dialog

Declaration:

```
DlgCloseDialog ( EwReturn )
```

Parameter:

EwReturn	The dialog's return value. Corresponds to the return value of the function Dialog().

Description:

The dialog is closed. This proceeds in three stages:

- The dialog window becomes invisible.
- The event 'Dialog End' is triggered.
- After the event sequence is run, the dialog is closed. The function Dialog(..), with which the dialog was originally created, returns the value which served here as the parameter as an event.

By default, the function is called in the event sequence 'Close dialog'. This event is triggered whenever the user has clicked either the button 'Close' in the window's title bar, or the corresponding system menu item, or pressed the combination of the ALT+F4 keys.

Otherwise, the function is often used in 'Button pressed' event sequences, in order to close the dialog at the push of a button, typically such buttons have the caption 'OK' or 'Cancel'.

In such cases it is recommended to supply a unique parameter value for each button. Since this value is reflected in the return value of the function Dialog(..), the one calling the function can recognize with which button the dialog was closed.

Examples:

A dialog 'Continue.dlg' consists of only the question 'Continue running the sequence?' and 2 buttons 'OK' and 'Cancel'.

--> Event sequence 'Button pressed' for the 'OK'-button:

```
DlgCloseDialog(1)
```

--> Event sequence 'Button pressed' for the 'Cancel'-button:

```
DlgCloseDialog(0)
```

--> Call the dialog:

```
Continue = Dialog("Continue.dlg", "", 0)
IF Continue = 0
   EXITSEQUENCE 0
END
;Next in the sequence...
...
```

See also:

Dialog

DlgDeleteItem

Scope: User-defined dialogs

Deletes one or all entries (list, table etc.)

Declaration:

`DlgDeleteItem ( TxElementName, Index )`

Parameter:

TxElementName	Name of the dialog element to be changed
Index	Index of the entry to be deleted. The first entry has the index 1. To delete all entries, enter a 0.

Applies to:

Listbox, Droplist, ComboBox, Tableview, Treeview

Examples:

In a list box with multi-selection, all selected entries are deleted:

```
Count = DlgGetItemCount("list1")
i = Count
WHILE i > 0
   IF DlgIsItemSelected("list1", i)
      DlgDeleteItem("list1", i)
   END
   i = i - 1
END
```

See also:

DlgGetItemCount, DlgGetItemText, DlgSetItemText, DlgInsertItem, DlgFindItem

DlgEnable

Scope: User-defined dialogs

The active dialog or the specified dialog element is disabled (made unavailable to the user) or restored.

Declaration:

```
DlgEnable ( TxElementName, OnOff )
```

Parameter:

TxElementName	Name of the dialog element to be changed. If an empty text is specified, the entire active dialog is enabled/ disabled.
OnOff	
	0 : Disable condition
	1 : Enable condition

Applies to:

All dialog elements (also menu items)

Examples:

A dialog for inquiring the date consists of a dialog element of the type 'Datepicker' and a button 'btnOk' for closing the dialog. The button 'btnOK' is only enabled once the user has entered a valid date.

--> Event sequence 'Dialog initialization'

```
;temporarily blocks the 'OK'-button
DlgEnable("btnOK", 0)
```

--> Event sequence 'Changed' of the 'calendar' element

```
;Test of the date set
_Date = DlgGetValue( PA1 )
IF _Date >= TimeSystem?()
   DlgEnable( "btnOK", 1)
ELSE
   DlgEnable( "btnOK", 0)
END
```

--> Event sequence 'Button pressed' for the 'btnOK'-button:

```
;Close dialog and return date
DlgCloseDialog(_Date)
DELETE _Date
```

See also:

DlgShow

DlgExpandTree

Scope: User-defined dialogs

Expansion or collapse of a node in a tree diagram

Declaration:

```
DlgExpandTree ( TxElementName, Index, Action )
```

Parameter:

TxElementName	Name of the dialog element to be changed
Index	Position of the desired tree entry. The first entry has the index 1. Enter 0 to have all entried expanded, or collapsed, respectively.
Action	Sets what action is to be performed
	0 : Collapse of the node
	1 : Expansion of the node
	2 : Expand entire sub-tree

Applies to:

Treeview

Examples:

Expands the currently selected node in a tree diagram

```
i = DlgGetSelectedItem("Tree")
IF i > 0
   DlgExpandTree("Tree", i, 1)
END
```

See also:

DlgInsertTreeItem, DlgGetItemLevel

DlgFileName

A dialog for selection/entry of a filename is called.

Declaration:

```
DlgFileName ( TxFolder, TxExtension, TxTitle, SvTask ) -> TxFileName
```

Parameter:

TxFolder	Folder in which the dialog should start. When an empty string is supplied, the default folder for loading files is used.
TxExtension	Default extension for filenames, e.g. "dat". Once a valid extension has been provided, only files having this extension are displayed upon initialization of the dialog. If the filename entered has no extension, this extension is appended to it. An empty text is allowed. As of Version 2022, it is also possible to specify multiple extensions, separated by semicolons, so for example "dat;raw".
TxTitle	The dialog's title. For an empty text, a default title is used.
SvTask	Task
	0 : "Open file"-dialog
	1 : "Save File"-dialog
TxFileName	
TxFileName	Entered filename (complete path); in case of failure (canceled by user), an empty text.

Description:

Using the standard Windows dialog for file selection, an existing file can be selected or a new filename can be entered.

- The position at which the dialog box appears can be specified with the help of the function SetBoxPos().
- You can also manually change the dialog's position (e.g. by moving it with the mouse). The position is remembered for the duration of the current session.
- Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

```
fileName = DlgFileName("", "dat", "", 0)
FileLoad(fileName, "", 0)
IF VarExist?("y")
   y2 = sin(y)
   saveFileName = DlgFileName("c:\imc\dat", "", "Please enter file name", 0)
   FileSave(saveFileName, "", 0, y2)
END
```

See also:

FsDlgSelectDirectory, FsDlgSelectFiles, BoxValue?, BoxOutput, BoxMessage, SetBoxPos

DlgFindItem

Scope: User-defined dialogs

Finds a (list-/ table-, etc.) entry with the specified contents.

Declaration:

```
DlgFindItem ( TxElementName, TxContents ) -> Index
```

Parameter:

TxElementName	Name of the dialog element in question
TxContents	Text for the entry for which to search
Index	
Index	Index of the entry to be found; (>=0) if found. 0 otherwise.

Description:

The function is not case-sensitive

Applies to:

Listbox, Droplist, ComboBox

Examples:

Looks for an entry in a list box (with single-selection). If the entry exists, it is selected and scrolled into view, if necessary.

```
i = DlgFindItem("list", "100.0")
IF i > 0
   DlgSelectItem("list", i)
END
```

See also:

DlgGetItemCount, DlgGetItemText, DlgSetItemText, DlgInsertItem, DlgDeleteItem

DlgGetBarMax

Scope: User-defined dialogs

Determines the value range's upper limit

Declaration:

```
DlgGetBarMax ( TxElementName ) -> Max
```

Parameter:

TxElementName	Name of the dialog element (Slider control) to be queried
Max	
Max	Value range maximum

Applies to:

Slider

Examples:

A Slider control is to be set to a new value. Beforehand, a test is conducted of whether the desired value lies within the permitted range and if appropriate, an error message is posted.

```
IF newValue >= DlgGetBarMin("Slider1") AND newValue <= DlgGetBarMax("Slider1")
   DlgSetValue( "Slider1", newValue)
ELSE
   BoxMessage("Error", "Limit exceeded", "1!")
END
```

See also:

DlgSetBarRange, DlgGetBarMin, DlgGetValue, DlgSetValue

DlgGetBarMin

Scope: User-defined dialogs

Determines the value range's lower limit

Declaration:

```
DlgGetBarMin ( TxElementName ) -> Min
```

Parameter:

TxElementName	Name of the dialog element (Slider control) to be queried
Min	
Min	Value range minimum

Applies to:

Slider

Examples:

A Slider control is to be set to a new value. Beforehand, a test is conducted of whether the desired value lies within the permitted range and if appropriate, an error message is posted.

```
IF newValue >= DlgGetBarMin("Slider1") AND newValue <= DlgGetBarMax("Slider1")
   DlgSetValue( "Slider1", newValue)
ELSE
   BoxMessage("Error", "Limit exceeded", "1!")
END
```

See also:

DlgSetBarRange, DlgGetBarMax, DlgGetValue, DlgSetValue

DlgGetCellText

Scope: User-defined dialogs

The content of the specified table cell is queried

Declaration:

```
DlgGetCellText ( TxElementName, Row, Colummn ) -> TxValue
```

Parameter:

TxElementName	Name of the dialog element (table) to be queried
Row	Column index. The first row has the index 1. The column header (if visible) has the index 0.
Colummn	Column index. The first column has the index 1. The row header (if visible) has the index 0.
TxValue	
TxValue	Current table cell contents

Applies to:

Tableview

Examples:

A table presents a variety of characteristic values for a data set, including the Y-unit. If the user changes the corresponding table cell, the new unit is adopted by the data set.

--> Event sequence 'Dialog initialization'

```
...
DlgSetCellText("Tab1", 1, 2, Unit?(MyData,1))
...
```

--> Event sequence 'Changed' of the table 'Tab1'

```
Row = PA2
Column = PA3
IF Row = 1 AND Column = 2
   TxUnit = DlgGetCellText(PA1, Row, Column)
   SetUnit(MyData, txUnit, 1)
END
```

See also:

DlgSetCellText, DlgGetCellValue, DlgSetCellValue

DlgGetCellValue

Scope: User-defined dialogs

Queries the value of the specified table cell

Declaration:

`DlgGetCellValue ( TxElementName, Row, Colummn ) -> Value`

Parameter:

TxElementName	Name of the dialog element to be queried
Row	Row index. The first row has the index 1. The column header (if visible) has the index 0.
Colummn	Column index. The first column has the index 1. The row header (if visible) has the index 0.
Value	
Value	Current value of table cell

Description:

If the current contents of the table cell queried cannot be converted to a number, a 0 is returned.

Applies to:

Tableview

Examples:

After the user changes the contents of a table cell, a check is made of whether the value entered is a number > 0. If not, the cell is marked red.

--> Event sequence 'Changed' of the table

```
Row = PA2
Column = PA3
newValue = DlgGetCellValue(PA1, Row, Column)
IF newValue <= 0
   DlgSetCellTextColor(PA1, Row, Column, RGB(255,0,0))
ELSE
   DlgSetCellTextColor(PA1, Row, Column, -2)
END
```

See also:

DlgSetCellValue, DlgGetCellText, DlgSetCellText

DlgGetItemCount

Scope: User-defined dialogs

Determines the number of entries in the specified dialog element (list, table etc.)

Declaration:

```
DlgGetItemCount ( TxElementName ) -> Count
```

Parameter:

TxElementName	Name of the dialog element to be queried
Count	
Count	Number of entries

Applies to:

Listbox, Droplist, ComboBox, Treeview, Tableview, Radiogroup

Examples:

In a list box with multi-selection, all selected entries are deleted:

```
Count = DlgGetItemCount("list1")
i = Count
WHILE i > 0
   IF DlgIsItemSelected("list1", i)
      DlgDeleteItem("list1", i)
   END
   i = i - 1
END
```

See also:

DlgGetItemText, DlgSetItemText, DlgInsertItem, DlgFindItem, DlgDeleteItem

DlgGetItemLevel

Scope: User-defined dialogs

Finds the level (indicated by the indent) of a tree diagram entry

Declaration:

```
DlgGetItemLevel ( TxElementName, Index ) -> Level
```

Parameter:

TxElementName	Name of the dialog element to be changed
Index	Position of the desired tree entry. The first entry has the index 1.
Level	
Level	An entry's level. Root entries have the level 0.

Applies to:

Treeview

Examples:

Whether a button is enabled depends on the current selection in a tree diagram. The button should be enabled if a root element (Level = 0) is selected.

```
i = DlgGetSelectedItem("Tree")
enable = (i > 0) AND (DlgGetItemLevel("Tree", i) = 0)
DlgEnable("button", enable)
```

See also:

DlgInsertTreeItem, DlgExpandTree

DlgGetItemText

Scope: User-defined dialogs

Returns the text for an entry into a dialog element (list box, tree diagram etc.).

Declaration:

```
DlgGetItemText ( TxElementName, Index ) -> TxContents
```

Parameter:

TxElementName	Name of the dialog element to be queried
Index	Index of the entry to be queried. The first entry has the index 1.
TxContents	
TxContents	Text for the specified entry

Applies to:

Listbox, Droplist, , ComboBox, Treeview

Examples:

The currently selected entry is read out of a list containing names of measurement files in FAMOS-format, and the corresponding file is opened.

```
i = DlgGetSelectedItem("listFiles")
IF i > 0
   FileName$ = DlgGetItemText("listFiles", i)
   fh = FileOpenDSF( FileName$,0)
   IF fh > 0
      ;...
      FileClose(fh)
   END
END
```

See also:

DlgGetItemCount, DlgSetItemText, DlgInsertItem, DlgFindItem, DlgDeleteItem

DlgGetPath

Scope: User-defined dialogs

Returns the complete path of the currently executed dialog file.

Declaration:

```
DlgGetPath ( Option ) -> TxPath
```

Parameter:

Option	Option
	0 : Complete path
	1 : Directory only
TxPath	
TxPath	Complete path or directory of the currently executed dialog file.

Description:

Examples:

When a dialog has been started with:

```
ret = Dialog( "c:\imc\seq\MyDialog.dlg", "", 0)
```

```
TxFullPath = DlgGetPath(0)
   ; TxFullPath contains "c:\imc\seq\MyDialog.dlg"
TxDir = DlgGetPath(1)
   ; TxDir contains "c:\imc\seq"
```

See also:

DlgSetTextColor, DlgSetCellTextColor

DlgGetSelectedItem

Scope: User-defined dialogs

Determines the entry currently selected in a list/ table with single-selection.

Declaration:

```
DlgGetSelectedItem ( TxElementName ) -> Index
```

Parameter:

TxElementName	Name of the dialog element in question
Index	
Index	Index of the selected entry (>=0). 0, if no entry is selected.

Applies to:

Listbox (single selection), DropDown-List, Combobox, Table (single selection), Treeview, Radiogroup

Examples:

The currently selected entry is read out of a list containing names of measurement files in FAMOS-format, and the corresponding file is opened.

```
i = DlgGetSelectedItem("listFiles")
IF i > 0
   FileName$ = DlgGetItemText("listFiles", i)
   fh = FileOpenDSF( FileName$,0)
   IF fh > 0
      ;...
      FileClose(fh)
   END
END
```

See also:

DlgSelectItem, DlgIsItemSelected

DlgGetSelectedItemCount

Scope: User-defined dialogs

Determines the number of selected entries in a list with multi-selection.

Declaration:

```
DlgGetSelectedItemCount ( TxElementName ) -> Count
```

Parameter:

TxElementName	Name of the dialog element in question
Count	
Count	Number of selected entries. 0, if no entry is selected.

Description:

Applies to:

Listbox (multi-selection), Tableview (multi-selection)

Examples:

The selected entries in a list box with multi-selection are to be evaluated at the push of a button. The button is only to be enabled if at least one entry is selected. For this purpose, the amount of entries selected in the event 'Selected' is checked:

--> The list box's event-sequence 'Selected'

```
Count = DlgGetSelectedItemCount(PA1)
DlgEnable("Button1", Count > 0)
```

See also:

DlgIsItemSelected, DlgSetItemSelection, DlgGetSelectedItem

DlgGetText

Scope: User-defined dialogs

DQueries the content or caption of the specified element.

Declaration:

```
DlgGetText ( TxElementName ) -> TxText
```

Parameter:

TxElementName	Name of the dialog element to be queried
TxText	
TxText	Current content or caption of the dialog element

Applies to:

The function can only be used for such dialog element which possess a caption or whose current state can be expressed by a text. Interpretation of the text depends on the element's type.

Element	Meaning
Button	Button caption
Label	Content of the text box
Editbox (single line)	Content of the input box
Editbox (multi-line)	Content of the input box
Checkbox	The element's caption
Groupbox	The element's caption
Radiogroup	Caption for the frame around the radiogroup. Only applicable for "Frame" = "With caption".
Listbox (Single selection)	Text of the currently selected entry.
Droplist	Text of the currently selected entry.
Treeview	Text of the currently selected entry.
Combobox	Content of the input box
Dialog	The dialog's title bar. For TxElementName, an empty text is to be entered.
Statusbar	Content of the dialog's status bar. For TxElementName, "#Status" is available.

For elements which aren't listed in the table above, the function isn't applicable.

Examples:

A dialog displays a variety of a data set's characteristic values; among others the Y-unit in an input box 'input_unit'. When closing the dialog with the 'OK' button, any changes of the unit made by the user are adopted in the data set.

--> Event sequence 'Dialog initialization'

```
...
DlgSetText("input_unit",  Unit?(MyData,1))
...
```

--> Event sequence 'Button pressed' for the 'OK'-button

```
TxUnit = DlgGetText("input_unit")
SetUnit(MyData, TxUnit, 1)
DlgCloseDialog(0)
```

See also:

DlgGetValue, DlgGetValue, DlgSetText

DlgGetValue

Scope: User-defined dialogs

Queries the specified element's current numerical value.

Declaration:

```
DlgGetValue ( TxElementName ) -> Value
```

Parameter:

TxElementName	Name of the dialog element to be queried
Value	
Value	Current dialog element value

Applies to:

The function can only be applied to such dialog elements whose current state can be expressed by a number. How the value is interpreted depends on the element's type.

Element	Value
Listbox (Single selection)	If the value currently selected in the list can be converted to a number, the number is returned. Otherwise 0.
Droplist	..
Combobox	If the text in the input box can be converted to a number, this number is returned. Otherwise 0.
Editbox (single line)	..
Checkbox	Returns 1 if the check box is checked, otherwise 0.
Radiogroup	Returns the index of the option currently selected. The first option has the index 1.
Slider	Returns the slider's current value.
Datepicker	Returns the current value in the FAMOS time format. This value can be processed with the functions in function group #18 "Date/Time".
Timepicker	Returns the current value in the FAMOS time format. This value can be processed with the functions in function group #18 "Date/Time".

For elements which aren't listed in the table above, the function isn't applicable.

Examples:

A dialog serves the purpose of setting filter parameters. A pop-down list 'CBoxType' contains the entries 'High-pass' and 'Low-pass'. A combination box 'CBox_Order' is for specifying the filter order, and an input box 'Input_Freq' for specifying the cutoff frequency. After that, the filtering is carried out at the push of a button:

```
Type  = DlgGetSelectedItem( "CBox_Type")
Order = DlgGetValue( "CBox_Order")
freq  = DlgGetValue("Input_Freq")
IF Order > 1 AND freq > 0
   IF Type = 1  ; Lowpass
      filtrat = FiltTP( MyData, 0, 0, Order, freq)
   ELSE          ; Highpass
      filtrat = FiltHP( MyData, 0, 0, Order, freq)
   END
END
```

See also:

DlgSetValue, DlgGetText

DlgInsertItem

Scope: User-defined dialogs

Adds a new entry to a dialog element (list, table etc.).

Declaration:

```
DlgInsertItem ( TxElementName, Index, TxContents, Option )
```

Parameter:

TxElementName	Name of the dialog element to be changed
Index	Insert position. The first entry has the index 1. To append the entry to the end, enter a 0.
TxContents	Text for the new entry
Option	Option parameter
	0 : Default
	1 : The new entry is scrolled into view (if necessary).

Applies to:

Listbox, Droplist, , ComboBox, Tableview

Examples:

A list box is filled with the names of all files located in the specified folder.

```
Dir$ = FsDlgSelectDirectory("Select folder", "", 0)
FileListID = FsFileListNew( Dir$, "*.*", 0, 0, 0)
FileCount = FsFileListGetCount(FileListID)
i = 1
WHILE i <= FileCount
   File$ = FsSplitPath(FsFileListGetName(FileListID, i), 4)
   DlgInsertItem("listFiles", i, File$, 0)
   i = i + 1
END
FsFileListClose(FileListID)
```

A list box is filled with the names of all variables which are in the FAMOS variables list when the program is started. Subsequently, the first entry is selected.

```
Count = VarGetInit(0)
i = 1
WHILE i <= Count
   DlgInsertItem("ListVariables", i, VarGetName?(i), 0)
   i = i + 1
END
DlgSelectItem("ListVariables", 1)
```

A list box is filled with all values belonging to a (short) data set.

```
Count = leng?(MyData)
i = 1
WHILE i <= Count
   TxVal = TForm( MyData[i], "")
   DlgInsertItem("list1", 0, TxVal, 0)
   i = i + 1
END
```

See also:

DlgGetItemCount, DlgGetItemText, DlgSetItemText, DlgFindItem, DlgDeleteItem

DlgInsertTreeItem

Scope: User-defined dialogs

Adds a dialog element (tree diagram) to a new entry

Declaration:

```
DlgInsertTreeItem ( TxElementName, Index, TxContents, Level, Option )
```

Parameter:

TxElementName	Name of the dialog element to be changed
Index	Insert position. The first entry has the index 1. To append the entry to the end, enter a 0.
TxContents	Text for the new entry
Level	Indent of the new entry. Root entries have an indent of 0.
Option	Option parameter
	0 : Default
	1 : The new entry is scrolled into view (if necessary).

Applies to:

Treeview

Examples:

At the beginning of a tree diagram, two new root entries with child elements are added.

```
DlgInsertTreeItem("Tree1", 1, "Root1", 0, 0)
DlgInsertTreeItem("Tree1", 2, "Leaf1", 1, 0)
DlgInsertTreeItem("Tree1", 3, "Leaf2", 1, 0)
DlgInsertTreeItem("Tree1", 4, "Root2", 0, 0)
DlgInsertTreeItem("Tree1", 5, "Leaf3", 1, 0)
```

See also:

DlgInsertItem, DlgDeleteItem

DlgIsItemSelected

Scope: User-defined dialogs

Determines whether an entry in the list/ table is selected with multi-selection

Declaration:

`DlgIsItemSelected ( TxElementName, Index ) -> IsSelected`

Parameter:

TxElementName	Name of the dialog element in question
Index	Index of the entry to be verified. the first entry has the index 1.
IsSelected	
IsSelected	The entry's selection status
	0 : Not selected
	1 : Selected

Applies to:

Listbox (multi-selection), Tableview (multi-selection)

Examples:

A list box in a dialog is filled with all of the values of a (short) data set. At the push of a button, all selected values are set to 0.

--> Event sequence 'Dialog initialization'

```
;Fills the list box with the data set's values:
Count = leng?(MyData)
i = 1
WHILE i <= Count
   TxVal = TForm( MyData[i], "")
   DlgInsertItem("list1", 0, TxVal, 0)
   i = i + 1
END
```

--> Event sequence 'Button pressed'

```
;The selected samples are set to 0.
Count = DlgGetItemCount("list1")
i = 1
WHILE i <= Count
   IF DlgIsItemSelected("list1", i)
      MyData[i] = 0
      DlgSetItemText("0")
   END
   i = i + 1
END
```

See also:

DlgGetSelectedItemCount, DlgSetItemSelection, DlgGetSelectedItem

DlgSelectItem

Scope: User-defined dialogs

Selects an entry in a list/ table with single-selection

Declaration:

`DlgSelectItem ( TxElementName, Index )`

Parameter:

TxElementName	Name of the dialog element to be queried
Index	Index of the entry to be selected. The first entry has the index 1. Specify 0 to clear the selection.

Applies to:

Listbox (single selection), DropDown-List, Combobox, Table (single selection), Treeview

Examples:

Looks for an entry in a list box (with single-selection). If the entry exists, it is selected and scrolled into view, if necessary.

```
i = DlgFindItem("list", "100.0")
IF i > 0
   DlgSelectItem("list", i)
END
```

See also:

DlgGetSelectedItem, DlgSetItemSelection

DlgSetBackColor

Scope: User-defined dialogs

Sets the background color for the selected element

Declaration:

`DlgSetBackColor ( TxElementName, RGBColor )`

Parameter:

TxElementName	Name of the dialog element to be changed. If an empty text specified, the dialog background is set.
RGBColor	Colour
	>=0 : RGBvalue of the desired color
	-1 : Transparent
	-2 : Automatic

Description:

The RGB-value supplied as the 2nd parameter represents the proportional intensities of the 3 fundamental colors Red, Green and Blue. To generate such a color value, you can use the FAMOS function RGB().

The value (-1) sets the background color to transparent. This option only makes sense and is only supported for certain dialog element types.

The value (-2) sets the background automatically. For the dialog, this means that the current Windows system configuration is used. For dialog elements, this option means that the dialog's background color is adopted ('inherited').

Applies to:

Function is not applicable for: Datepicker, Timepicker, Curvewindow, Menu

Examples:

The data set's maximum value is checked against a cutoff limit and the result is displayed in a text box ("Label1") in the dialog. If the limit has been exceeded, the text box's background and text color are chaged (red/ white) and a warniong message is posted.

```
_max = Max(Daten)
IF _max > 12
   DlgSetTextColor("Label1", RGB(255,255,255))
   DlgSetBackColor("Label1", RGB(255,0,0))
   DlgSetText("Label1", "Limit exceeded!")
ELSE
   DlgSetTextColor("Label1", -2)
   DlgSetBackColor("Label1", -2)
   DlgSetText("Label1", "Dataset OK.")
END
```

See also:

DlgSetTextColor, DlgSetCellTextColor

DlgSetBarRange

Scope: User-defined dialogs

Configures a Slider control in terms of range minimum and -maximum, as well as step size.

Declaration:

`DlgSetBarRange ( TxElementName, Minimum, Maximum, StepSmall, StepLarge )`

Parameter:

TxElementName	Name od dialog element (Slider control) to be changed.
Minimum	Lower limit of value range
Maximum	Upper limit of value range
StepSmall	Sets the size of small steps (navigation keys 'Left', 'Right' ...).
StepLarge	Sets the size of large steps('PgUp' and 'PgDn' keys)

Applies to:

Slider

Examples:

A Slider control's value range is set. The range is specified as 0 to 100 (percent). The arrow-keys change the instrument value by +/-1; keys 'PgUp' and 'PgDn' by +/-10. The slider is initially centered.

```
DlgSetBarRange("Slider1", 0, 100, 1, 10)
DlgSetValue("Slider1", 50)
```

See also:

DlgGetBarMin, DlgGetBarMax, DlgGetValue, DlgSetValue

DlgSetCellBackColor

Scope: User-defined dialogs

Sets the background color for the specified table cell

Declaration:

```
DlgSetCellBackColor ( TxElementName, Row, Colummn, RGBColor )
```

Parameter:

TxElementName	Name of the table to be changed
Row	Row index. The first row has the index 1. If 0, the color is used for the whole column.
Colummn	Column index. The first column has the index 1. If 0, the color is used for the entire row.
RGBColor	RGB-value of the desired color or (-2) for automatic.

Description:

The RGB-value supplied as the 2nd parameter represents the proportional intensities of the 3 fundamental colors Red, Green and Blue. To generate such a color value, you can use the FAMOS function RGB().

The setting 'automatic' (-2) means here that the column's default background color is used

Applies to:

Tableview

Examples:

A table presents a variety of a data set's characteristic values. If the maximum exceeds a cutoff limit, the corresponding table cell appears in a distinct color.

```
DlgSetCellText("Table", 1, 1, "Length ")
DlgSetCellValue("Table", 1, 2, leng?(MyData))
DlgSetCellText("Table", 2, 1, "Sampling time ")
DlgSetCellValue("Table", 2, 2, xdel?(MyData))
DlgSetCellText("Table", 3, 1, "Maximum ")
value = max(MyData)
DlgSetCellValue("Table", 3, 2, value)
IF value > 100
   DlgSetCellTextColor( "Table", 3, 2, RGB(255,0,0))
   DlgSetCellBackColor( "Table", 3, 2, RGB(127,127,127))
ELSE
   DlgSetCellTextColor( "Table", 3, 2, -2)
   DlgSetCellBackColor( "Table", 3, 2, -2)
END
```

See also:

DlgSetCellTextColor, DlgSetTextColor, DlgSetBackColor

DlgSetCellText

Scope: User-defined dialogs

Sets new content for the specified table cell.

Declaration:

`DlgSetCellText ( TxElementName, Row, Colummn, TxContents )`

Parameter:

TxElementName	Name of the dialog element to be edited (table).
Row	Row index. The first row has the index 1. The column header (if visible) has the index 0.
Colummn	Column index. The first column has the index 1. The row header (if visible) has the index 0.
TxContents	Text to be set for the cell.

Applies to:

Tableview

Examples:

A table presents a variety of a data set's characteristic values. If the maximum exceeds a cutoff limit, the corresponding table cell appears in a distinct color.

```
DlgSetCellText("Table", 1, 1, "Length ")
DlgSetCellValue("Table", 1, 2, leng?(MyData))
DlgSetCellText("Table", 2, 1, "Sampling time ")
DlgSetCellValue("Table", 2, 2, xdel?(MyData))
DlgSetCellText("Table", 3, 1, "Maximum ")
value = max(MyData)
DlgSetCellValue("Table", 3, 2, value)
IF value > 100
   DlgSetCellTextColor( "Table", 3, 2, RGB(255,0,0))
   DlgSetCellBackColor( "Table", 3, 2, RGB(127,127,127))
ELSE
   DlgSetCellTextColor( "Table", 3, 2, -2)
   DlgSetCellBackColor( "Table", 3, 2, -2)
END
```

See also:

DlgSetCellValue, DlgGetCellText

DlgSetCellTextColor

Scope: User-defined dialogs

Sets the font color for the specified table cell.

Declaration:

`DlgSetCellTextColor ( TxElementName, Row, Colummn, RGBColor )`

Parameter:

TxElementName	Name of the table to be changed
Row	Row index. The first row has the index 1. If 0, the color is used for the whole column.
Colummn	Column index. The first column has the index 1. If 0, the color is used for the entire row.
RGBColor	RGB-value of the desired color or (-2) for automatic.

Description:

The RGB-value supplied as the 2nd parameter represents the proportional intensities of the 3 fundamental colors Red, Green and Blue. To generate such a color value, you can use the FAMOS function RGB().

The setting 'automatic' (-2) means here that the column's default text color is used.

Applies to:

Tableview

Examples:

After the user changes the contents of a table cell, a check is made of whether the value entered is a number > 0. If not, the cell is marked red.

--> Event sequence 'Changed' of the table

```
Row = PA2
Column = PA3
newValue = DlgGetCellValue(PA1, Row, Column)
IF newValue <= 0
   DlgSetCellTextColor(PA1, Row, Column, RGB(255,0,0))
ELSE
   DlgSetCellTextColor(PA1, Row, Column, -2)
END
```

See also:

DlgSetCellBackColor, DlgSetTextColor, DlgSetBackColor

DlgSetCellValue

Scope: User-defined dialogs

Resets the value of the specified table cell

Declaration:

```
DlgSetCellValue ( TxElementName, Row, Colummn, Value )
```

Parameter:

TxElementName	Name of the dialog element to be edited (table).
Row	Row index. The first row has the index 1. The column header (if visible) has the index 0.
Colummn	Column index. The first column has the index 1. The row header (if visible) has the index 0.
Value	Numerical value to which the cell is to be set

Description:

Conversion of the numerical value to displayed text proceeds according to the data type set for the table cell (property: "Data linking/ Data type"). For "Real number" or "automatic", this corresponds to the result of the function TForm(..., "").

Applies to:

Tableview

Examples:

A table presents a variety of a data set's characteristic values. If the maximum exceeds a cutoff limit, the corresponding table cell appears in a distinct color.

```
DlgSetCellText("Table", 1, 1, "Length ")
DlgSetCellValue("Table", 1, 2, leng?(MyData))
DlgSetCellText("Table", 2, 1, "Sampling time ")
DlgSetCellValue("Table", 2, 2, xdel?(MyData))
DlgSetCellText("Table", 3, 1, "Maximum ")
value = max(MyData)
DlgSetCellValue("Table", 3, 2, value)
IF value > 100
   DlgSetCellTextColor( "Table", 3, 2, RGB(255,0,0))
   DlgSetCellBackColor( "Table", 3, 2, RGB(127,127,127))
ELSE
   DlgSetCellTextColor( "Table", 3, 2, -2)
   DlgSetCellBackColor( "Table", 3, 2, -2)
END
```

See also:

DlgSetCellText, DlgGetCellValue

DlgSetItemSelection

Scope: User-defined dialogs

Sets of cancels the selection for an entry in a list with multi-selection.

Declaration:

```
DlgSetItemSelection ( TxElementName, Index, OnOff )
```

Parameter:

TxElementName	Name of the dialog element in question
Index	Index of the entry to be changed. The first entry has the index 1. In order to (de-)select all entries, enter 0.
OnOff	Selection On/ Off
	0 : Cancel selection
	1 : Selects an entry

Applies to:

Listbox (multi-selection), Tableview (multi-selection)

Examples:

Inverts the current selection in a list with multi-selection:

```
Count = DlgGetItemCount("list1")
i = 1
   WHILE i <= Count
   Sel = DlgIsItemSelected("list1", i)
   DlgSetItemSelection("list1", i, NOT(Sel))
   i = i + 1
END
```

See also:

DlgGetSelectedItemCount, DlgIsItemSelected, DlgSelectItem

DlgSetItemText

Scope: User-defined dialogs

Replaces an entry (list box, tree diagram etc.) with the specified text.

Declaration:

`DlgSetItemText ( TxElementName, Index, TxNewContents )`

Parameter:

TxElementName	Name of the dialog element to be changed
Index	Index of the entry to be replaced. The first entry has the index 1.
TxNewContents	New text for the entry to be replaced

Applies to:

Listbox, Droplist, , ComboBox, Treeview

Examples:

A list box in a dialog is filled with all of the values of a (short) data set. At the push of a button, all selected values are set to 0.

--> Event sequence 'Dialog initialization'

```
;Fills the list box with the data set's values:
Count = leng?(MyData)
i = 1
WHILE i <= Count
   TxVal = TForm( MyData[i], "")
   DlgInsertItem("list1", 0, TxVal, 0)
   i = i + 1
END
```

--> Event sequence 'Button pressed'

```
;The selected samples are set to 0.
Count = DlgGetItemCount("list1")
i = 1
WHILE i <= Count
   IF DlgIsItemSelected("list1", i)
      MyData[i] = 0
      DlgSetItemText("0")
   END
   i = i + 1
END
```

See also:

DlgGetItemCount, DlgGetItemText, DlgInsertItem, DlgFindItem, DlgDeleteItem

DlgSetText

Scope: User-defined dialogs

Sets a new content or caption of the specified element.

Declaration:

```
DlgSetText ( TxElementName, TxText )
```

Parameter:

TxElementName	Name of the dialog element to be changed
TxText	Text to which the dialog element is to be set

Applies to:

The function can only be used for such dialog element which possess a caption or whose current state can be expressed by a text. Interpretation of the text depends on the element's type.

Element	Meaning
Button	Button caption
Label	Content of the text box
Editbox (single line)	Content of the input box
Editbox (multi-line)	Content of the input box
Checkbox	The element's caption
Radiogroup	Sets the caption for the frame around the radiogroup. Only applicable for "Frame" = "With caption".
Groupbox	The element's caption
Listbox (Single selection)	If a list entry with the specified text exists, it is selected. If there is none, the current selection is retained.
Droplist	Like Listbox
Treeview	Like Listbox
Combobox	Content of the input box
Dialog	The dialog's title bar. For TxElementName, an empty text is to be entered.
Statusbar	Content of the dialog's status bar. For TxElementName, "#Status" is available.

For elements which aren't listed in the table above, the function isn't applicable.

Examples:

A dialog displays a variety of a data set's characteristic values; among others the Y-unit in an input box 'input_unit'. When closing the dialog with the 'OK' button, any changes of the unit made by the user are adopted in the data set.

--> Event sequence 'Dialog initialization'

```
...
DlgSetText("input_unit",  Unit?(MyData,1))
...
```

--> Event sequence 'Button pressed' for the 'OK'-button

```
TxUnit = DlgGetText("input_unit")
SetUnit(MyData, TxUnit, 1)
DlgCloseDialog(0)
```

See also:

DlgGetValue, DlgGetValue, DlgGetText

DlgSetTextColor

Scope: User-defined dialogs

Sets the font color

Declaration:

```
DlgSetTextColor ( TxElementName, RGBColor )
```

Parameter:

TxElementName	Name of the dialog element to be changed. If an empty text is specified, the default value is set for the entire dialog.
RGBColor	RGB-value of the desired color or (-2) for automatic.

Description:

The RGB-value supplied as the 2nd parameter represents the proportional intensities of the 3 fundamental colors Red, Green and Blue. To generate such a color value, you can use the FAMOS function RGB().

If you changed the text color for the whole dialog, this also changes the text color of all those dialog elements whose text color is set to "automatic".

The value (-2) sets the text color to automatic. For the dialog this means that the current Windows system configuration is used. For dialog elements, this option means that the dialog's text color is adopted ('inherited').

Applies to:

This function can not be applied to: sliders, pictures, curve windows, menu items, calendars, clock

Examples:

The data set's maximum value is checked against a cutoff limit and the result is displayed in a text box ("Label1") in the dialog. If the limit has been exceeded, the text box's background and text color are chaged (red/ white) and a warniong message is posted.

```
_max = Max(Daten)
IF _max > 12
   DlgSetTextColor("Label1", RGB(255,255,255))
   DlgSetBackColor("Label1", RGB(255,0,0))
   DlgSetText("Label1", "Limit exceeded!")
ELSE
   DlgSetTextColor("Label1", -2)
   DlgSetBackColor("Label1", -2)
   DlgSetText("Label1", "Dataset OK.")
END
```

See also:

DlgSetBackColor, DlgSetCellTextColor

DlgSetValue

Scope: User-defined dialogs

Sets a new value for the specified element.

Declaration:

```
DlgSetValue ( TxElementName, Value )
```

Parameter:

TxElementName	Name of the dialog element to be set
Value	Numerical value to which the dialog element is to be set

Applies to:

The function can only be applied to such dialog elements whose current state can be expressed by a number. How the value is interpreted depends on the element's type.

Element	Value
Label	The contents of the Label is set to the given value. The formatting depends from the Label's property "Data format".
Listbox (Single selection)	If available, the corresponding numeric value in the list is selected. If not found, the current selection is not changed. The conversion of the number is analogous to the function call TForm(...,"") . Practicable for integer numbers only.
Droplist	Like Listbox
Combobox	The number is converted to text and the input box is filled accordingly. Conversion of the number is performed analogously to calling the function TForm(...,"").
Editbox (single line)	Like Combobox
Checkbox	Allowed: 1 or 0. The check-box's status is set accordingly. No other values are allowed.
Radiogroup	Sets the option to be selected. The first option has the index 1.
Slider	Sets the slider to the value specified. If the value is outside of the range limits set for the control, the slider is positioned at the respective range edge.
Datepicker	Sets the element to the specified value for the date. The value must be stated in the FAMOS time format and can be generated and processed using the functions belonging to Function Group #18, 'Date/Time'.
Timepicker	Sets the element to the specified clock time. The value must be enterd in the FAMOS time format and can be generated and processed using the functions belonging to Function Group #18, 'Date/Time'.

For elements which aren't listed in the table above, the function isn't applicable.

Examples:

A slider control's value range is set and the midpoint initialized.

```
DlgSetBarRange("Slider1", 0, 100, 1, 10)
DlgSetValue("Slider1", 50)
```

Various characteristic values of a data set are displayed in a dialog, among others the trigger time, in a 'Datepicker'- and a 'Timepicker'-element. Upon closing the dialog by means of the 'OK' button, any changes to the time made by the user are adopted by the data set.

--> Event sequence 'Dialog initialization'

```
...
Time = Zeit?(MyData)
DlgSetValue("time", time)
DlgSetValue("date", time)
...
```

--> Event sequence 'Button pressed' for the 'OK'-button

```
date = DlgGetValue("time")
time = DlgGetValue("date")
SetTime(Mydata, TimeAdd(date, time))
```

See also:

DlgGetValue, DlgGetText, DlgSetText

DlgShow

Scope: User-defined dialogs

The active dialog or a special dialog element is hiddent or showed (again).

Declaration:

```
DlgShow ( TxElementName, OnOff )
```

Parameter:

TxElementName	Name of the dialog element to be changed. If an empty text is specified, the entire active dialog is displayed or hidden.
OnOff	
	0 : Hide element
	1 : Show element

Applies to:

The function can be used for all dialog elements except menus, toolbars and the status bar.

Examples:

The results of a calculation are saved to an EXCEL-file. Subsequently, EXCEL is opened and the file is displayed for checking. As long as EXCEL is open, the current dialog is hidden.

```
Filtrat =  FiltTP(Daten, 0, 0, 6, 100)
fh = FileOpenXLS2("z:\dat\res.xls", "Template #2", 1, 1, 1)
IF fh > 0
   err = FileObjWrite( fh, Filtrat)
   err = FileClose(fh)
   DlgShow("", 0)
   Execute("Excel", "z:\dat\res.xls", "", 0, -1)
   DlgShow("", 1)
END
```

See also:

DlgEnable

DrTitleN

Scope: Report Generator

Retrieves object title.

Declaration:

```
DrTitleN ( ObjectIndex, Zero ) -> TxTitle
```

Parameter:

ObjectIndex	Object is selected with this index.
Zero	Reserved, always set to 0
TxTitle	
TxTitle	Title of the addressed object.

Description:

The title of the object specified by SvIndex is retrieved. If the object has no title or if SvIndex is greater than the number of objects in the Report Generator, an empty string ("") is returned.

- This function is provided for reasons of compatibility. For newly created sequences or programs, RgObjGetTitle should be used.
- SvIndex always starts at 1.

Examples:

```
Amount = PrTitleI(0)
Title$ = PrTitleN(Amount, 0)
```

The last object's title is determined.

See also:

RgObjGetTitle, RgObjGetCount, RgObjGetType, DrTitleI

DSPLOAD

Loads a DIGISKOP(c)-file (Copyright remes GmbH)

Declaration:

```
DSPLOAD SvBoard SvChannel Filename VariableName
```

Parameter:

SvBoard	Board to load (1..5)
SvChannel	Channel to be loaded (1...160)
Filename	Name of the file to be loaded
VariableName	Variable in to which the file excerpt is entered

Description

A DIGISKOP(c) file group, defined by the remes company, is loaded in FAMOS. This DIGISKOP(c) file group consists of a *.DSP, *.PTR and *.DRD file, differentiated only by their file name extension. These files contain information and measurement values from 1 to 5 data acquisition boards and 1 to 160 channels per board. The desired file section is read using the parameters "Board" and "Channel". The parameter "Board" can accept values between 1 and 5, the parameter "Channel" values between 1 and 160.

The selected file excerpt is loaded with the designation specified under [VariableName].

- The filename can be a complete pathname included folder and filename extension, but can also be specified without either. Then, the system searches for the file in the folder from which FAMOS loads data. The extension is selected automatically.
- The filename can also be specified to contain quotation marks. This can be necessary, if, for instance, the path contains spaces.

Examples:

```
DSPLOAD 4 1 DSCP
DSPLOAD 4 1 c:\dig\DSCP.DSP karte4_1
```

In both cases, the first channel of the fourth board in read from the file group "DSCP.*" and entered in imc FAMOS under the name "board4_1".

```
board = 4
chan = 1
DSPLOAD kart kan DSCP daten
```

Two variables define the desired range of the file group "DSCP". The range is entered under the name "data".

```
DSPLOAD 4 1 "c:\Meine Daten\DSCP.DSP" karte4_1
```

The pathname contains spaces and must therefore be written inside of quotation marks.

See also:

FileLoad, FileOpenFAS, LDIR

e

Euler number e = 2.718...

Examples:

The natural logarithm of the pre-defined constant e is 1:

```
one = ln(e)
```

The following two formulas are equivalent:

```
y = exp(x)
y = e ^ x
```

See also:

ln, exp

eFit

Exponential regression; fits data set to an exponential function

Declaration:

```
eFit ( Data ) -> eFunction
```

Parameter:

Data	Data set to be fitted to the exponential function [ND],[XY]
eFunction	
eFunction	Data set fitted to the exponential function

Description:

A data set given by the equation

```
f(x) = A * exp(B*x)
```

is determined, which best approximates the original data set.

After completion of the function, the equation determined is displayed in the output box. The equation is displayed with the units and has the following form:

```
f(x) = 13.81 [V] * exp (116.8 [Ohm] * x)
```

Calling the function with an empty text as the parameter returns the last coefficients computed, B and A, in the form of a data set with 2 values.

Normal data set [NW]:

The data set generated has the length and unit of the specified data set, with a maximum length of 1000. If the length is shortened (truncated), the sampling time is changed so that the specified data set is defined for the same interval.

XY -data set [XY]

The data set generated has the length 1000. It is defined by the same x-interval as the source data set.

- The numerical values of the specified data set must be greater than zero.
- If numerical values below zero occur, a warning will appear along with the equation determined. This equation is meaningless and must not be considered correct under any circumstances. For this reason, in a sequence one should always check the coefficients found. If they are 1 and 0, the result is not valid.
- Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

Calculates the exponential regression for a data set. Subsequently, the two cofficients of the e-function are determined:

```
ExpRegression = eFit(Daten)
coeff = eFit("")
factorB = coeff[1]
factorA = coeff[2]
```

See also:

LFit, exp, ln, Appro, ApproNonLin, Poly

ELSE

In an IF- branching, this initializes the instructions to be performed if neither the IF-condition nor any exisiting ELSEIF-conditions are met.

Declaration:

```
ELSE
```

Description

The end of the instructions belonging to this block is denoted by a subsequent END.

An IF-block may contain any arbitrary amount of ELSEIF-branchings. If any ELSE-branching is also specified, it must appear at the last position of the chain.

Instead of IF/ELSEIF/ELSE, for many applications it is easier to use the SWITCH/CASE/DEFAULT-instruction.

Examples:

If the maximum value of a variable is greater than 0, the variable is displayed in a curve window. Otherwise, the variable is deleted.

```
Maxi = Max(data)
IF Maxi
   SHOW data
ELSE
   DELETE data
END
```

When a file is loaded, its return value is checked and an error message posted if appropriate.

```
fh = FileOpenDSF("Channel1.dat", 0)
IF fh = 0
   Pause ==> Can't load file <==
ELSE
 ; ...
   FileClose(fh)
END
```

A portion of an equidistantly sample data set [data] is to be cut out. The two boundaries [xmin] and [xmax] have previously been entered by the user and are now being checked for validity:

```
IF xmin < xoff?(data) OR xmin <= 0
   txError = "Illegal lower limit"
ELSEIF xmax > (xoff?(data)+(leng?(data)-1)*xdel?(data))
   txError = "Illegal upper limit"
ELSEIF xmin >= xmax
   txError = "Illegal limit relation"
ELSE
   result = Cut(data, xmin, xmax)
END
```

See also:

ELSEIF, ELSE, SWITCH

ELSEIF

Initializes an additional condition branch in an IF-branch. This is run if the previous condition branches were not run, and if the condition stated here has been met.

Declaration:

`ELSEIF Condition`

Parameter:

Condition	The following block's instructions are only carried out if the condition is met (evaluation returns a value >0).

Description

The end of the instructions belonging to this block is denoted by a subsequent, additional ELSEIF, an ELSE or END.

As the condition, it is possible to specify, for example, a single value variable or a complex expression using logical operators (AND, OR..) and/or comparison operators (<, =, ...).

An IF-block can contain any arbitrary amount of additional ELSEIF-branchings. Any ELSE-branch specified must appear at the last position in the chain.

Instead of IF/ELSEIF/ELSE, for many applications it is easier to use the SWITCH/CASE/DEFAULT-instruction.

Examples:

A portion of an equidistantly sample data set [data] is to be cut out. The two boundaries [xmin] and [xmax] have previously been entered by the user and are now being checked for validity:

```
IF xmin < xoff?(data) OR xmin <= 0
   txError = "Illegal lower limit"
ELSEIF xmax > (xoff?(data)+(leng?(data)-1)*xdel?(data))
   txError = "Illegal upper limit"
ELSEIF xmin >= xmax
   txError = "Illegal limit relation"
ELSE
   result = Cut(data, xmin, xmax)
END
```

See also:

IF, ELSE, SWITCH

EMPTY

Empty data set of length 0.

Examples:

A typical application of these constants is the initialization of a data set which is to be constructed step-by-step in a subsequent loop.

```
result = EMPTY
FOR i = 1 to ...
   newValue = ...
   result = Join(result, newValue)
END
```

See also:

Join

END

The command denotes the end of a sequence of instructions in a loop or branching.

Declaration:

```
END
```

Description

Last command in a loop (WHILE, FOR, FOREACH), of a conditional branching (IF , ELSEIF), or of a case distinction (SWITCH, CASE, DEFAULT).

Examples:

If a variable's maximum is greater than 0, it is displayed in a curve window.

```
Maxi = Max(data)
IF Maxi
   SHOW data
END
```

A descriptive text is formulated for a value nomally lying within the range from 0 to 100. If it is ouside of this range, an error message is displayed.

```
SWITCH Round(value, 1)
CASE 0 TO 48
    Tx = "Lower half"
CASE 49,50,51
    Tx = "Center"
CASE 52 To 100
   Tx = "Upper half"
DEFAULT
   PAUSE Invalid Value
END
```

See also:

IF, SWITCH, FOR, WHILE, FOREACH, BREAK, CONTINUE

END_PARALLEL

Available in: Professional Edition and above

Declaration:

`END_PARALLEL`

Description

A parallel execution block opened with BEGIN_PARALLEL is closed. The system waits until all previously started parallel executions are concluded.

See also:

BEGIN_PARALLEL

Supported since:

Version 2022, Editionen: Professional, Enterprise, Runtime

Envelope1

Constructs the upper and lower envelope curve according to the interval-secant-method.

Declaration:

```
Envelope1 ( Data, SvTimeInterval, SvOption, Zero ) -> ErgHüllkurve
```

Parameter:

Data	Data set for which to construct an envelope curve [ND]
SvTimeInterval	Time interval, scaled in the data set's x-unit
SvOption	Defines the calculation type
	1 : Upper envelope curve
	2 : Lower envelope curve
	3 : The upper envelope curve of the absolute value of the input data
	4 : Upper envelope curve, but with additional, interpolated values in the places where the lower envelope curve has nodes
	5 : Lower envelope curve, but with additional, interpolated values in the places where the upper envelope curve has nodes
Zero	Reserved parameter, always 0
ErgHüllkurve	
ErgHüllkurve	Resulting envelope curve

Description:

The algorithm demonstrated on the basis of the upper envelope curve:

The first input data value is stored. The secant in the interval with the greatest slope is sought, starting from the interval's beginning. This 2nd intersection point of the secant is stored. A new interval begins from this point onward. This procedure is then repeated. The last value is also stored.

- Slopes are regarded as equal if they differ from each other by 1e-13. If the same slope is found in several places in the interval, the last occurrence is given preference.
- Options 4 and 5 are much more demanding on computational capacity than ordinary envelope curves. The options are good choices when the upper and lower curves are to be combined in formulas. In that case, both curves should be computed with Options 4 or 5, respectively.
- Using the function Sub() the difference between the two envelope curves can be computed in conjunctions with Options 1 and 2, even if the nodes are not at the same locations.

Examples:

The oscillations in a waveform have a period of up to 2.0 seconds. This is rounded up to 5.0 sec. The function is supposed to join all the peak values in an upper envelope curve without dipping down into the valleys.

```
Envel = Envelope1(TimeData, 5.0, 1, 0)
```

For quickly finding the difference between the upper and lower envelopes:

```
UpperEnvel = Envelope1(TimeData, 1e-3, 1, 0)
LowerEnvel = Envelope1(TimeData, 1e-3, 2, 0)
Difference = Sub(UpperEnvel, LowerEnvel, 0)
```

An inefficient way of determining the difference:

```
UpperEnvel = Envelope1(TimeData, 1e-3, 4, 0)
LowerEnvel = Envelope1(TimeData, 1e-3, 5, 0)
Difference = XYof(UpperEnvel.x, UpperEnvel.y - LowerEnvel.y)
```

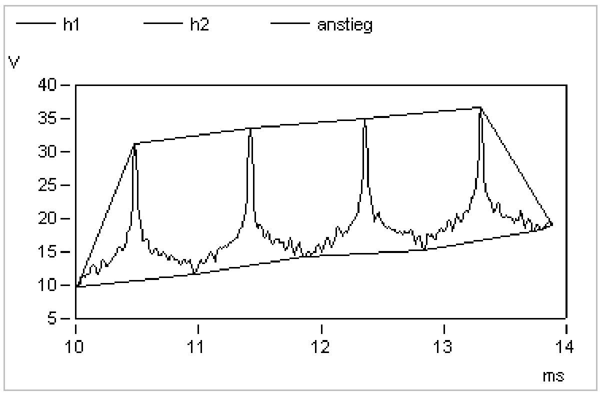
Comparison of original data set and envelope, sample for sample:

```
UpperEnvel = Envelope1(TimeData, 1e-3, 1, 0)
Oe = XYdt(UpperEnvel.x, UpperEnvel.y, xdel?(TimeData))
; "UE" is now a data set which is sampled exactly like "Timedata".
deviation = Oe - TimeData
```

The data set "slope.dat", provided with the imc FAMOS package, can be manipulated in the following way to demonstrate this function's workings:

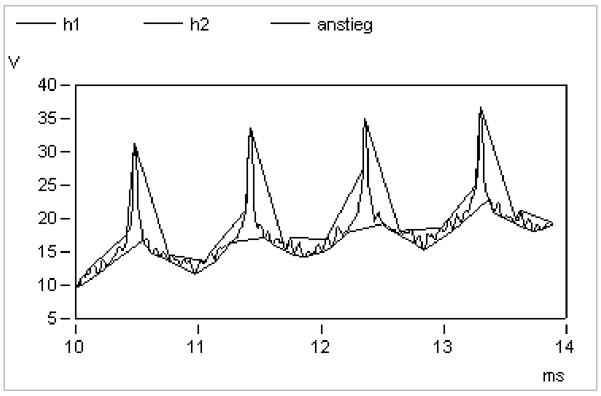
```
UpperEnvel = Envelope1(slope, 1e-3, 1, 0)
```

```
LowerEnvel = Envelope1(slope, 1e-3, 2, 0)
```



Using a smaller interval makes the envelope tighter:

```
h1 = Envelope1(slope, 3e-4, 1 ,0)
h2 = Envelope1(slope, 3e-4, 2 ,0)
```



See also:

Envelope2, Sub, XYof

Envelope2

Constructs the upper and lower envelope curve according to the interval-secant-method using specified nodes.

Declaration:

```
Envelope2 ( Data, SvTimeInterval, SvOption, Zero, Nodes ) -> ErgHüllkurve
```

Parameter:

Data	Data set for which to construct an envelope curve [ND]
SvTimeInterval	Time interval, scaled in the data set's x-unit
SvOption	Defines the calculation type
	1 : Upper envelope curve
	2 : Lower envelope curve
	3 : The upper envelope curve of the absolute value of the input data
	4 : Upper envelope curve, but with additional, interpolated values in the places where the lower envelope curve has nodes
	5 : Lower envelope curve, but with additional, interpolated values in the places where the upper envelope curve has nodes
Zero	Reserved parameter, always 0
Nodes	Data set of time points or x-coordinates; must be ordered to increase strictly monotonically! At these time points, the envelope curve is forced to coincide with an input sample.
ErgHüllkurve	
ErgHüllkurve	Resulting envelope curve

Description:

The algorithm demonstrated on the basis of the upper envelope curve:

The first input data value is stored. The secant in the interval with the greatest slope is sought, starting from the interval's beginning. This 2nd intersection point of the secant is stored. A new interval begins from this point onward. This procedure is then repeated. The last value is also stored. Additionally, the results contain all points specified in the list of nodes.

- Slopes are regarded as equal if they differ from each other by 1e-13. If the same slope is found in several places in the interval, the last occurrence is given preference.
- Options 4 and 5 are much more demanding on computational capacity than ordinary envelope curves. The options are good choices when the upper and lower curves are to be combined in formulas. In that case, both curves should be computed with Options 4 or 5, respectively.
- Using the function Sub() the difference between the two envelope curves can be computed in conjunctions with Options 1 and 2, even if the nodes are not at the same locations.

See also:

Envelope1, Sub, XYof

Equal

Available in: Professional Edition and above

Compares two data sets for equality. Each measurement value of one data set is compared with the corresponding measurement value of the other data set. The data sets are only equal of all values are exactly equal.

Declaration:

```
Equal ( Value1, Value2 [, Tolerance] [, Calculation] [, Tolerance2] [, Calculation2] ) -> IsEqual
```

Parameter:

Value1	1st comparison value
Value2	2nd comparison value
Tolerance	Tolerance to apply to the comparison. The two values to be compared may only differ by a maximum of this tolerance value. >= 0. (optional , Default value: 0)
Calculation	How is the tolerance calculated? (optional , Default value: "absolute")
	"absolute" : Absolute deviation
	"relative" : Relative deviation
Tolerance2	Tolerance of 2nd component. Only needed if the data sets to be compared are either XY or complex. (optional , Default value: 0)
Calculation2	How is the tolerance of the 2nd component calculated? Only needed if the data sets to be compared are either XY or complex. (optional , Default value: "absolute")
	"absolute" : Absolute deviation
	"relative" : Relative deviation
IsEqual	
IsEqual	IsEqual, so 1 for equality, 0 for deviation

Description:

This function is used to incorporate self-verification into sequences in a compact way.

Only the measured readings are compared. Other properties such as the trigger time, sampling time, or units are not compared.

If the measurement readings to be compared are real numbers, it is generally necessary to apply a tolerance.

A relative deviation of 1e-7 is near the 4-Byte real number resolution; 1e-14 is near to the 8-Byte real number resolution.

Equality can only be achieved if the number of samples is the same in both data sets.

This function can be applied to input data having events or segments. Equality is only achieved when the number of segments and events is also equal.

With absolute deviation, the absolute value of the difference between Value1 and Value2 may not exceed the tolerance.

With relative deviation, the ratio of Value1 to Value2 may deviate only by a certain proportion: Value1 / (1+Tolerance) <= Value2 <= Value1 * (1+Tolerance)

If data have 2 components (XY or complex), then both components are compared. In that case, Tolerance2 and Calculation2 must also be observed.

Examples:

Successful checking of two almost equal numbers which differ by a small percentage

```
A=2
B=sqrt(2)^2 ; A <> B !
eq = equal( A, B, 1e-10, "relative" )
verify( eq )
```

Checking of two time series which actually should be equal, but due to the numerical math only are approximately equal. The difference between the two is always below 1e-3.

```
ra1=ramp(0,1,1024) ; test data
ra2 = ifft(fft(ra1))
eq = equal( ra1, ra2, 1e-3, "absolute" )
```

See also:

Verify

EventAppend

One or more events are appended to an event-based data set

Declaration:

```
EventAppend ( EventData, NewEventData, Zero )
```

Parameter:

EventData	Event-based data set
NewEventData	Data to append
Zero	Reserved parameter. Always 0.

Description:

A data record is appended to an event-based data set.

If the 2nd parameter itself does not have an event structure, the data set is appended as a new event.

If the 2nd parameter is already evented itself, all its events are appended.

The two data sets must have the same data type (both normal real or both XY-data sets). The data format of the result remains intact. The properties x-offset (pretrigger), x-delta (sampling rate) and trigger time of the 2nd parameter are applied to the new event. All other properties remain intact.

To generate an event-based data set, either use the function EventNew() or pass a non-event-based data set as the first parameter to the EventAppend()-function. This is then automatically converted to an event-based data set before the second event is appended.

Instead of

```
a = EventNew(a, 0)
EventAppend(a, b, 0)
```

can thus also be written more briefly:

```
EventAppend(a, b, 0)
```

When a data set is structured in events, the individual events may have different lengths; other properties such as the trigger time and pre-trigger (or x-offset) may also be event-specific. For example, it is possible to join consecutive measurements together on one physical measurement channel in an event-based data set, for simplicity of management.

Many imc FAMOS functions can not process multi-shot channels directly, but must be performed by a loop over all the data set's events.

Examples:

A new, three-event data set is created from three individual data sets:

```
DataAllDay = EventNew(Data0_8h, 0)
EventAppend(DataAllDay, Data8_16h, 0)
EventAppend(DataAllDay, Data16_24h, 0)
```

See also:

EventNew, EventGet, EventDel, EventNum?, EventJoin, Join, JoinEx

EventDel

One event is deleted from an event-based data set.

Declaration:

`EventDel ( EventData, SvEventIndex, Zero )`

Parameter:

EventData	Event-based data set
SvEventIndex	Index of the event to be deleted; in the range 1.. Event Count
Zero	Reserved parameter. Always 0.

Description:

The event specified is deleted. The total length of the data set is reduced by the length of the deleted event.

If you wish to perform this function in a loop on all the events of a data set, remember that by deleting an event the total number of events in the data set decreases, thus altering the (addressing) structure within the data set. Therefore, it is recommendable to delete events from a data set from the end backwards (see example).

Examples:

All events comprising less than 10 measurement values are deleted from a data set:

```
count = EventNum?(signal)
index = count
WHILE index >= 1
   eventLeng = EventProp?(signal, index, 3)
   IF eventLeng < 10
      EventDel(signal, index, 0)
   END
   index = index -1
END
```

See also:

EventGet, EventSet, EventNum?

EventGet

One event is copied from an event-based data set.

Declaration:

```
EventGet ( EventData, SvEventIndex, Zero ) -> EventCopy
```

Parameter:

EventData	Event-based data set
SvEventIndex	Index of the desired event; in the range 1.. Event Count
Zero	Reserved parameter. Always 0.
EventCopy	
EventCopy	Copy of the specified event

Description:

This function returns a copy of a specified event. All the event's properties, including the data format, are adopted. The resulting data set has no event structure.

This function is generally used to enumerate the individual events. Many imc FAMOS functions can not process multi-shot channels directly, but must be performed by a loop over all the data set's events.

In imc FAMOS an event can also be accessed directly via its index by means of [..]:

```
singleEvent = EventData[ Index ]
```

Examples:

A loop is executed on all the events of a data set. Each event is smoothed and copied back into the original data set.

```
count = EventNum?(signal)
FOR index = 1 TO count
   thisEvent = EventGet(signal, index, 0)
      ;; also possible:
      ;; thisEvent = signal[index]
   thisEvent = Smo5(thisEvent)
   EventSet(signal, thisEvent, index, 0)
      ;; also possible:
      ;; signal[index] = thisEvent
END
```

Note: Such tasks can often be solved more elegantly by means of a FOREACH EVENT-loop.

See also:

EventSet, EventNum?, EventDel, FOREACH

EventJoin

Dissolves the event-structure of a data set.

Declaration:

```
EventJoin ( EventData, Zero )
```

Parameter:

EventData	Event-based data set
Zero	Reserved parameter. Always 0.

Description:

All events in a data set are attached to each other to a continuous stream of data; the event structure is canceled in the process. The properties of the first event in the data set are adopted by the entirety of the new data set.

CAUTION: The event-specific properties of the latter events are lost, this includes trigger time, pretrigger, sampling rate.

The total length of the data set remains intact.

If the first parameter has no event structuring, it remains unchanged by this function. A warning message is generated.

This function is generally used in preparation for further processing.. Many imc FAMOS functions can not process multi-shot channels directly, but must be performed by a loop on all the data set's events, or on a data set previously restructured by this function.

Examples:

The absolute maximum across all events of an event-based data set is determined:

```
EventJoin(signal, 0)
Maximum = Max(signal)
```

See also:

EventSet, EventGet, JoinEx

EventNew

From an unstructured data set, an event-based data set is generated.

Declaration:

```
EventNew ( FirstEvent, Zero ) -> EventData
```

Parameter:

FirstEvent	Data set from which the first event is formed
Zero	Reserved parameter. Always 0.
EventData	
EventData	Event-based data set

Description:

A copy of the source data, incorporating or multi-sot structure, is created. Subsequent use of the EventAppend function can be used to expand the data set by additional events.

When a data set is structured in events, the individual events may have different lengths; other properties such as the trigger time and pre-trigger (or x-offset) may also be event-specific. For example, it is possible to join consecutive measurements together on one physical measurement channel in an event-based data set, for simplicity of management.

Examples:

A new, three-event data set is created from three individual data sets:

```
DataAllDay = EventNew(Data0_8h, 0)
EventAppend(DataAllDay, Data8_16h, 0)
EventAppend(DataAllDay, Data16_24h, 0)
```

See also:

EventAppend, JoinEx

EventNum?

Gets the number of events in a data set.

Declaration:

```
EventNum? ( EventData ) -> SvEventCount
```

Parameter:

EventData	Event-based data set
SvEventCount	
SvEventCount	EventNumber
	>=0 : Number of events included
	-1 : The parameter is not an event-based data set.

Description:

The number of events present in a multi-shot channel is determined. A value of 1 is returned if the data set is not accordingly structured.

This function is generally used to prepare for subsequent enumeration of the data set's individual events. Many imc FAMOS functions can not process event-based data sets directly, but must be called for all the data set's events by means of a loop.

Examples:

A loop is executed on all the events of a data set. Each event is smoothed and copied back into the original data set.

```
count = EventNum?(signal)
FOR index = 1 TO count
   thisEvent = signal[index]
   thisEvent = Smo5(thisEvent)
   signal[index] = thisEvent
END
```

Note: Such tasks can often be solved more elegantly by means of a FOREACH EVENT-loop.

See also:

EventSet, EventGet, Join

EventProp

Sets event-specific properties

Declaration:

```
EventProp ( EventData, SvEventIndex, SvTask, SvNewProperty )
```

Parameter:

EventData	Event-based data set to be changed
SvEventIndex	Index of the event to be changed; in the range 1.. Event Count
SvTask	Selection of the characteristic value to be changed
	0 : The event's trigger time (expressed in the internal time format).
	1 : Sampling interval/x-Delta
	2 : Pretrigger time/x-Offset
	5 : z-Offset
SvNewProperty	New value

Description:

For a specified event in a data set, event-specific properties are changed.

When a data set is structured in events, the individual events may have different lengths; other properties such as the trigger time and pre-trigger (or x-offset) may also be event-specific. For example, it is possible to join consecutive measurements together on one physical measurement channel in an event-based data set, for simplicity of management.

Examples:

All events of an event-basedd data set are assigned the same trigger time.

```
time = TimeJoin(11, 2, 1997, 15, 0, 0)
count = EventNum?(signal)
FOR index = 1 TO count
   EventProp(signal, index, 0, time)
END
```

See also:

EventSet, EventNum?, EventProp?

EventProp?

Queries event-specific properties

Declaration:

```
EventProp? ( EventData, SvEventIndex, SvTask ) -> SvProperty
```

Parameter:

EventData	Event-based data set to be queried
SvEventIndex	Index of the desired event; in the range 1.. Event Count
SvTask	Selection of the characteristic value to be requested
	0 : The event's trigger time (expressed in the internal time format).
	1 : Sampling interval/x-Delta
	2 : Pretrigger time/x-Offset
	3 : Length (number of data points)
	4 : Index of the data sets first point belonging to the relevant event. For the first event this is 1. For all subsequent events it is the sum of the lengths of preceding events, plus 1.
	5 : z-Offset
SvProperty	
SvProperty	Value requested

Description:

For a specified event in a data set, event-specific values are requested.

When a data set is structured in events, the individual events may have different lengths; other properties such as the trigger time and pre-trigger (or x-offset) may also be event-specific. For example, it is possible to join consecutive measurements together on one physical measurement channel in an event-based data set, for simplicity of management.

Examples:

All events comprising less than 10 measurement values are deleted from a data set:

```
count = EventNum?(signal)
index = count
WHILE index >= 1
   EventLeng = EventProp?(signal, index, 3)
   IF EventLeng < 10
      EventDel(signal, index, 0)
   END
   index = index - 1
END
```

See also:

EventProp, EventSet, EventNum?

EventSet

A selected event in a data set is replaced.

Declaration:

```
EventSet ( EventData, NewEvent, SvEventIndex, Zero )
```

Parameter:

EventData	Event-based data set
NewEvent	New data for the specified event
SvEventIndex	Index of the event to be changed; in the range 1.. Event Count
Zero	Reserved parameter. Always 0.

Description:

In an event-based data set, a specified event is replaced. The replacement data set may not itself have a multi-event structure.

The first two parameters must have the same data type (both real, complex, or XY-data sets). The data format of the first parameter is retained.

This function is generallyused to separate an individual event from the data set, and later restore it after having first processed it. Many imc FAMOS functions can not process multi-shot channels directly, but must be performed by a loop over all the data set's events.

In imc FAMOS an event can also be accessed directly via its index by means of [..]:

```
EventData[ Index ] = NewData
```

Examples:

A loop is executed on all the events of a data set. Each event is smoothed and copied back into the original data set.

```
count = EventNum?(signal)
FOR index = 1 TO count
   thisEvent = EventGet(signal, index, 0)
      ;; also possible:
      ;; thisEvent = signal[index]
   thisEvent = Smo5(thisEvent)
   EventSet(signal, thisEvent, index, 0)
      ;; also possible:
      ;; signal[index] = thisEvent
END
```

Note: Such tasks can often be solved more elegantly by means of a FOREACH EVENT-loop.

See also:

EventGet, EventNum?, EventDel, FOREACH

Execute

A program is started and/or an action with a document file is carried out.

Declaration:

```
Execute ( TxFileName, TxParameter, TxAction, SvDisplayOption, SvWaitOption ) -> SvExitCode
```

Parameter:

TxFileName	Name of the executable file (*.exe, *.bat, *.com) or name of the document-file to be edited. A document-file must be linked to an application via the Windows system settings.
TxParameter	Parameter to be used when starting application, separated by a space. For document-files an empty text.
TxAction	States the nature of action to be performed. The available action verbs depend on the specified file's type. To run an application, an empty string or "Open".
SvDisplayOption	Display of the program window
	0 : Starts/shows the application in a normal window. The application is activated.
	1 : Starts/shows the application in a normal window. The application is not activated.
	2 : Starts/shows the application as an icon.
	3 : Starts/shows the application in a maximized window.
SvWaitOption	Waits for the end
	0 : The application is started. The function returns immediately.
	-1 : The application is started. The function returns once the application has been completed. Useful for calls of the command interpreter and other applications no having an explicit user interface.
	>0 : Maximum waiting time (in seconds). The function returns when the application which was started is completed OR when the waiting time has elapsed.
SvExitCode	

SvExitCode	Status
	>=0 : Function has been successfully executed and completed. The particular value depends on the application and corresponds to the code returned to the operating system by the application. This code can, for instance, report success in executing the function.
	-2 : The application was started successfully, but isn't completed within the specified waiting time.
	-3 : The desired action was carried out by an already open instance of the application and thus no new application was started.

Description:

Notes on editing document files:

If a document-file is given as the first parameter, the application with which this file is edited must be known. A corresponding link can be established in the Windows system settings (Explorer / View / Options / File types) and is identified by the filename extension.

Along with the Windows default settings, such links are normally creeated automatically upon installation of an application, but can be manually revised later by the user. Here, so-called action verbs are defined, which describe various processing actions. An action defined in this way must then be supplied as the 3rd parameter [TxAction].

The following action verbs are common for documents:

"Open"	Starts the file (executable file or document)
"Edit"	Opens an editor and loads the document.
"Explore"	Opens the Windows Explorer and displays the specified directory.
"Find"	Starts a search in the specified directory.
"Print"	Prints out the document-file.
"Properties"	Opens the Properties dialog for the specified file.

Which verbs are actually available depends on the Windows system settings. An application itself can determine which actions are supported and define additional verbs (e.g. "play" for video and sound-files).

Remarks

- If no full pathname is specified for executable files, a search of the folders listed by the environment variable [PATH] is conducted. For document-files, by contrast, the complete pathname must be specified.
- In case of an error (e.g. invalid pathname or unknown filename-extension), an error message is displayed and sequence processing is aborted.
- While the function is waiting for the end of the application started ([WaitOption] = -1 or >0), FAMOS can't be operated (exception: curve window). However, a running sequence can be interrupted in the usual ways ("Sequence"-symbol with context menu in System Tray or [Ctrl+Break]). During the waiting time, FAMOS is partially accessible via DDE: data can be received, but no commands executed.

- If a parameter in [TxParameter] contains spaces, it must be enclosed in quotation marks. See the example.
- Query of return value is optional in FAMOS.
- The parameter [WaitOption] is passed to the application started. However, it depends on the application whether it is taken into account.
- If some instance of the desired application is already open, the resulting response also depends on the application. Either a new, addtional program instance is started or the already open one re-used. In the latter case the system doesn't wait for the end, regardless of the parameter [WaitOption], the function returns immediately (denoted by a return value of [-3]).

Examples:

Starts the Editor [Notepad] and loads the specified file. The function returns immediately.

```
ret = Execute("notepad.exe", "c:\tmp\MyTest.txt", "", 0, 0)
```

Parameters with spaces must be enclosed in quotation marks:

```
ret = Execute("notepad.exe", """c:\tmp\My Text.txt""", "", 0, 0)
```

The same result can be obtained by making the following call (Assuming: Extension "txt" is linked to [Notepad]):

```
ret = Execute("c:\tmp\My Text.txt", "", "open", 0, 0)
```

Starts MS-EXCEL and loads the table specified. The function returns when EXCEL closes again, but after 60s at the latest. Both calls are equivalent.

```
ret = Execute("excel.exe", "c:\tmp\1.xls", "", 0, 60)
ret = Execute("c:\tmp\1.xls", "", "open", 0, 60)
```

Prints out the specified MS-EXCEL file.

```
ret = Execute("c:\tmp\1.xls", "", "print", 2, 0)
```

Calls the packing program "7-Zip" for unpacking a file:

```
fullname$ = "c:\files\archive.zip" ; input file
dir$ = "c:\files\unpacked" ; output folder

para$ = "e """ + fullname$ +""" -o""" + dir$ + """"
; Please note the extra quotation marks around the filenames. These are needed in case the filenames contain spaces.
ret$ = Execute("c:\Program Files\7-Zip\7z", para$, "", 0, -1)
```

Starts MS-EXCEL and loads the table specified. The function returns when EXCEL closes again, but after 60s at the latest. Both calls are equivalent.

```
ret = Execute("excel.exe", "c:\tmp\1.xls", "", 0, 60)
ret = Execute("c:\tmp\1.xls", "", "open", 0, 60)
```

Prints out the specified MS-EXCEL file.

```
ret = Execute("c:\tmp\1.xls", "", "print", 2, 0)
```

To execute an operating system command, the command processor can be started with the option "/C", followed by the desired command. In the example, a folder is created. The console window is shown in minimized mode. The function returns when the Command-interpreter is finished, in other words, after the folder was actually created.

```
ret = Execute("cmd.exe", "/C MkDir z:\NewFolder", "", 2, -1)
```

Calls the "Properties"-dialog for the specified file:

```
ret = Execute("c:\tmp\data.raw", "", "properties", 0, 0)
```

Comment: Here, the standard Windows Explorer is used to display the Properties dialog. Since this is always active, a (-3) is returned here and the last parameter [WaitOption] is ignored.

See also:

APPLICATION

EXITSEQUENCE

Quit sequence execution

Declaration:

```
EXITSEQUENCE SvOption
```

Parameter:

SvOption	Option
	0 : Sequence processing is stopped completely. Default for classic sequences, not permitted within sequence functions.
	1 : Skip to the end of the current sequence. If this is a sub-sequence, processing of the calling sequence resumes. Default for sequence functions.
	2 : Skip to the end of the current sequence. If that is a cyclical sub-sequence (called with parameters containing wildcards), the system resumes with the next cycle. Otherwise, the system resumes processing of the calling sequence. Not permitted within sequence functions.

Description

The sequence processing is stopped. Depending on this option chosen, either only the running sub-sequence/sequence function is stopped (skip back to the calling sequence), or the entire sequence processing is terminated.

The parameter is optional. If not specified, then in case of a classic sequence, the system assumes 0 (complete Cancel), or in the case of a sequence function, it assumes 1 (skip to end of the sequence).

Examples:

Cancelling a sequence after a user input:

```
Stop = BoxMessage("Evaluation", "Cancel evaluation ?", "?4")
IF Stop
   EXITSEQUENCE
END
```

Calling the following sequence with

```
SEQUENCE Evaluation *.dat
```

causes all files named "*.dat" which belong to the current folder to be evaluated and which could be successfully loaded in imc/FAMOS format to be evaluated.

```
; Sequence "Evaluation.seq"
fh = FileOpenDSF("PA1", 0)
IF fh < 1
   EXITSEQUENCE 2
END
...
; Extensive evaluation follows
...
```

Conversely, if the EXITSEQUENCE parameter is changed to 1, execution stops as soon as opening of a file fails (e.g. because the file is not saved in the imc/FAMOS format).

See also:

PAUSE

exp

Exponential function (exponent of base e)

Declaration:

```
exp ( Parameter ) -> Result
```

Parameter:

Parameter	Parameter. Allowed types: [ND],[XY].
Result	
Result	Result from calculation of the e-function.

Description:

The exponential function raises the base e (Eulersche number = 2.718...) to the parameter in the exponents.

The exponential function can also be expressed using the function ^ (Power). The following formulas have the same effect:

```
y = exp(x)
y = e ^ x
```

Remarks

- The x-coordinate(s) of the parameter and the result are the same.
- The exponential function increases rapidly in a positive argument. The valid numerical range is exceeded in arguments with values over 46.
- Calling ln (exp (x)) represents an identity as long as the intermediate value does not exceed the permitted range of values.
- The parameter of the exponential function should not have any unit.
- The parameter may be structured (events/segments).

Examples:

Values from a measurement device are corrected according to a exponential characteristic:

```
data_corrected = 2 * exp(0.1 * data)
```

See also:

ln, ^(hoch)

ExpoRMS

Moving RMS with exponential averaging

Declaration:

```
ExpoRMS ( Data, SvTimeConstant, SvReduction ) -> MovingRMS
```

Parameter:

Data	Data set to be averaged. [ND]
SvTimeConstant	Time constant/weighting of the filter
	>=0 : Time constant of filter
	-1 : FAST weighting
	-2 : SLOW weighting
	-3 : IMPULSE weighting
	-4 : PEAK weighting
SvReduction	Width of the reduction interval (in x-units).
MovingRMS	
MovingRMS	Moving RMS value

Description:

The floating RMS with exponential weighting and subsequent resampling is computed.

The exponential weighting used here can be considered as a 1st order filter. The time constant can be greater or less than the sampling rate.

The floating RMS is calculated in the following manner: The signal is squared, then the exponentially weighted mean is taken, and then the mean's square root. A conventional RMS weights all squares equally, whereas here a time-weighting is performed for the moving RMS calculation.

The function MvRMS provides a floating RMS featuring equal weighting across the width of the window. 'Older' measured values are weighted less here.

After filtering the result is resampled over the interval width [SvReduction]. If [SvReduction] is smaller than or equal to the sampling rate, no data reduction takes place.

This function operates analogously to the RMS computation in the functions KBT (KB-Evaluation) , OctA (octave analysis), and ABCRating..

The predefined time-weighting modes are defined as per the standard IEC 651 as follows::

FAST	Time constant = 0.125 s.
SLOW	Time constant = 1 s.
IMPULSE	For increasing amplitudes the time constant is 35 ms, for decreasing amplitudes 1.5 s. Thus impulse-shaped signals are captured quickly, the response decays slowly.
PEAK	Extreme response for very short impulses; ensuring capture of the peak value. Time constant is zero during increasing amplitude (can be performed exactly by computer, by analog operation only in approximation); during decreasing amplitude 3 s.

Examples:

```
resultRMS = ExpoRMS(timeData, 0.125, 0.05)
```

The floating RMS is determined every 0.125 s (FAST-weighting). The value at every 0.05s of the result is used.

```
resultRMS = ExpoRMS(timeData, -2, 0)
```

Calculation of RMS with 1s time constant (SLOW-weighting). No data reduction performed.

See also:

MvRMS, MvMean, KBT, RMS, ABCRating

FAMOS

Sets the imc/FAMOS format for subsquent loading of files using th command LOAD

Declaration:

```
FAMOS
```

Description

The command is obsolete; instead of the command combination FAMOS/LOAD, you can use the more powerful and user-friendly functions FileLoad() or FileOpenDSF().

The file format for loading files is set to the FAMOS format.

Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
FAMOS
LOAD DATA
```

The file format for loading files is set to FAMOS format. The file DATA.DAT is loaded and entered in the variable list under the name DATA.

See also:

LOAD, FileLoad, FileOpenDSF

FASLOAD

Loads a file in a user-defined format (File Assistant or import/export extension library)

Declaration:

```
FASLOAD TxFileName TxFormat VariableName
```

Parameter:

TxFileName	Name of the file to be loaded. To be passed as a text constant in "" or as text variable.
TxFormat	Specification of the file format
VariableName	Name under which the variable is entered in the Variables list. This name is used only when the file contains exactly one data object. Otherwise, the name specified in the format definition is used (transfer as text constant in "" or as text variable)

Description

This command is obsolete; instead of it you can use the more powerful and convenient function FileLoad().

There are two ways of specifying the format in the parameter <TxFormat>:

Import filter (File Assistant)

The format is determined by an importation description file (an import filter) created by means of the imc File Assistant. Such an import filter file usually takes the extension ".fas". This file's name is then supplied as the parameter. Unless a complete pathname is provided, the system searches in the default definitions' folder ("Extra"/ "Options"/ "Folders").

Import/Export extension library

The necessary functionality is provided by an import/export expansion library. Such libraries (Dynamic Link Library, DLL) are offered by imc or our distribution partners and can be seamlessly integrated into the FAMOS user's interface and function library. This means that FAMOS can be flexibly adapted to support additional file formats. One example is the DLL for importing and exporting the MATLAB file format which is included in the standard FAMOS package.

The syntax of the parameter <TxFormat> is then as follows:

```
"#DLLName|FormatName|Parameter"
```

[DLLName] specifies the expansion library's filename; the extension ".dll" can be omitted.

[FormatName] specifies the name of the desired format (an expansion library can certainly support multiple formats). This can be omitted if the DLL provides only one format and thus needs no additional parameters.

[Parameter] is for passing additional optional parameters to the library. Whether parameters are needed, and what theur syntax depends on the respective extension library. Often not required, in which case it can be omitted.

Tip:Use the Functions Assistant to parameterize the function. Here, you can conveniently make a selecton from a list of all installed file filters.

Examples

```
FASLOAD "1.txt" "#MyFormat.DLL"
FASLOAD "1.txt" "#MyFormat.DLL|Txt"
FASLOAD "1.asc" "#MyFormat.DLL|Asc|Header=yes"
```

An overview of the import filters included with the FAMOS product package, and how to parameterize them, is available in the online help "FAMOS User's Manual"=>"Import/Export Filter"

- If the filename does not contain a full filepath, the folder currently set for file loading is used. After launching imc FAMOS, this path is the default path set in the dialog "Options"/ "Folders" and can be modified with the command LDIR or the function SetOption(), for example.
- Wildcards can also be specified in file names to load a series of files. The wildcard '?' stands for an exact character, '*' stands for an undefined number of any characters.
- The variable name is optional. It is used only when the file to be read contains exactly one data object. Otherwise, the variable names specified with the File Assistant are used.
- To load user-defined files, you can also use the function FileOpenFAS(). This is especially preferable when working with multi-channel files.

Examples:

```
FASLOAD "Daten.dat" "Oszi2311.fas" "Kanal1"
```

The file DATA.DAT is read using the specified format description format and saved in imc FAMOS as a variable with the name "Channel1".

```
FASLOAD "Daten.mat" "#MatLabImportExport.DLL"
```

The file DATA.MAT in the MatLab® -format is loaded.

```
TxFormatFile = "myformat.fas"
FASLOAD "c:\imc\*.*" TxFormatFile
```

All files are loaded in the directory c:\imc.

```
FASLOAD "a??.*" TxFormatFile
```

All files whose names begin with an 'a' and consist of three characters are loaded in the current directory. Any file name extension can be used.

```
FASLOAD "*a1*.dat" TxFormatFile
```

All files with the extension 'dat', whose names contain the string '1a' (at the beginning or end) are loaded.

See also:

FileLoad, FileOpenFAS, LDIR

FFT

Spectrum according to the FFT algorithm

Declaration:

```
FFT ( Data [, SvWindow] [, SvMode] ) -> Spectrum
```

Parameter:

Data	Data set from which the spectrum should be calculated [NW].
SvWindow	Window-function (optional , Default value: -1)
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman/Harris
	5 : Flat-Top
	-1 : The global preset (dialog: 'Options'/'Functions', SetOption(), FFTOPTION) is used. Compatibe with older versions of FAMOS (< V2022).
SvMode	Procedure (optional , Default value: -1)
	0 : Radix 2: Before the calculation, the length of the data set is reduced (truncated) to the next lower power of 2.
	1 : Radix 2: Before the calculation, the data set's length is increased (by adding zeroes) up to the next higher power of 2.
	2 : Radix 2: Before the calculation, the data set is re-sampled so that the data set's length becomes a power of 2. This corresponds to a previous application of the function Red2().
	3 : Mixed-Radix: The length of the data set needs to be a product of powers of the numbers 2, 3 and 5. Other data set lengths can not be processed.
	-1 : The global preset (dialog: 'Options'/'Functions', SetOption(), FFTOPTION) is used. Compatibe with older versions of FAMOS (< V2022).
Spectrum	
Spectrum	Discrete spectrum of the data according to FFT-algorithm [MP].

Description:

The FFT-function calculates a data set's discrete spectrum according to the very fast FFT-algorithm. In this implementation, the classic algorithm by Cooley and Tukey is used in the Radix-2- or Mixed-Radix form. The FFT (Fast Fourier-Transformation) is a special case of the general discrete Fourier-Transformation (DFT). In this special case, the procedure exploits the fact that the length of the data set is a power of 2 (Radix 2) or the product of powers of the numbers 2, 3 and 5 (Mixed-Radix).

The spectrum generated by FFT indicates the magnitudes and phases of the harmonics contained in the signal. The values of the spectrum are arranged by frequency, with the first value belonging to the frequency 0 and indicating the mean value of the signal.

Note that the magnitudes of the harmonics cannot be interpreted directly as amplitudes; if this is desired, use the Spec() function.

Please note that a continual spectrum is not calculated, only a discrete spectrum. This means that the spectrum is only calculated for certain discrete frequencies: those which have and integer number of periods within the calculation interval. For example, to calculate the FFT of a data set with a length of 0.5 s , the spectrum would be determined at the frequencies 0, 2 Hz, 4 Hz, 6 Hz, 8 Hz ....

An important principle in this context is that a discrete spectrum is the spectrum of a periodic signal. In calculating the spectrum, the signal is interpreted as if it consisted of a long chain of identical signals. If the actual signal represents only one pulse, the spectrum is calculated not for this pulse, but for a signal containing a series of pulse identical to it. This periodic concatenation is noticeable whenever a non-integer number of periods are contained in a signal.

For example, a data set contains 3.5 periods of a sinusoidal signal. One harmonic is expected as the spectrum; however, since the signal is extended periodically for the spectrum calculation, the signal no longer appears as a sine wave, rather a series of sine-shaped sections with the appearance of having been put together incorrectly. The spectrum of this signal will include many harmonics, which are particularly strong in the vicinity of the actual frequency. This apparent disadvantage of the FFT function can be avoided in two ways. The first option is to specify a section of the signal which contains an integer number of periods before calculation of the spectrum, a standard task for periodic signals. The other option is to use a window function. These methods can also be combined.

A window function assigns different significance to different values of a data set. The edges of the signal are especially weak, the center values influence the result very strongly.

Six different window functions are available:

- Rectangle
- Hanning
- Hamming
- Blackman
- Blackman-Harris.
- Flat Top

The rectangular window treats all values equally, representing the standard setting for which no extra calculation is required. The Flat Top-window is the opposite extreme. The "windowing" effect increases from the top to the bottom of this list. The effect of the window function is demonstrated using a 3.5-period sinusoidal signal.

The spectrum exhibits additional harmonics around the frequency, whose amplitudes decrease with increasing distance from the actual frequency. The rectangular window shows a noticeable, sharp peak around 3 and 4, but even at greater distances a distorting influence on the spectrum is present. Using the other windows will increase the width of the peak, but the greater the distance from the actual frequency, the less distortion is apparent. Each window function represent a compromise between the two extremes.

To differentiate between harmonics located very closely, the rectangular window is appropriate; to ascertain the exact amplitudes and phases of harmonics located further away from each other, use the Blackman window.

- Note that, in contrast to the Spec() function(), the middle harmonics are not multiplied by a factor of 2, and therefore cannot be interpreted as amplitudes.
- If direct interpretation of the amplitudes of the spectrum is important for the specific application, use the Spec() function. The FFT() function should be used to calculate the spectrum if additional operations, such as filtering or convolution are to be performed in the frequency range.
- The **Spectral package** (included in the Professional/Enterprise edition) contains additional functions which are substantially more powerful and closely adapted to the requirements of signal analysis, for the calculation of spectra. Example: **AmpSpectrumRMS()** calculates an amplitude spectrum (harmonics as RMS-value), with moving windows and linear averaging.
- The optional parameters for the window function and mode are suported as of FAMOS 2022. If these parameters are omitted, then as with earlier versions the global settings set by means of the dialog: 'Options'/'Functions', by means of the command FFTOPTION or by means of the function SetOption() are in force.
- If the length of the data set specified as a parameter exceeds 2^27, an appropriate warning message is generated and calculation is not possible. Use the functions Leng() or Red2() to shorten the data set. The maximum amount of points which can be processed is 134.217.728; resulting in a complex data set with 67.108.865 points.
- FAMOS implements an FFT which returns only the positive side of the spectrum. For the real data set used here, the spectrum is always conjugated symmetrically, so that the other half does not contain any additional information. The representation of negative frequencies or of frequencies greater than half the sampling rate are meaningless.
- If the data set provided has N points and if N is even, then the spectrum generated has N/2 + 1 points. Since the first and last value (center value and highest harmonic) are always only real while the other spectral lines are complex, it follows that the spectrum has exactly as many significant digits as the data set contains, so that no information has been lost. If N is odd (only possible with Mixed-Radix procedure), then the spectrum generated has (N-1)/2 + 1 points. The highest harmonic is then complex and not multiplied by a factor of 2.
- The **Mixed-Radix-procedure** has the advantage of returning spectra with "round" frequency line distances from data sets having common sampling frequencies (e.g. 1kHz and multiples in the pattern1/2/5) and an appropriate selection of the input length. For a data set sampled at 1kHz, for instance, and a length of 1000 samples (1000 = 2^3 * 5^3), the procedure returns a spectrum having a frequency line distance of 1Hz; for a length of 20000 samples (= 2^5 * 5^4), the frequency line distance is 50mHz.
- The data set can be reconstructed from the spectrum if a rectangular window was used for spectrum calculation
- The x-unit of the spectrum is always Hz; the y-unit of the magnitude corresponds to that data set. The phase is given in degrees.
- The x-offset of the specified data set should be zero. If it is not, a warning message is generated and the x-offset is treated as zero. To include the influence of an x-offset in the phase, adjust the phase after the spectrum calculation, correcting it with a linear function.

Examples:

```
MPfft = FFT(NWdata, 1, 0)
```

Simple application for calculating the spectrum of a data set whose length is a power of 2, and in which direct interpretability of the amplitudes is not important (Hamming window).

```
DPTransfer=dB(FFT(NWout, 0, 2)/FFT(NWin, 0, 2))
; is equivalent to:
DPTransfer=dB(FFT(Red2(NWout), 0, 0)/FFT(Red2(NWin), 0, 0))
```

Calculation with a transfer function in dB of two data sets with equal length, whose length is not a power of two.

```
First1000Samples = CutIndex(data1kHz, 1, 1000)
FFT_with_1Hz_distance = FFT(First1000Samples, 0, 3)
```

The FFT of the first 1000 samples of a data set sampled at 1kHz is calculated. The result has 501 spectral llines at the frequencies 0, 1, 2, ... 500 Hz.

```
; Pay attention when the input length is odd:
First125amples = CutIndex(data1kHz, 1, 125)
FFT_with_8Hz_distance = FFT(First125Samples, 0, 3)
```

The FFT of the first 125 samples of a data set sampled at 1 kHz is calculated. The result has 63 spectral lines at frequencies 0, 8, 16, ... 496 Hz. The last spectral line is complex and does not lie at half the sampling frequency.

```
NDwindow = iFFT(FFT(Ramp(0, 1, 256) * 0 + 1, 2, 0))
```

A data set whose value is constant at value = 1 is generated. The currently set window function weights this data set when calculating the FFT, the iFFT then shows the window function (here: Hanning).

See also:

AmpSpectrumRMS_1, AmpSpectrumRMS, Spec, iFFT, Red2, dB, ACF

Supported since:

Optional parameters and Mixed-Radix-procedure are supported as of Version 2022.

FFTOPTION

Set FFT-settings

Declaration:

```
FFTOPTION SvWindow SvMode
```

Parameter:

SvWindow	Window function
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman/Harris
	5 : Flat-Top
SvMode	Mode
	0 : Reduce waveform length to next power of two
	1 : Increase waveform length to next power of two and fill with zeros

Description

The FFTOPTION command is obsolete; instead, the corresponding optional parameters of the FFT() and Spec() functions or the SetOption() function should be used in newly created sequences.

This command is essentially an automation of the corresponding settings in the dialog box under 'Extra'/'Options'/'Functions'.

These settings affect the functions FFT(), Spec(), CCF() and ACF().

Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
FFTOPTION 4 1
FFTresult = FFT(data)
```

A Blackman/Harris window function was selected. The waveform is extended to the next power of two and filled with zeros.

See also:

FFT, Spec, SetOption

FileClose

Closes an opened file

Declaration:

```
FileClose ( SvFileID ) -> SvStatus
```

Parameter:

SvFileID	ID of the open file, which was returned with a FileOpen*() function when the file is opened
SvStatus	
SvStatus	Success of the function
	0 : Function executed successfully
	-1 : Error in executing the function

Description:

The function closes a file which was opened with one of the functions FileOpenDSF(), FileOpenFAS(), FileOpenASCII(), FileOpenASCII2(), FileOpenXLS() or FileOpenXLS2(). If the file was opened for writing, if it was changed, the file will actually be written here.

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of the error can be inquired using the functions FileErrText?() or FileErrCode?().
- Querying the return value is optional in imc FAMOS, however, this value should be evaluated especially when writing to files.

Examples:

A file "xxx.dat" is opened to write in imc FAMOS format; two variables are added to this file and stored by calling the function FileClose.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 1)
IF idFile > 0
   data = Ramp(0, 1, 100)
   sinData = sin(data)
   status= FileObjWrite(idFile, data)
   status= FileObjWrite(idFile, sinData)
   status= FileClose(idFile)
END
```

See also:

FileOpenDSF, FileOpenFAS, FileOpenASCII, FileOpenXLS, FileResetAll

FileComm?

Gets the comment on a file in the imc/FAMOS-format

Declaration:

```
FileComm? ( SvFileID ) -> TxComment
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF()
TxComment	
TxComment	File comment; empty text if a comment is not present of if error occurs.

Description:

The function inquires the file comment of an opened imc FAMOS file. The file must have been opened using the function FileOpenDSF.

- How the function responds to an error depends on the global presetting for "No error messages for file functions", which is set in the dialog called by "Options / Functions", or using the function SetOption. If this option is not selected, an output box containing the error message is displayed and the return text is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, an empty text is returned and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.

Examples:

A imc FAMOS file "xxx.dat" is opened in imc FAMOS format for reading and writing. An inquiry of the comment is made and if one is not present, a comment is entered and the file is closed.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 2)
txComment = FileComm?(idFile)
IF TLeng(txComment) = 0
   FileSetComm(idFile, "Checked")
END
FileClose(idFile)
```

See also:

FileOpenDSF, FileSetComm

FileErrCode?

Get error code of the last file function executed

Declaration:

```
FileErrCode? ( ) -> SwErrorCode
```

Parameter:

SwErrorCode	
SwErrorCode	Last error code

Description:

The function provides the error code of the last executed function in the group of File*()-functions.

General error codes:

0	The function was successful.
1	Insufficient memory
2	Too many files are open.
3	The idientifier is invalid
4	No opened file exists for this identifier
5	Canceled by user
6	Invalid object-index.
7	Attempt to change a file opened in "Read only" mode
8	Function allowed only for DSF files.
9	Invalid option when opening the file.
10	Internal error
11	Unable to open file
12	This function cannot be applied to ASCII-files.
13	This function can only be applied to ASCII-files.
14	Error in writing to file (data carrier full?)
15	Error in reading the file
16	Invalid attempt to read from file opened in "write"-mode.
17	The library "im22form.dll" either isn't present or cannot be opened.
18	This function can only be applied to XLS-files.

The following error messages refer to files in the imc FAMOS format, which were opened using FileOpenDSF(..):

21	Insufficient temporary memory
22	Writing error; drive full
23	Unable to create file
24	Too many large objects in file; file size no longer manageable.
25,26	Error when writing temporary file
27	File is write-protected
28	Unable to open file
29..69	Format error in the file

The following error messages refer to user-defined file format, which were opened using FileOpenFAS(..):

71	Canceled by user
72	Insufficient memory
73	Error when opening format definition file

74	Error in opening measurement value file
75	Unable to fine Assistant-DLL
76	Incorrect version of definitions file
77	Error in reading the format definitions file
78	Error in reading the measurement value file

The error messages below pertain to files previously opened by FileOpenXLS or FileOpenXLS2.

121	Version conflict with library "im22form.dll". The file is too old.
122	Version conflict with library "im22form.dll". The file is too new.
123	Implementation problem
124	Unknown internal error
136	At least one of the data sets to be saved is of an invalid file type.
137	Insufficient memory
138	Data could not be saved. All data sets transferred are of invalid type (e.g. Text) or of length 0.
139	Data of type "Time stamp-ASCII" can only be saved individually and not together with other data sets in the file. Also, they may not contain events or segments.
140	For the "Common scaling column" mode, all data sets must have a monotonous x- or time track.
152	The specified template file contains no valid templates.
153	The specified template file is of an invalid or non-matching version.
154	The specified template file is of the wrong type.
168	The specified file cannot be opened. Please check the filename.
169	Error in writing to output file. The data carrier may be full.
184	Error in activating EXCEL
186	The scaling column is empty or contains invalid values.
187	No readable values at the specified table position
188	Unable to start Excel.

Examples:

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   status = FileObjWrite(idFile, data)
   IF status = -1
      BoxOutput("Error writing file: " + FileErrText?(), FileErrCode?(), "", 0)
   END
END
```

If the variable cannot be written to the file, an output box containing the error text and error code is generated.

See also:

FileErrText?

FileErrText?

Get error text of the last file function executed

Declaration:

```
FileErrText? ( ) -> TxErrorText
```

Parameter:

TxErrorText	
TxErrorText	Last error text

Description:

Error text of the last executed function in the file function group (File*(..))

Examples:

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   status = FileObjWrite(idFile, data)
   IF status = -1
      BoxOutput("Error writing file: " + FileErrText?(), FileErrCode?(), "", 0)
   END
END
```

If the variable cannot be written to the file, an output box containing the error text and error code is generated.

See also:

FileErrCode?

FileLineRead

Read the next row(s) from an opened ASCII-file

Declaration:

```
FileLineRead ( SvFileID, TxLine, Zero ) -> SvStatus
```

Parameter:

SvFileID	ID of the open file, which was returned by the function FileOpenASCII when the ASCII-file was opened.
TxLine	Text variable or text array for containing the text read
Zero	Reserved parameter. Always 0.
SvStatus	
SvStatus	Success of the function (optional).
	0 : Function executed successfully
	1 : End of file reached
	-1 : An error occurred

Description:

This function reads the next row(s) from an opened ASCII-file. The file needs to have been previously opened using the function FileOpenASCII().

The combination of characters "Carriage Return/ Linefeed" (ASCII-characters 13 and 10), or only "line Feed", is identified as the end of the line of text. The read text does not contain these characters.

The variable passed to the function as the 2nd parameter must already exist when the function is called. This can be accomplished by means of a simple assignment, for example:

```
TxLine = ""
```

or for text arrays:

```
TxAllLines= TxArrayCreate(0)
```

If a text array is passed to the function as the 2nd parameter, all remaining rows up to the end of the file are read and these are appended to the text array as new elements.

- This function is useful for reading in log files or short numerical series in ASCII-format (for subsequent conversion of text to numbers). For reasons of speed, this command is generally not well suited to reading long files of measurement values.
- For reading such files, it is recommended to use the ASCII-Import-Assistant, or to create a file filter using the File Assistant.
- In case of an error (SvStatus=-1), the cause can be inquired through the use of the functions FileErrText? or FileErrCode?.

Examples:

From a multi-column text file with numerical values (columns separated by a comma), the first column is to be imported:

```
idFile = FileOpenASCII("z:\000.txt", 0)
TxLine = ""
channel1 = EMPTY ; one blank channel
WHILE 1
   isEOF = FileLineRead(idFile, TxLine, 0) ; read one text line from file
   IF isEOF = 1
      BREAK  ; End Of File
   END
   TxLineSplit = TxSplit(TxLine, ",") ; split comma separated values from line
   TxChannel1 = TxLineSplit[1] ; get 1. separated value for channel 1
   SvChannel1 = TtoSV(TxChannel1, "f") ; convert text to float value
   AppendLoop(channel1, SvChannel1) ; attach value to value
END
AppendLoopEnd(channel1)
FileClose(idFile) ; close file processing
```

Remarks: Import tasks of this type can often be accomplished more conveniently using the ASCII-Import-Assistant.

The same task can be solved with a text array in a much more elegant and quick way:

```
idFile = FileOpenASCII("z:\000.txt", 0)
TxaLines = TxArrayCreate(0)
ret = FileLineRead(idFile, TxaLines, 0) ; read all text lines from file
IF ret = 0
   TxaLines = TxRegexMatch(TxaLines, "^[^,]+", " ", 0, 0) ; split 1st column (comma separated) from line
   ; To get the 3rd column, you could use the following line (note the '{2}' for the the zero based column index):
   ; TxaLines = TxRegexMatch(TxaLines, "^(?:[^,]+,){2}([^,]+)", " ", 0, 1)
   channel = TxArrayToChannel(TxaLines, 0);
END
FileClose(idFile) ; close file processing
```

Events are recorded in a report file line by line according to the following pattern: "12.2.95 13:00:42 Alarm # 123 File::c12_34.dat". These records are displayed consecutively. The user can opt to load and display the data set indicated.

```
idFile = FileOpenASCII("c:\dat\prot.dat", 0)
txRow = ""
ok = FileLineRead(idFile, txRow, 0)
WHILE ok = 0
   yes = BoxMessage("Display?", txRow, "?4")
   IF yes
      txFile= TPart(txRow, TxWhere(txRow, "::")+2, 100)
      FileLoad(txFile + ".dat, "", 0)
      SHOW <txFile>
   END
   ok = FileLineRead(idFile, txRow, 0)
END
FileClose(idFile)
```

See also:

FileOpenASCII, FileOpenASCII2, FileLineWrite, FileClose

FileLineWrite

Appends row(s) to an opened ASCII-file

Declaration:

```
FileLineWrite ( SvFileID, TxLines, SvOption ) -> SvStatus
```

Parameter:

SvFileID	ID of the open file, which was returned by the function FileOpenASCII when the ASCII-file was opened.
TxLines	Text to write. The data types allowed are Text and Text Array.
SvOption	Options
	0 : Append line break
	1 : No line break
SvStatus	
SvStatus	Success of the function
	0 : Function executed successfully
	-1 : An error occurred

Description:

This function appends one row (if a text is passed as the 2nd parameter) or multiple rows (if a text array is passed as the 2nd parameter) to an opened ASCII-file. The file needs to have previously been opened for writing using the function FileOpenASCII().

A 'CarriageReturn'- (ASCII 13) and a 'LineFeed'-character (ASCII 10) are (optionally) appended to the row of text in order to ensure a line return appears after the text written.

If a text array is specified, each element is written as a new row. The Options parameter is then ignored.

The text is first temporarily stored. The actual writing to the file takes place upon calling of FileClose().

- This function is useful for writing log files or short numerical series in ASCII-format (for subsequent conversion of text to numbers). For reasons of speed, this function is not well suited to writing long files of measurement values in ASCII format. For such purposes, you can use the function FileOpenASCII2().
- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- Querying and evaluating the return value is not absolutely necessary in imc FAMOS, but recommended.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.

Examples:

In a complex evaluation procedure, error and status messages are recorded together with the current time, e.g.:. "12.10.23 13:00:02 : Measurement xyz violates tolerances".

```
idFile = FileOpenASCII("c:\dat\prot.dat", 2)
;; ... Evaluation ...
;; ...
IF TLeng(TxStatus) > 0
   txTime = TimeToText(TimeSystem?(), 0)
   txRow = txTime + TxStatus
   ret= FileLineWrite(idFile, txRow, 0)
END
;; ... more FileLineWrite ...
;; ...
ret= FileClose(idFile)
```

A text-file with multiple columns, separated by commas, is read. The 3rd column is extracted and saved in a new file.

```
; Read file with at least 3 comma separated columns
idFile = FileOpenASCII("z:\000.txt", 0)
TxaLines = TxArrayCreate(0)
FileLineRead(idFile, TxaLines, 0) ; read all text lines from file
TxaLines = TxRegExMatch(TxaLines, "^(?:[^,]+,){2}([^,]+)", " ", 0, 1) ; split 3rd column (comma separated) from line
FileClose(idFile) ;
; Write new file with 1 column
idFile = FileOpenASCII("z:\000_col3.txt",1)
FileLineWrite(idFile, TxaLines, 0) ; write all text lines to file
FileClose(idFile) ;
```

See also:

FileOpenASCII, FileOpenASCII2, FileLineRead, FileClose

FileLoad

Loads the file of measured data in the specified format and enters all data objects it contains in FAMOS' Variables list.

Declaration:

```
FileLoad ( TxFile, TxFormat, SvOptions ) -> SvSuccess
```

Parameter:

TxFile	Name of the file. Unless a full pathname is provided, the pre-set loading folder is used.
TxFormat	Specification of the file format
SvOptions	Options
	0 : Default behavior
	1 : Any measurement association determined by the import filter will be ignored.
SvSuccess	
SvSuccess	Success of the function
	-1 : Error loading the file. The error cause can be inquired using the function GetLastError().
	>=0 : Amount of data objects to load (optional).

Description:

The content of the specified file(s) is loaded completely and the data objects created are adopted in FAMOS' Variables list. The function does not return any information on whether variables (under which name) have been generated and it is thus suitable for applications in which the structure of the files to be loaded, and thus the results of the loading process, are predictable.

For files having unknown or variable content, or if you wish to load only selected channels from a multi-channel file, it is better to use the powerful functions FileOpenDSF() or FileOpenFAS().

It is possible to place wildcard characters in the filename in order to open multiple files. The wildcard character '?' stands for exactly one arbitrary character. The wildcard character '*' stands for a non-specific amount of arbitrary characters.

Tip: Use the Function Assistant to parameterize the function. By this means, you are easily able to select any kind of import format from a list.

For specifying the format, multiple different ways are possible:

Open in the default imc/FAMOS-format
For <TxFormat>, either an empty text or the identifier "imc/FAMOS" must be entered.

if the file name supplied has no extension, then ".dat" is appended automatically.

You can also specify that the option 'Quick Load' is to be used; in this case, use the format identifier "imc/FAMOS|Quick". The effect of this option is that the file is not copied when loaded, but is handled internally as a reference to the original file. This can accelerate the loading of very large data sets, but is only sensible to use of you do not wish to alter the data sets later.

Loads a text variable from a text file
In order to transfer the content of a text file to a text variable, enter the format identifier "imc/Text".

If the filename supplied has no extension, ".txt" is appended automatically. The text variable created has the same name as the file.

You can also specify the character set encoding (Code page) of the file. The current Windows code page is standard.

"imc/Text"	The current Windows ANSI code page is assumed.
"imc/Text/auto"	The coding is determined automatically. For this purpose, it is checked whether the file begins with a defined BOM (byte order mark). The BOM for UTF-8 and UTF-16 ("Big endian" and "Little endian" byte order) are recognized. If no such BOM is recognized, the current Windows code page is assumed.
"imc/Text/UTF8"	UTF-8-Encoding
"imc/Text/UTF16_LE"	UTF-16 LE (Byte order "Little endian", least significant Byte first)
"imc/Text/UTF16_BE"	UTF-16 BE (Byte order "Big endian", most significant Byte first)

Import with format description file (*.FAS, imc File Assistant).
The format is determined by an import description file (Importfilter), which had been created using the imc File Assistant. Such an import filter file typically takes the extension ".fas" and must be located in the pre-set definitions folder ("Extras/Options/Folders"). This file's name is then passed as a parameter, and the extension ".fas" can be omitted.

Import with an appropriate extension library (*.DLL).
The syntax for <TxFormat> is then:

```
"#DLLName|FormatName|Parameter"
```

- DLLName: specifies the filename of the extension library; the extension ".DLL" can be omitted.

- FormatName: specifies the name of the desired format (an extension library may certainly support multiple formats). Can be omitted if the DLL only provides one single format and does not require any additional parameters.

- Parameters: serves for the optional transfer of further parameters to the library. Whether parameters are required and their syntax depends on the respective extension library. Often not required, can then be omitted. If necessary, look for the name of the format used in the help, where the format-specific use of the parameter is described.

The FAMOS product package contains a number of extension libraries for the import of common file formats (such as MatLab).

The ASCII-Import-Assistant built into FAMOS is also an extension library (ImportAscii1.dll). Any filters creating with it can also be used here (see example).

Import with a derivative import filter
Import is performed by a derivative import filter. The parameter's syntax <TxFormat> is then as follows:

```
"$FormatName"
```

[FormatName] matches the name assigned by the user, such as it is displayed in the FAMOS dialog "Options/Import filter", for example.

Special format "ByteBlob": Uninterpreted binary data
The the complete file content is written to a data set in a binary and uninterpreted format. For this, specify the format ID "imc/ByteBlob".

The resulting data format is '1 Byte unsigned'. Normally required only for special applications, e.g. in conjunction with the FAMOS-Database Kit for the purpose of being able to save and retrieve any file types (e.g. PDF, images, videos) to and from databases jointly with the actual measurement data.

- Any existing variables with the same name can be overwritten without warning.
- The function's behavior at fault condition depends on the global presetting 'No error boxes with file functions' (which is set in the dialog "Options/Functions" or by means of the function SetOption). If this option is deactivated, an error message box is displayed and the return value is not generated. This represents the default behavior of imc FAMOS functions. On the other hand, if the option is activated, a 0 is returned and no error message is displayed. If needed, the cause of the error can be queried using the function GetLastError().

Examples:

Opens a file in the FAMOS format:

```
FileLoad("test","", 0)
```

..or with 'Quickload'-option:

```
FileLoad("test","imc/Famos|Quick", 0)
```

Loads a text file and assignment of the content as a user-defined property to a data set:

```
FileLoad("mycomment.txt","imc/Text", 0)
UserPropSet(Channel1, "UserComment", mycomment, 1, 0)
```

Opens a MatLab file (with extension DLL)

```
FileLoad("1.mat","#MatLabImportExport", 0)
```

Import filters defined using the ASCII Import Assistant can also be used.

```
FileLoad("1.txt","#ImportAscii1.dll|FilterName", 0)
```

[FilterName] corresponds to the name under which the filter had been saved in the ASCII Import-dialog, and is identical with the format name displayed in the "Load file"-dialog.

Opens a file in LeCroy format (with File Assistant, 'lecroy.fas')

```
FileLoad("1.lec","lecroy", 0)
```

Opens an EXCEL file with a derivative import filter which had been created on the basis of the EXCEL format. The private settings are set for the cells B2 through C300 to be read. The import filter was saved under the name 'EXCEL (B2-C300)'.

```
FileLoad("c:\test\result.xls","$EXCEL (B2-C300)", 0)
```

See also:

FileSave, FileOpenDSF, FileOpenFAS

FileName?

Determines the name of the file from which a data object is to be loaded.

Declaration:

`FileName? ( Dataobject ) -> TxFilename`

Parameter:

Dataobject	Data object whose origin is to be identified
TxFilename	
TxFilename	Filename (complete path)

Description:

If the data object was generated in imc FAMOS by loading a file, the file's complete pathname is returned.

If the data object was not generated in imc FAMOS by loading a file, an empty text is returned.

Examples:

```
FileLoad("c:\imc\dat\slope.dat", "", 0)
RENAME slope TestData
TxFileName = FileName?(TestData)
```

TxFileName contains the text "c:\imc\dat\slope.dat".

See also:

Comm?

FileObjDel

The function removes an object from the table of contents of an open file.

Declaration:

```
FileObjDel ( SvFileID, SvObjectIndex ) -> SvStatus
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF()
SvObjectIndex	Index of object to delete. The first object takes the index 1.
SvStatus	
SvStatus	Success of the function
	0 : Function executed successfully
	-1 : Error in executing the function

Description:

This function deletes the specified object from the list of file contents. The altered file is written physically only when FileClose() is called.

The file must previously have been opened for writing (Mode 2 = "Append/ Edit") by means of the function FileOpenDSF().

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of the error can be inquired using the functions FileErrText?() or FileErrCode?().
- Querying and evaluating the return value is not absolutely necessary in imc FAMOS, but recommended.

Examples:

The file "test.dat" in imc FAMOS-format is opened. All channels with a maximum value over 20 are deleted. Note that the channels are counted backwards. This is simpler because deleting a channel would change the indices of all subsequent channels in the list.

```
idFile = FileOpenDSF("c:\dat\test.dat", 2)
IF idFile >= 1
   num = FileObjNum?(idFile)
   index = num
   WHILE index >= 1
      data = FileObjRead(idFile, index)
      IF max(data) > 20
         err = FileObjDel(idFile, index)
      END
      index = index-1
   END
   err = FileClose(idFile)
END
```

See also:

FileOpenDSF, FileObjWrite, FileObjRead

FileObjFind

Searches for a data object with a specified name in an opened file.

Declaration:

```
FileObjFind ( SvFileID, TxSearchName, SvStartIndex ) -> SvIndex
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF or FileOpenFAS
TxSearchName	Name of the object to find
SvStartIndex	Starting location of the search; 1= from the beginning
SvIndex	
SvIndex	Index of the object when successful
	>0 : Found object's index
	0 : No object with the search name was found
	-1 : Error in executing the function

Description:

The function searches for a data object with the specified name in the content list of an opened file. If the object is found, the index is returned; 0 means that the object could not be found.

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.
- The index of the found object is valid only until the next time the file is accessed for writing. For example, if a data set is deleted before the specified data set is found, this index would then indicate the following object in the file.

Examples:

The program searches for a data set named "Channel1" in the "test.dat" file in the imc/FAMOS format. If this data set is found, it is loaded and displayed

```
idFile = FileOpenDSF("c:\dat\test.dat", 0)
IF idFile >= 1
   index = FileObjFind(idFile, "channel1", 1)
   IF index > 0
      data = FileObjRead(idFile, index)
      SHOW data
   END
END
```

See also:

FileOpenDSF, FileOpenFAS, FileObjRead

FileObjName?

Determines the name of a data object in an opened file.

Declaration:

```
FileObjName? ( SvFileID, SvObjectIndex ) -> TxName
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF or FileOpenFAS
SvObjectIndex	Index of the object to be checked. The first object has the index 1.
TxName	
TxName	Name of the data object. Empty text at fault condition.

Description:

The function returns the name of a data object in a file opened using FileOpenFAS or FileOpenDSF.

The index must lie between 1 and the total number of objects present.

- How the function responds to an error depends on the global presetting for "No error messages for file functions", which is set in the dialog called by "Options / Functions", or using the function SetOption. If this option is not selected, an output box containing the error message is displayed and the return text is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, an empty text is returned and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.

Examples:

A file is opened in imc FAMOS format. By means of a loop through all data objects in the file, a list of all names is outputted in the output box of the imc FAMOS main windo:

```
idFile = FileOpenDSF("c:\dat\test1.dat", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      TxName = FileObjName?(idFile, index)
      BoxOutput(TxName, EMPTY, "", 1)
      index = index +1
   END
   ret=FileClose(idFile)
END
```

See also:

FileOpenDSF, FileOpenFAS, FileObjType?, FileObjFind, FileObjRead

FileObjNum?

The number of data objects in an opened file is determined.

Declaration:

```
FileObjNum? ( SvFileID ) -> SvCount
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF or FileOpenFAS
SvCount	
SvCount	Number of data objects (data sets, data groups, texts) in the file. -1 in case of error (see remarks)

Description:

The function returns the number of objects (groups, texts, data sets) in an opened file.

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.

Examples:

A file is opened for reading in imc FAMOS format. All data sets are read in imc FAMOS in a loop under their original names. Texts and data groups are ignored.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      type = FileObjType?(idFile, index)
      IF type = 1
         TxName = FileObjName?(idFile, index)
          <TxName> = FileObjRead(idFile, index)
      END
      index = index + 1
   END
   ret = FileClose(idFile)
END
```

See also:

FileOpenDSF, FileOpenFAS, FileObjRead, FileObjName?

FileObjRead

A data object is read from an opened file.

Declaration:

```
FileObjRead ( SvFileID, SvObjectIndex ) -> Variable
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF or FileOpenFAS
SvObjectIndex	Index of the object to be read. The first object has the index 1.
Variable	
Variable	The data set, text or data group read in the file

Description:

The function reads an object from an opened file and assigns a target variable to the object. In case of error, the function is not executed and the existing variable is not changed.

Examples:

A file is opened for reading in imc FAMOS format. All data sets are read in imc FAMOS in a loop under their original names. Texts and data groups are ignored.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      type = FileObjType?(idFile, index)
      IF type = 1
         TxName = FileObjName?(idFile, index)
          <TxName> = FileObjRead(idFile, index)
      END
      index = index + 1
   END
   ret = FileClose(idFile)
END
```

See also:

FileOpenDSF, FileOpenFAS, FileObjNum?, FileObjWrite

FileObjType?

Queries type of data object in an open file.

Declaration:

```
FileObjType? ( SvFileID, SvObjectIndex ) -> SvType
```

Parameter:

SvFileID	ID of opened file, which was returned when file is opened using function FileOpenDSF or FileOpenFAS
SvObjectIndex	Index of the object to be checked. The first object has the index 1.
SvType	
SvType	The data object's type
	0 : Data group
	1 : Data set
	2 : Text
	-1 : Error in executing the function

Description:

The function yields the type of an object in an opened file, differentiating between data set, text and data group.

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.

Examples:

A file is opened in imc FAMOS format. All texts in the file are loaded into imc FAMOS in a loop under their original names. Data sets and data groups in the file are skipped.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      type = FileObjType?(idFile, index)
      IF type = 2
         TxName = FileObjName?(idFile, index)
         <TxName>= FileObjRead(idFile, index)
      END
      index = index + 1
   END
   ret=FileClose(idFile)
END
```

See also:

FileOpenDSF, FileOpenFAS, FileObjName?, FileObjFind, FileObjRead

FileObjWrite

A data object is written to an opened imc FAMOS file.

Declaration:

```
FileObjWrite ( SvFileID, Variable ) -> SvStatus
```

Parameter:

SvFileID	ID of the opened file, which was determined when opened using one of the functions FileOpenDSF, FileOpenFas, FileOpenASCII2, or FileOpenXLS2.
Variable	Data set, text or data group to be saved
SvStatus	
SvStatus	Success of the function
	0 : Function executed successfully
	-1 : Error in executing the function

Description:

The function adds a variable to the content list of an open file. The file must have been opened using the function FileOpenDSF to write (option 1 or 2) or with one of the functions FileOpenASCII2 or FileOpenXLS2.

A copy of the transferred variables is added to the contents; the content list is actually saved the first time FileClose is called.

When saving a single element from a group, the group information is lost.

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of the error can be inquired using the functions FileErrText?() or FileErrCode?().
- Querying and evaluating the return value is not absolutely necessary in imc FAMOS, but is recommended.

Examples:

A file "xxx.dat" is opened to write in imc FAMOS format; two variables are added to this file and stored by calling the function FileClose.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 1)
IF idFile > 0
   data = Ramp(0, 1, 100)
   sinData = sin(data)
   status= FileObjWrite(idFile, data)
   status= FileObjWrite(idFile, sinData)
   status= FileClose(idFile)
END
```

See also:

FileOpenDSF, FileOpenFAS, FileOpenASCII2, FileOpenXLS2, FileLineWrite

FileOpenASCII

Opens a file in ASCII-format for reading or writing line by line.

Declaration:

```
FileOpenASCII ( TxFile, SvMode [, SvCodePage] ) -> SvFileID
```

Parameter:

TxFile	Name of the file. Unless a full pathname is provided, the pre-set loading folder is used.
SvMode	Mode
	0 : Opens file for reading
	1 : Opens the file for writing. Any existing file of the same name will be overwritten.
	2 : Opens file for writing. If the file already exists, the new content is appended.
SvCodePage	If the file is to be opened for reading with option 0, the character set encoding (Code page) of the file can be specified here. The current Windows code page is standard. (optional , Default value: 0)
	0 : Current Windows ANSI code page
	1 : The encoding is determined automatically. For this purpose, it is checked whether the file begins with a defined BOM (byte order mark). The BOM for UTF-8 and UTF-16 ("Big endian" and "Little endian" byte order) are recognized. If no such BOM is recognized, the current Windows code page is assumed.
	2 : UTF-8-Encoding
	3 : UTF-16 with "little endian" byte order (least significant byte first).
	4 : UTF-16 with "Big Endian" byte order (most significant byte first).
SvFileID	
SvFileID	ID of the opened file, or error code
	0 : Error in opening the file. The cause of error can be inquired using the functions FileErrText? or FileErrCode?.
	>0 : Identifier of the opened file. It is passed as the parameter for all subsequent file editing functions.

Description:

A file in ASCII-format is opened. Subsequently, the functions FileLineRead() and FileLineWrite() can be used for line-by-line to read/write to the file.

If no complete filename is entered, the system searches the current loading folder. The current loading folder is set according to the default defined under "Extra/Options/Folders" when imc FAMOS is started. The default folder can be changed using either the command LDIR or the function SetOption. Otherwise the folder last specified when a file was loaded is used.

The identifier of the opened file must be specified with a file function each time the file is accessed. This identifier is valid until the function FileClose() is called.

Every file opened with FileOpenASCII2 must be closed again afterwards by calling FileClose.

imc FAMOS can administer a maximum of 10 open files at once. This amount can easily be reached by neglecting to call FileClose. An appropriate error message will appear in such a case.

The function FileResetAll closes all files presently open. This can be useful if files whose identifiers are no longer known remain open while testing imc FAMOS sequences in single-step mode.

- This function, in conjunction with FileLineRead and FileLineWrite, is designed for reading and writing log files or short numerical series in ASCII-format (previously converted from numbers to text or vice-versa). For long measurement files in ASCII-format it can not be recommended, for reasons of speed.
- For writing such files, you should use the function FileOpenASCII2().
- For reading such files, it is recommended to use the ASCII-Import-Assistant, or to create a file filter using the File Assistant.
- How the function responds to an error depends on the global presetting for "No error messages for file functions", which is set in the dialog called by "Options / Functions", or using the function SetOption. If this option is not selected, an output box containing the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, the value 0 is returned and an error message is not displayed. If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.
- Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

In a complex evaluation procedure, error and status messages are recorded together with the current time, e.g.:. "12.10.23 13:00:02 :

Measurement xyz violates tolerances".

```
idFile = FileOpenASCII("c:\dat\prot.dat", 2)
;; ... Evaluation ...
;; ...
IF TLeng(txStatus) > 0
   txTime = TimeToText(TimeSystem?(), 0)
   txRow = txTime + txStatus
   ret= FileLineWrite(idFile, txRow, 0)
END
;; ... more FileLineWrite ...
;; ...
ret = FileClose(idFile)
```

See also:

FileOpenASCII2, FileLineRead, FileLineWrite, FileClose, FileOpenXLS, FileOpenXLS2

FileOpenASCII2

An ASCII file for writing data sets in columns is opened. The format specificaton is performed by means of a pre-defined ASCII export template.

Declaration:

```
FileOpenASCII2 ( TxFile, TxExportTemplate, SvMode ) -> SvFileID
```

Parameter:

TxFile	Name of the ASCII file. Unless a complete pathname is specified, the default folder for opening files is used.
TxExportTemplate	Name of the ASCII export template to use
SvMode	Mode
	1 : Opens the file for writing. Any existing file of the same name will be overwritten.
	2 : Opens file for writing. If the file already exists, the new content is appended.
SvFileID	
SvFileID	ID of the opened file, or error code
	0 : Error opening the file. The cause of the error can be queried with the functions FileErrCode?() or FileErrText?().
	>0 : Identifier for the opened file. This is provided as a parameter to all subsequent functions for editing this file.

Description:

This function enables writing of ASCII files, in which data sets are written in columns.

Once a file has been opened with this function, the data sets to save can be added using the function FileObjWrite(..).

Every file opened with FileOpenAscii2() must be closed by calling FileClose(). Only then is the file physically written to the data medium.

FileObjWrite() and FileClose() are the only functions which can be applied to files opened using FileOpenAscii2(). Additional headers and footers can be added by preceding or subsequent calls of FileOpenASCII()/FileClose() in combination with FileLineWrite().

Concrete definition of the file format (e.g. column headers, scaling columns, numerical format..) is accomplished by means of the specified ASCII export template. Creation and administration of such templates is performed by means of the menu item "Extra / Options / File formats / ASCII storage"

Tip: Use the Function Assistant for parameterizing the function. It lets you easily select from a list of all ASCII export templates.

If no complete filename is entered, the system searches the current loading folder. The current loading folder is set according to the default defined under "Extra/Options/Folders" when imc FAMOS is started. The default folder can be changed using either the command LDIR or the function SetOption. Otherwise the folder last specified when a file was loaded is used.

If the template file (extension: "*.aet") is not located in the Definitions folder currently set, the complete pathname including the filename extension must be specified.

This function does not verify the validity of the filename and template name entered. Verification only happens with the corresponding call to FileClose(), once the file is actually written.

How the function behaves at fault condition depends on the setting "No error boxes for file functions" in the "Options"/"Functions" -dialog (or the corresponding function SetOption("Func.ErrorBoxes", ..). If this option is deactivated, an output box containing the error message is displayed and execution of the sequence is cancelled (default behavior of FAMOS-functions). On the other hand, if the option is activated, a 0 is returned and no error message is displayed.

If desired, the error cause can be queried using either of the functions FileErrText?() or FileErrCode?().

imc FAMOS can administer a maximum of 10 open files at once. This amount can easily be reached by neglecting to call FileClose. An appropriate error message will appear in such a case. The function FileResetAll closes all files open at the moment. This can be useful if, for instance, while testing sequences in the single-step mode, some files whose ID's you've forgotten remain open.

Texts/Textarrays: From version 2023, text or textarray variables can also be specified with FileObjWrite(). Texts are truncated to a maximum of 32767 characters when saved.

Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

A data set is filtered and subsequently saved along with the filtering results to an ASCII file. The exact format is defined by the export template "Template #2", which was previously created using the command "Extra / Options / File Formats / ASCII-Storage".

```
signal = ...
filtered =  FiltLP(signal, 0, 0, 6, 100)
fh = FileOpenAscii2("z:\dat\result.txt", "Template #2", 1)
IF (fh > 0)
   err = FileObjWrite(fh, signal)
   err = FileObjWrite(fh, filtered)
   err = FileClose(fh)
```

```
END
```

The following sequence saves the selected variables column-by-column in an ASCII file. Subsequently, an additional footer is appended to the file.

```
TxFileName = "c:\dat\protocol.txt"

fh = FileOpenAscii2(TxFileName, "Template #2", 1)
IF (fh > 0)
   count = VarGetInit(1)
   n = 1
   WHILE (n <= count)
      TxVarName = VarGetName?(n)
      err = FileObjWrite(fh, <TxVarName>)
      n = n+1
   END
   err = FileClose(fh)
END

fh = FileOpenAscii(TxFileName, 2) ; Option 2: append
IF (fh > 0)
   err = FileLineWrite(fh, "User: Paul Smithr", 0)
   err = FileClose(fh)
END
```

See also:

FileOpenAscii, FileObjWrite, FileClose, FileOpenXLS, FileOpenXLS2

FileOpenDSF

Opens a measurement value file in the imc/FAMOS-format.

Declaration:

```
FileOpenDSF ( TxFile, SvMode ) -> SvFileID
```

Parameter:

TxFile	Name of the file. Unless a full pathname is provided, the pre-set loading folder is used.
SvMode	Mode
	0 : Opens file for reading
	1 : Opens the file for writing. Any existing file of the same name will be overwritten.
	2 : Opens file for writing. If the file already exists, the new content is appended.
SvFileID	
SvFileID	ID of the opened file, or error code
	0 : Error opening the file. The cause of the error can be queried with the functions FileErrCode?() or FileErrText?().
	>0 : The identifier for the opened file. It is passed as a parameter to all subsequent file editing functions.

Description:

A measurement value file in imc FAMOS format is opened. If a complete file name is not specified, the program searches in the current directory for loading files.

This directory for loading files is set to the presettings when imc FAMOS is started (dialog "Options" / "Directories"). Use the command LDIR to change the load directory; otherwise the directory last specified to load a file is used.

The identifier of the opened file must be specified with a file function each time the file is accessed. This identifier is valid until the function FileClose() is called.

Each file opened using FileOpenDSF() must be closed by calling FileClose().

imc FAMOS can manage a maximum of 10 open files at the same time. If FileClose is not called enough, the maximum number of files can be reached quickly and an error message is generated.

The function FileResetAll closes all files presently open. This can be useful if files whose identifiers are no longer known remain open while testing imc FAMOS sequences in single-step mode.

If the file was opened for reading with the options 0, additional flags can be specified with <SvMode>:

Add +10:	Fast loading. Upon subsequent calling of the function FileObjRead the data are not copied, but rather managed by reference. This is useful only if you do not intend to change the data sets later.
Add +100:	No data groups. The data sets or texts contained in the data groups are expanded.

Additional options concerning compatibility with DSF-files, which were generated by programs or devices not produced by imc:

Add +1000:	Offset correction. This becomes necessary if you observe an offset of the data set in the y-direction.
Add +2000:	Exchanges the values for the day and month when reading the trigger time. Activate this option if you receive an error message stating "Invalid entry in |NTKey", or if the day and month of a trigger time are interchanged in the loaded data set.
Add +20000:	This option can be used to load files which were not concluded correctly during measurement. FAMOS then tries to load such files partially and to read the valid portion of measured data. It is recommended that you subsequently carefully check the data loaded. This option only affects files in the "imc3" file format.

If the file is to be opened for writing using Option 1 or 2, it is possible to specify adiitional flags with [SvMode]:

Add +10000:	As of FAMOS 7.4, a new and improved file format (Version 3) is used, which may not be readable by older imc programs (imc FAMOS 7.3, imc STUDIO 5.x). With this option, you can force use of the 'old' file format (Version 2).

- How the function responds to an error depends on the global presetting for "No error messages for file functions", which is set in the dialog called by "Options / Functions", or using the function SetOption. If this option is selected, an output box containing the error message is displayed and the return value is not generated. This corresponds to the standard response of imc FAMOS functions. If the option is not selected, the value 0 is returned and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.
- Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

A file in imc FAMOS format is opened for reading. All texts are loaded into imc FAMOS in a loop under the original name. Data sets and data groups in the file are skipped.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      Typ = FileObjType?(idFile, index)
      IF Typ = 1
         TxName  = FileObjName?(idFile, index)
          <TxName> = FileObjRead(idFile, index)
      END
      index = index + 1
   END
   ret = FileClose(idFile)
END
```

See also:

FileLoad, FileSave, FileOpenFAS, FileOpenASCII, FileOpenXLS

FileOpenFAS

Opens a measurement value file in a user-defined format

Declaration:

```
FileOpenFAS ( TxFile, TxFormat, SvMode ) -> SvFileID
```

Parameter:

TxFile	Name of the file. Unless a full pathname is provided, the pre-set loading folder is used.
TxFormat	Specification of the file format
SvMode	Mode
	0 : Opens file for reading
	100 : Opens the file for reading. Any groups included are expanded.
	1 : Opens the file for writing. Any existing file of the same name will be overwritten.
	2 : Opens the file for writing. If the file already exists, the new content is appended (provided 'Append' is supported by the export filter used).
SvFileID	
SvFileID	ID of the opened file, or error code
	0 : Error opening the file. The cause of the error can be queried with the functions FileErrCode?() or FileErrText?().
	>0 : Identifier for the opened file. This is provided as a parameter to all subsequent functions for editing this file.

Description:

Opens a file in a user-defined format (File Assistant or import/export extension library or derived import format).

Three variants are possible for specifying the format:

Import with format description file (*.FAS, imc File Assistant).

The format is determined by an importation description file (an import filter) created by means of the imc File Assistant. Such an import filter file usually takes the extension ".fas".

<TxFormat> specifies the name of the format describing file. Unless a complete pathname is provided, the system searches in the default definitions' folder ("Extra"/ "Options"/ "Folders").

With this variant, files of measured data can only be read, not written; thus, <Mode> must always be set to 0.

Import or export with an associated extension library (*.DLL).

The necessary functionality is provided by an import/export extension library. Such libraries (Dynamic Link Library, DLL) are offered by imc or our distribution partners and can be seamlessly integrated into the imc FAMOS user's interface and function library. This means that imc FAMOS can be flexibly adapted to support additional file formats. One example is the DLL for importing and exporting the MATLAB file format which is included in the standard imc FAMOS package.

The syntax of <TxFormat> is then as follows:

```
"#DLLName|FormatName|Parameter"
```

- DLLName: specifies the filename of the extension library; the extension ".DLL" can be omitted.
- FormatName: specifies the name of the desired format (an extension library may certainly support multiple formats). Can be omitted if the DLL only provides one single format and does not require any additional parameters.

- Parameters: serves for the optional transfer of further parameters to the library. Whether parameters are required and their syntax depends on the respective extension library. Often not required, can then be omitted. If necessary, look for the name of the format used in the help, where the format-specific use of the parameter is described.

Import with a derivative import filter

Import is performed by means of a derived import filter. The syntax for <TxFormat> is then as follows:

```
"$FormatName"
```

"FormatName" is the name of the derived filter, as displayed in the FAMOS dialog "Options"/Import filter".

Note: A measurent association defined in the derived import filter is ignored when loading channels.

Loading a text variable from a text file

To transfer the content of a text file to a text variable, enter the format identifier "imc/Text".

You can also specify the character set encoding (Code page) of the file. The current Windows code page is standard.

"imc/Text"	The current Windows ANSI code page is assumed.
"imc/Text/auto)"	The encoding is determined automatically. For this purpose, it is checked whether the file begins with a defined BOM (byte order mark). The BOM for UTF-8 and UTF-16 ("Big endian" and "Little endian" byte order) are recognized. If no such BOM is recognized, the current Windows code page is assumed.
"imc/Text/UTF8"	UTF-8-Encoding
"imc/Text/UTF16_LE"	UTF-16 LE (Byte order "Little endian", least significant Byte first)
"imc/Text/UTF16_BE"	UTF-16 BE (Byte order "Big endian", most significant Byte first)

Tip: Use the Function Assistant for setting the function's parameters. With this tool, you can conveniently select from a list of all file filters installed.

Examples:

```
FileOpenFAS("1.txt","#MyFormat.DLL", 0)
FileOpenFAS("1.txt","#MyFormat.DLL|Txt", 0)
FileOpenFAS("1.asc","#MyFormat.DLL|Asc|Head=y", 1)
FileOpenFAS("1.lec","$Lecroy #1", 0)
```

Import of MatLab-files:

```
FileOpenFAS("1.mat","#MatLabImportExport.DLL", 0)
```

Export of MatLab-files:

```
FileOpenFAS("1.mat","#MatLabImportExport.DLL|Matlab 4 Format", 1)
  ; or
FileOpenFAS("1.mat","#MatLabImportExport.DLL|Matlab 5 Format", 1)
```

ASCII-Import-Assistant

Import filters defined using the ASCII Import Assistant can also be used with the function FileOpenFAS().

```
FileOpenFAS("1.txt","#ImportAscii1.dll|FilterName", 0)
```

"FilterName" represents the name under which the filter was saved in the ASCII-Import dialog, and is identical to the format name displayed in the "Load File" dialog.

- Unless a complete pathname is provided for the measured value file, the system searches in the current loading folder. Upon starting imc FAMOS, the current loading folder is set to the presetting (dialog "Options"/ "Folders"). It can be reset with the command LDIR or the function SetOption. Otherwise, the folder specified the last time a file was loaded is used.
- The identifier of the opened file must be stated each time the file is subsequently accessed with a file function. It remains valid until FileClose is called.
- Every file opened with FileOpenFAS must be closed again by calling FileClose.
- imc FAMOS can administer a maximum of 10 open files simultaneously. If calls of FileClose are missing this number can quickly be reached. In that case, an error message will be posted.
- The function FileResetAll closes all files open at the moment. This can be useful if, for example, in a test of imc FAMOS-sequences in the single-step mode some files stayed open whose identifiers are no longer known.
- How the function responds in fault condition depends on the global presetting "No error boxes for file functions" (available on the "Options"/ "Functions"-dialog or with the function SetOption). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This conforms to the default behavior of the imc FAMOS functions. If, on the other hand, the option was selected, a 0 is returned and no error message is generated.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.
- Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

A file is opened for reading in a user-defined format. The format is described in the file 'format1.fas'. In a loop through all objects, all of the data sets are created in FAMOS under their original names. Strings and data groups in the file are skipped.

```
idFile = FileOpenFAS("c:\imc\dat\xxx.dat", "c:\imc\def\format1.fas", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      Typ = FileObjType?(idFile, index)
      IF Typ = 1
         TxName = FileObjName?(idFile, index)
          <TxName>= FileObjRead(idFile, index)
      END
      index = index + 1
   END
   ret = FileClose(idFile)
```

```
END
```

The variable 'MyChannel' is saved in MatLab5 format.

```
MyChannel = ...
idFile = FileOpenFAS("z:\tmp\export.mat", "#MatlabImportExport.dll|Matlab 5 Format", 1)
IF idFile >= 1
   TxName  = FileObjWrite(idFile, MyChannel)
   ret=FileClose(idFile)
END
```

See also:

FileLoad, FileOpenASCII, FileOpenXLS2, FileXLSColumnRead, FileXLSCellRead

FileOpenXLS

An Excel table in XLS-format is opened for reading and/or cell-by-cell writing.

Declaration:

```
FileOpenXLS ( TxFile, SvMode ) -> SvFileID
```

Parameter:

TxFile	Name of the XLS file. If no complete pathname is specified, the default folder for opening files is used.
SvMode	Mode
	0 : The file is opened in Read-Only mode.
	1 : Opens the file for writing. Any existing file of the same name will be overwritten.
	2 : Opens file for writing. If the file already exists, the new content is appended.
SvFileID	
SvFileID	ID of the opened file, or error code
	0 : Error opening the file. The cause of the error can be queried with the functions FileErrCode?() or FileErrText?().
	>0 : Identifier for the opened file. This is provided as a parameter to all subsequent functions for editing this file.

Description:

This function enables reading and writing of files in the EXCEL-XLS format. After the file has been opened with this function, it is subsequently possible to use the functions FileXLSColumnRead(), FileXLSCellRead() or FileXLSCellWrite() to access the table.

The function starts a hidden instance of EXCEL which is instructed by remote control (so-called OLE automation) to load the file. Reading functions used subsequently import the desired data also by means of EXCEL remote control.

Thus, the function can only be used if a version of EXCEL is installed on the computer (versions Excel95 .. Excel2016).

Every file opened with FileOpenXLS() must be closed subsequently by calling FileClose(). In the process, any file changed is saved and the hidden instance of the Excel program is closed.

With this function, only one EXCEL file can be open at a time. If any XLS file opened before the call to FileOpenXLS() is still open (meaning: the function FileClose() has not yet been called), the previously open file is closed automatically.

imc FAMOS can administer a maximum of 10 open files at once. This amount can easily be reached by neglecting to call FileClose. An appropriate error message will appear in such a case. The function FileResetAll closes all files open at the moment. This can be useful if, for instance, while testing sequences in the single-step mode, some files whose ID's you've forgotten remain open.

FileXLSColumnRead(), FileXLSCellRead(), FileXLSCellWrite(), FileXLSSelectSheet() and FileClose() are the only functions which can be applied to files opened with FileOpenXLS().

To write to XLS-files column-by-column, you can use the function FileOpenXLS2().

If no complete filename is entered, the system searches the current loading folder. The current loading folder is set according to the default defined under "Extra/Options/Folders" when imc FAMOS is started. The default folder can be changed using either the command LDIR or the function SetOption(). Otherwise the folder last specified when a file was loaded is used.

How the function responds to errors depends on the setting "No Error boxes for file functions" in the dialog "Options"/"Functions" (or the corresponding function SetOption("Func.ErrorBoxes", ..). When this option is deactivated, an output box with the error message appears and processing of the sequence is aborted (FAMOS-functions' default response). However, if the option is active, empty text is returned and no error message is posted.

If desired, the error cause can be queried using either of the functions FileErrText?() or FileErrCode?().

Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

An XLS-file is opened. It contains a table with 2 columns each containing a value pair (time, reading). The values begin at the 2nd row. In the 1st row stands the data set's name alongside the y-unit.

```
fh = FileOpenXLS("c:\dat\table.xls", 0)
IF (fh > 0)
   x = FileXLSColumnRead(fh, 1, 2, 65336, 0)
   y = FileXLSColumnRead(fh, 2, 2, 65336, 0)
   TxName = FileXLSCellRead(fh, 1, 1, 0)
   TxUnit = FileXLSCellRead(fh, 2, 1, 0)
   err = FileClose(fh)
   data = XYof(x,y)
   ; 1st value of the time track = Trigger time
   first_time = data[1].x
   SetTime(data, first_time)
   ; scales time track in terms of trigger time
   data.x = data.x-first_time
   ; sets unit and name
   SetUnit(data, TxUnit, 1)
   RENAME data <TxName>
END
```

A data set is saved in an XLS file as of the 2nd row. Subsequently, the mean value of the saved data is entered in the first row.

```
signal = ...
fh = FileOpenXLS2("z:\dat\result.xls", "Template #2", 1, 1, 2)
IF (fh > 0)
   err = FileObjWrite(fh, signal)
   err = FileClose(fh)
END
fh = FileOpenXLS("z:\dat\result.xls", 2)
IF (fh > 0)
   err = FileXLSCellWrite(fh, 1, 1, "Mean Value:")
   err = FileXLSCellWrite(fh, 2, 1, Mean(signal))
   err = FileClose(fh)
END
```

See also:

FileOpenASCII, FileOpenXLS2, FileXLSColumnRead, FileXLSCellRead, FileXLSCellWrite

FileOpenXLS2

An XLS file is opened in which to export data sets in columns. Specification of the format is performed by means of a pre-defined XLS export template.

Declaration:

```
FileOpenXLS2 ( TxFile, TxExportTemplate, SvMode, SvColumn, SvRow ) -> SvFileID
```

Parameter:

TxFile	Name of the XLS file. If no complete pathname is specified, the default folder for opening files is used.
TxExportTemplate	Name of the XLS export template used
SvMode	Mode
	1 : A file in which to write data is opened. Any existing file having the same name is completely overwritten.
	2 : A file to which to write data is opened. Any existing file of the same name is opened and the data sets to add are copied into the existing table, otherwise the original file is not changed.
SvColumn	Column number (1..256). Along with the next parameter, it indicates to which cell data is written first; the top left is [1,1].
SvRow	
SvFileID	
SvFileID	ID of the opened file, or error code
	0 : Error opening the file. The cause of the error can be queried with the functions FileErrCode?() or FileErrText?().
	>0 : Identifier for the opened file. This is provided as a parameter to all subsequent functions for editing this file.

Description:

This function enables writing to files in the EXCEL-XLS format, in which data sets are saved colmn-by-column. The format specification is performed by means of a pre-defined XLS-export template.

The function can only be used if a version of EXCEL is installed on the computer (versions Excel95.. Excel2016).

The function starts a hidden instance of EXCEL which is instructed by remote control (so-called OLE automation) to load the file/create a new table.

Once a file has been opened with this function, the data sets to save can be added using the function FileObjWrite(..).

By default, the first table sheet is always accessed. If a different page is to be written, the function FileXLSSelectSheet() should accordingly be called beforehand.

Every file opened with FileOpenXLS2 must afterwards be closed by a call to FileClose(). In the process, the file is also saved and the hidden Excel program instance is closed.

FileObjWrite(), FileXLSSelectSheet() and FileClose() are the only functions which can be applied to files opened with FileOpenXLS2().

Concrete definition of the file format (e.g. column headers, scaling columns, numerical format..) is accomplished by means of the specified XLS export template. Creation and administration of such templates is performed by means of the menu item "Extra / Options / File-Save / EXCEL"

Tip: Use the Function Assistant for parameterizing the function. It lets you easily select from a list of all XLS export templates.

To import XLS files, you can use the function FileOpenXLS().

If no complete filename is entered, the system searches the current loading folder. The current loading folder is set according to the default defined under "Extra/Options/Folders" when imc FAMOS is started. The default folder can be changed using either the command LDIR or the function SetOption(). Otherwise the folder last specified when a file was loaded is used.

This function doesn't check the validity of the filenames and template names passed to it. This is only done later in connection with the corresponding call of FileClose(), when the file is actually written.

If the template file (extension: "*.aet") is not located in the definitions folder currently set, the complete pathname including the filename exptension must be specified.

imc FAMOS can administer a maximum of 10 open files at once. This amount can easily be reached by neglecting to call FileClose. An appropriate error message will appear in such a case. The function FileResetAll closes all files open at the moment. This can be useful if, for instance, while testing sequences in the single-step mode, some files whose ID's you've forgotten remain open.

How the function behaves at fault condition depends on the setting "No error boxes for file functions" in the "Options"/"Functions" -dialog (or the corresponding function SetOption("Func.ErrorBoxes", ..). If this option is deactivated, an output box containing the error message is displayed and execution of the sequence is cancelled (default behavior of FAMOS-functions). On the other hand, if the option is activated, a 0 is returned and no error message is displayed.

If desired, the error cause can be queried using either of the functions FileErrText?() or FileErrCode?().

Texts/Textarrays: From version 2023, text or textarray variables can also be specified with FileObjWrite(). Texts are truncated to a maximum of 32767 characters when saved.

Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

A data set is filtered and subsequently saved together with the filtering results in an XLS file. The exact format is defined by the export template "Template #2", which was previously created using the command "Extra / Options / File Saving / EXCEL".

```
signal = ...
filtered =  FiltLP(signal, 0, 0, 6, 100)
fh = FileOpenXLS2("z:\dat\result.xls", "Template #2", 1, 1, 1)
IF (fh > 0)
   err = FileObjWrite(fh, signal)
   err = FileObjWrite(fh, filtered)
   err = FileClose(fh)
END
```

The following sequence saves the selected variables column-by-column in an XLS-file. The output begins in the 3rd column, 2nd line.

```
TxFileName = "c:\dat\protocol.xls"

fh = FileOpenXLS2(TxFileName, "Template #2", 1, 3, 2)
IF (fh > 0)
   count = VarGetInit(1)
   n = 1
   WHILE (n <= count)
      TxVarName = VarGetName?(n)
      err = FileObjWrite(fh, <TxVarName>)
      n = n + 1
   END
   err = FileClose(fh)
END
```

See also:

FileOpenASCII, FileOpenASCII2, FileOpenXLS, FileXLSSelectSheet, FileObjWrite, FileClose

FileResetAll

Closes all opened files

Declaration:

```
FileResetAll ( )
```

Parameter:

Description:

All file lists opened with a FileOpen*() function (e.g. FileOpenDSF or FileOpenFAS) are closed. There is no writing from file lists if changes were made.

These functions should only be used to recreate a defined basic state when testing sequences or programs; the function FileClose() should always be used.

imc FAMOS can manage a maximum of 10 open files at the same time. If FileClose is not called enough, the maximum number of files can be reached quickly and an error message is generated.

The function FileRestAll can be used if files, whose identifiers are unknown, remain open while testing imc FAMOS sequences in single-step mode.

Multithreading: The function only closes files opened in the current execution thread.

See also:

FileClose, FileOpenDSF, FileOpenFAS, FileOpenASCII, FileOpenXLS

FileSave

Saves the data supplied in a file of selectable format.

Declaration:

```
FileSave ( TxFile, TxFormat, SvOptions, Data [, Data2] [, Data3] [, Data4] [, Data5] [, Data6] ) -> SvSuccess
```

Parameter:

TxFile	Name of the file. Unless a full pathname is provided, the pre-set loading folder is used.
TxFormat	Specification of the file format
SvOptions	Options
	0 : Default behavior
	1 : Any groups supplied are expanded, i.e. what is saved is the list of the channels contained, without any group association. The option is helpful when, for instance, a complex analysis generates a large number of resulting variables which you ultimately wish to save to a file. Add all of these variables right after they are created to a temporary group and then at the end of the analysis, provide it as the Save parameter here.
	10000 : Only taken into account when the "imc/FAMOS"-file format is used. As of FAMOS 7.4, a new and improved file format (Version 3) is used by default, wheich may not be readable by older imc programs (imc FAMOS 7.3, imc STUDIO 5.x). Add 10000, to force use of the 'old' file format (Version 2).
Data	Data to be saved
Data2	Additional data (optional) (optional)
Data3	Additional data (optional) (optional)
Data4	Additional data (optional) (optional)
Data5	Additional data (optional) (optional)
Data6	Additional data (optional) (optional)
SvSuccess	
SvSuccess	Success of the function (optional)
	0 : Error in saving the file. The error cause can be inquired with either of the functions FileErrCode?() or FileErrText?().
	1 : The file has been successfully saved.

Description:

Tip: Use the Function Assistant to parameterize the function. By this means, you are easily able to select any kind of export format from a list.

For specifying the format, multiple different ways are possible:

Saving in the default imc/FAMOS format
For <TxFormat>, either an empty text or the identifier "imc/FAMOS" must be entered.

if the file name supplied has no extension, then ".dat" is appended automatically.

When saving a single element from a group, the group information is lost.

Alternative function: FileOpenDSF().

Saves a text variables to a text file.
In order to save the content of a text- or textarray variable in a text file, enter the format ID "imc/Text". Only one variable can be saved per file, so the parameters Data2 to Data6 should be omitted.

For a text array, each element is saved as a line in the file. The line break is guaranteed by a 'CarriageReturn' (ASCII 13)/'LineFeed' (ASCII 10) character combination. No line break is appended after the last line.

if the file name supplied has no extension, then ".txt" is appended automatically.

Export with a corresponding extension library (*.DLL).
The syntax of <TxFormat> is then as follows:

```
"#DLLName|FormatName|Parameter"
```

- DLLName: specifies the filename of the extension library; the extension ".DLL" can be omitted.
- FormatName: specifies the name of the desired format (an extension library may certainly support multiple formats). Can be omitted if the DLL only provides one single format and does not require any additional parameters.
- Parameters: serves for the optional transfer of further parameters to the library. Whether parameters are required and their syntax depends on the respective extension library. Often not required, can then be omitted. If necessary, look for the name of the format used in the help, where the format-specific use of the parameter is described.

The FAMOS product package contains a number of extension libraries for creation of common file formats.

Example: Export of 3 data sets to a MatLab file:

```
FileSave("1.mat","#MatLabImportExport|Matlab 5 Format", 0, Channel1, Channel2, Channel3)
```

Alternative function: FileOpenFAS().

Export by means of an ASCII export template
The data sets transferred are each written to a column in an ASCII file. Specification of the format is accomplished by means of a pre-defined ASCII export template containing concrete file format definitions (e.g. column headers, scaling columns, numerical format). Creation and management of such templates is accomplished by means of the FAMOS menu item "Extras/Options/Save File/Export/ASCII".

The syntax of <TxFormat> is then as follows:

```
"[ASC] ExportTemplateName"
```

Alternative function: FileOpenASCII2().

Export by means of an EXCEL export template
The data sets transferred are each written to a column in an EXCEL file. Specification of the format is accomplished by means of a pre-defined EXCEL export template containing concrete file format definitions (e.g. column headers, scaling columns). Creation and management of such templates is accomplished by means of the FAMOS menu item "Extras/Options/Save File/Export / EXCEL".

The syntax of <TxFormat> is then as follows:

```
"[XLS] ExportTemplateName"
```

With this function, a hidden EXCEL instance is started and it is requested by means of (the OLE-Automation's) remote control to fill and save a new table. The function can also only be run if a supported version of EXCEL is installed. The data are always written to the first worksheet, beginning at the 1st row and 1st column.

Alternative function: FileOpenXLS2().

Special format "ByteBlob": Uninterpreted binary data
The content of the variables (only the actual data) is written to a file in a binary and uninterpreted format. For this, specify the format ID "imc/ByteBlob".

The format can be used for single-component, unstructured data sets with simple data formats (real, integer, but not digital) or TimeStampASCII. Normally required only for special applications, e.g. in conjunction with the FAMOS-Database Kit for the purpose of being able to save and retrieve any file types (e.g. PDF, images, videos) to and from databases jointly with the actual measurement data.

- The respective alternative function for saving in a particular format is more complex to use but offers additional possibilities (e.g. any amounnt of data sets, appended to existing files).
- If the template file (extension: "*.aet") is not located in the currently set default definitions folder upon ASCII- or EXCEL-export, the complete pathname must be specified.
- The function's behavior at fault condition depends on the global presetting 'No error boxes with file functions' (which is set in the dialog "Options/Functions" or by means of the function SetOption). If this option is deactivated, an error message box is displayed and the return value is not generated. This matches the default behavior of FAMOS functions. On the other hand, if this option is activated, a 0 is returned and no error message is displayed. If may be possible to inquire the error cause with the functions FileErrText?() or FileErrCode?().

Examples:

The variables 'MyChannel' through 'MyChannel3' are saved in the FAMOS format in file 'result.dat'.

```
FileSave("z:\tmp\result", "", 0, MyChannel1, MyChannel2, MyChannel3)
```

The comment belonging to the data set 'MyChannel' is saved in a text file.

```
FileSave("z:\tmp\comment.txt", "imc/Text", 0, Comm?(MyChannel1))
```

The group 'result' contains the channels 'c1' through 'c3' and is saved in EXCEL format. The group is expanded, the file contains the channels c1,c2,c3 without group association.

```
FileSave("z:\tmp\c.xls", "[XLS] XLSX, individual scaling", 1, result)
```

Saves a variable in MatLab4 format.

```
FileSave("z:\tmp\export.mat", "#MatlabImportExport|Matlab 4 Format", 0, MyChannel)
```

See also:

FileOpenDSF, FileOpenASCII2, FileOpenXLS2, FileLoad

FileSetComm

4.109 FileSetComm Function The function assigns a specified file comment to an opened file.

Declaration:

```
FileSetComm ( SvFileID, TxComment ) -> SvStatus
```

Parameter:

SvFileID	File identifier, received during a previous call of FileOpenDSF, FileOpenXLS2 or FileOpenASCII2
TxComment	New comment for the file
SvStatus	
SvStatus	Success of the function
	0 : Function executed successfully
	-1 : Error in executing the function

Description:

The function sets the file comment of an open imc FAMOS file. The file must be opened for writing, and the function FileOpenDSF needs to have been used with option 1 or 2. The new comment is memorized internally, but the changed file is actually written when the function FileClose is called.

When writing by means of export templates (FileOpenASCII2/FileOpenXLS2), the placeholder %FILECOMMENT% in the template is replaced accordingly.

- The behavior of the function if an error occurs depends on whether the global presetting "No error boxes in file functions" is set in the "Options"/ "Functions" dialog or with the function SetOption(). If this option is not selected, an output box with the error message is displayed and the return value is not generated. This corresponds to the standard behavior of imc FAMOS functions. If the option is selected, a return value of -1 is generated and an error message is not displayed.
- If necessary, the cause of error can be inquired using the functions FileErrText? or FileErrCode?.
- Querying and evaluating the return value is not absolutely necessary in imc FAMOS, but recommended.

Examples:

A imc FAMOS file "xxx.dat" is opened in imc FAMOS format for reading and writing. An inquiry of the comment is made and if one is not present, a comment is entered and the file is closed.

```
fileID = FileOpenDSF("c:\imc\dat\xxx.dat", 2)
TxComm = FileComm?(fileID)
IF TLeng(TxComm) = 0
   status = FileSetComm(fileID, "checked")
END
status = FileClose(fileID)
```

See also:

FileOpenDSF, FileComm?, FileOpenASCII2, FileOpenXLS2

FileXLSCellRead

A cell in an EXCEL-table (XLS-format) is imported (as text).

Declaration:

```
FileXLSCellRead ( SvFileID, SvColumn, SvRow, Zero ) -> TxContents
```

Parameter:

SvFileID	ID of the opened XLS-file, returned upon execution of the command FileOpenXLS().
SvColumn	Number of the column to be read (1..256 or 1..16384 since Excel 2007).
SvRow	Row number of the cell to be read. The top row is numbered 1. The last (maximum possible) row has the number 1048576 (since Excel 2007) or 65536 in older versions.
Zero	Reserved parameter. Always set to 0.
TxContents	
TxContents	Cell content (as text).

Description:

This function reads the content of an EXCEL table's cell in XLS-format and returns it as text.

The file has to have been opened previously using the function FileOpenXLS().

By default, the first table sheet is accessed. If the data are to be written to a different sheet, first call the function FileXLSSelectSheet() accordingly.

For reading numerical values, the function FileXLSColumnRead() should always be used.

How functions behave at fault condition depends on the setting "No error boxes for file functions" in the "Options"/"Functions" -dialog (or the corresponding function SetOption("Func.ErrorBoxes", ..). If this option is deactivated, an output box containing the error message is displayed and execution of the sequence is cancelled (default behavior of FAMOS-functions). On the other hand, if the option is activated, an empty string is returned and no error message is displayed.

If desired, the error cause can be queried using either of the functions FileErrText?() or FileErrCode?().

Examples:

An XLS-file is opened. It contains a table with 2 columns each containing a value pair (time, reading). The values begin at the 2nd row. In the 1st row stands the data set's name alongside the y-unit.

```
fh = FileOpenXLS("c:\dat\table.xls", 0)
IF (fh > 0)
   x = FileXLSColumnRead(fh, 1, 2, 1048575, 0)
   y = FileXLSColumnRead(fh, 2, 2, 1048575, 0)
   TxName = FileXLSCellRead(fh, 1, 1, 0)
   TxUnit = FileXLSCellRead(fh, 2, 1, 0)
   err = FileClose(fh)
   data = XYof(x,y)
   ; 1st value of the time track = Trigger time
   first_time = data[1].x
   SetTime(data, first_time)
   ; scales time track in terms of trigger time
   data.x = data.x-first_time
   ; sets unit and name
   SetUnit(data, TxUnit, 1)
   RENAME data <TxName>
END
```

See also:

FileOpenXLS, FileXLSColumnRead, FileClose

FileXLSCellWrite

Write data to a cell in an EXCEL table (XLS-format).

Declaration:

```
FileXLSCellWrite ( SvFileID, SvColumn, SvRow, Content ) -> SvStatus
```

Parameter:

SvFileID	ID of the opened XLS-file, returned upon execution of the command FileOpenXLS().
SvColumn	Column number of the cell to fill (1..256).
SvRow	Row number of the cell to fill. he top row is numbered 1.
Content	Content to write. Either a numerical single value (or data set of length 1) or text is allowed.
SvStatus	
SvStatus	Success of the function
	0 : Function has been carried out successfully
	-1 : Function failed. The cause can be queried using the function FileErrCode?() or FileErrText?().

Description:

This function sets the content of an EXCEL table cell in XLS format. Either a text or a single numerical value can be entered.

The file must previously have been opened with the function FileOpenXLS() in Write Mode.

For writing longer series of data to a table column, the functions FileOpenXLS2()/FileObjWrite() are preferable to repeated calling of this function.

By default, the first table sheet is accessed. If the data are to be written to a different sheet, first call the function FileXLSSelectSheet() accordingly.

How the function behaves at fault condition depends on the setting "No error boxes for file functions" in the "Options"/"Functions" -dialog (or the corresponding function SetOption("Func.ErrorBoxes", ..). If this option is deactivated, an output box containing the error message is displayed and execution of the sequence is cancelled (default behavior of FAMOS-functions). On the other hand, if the option is activated, -1 is returned and no error message is displayed.

If desired, the error cause can be queried using either of the functions FileErrText?() or FileErrCode?().

Examples:

A data set is saved in an XLS file as of the 2nd row. Subsequently, the mean value of the saved data is entered in the first row.

```
data = ...
fh = FileOpenXLS2("z:\dat\result.xls", "Template #2", 1, 1, 2)
IF (fh > 0)
   err = FileObjWrite(fh, data)
   err = FileClose(fh)
END
fh = FileOpenXLS("z:\dat\result.xls", 2)
IF (fh > 0)
   err = FileXLSCellWrite(fh, 1, 1, "Mean Value:")
   err = FileXLSCellWrite(fh, 2, 1, Mean(data))
   err = FileClose(fh)
END
```

See also:

FileOpenXLS, FileXLSCellRead, FileClose

FileXLSColumnRead

One column of an EXCEL-table (XLS-Format) is read out.

Declaration:

```
FileXLSColumnRead ( SvFileID, SvColumn, SvRow, SvMaxCount, Zero ) -> Data
```

Parameter:

SvFileID	ID of the opened XLS-file, returned upon execution of the command FileOpenXLS().
SvColumn	Number of the column to be read (1..256 or 1..16384 since Excel 2007).
SvRow	Row number at which to begin reading. The top row is numbered 1. The last (maximum possible) row has the number 1048576 (since Excel 2007), or 65536 in older versions.
SvMaxCount	Maximum amount of values to be read.
Zero	Reserved parameter. Always set to 0.
Data	
Data	Column contents (as data set).

Description:

This functions reads out columns of EXCEL-table data in XLS-format.

The file has to have been opened previously using the function FileOpenXLS().

The first value of the created data set is read out from the cell specified by the parameters [Column] and [Row]. Then, read-out continues down the column, until either the first cell not having a numerical value is found, or the maximum number of values to read (corresponding to the parameter [MaxAmount]) is reached.

By default, the first table sheet is accessed. If the data are to be written to a different sheet, first call the function FileXLSSelectSheet() accordingly.

If the column to be read is formatted as Date/Time, the values are converted from EXCEL time format to imc time format. The values can be converted to legible text using the function TimeInText(), for example, or be assigned directly as the X-component of an XY-data set.

The result's data format is always 8Byte Real, (Double).

To read text, the function FileXLSCellRead can be used.

How the function responds to errors depends on the setting "No Error boxes for file functions" in the dialog "Options"/"Functions" (or the corresponding function SetOption("Func.ErrorBoxes", ..). When this option is deactivated, an output box with the error message appears and processing of the sequence is aborted (FAMOS-functions' default response). However, if the option is active, empty text is returned and no error message is posted.

If desired, the error cause can be queried using either of the functions FileErrText?() or FileErrCode?().

Examples:

An XLS-file is opened. It contains a table with 2 columns each containing a value pair (time, reading). The values begin at the 2nd row. In the 1st row stands the data set's name alongside the y-unit.

```
fh = FileOpenXLS("c:\dat\table.xls", 0)
IF (fh > 0)
   x = FileXLSColumnRead(fh, 1, 2, 1048575, 0)
   y = FileXLSColumnRead(fh, 2, 2, 1048575, 0)
   TxName = FileXLSCellRead(fh, 1, 1, 0)
   TxUnit = FileXLSCellRead(fh, 2, 1, 0)
   err = FileClose(fh)
   data = XYof(x,y)
   ; 1st value of the time track = Trigger time
   first_time = data[1].x
   SetTime(data, first_time)
   ; scales time track in terms of trigger time
   data.x = data.x - first_time
   ; sets unit and name
   SetUnit(data, TxUnit, 1)
   RENAME data <TxName>
END
```

See also:

FileOpenXLS, FileXLSCellRead, FileClose

FileXLSSelectSheet

A table sheet in an EXCEL spreadsheet (XLS-format) is selected for subsequent read/write access.

Declaration:

`FileXLSSelectSheet ( SvFileID, SvTableIndex ) -> SvStatus`

Parameter:

SvFileID	ID of the opened XLS file, which was returned when opening the file with the command FileOpenXLS() or FileOpenXLS2().
SvTableIndex	Index of the table sheet to be selected (1..).
SvStatus	
SvStatus	Success of the function
	0 : Function has been carried out successfully
	-1 : Function failed. The cause can be queried using the function FileErrCode?() or FileErrText?().

Description:

Reading and cell-by-cell writing:

This function selects the worksheet to be subsequently accessed for reading by FileXLSCellRead(), FileXLSColumnRead() or FileXLSCellWrite(). The file needs to have been previously opened using the function FileOpenXLS() with the mode 'Read Only' or 'Write/Append'.

Column-by-column writing:

This function selects the table sheet in which values written with FileObjWrite() are to be saved. The file must previously be opened using the function FileOpenXLS2() with the option 'Combine' (parameter 'Mode' = 2). The table worksheet specified here must already exist in the opened file, so no new worksheet is created.

Only one call of FileXLSSelectSheet() is allowed for each FileOpenXLS2()/FileClose()-block. If multiple sheets are to be written in an EXCEL file, then a separate Open/Close-block is necessary for each sheet.

Examples:

An XLS-file with 3 table sheets is opened. Each table sheet contains 2 columns containing respective counterparts of value pairs (time, measured value). The values begin in the 2nd row. In the 1st row, the data set's name is noted.

```
fh = FileOpenXLS("c:\dat\table.xls", 0)
IF (fh > 0)
   I = 1
   WHILE I <= 3
      FileXLSSelectSheet(fh, I)
      x = FileXLSColumnRead(fh, 1, 2, 65336, 0)
      y = FileXLSColumnRead(fh, 2, 2, 65336, 0)
      TxName = FileXLSCellRead(fh, 1, 1, 0)
      <TxName> = XYof(x,y)
      I = I + 1
   END
   err = FileClose(fh)
END
```

A prepared EXCEL file contains 3 table sheets. In the 2nd sheet, the values of the data set "Channel1" are to be entered, and in the 3rd sheet the results of the data set's FFT. The template is first copied to the output folder and then filled out accordingly.

```
TxOutputFileName = "c:\out\result1.xls"
Channel1= ...
FFT_Channel1 = FFT(Channel1)
res = FsCopyFile("c:\templates\result.xls", TxOutputFileName, 2, 0)
fh = FileOpenXLS2(TxOutputFileName, "XLS Template Data", 2, 1, 1)
IF (fh > 0)
   err = FileXLSSelectSheet(fh, 2)
   err = FileObjWrite(fh, Channel1)
   err = FileClose(fh)
END
fh = FileOpenXLS2(TxOutputFileName, "XLS Template FFT", 2, 1, 1)
IF (fh > 0)
   err = FileXLSSelectSheet(fh, 3)
   err = FileObjWrite(fh, FFT_Channel1)
   err = FileClose(fh)
END
```

See also:

FileOpenXLS, FileXLSColumnRead, FileClose

FiltBP

Band-pass filter

Declaration:

```
FiltBP ( Data, SvCharacter, SvParameter, SvOrder, SvFreqLower, SvFreqUpper ) -> Filtrate
```

Parameter:

Data	Data set to be filtered [NW].
SvCharacter	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Chebychev
	3 : Critical damping
SvParameter	In Chebychev characteristic, the desired ripple (0..3) is set in dB. Otherwise set to 0.
SvOrder	Filter order. Bessel: Range 1..40, other 1..100.
SvFreqLower	Desired lower cut-off frequency in Hz.
SvFreqUpper	Desired upper cut-off frequency in Hz.
Filtrate	
Filtrate	Filtered data set

Description:

The transferred data set is filtered by a band-pass filter.

The filter coefficients are calculated from the transferred parameters using bilinear transformation.

For accurate filtering, both cut-off frequencies should lie below half of the sampling frequency of the output signal. When the frequencies are too near to the sampling frequency, the amplitude response of the filter becomes more inaccurate.

Examples:

```
BandPassFiltered = FiltBP(signal, 2, 1, 10, 100, 10000)
```

The data set is filtered with a band-pass filter with Chebychev characteristic. The ripple should be 1dB. A filter of the tenth order is used and the cut-off frequencies are 100Hz and 10kHz.

See also:

FiltBS, FiltHP, FiltLP, DFilt, FiltBpZ

FiltBpZ

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Band-pass filter without phase shift

Declaration:

```
FiltBpZ ( InputChannel, Characteristic, Parameter, Order, Lower cutoff frequency, Upper cutoff frequency ) ->
Result
```

Parameter:

InputChannel	The waveform to be filtered, time scaled in seconds.
Characteristic	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Tschebyschew
	3 : Critical damping
Parameter	With Tschebyschew characteristics, the ripple in dB (0..3). Otherwise set to 0.
Order	Filter order: 4, 8, 12, 16, 20
Lower cutoff frequency	Lower cutoff frequency in Hz
Upper cutoff frequency	Upper cutoff frequency in Hz
Result	
Result	Filtered waveform

Description:

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter coefficients are computed from the parameters specified with bilinear transformation.

For filtering to be useful, the cutoff frequency should be significantly below one-half of the output signal's sampling frequency. The nearer the frequency is to the sampling frequency, the more precise the filter's amplitude response.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

```
f = FiltBpZ ( Vibration, 0, 0, 4, 45, 55 )
```

A 4th order Butterworth band-pass filter with a frequency of 45Hz to 55Hz is calculated.

See also:

FiltLpZ, FiltBp

FiltBS

Band-stop filter

Declaration:

```
FiltBS ( Data, SvCharacter, SvParameter, SvOrder, SvFreqLower, SvFreqUpper ) -> Filtrate
```

Parameter:

Data	Data set to be filtered [NW].
SvCharacter	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Chebychev
	3 : Critical damping
SvParameter	In Chebychev characteristic, the desired ripple (0..3) is set in dB. Otherwise set to 0.
SvOrder	Filter order. Bessel: Range 1..40, other 1..100.
SvFreqLower	Desired lower cut-off frequency in Hz.
SvFreqUpper	Desired upper cut-off frequency in Hz.
Filtrate	
Filtrate	Filtered data set

Description:

The transferred data set is filtered with a band-stop filter

The filter coefficients are calculated from the transferred parameters using bilinear transformation.

For accurate filtering, both cut-off frequencies should lie below half of the sampling frequency of the output signal. When the frequencies are too near to the sampling frequency, the amplitude response of the filter becomes more inaccurate.

Examples:

```
BandStopFiltered = FiltBS(signal, 2, 1, 10, 100, 10000)
```

The data set is filtered with a band-stop filter with Chebychev characteristic. The ripple should be 1dB. A filter of tenth order is used and the cut-off frequencies should be 100Hz and 10kHz.

See also:

FiltBP, FiltHP, FiltLP, DFilt, FiltBsZ

FiltBsZ

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Band-stop filter without phase shift

Declaration:

```
FiltBsZ ( InputChannel, Characteristic, Parameter, Order, Lower cutoff frequency, Upper cutoff frequency ) ->
Result
```

Parameter:

InputChannel	The waveform to be filtered, time scaled in seconds.
Characteristic	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Tschebyschew
	3 : Critical damping
Parameter	With Tschebyschew characteristics, the ripple in dB (0..3). Otherwise set to 0.
Order	Filter order: 4, 8, 12, 16, 20
Lower cutoff frequency	Lower cutoff frequency in Hz
Upper cutoff frequency	Upper cutoff frequency in Hz
Result	
Result	Filtered waveform

Description:

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter coefficients are computed from the parameters specified with bilinear transformation.

For filtering to be useful, the cutoff frequency should be significantly below one-half of the output signal's sampling frequency. The nearer the frequency is to the sampling frequency, the more precise the filter's amplitude response.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

```
f = FiltBsZ ( Vibration, 0, 0, 4, 45, 55 )
```

A 4th order Butterworth band-stop filter with a frequency of 45Hz to 55Hz is calculated.

See also:

FiltLpZ, FiltBp

FilterAnalog

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

A waveform is processed with a filter whose analog coefficients must be supplied to the transfer function H(s).

Declaration:

```
FilterAnalog ( InputChannel, Coefficients ) -> Result
```

Parameter:

InputChannel	The waveform to be filtered, time scaled in seconds.
Coefficients	A waveform in which the coefficients appear in a set order. See below.
Result	
Result	Filtered waveform

Description:

The sampling time must be small enough so that the relevant bends or resonances of the frequency responce will be reflected.

The analog coefficients are used to design a digital filter by means of bilinear transformation. The digital filter is usually only a moderately good approximation of an analog filter's actual behaviour.

The largest discrepancies result for the highest frequencies (those near half the sampling rate). Thus differentiators cannot be realized.

The digital filter used always is recursive, which generally tends to distort the phase slightly.

The function is only suited to stable filters.

Order of coefficients:

The transfer function is represented as the product of 2nd order terms (biquads). There are 6 coefficients for each biquad.

Let:

- p = i * 2 * PI * f, f frequency, i = sqrt(-1).

Then the transfer function (here 2nd order) is given by:

- A (p) = (d0 + d1 * p + d2 * p^2) / (c0 + c1 * p + c2 * p^2)

Note the differences between the exponents of p in both equations!

Alternative expression:

- A (p) = (d0 * p^-2 + d1 * p^-1 + d2) / (c0 * p^-2 + c1 * p^-1 + c2)

The coefficients are entered in the following order:

- d0, d1, d2, c0, c1, c2

If there is more than one biquad, the process is repeated, and the coefficient list is continued accordingly:

- d0_1, d1_1, d2_1, c0_1, c1_1, c2_1, d0_2, d1_2, d2_2, c0_2, c1_2, c2_2, ...

Up to 1000 biquads can be applied.

Examples:

```
d0 = 1
d1 = 0.08
d2 = 0
c0 = 1
c1 = 0.125
c2 = 0.08 * 0.08
acoeff = leng( 0, 6 )
acoeff[1] = d0
acoeff[2] = d1
acoeff[3] = d2
```

```
acoeff[4] = c0
acoeff[5] = c1
acoeff[6] = c2
Filtered =  FilterAnalog ( Acceleration, acoeff)
```

The acceleration has a sampling time of 1ms. The transfer function takes the form:

A(p) = (1 + 0.08 * p) / (1 + 0.125 * p + (0.08 * p)^2)

See also:

FiltLP, VibrationFilter, dFilt, ExpoRms

FiltHP

High-pass filter

Declaration:

```
FiltHP ( Data, SvCharacter, SvParameter, SvOrder, SvCutOffFreq ) -> Filtrate
```

Parameter:

Data	Data set to be filtered [NW].
SvCharacter	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Chebychev
	3 : Critical damping
SvParameter	In Chebychev characteristic, the desired ripple (0..3) is set in dB. Otherwise set to 0.
SvOrder	Filter order. Bessel: Range 1..20, Chebychev 1..50, others 1..100.
SvCutOffFreq	Desired cut-off frequency in Hz.
Filtrate	
Filtrate	Filtered data set

Description:

The transferred data set is filtered with a high-pass filter.

The filter coefficients are calculated from the transferred parameters using bilinear transformation.

For accurate filtering, the cut-off frequency should lie below half of the sampling frequency of the output signal. When the frequency is too near to the sampling frequency, the amplitude response of the filter becomes more inaccurate.

Examples:

```
HighPassFiltered = FiltHP(signal, 2, 1, 10, 100)
```

The data set is filtered with a high-pass filter with Chebychev characteristic. The ripple should be 1dB. A filter of tenth order is used and the cut-off frequency should be 100Hz.

See also:

FiltBS, FiltBP, FiltLP, DFilt, FiltHpZ

FiltHpZ

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

High-pass filter without phase shift

Declaration:

```
FiltHpZ ( InputChannel, Characteristic, Parameter, Order, Cutoff frequency ) -> Result
```

Parameter:

InputChannel	The waveform to be filtered, time scaled in seconds.
Characteristic	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Tschebyschew
	3 : Critical damping
Parameter	With Tschebyschew characteristics, the ripple in dB (0..3). Otherwise set to 0.
Order	Filter order: 2, 4, 6, ... 20
Cutoff frequency	Lower cutoff frequency in Hz
Result	
Result	Filtered waveform

Description:

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter coefficients are computed from the parameters specified with bilinear transformation.

For filtering to be useful, the cutoff frequency should be significantly below one-half of the output signal's sampling frequency. The nearer the frequency is to the sampling frequency, the more precise the filter's amplitude response.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

```
f = FiltHpZ ( Vibration, 0, 0, 2, 5 )
```

A 2nd order Butterworth high-pass filter with a lower cutoff frequency of 5Hz is calculated.

See also:

FiltLpZ, FiltHp

FiltLP

Low-pass filter

Declaration:

```
FiltLP ( Data, SvCharacter, SvParameter, SvOrder, SvCutOffFreq ) -> Filtrate
```

Parameter:

Data	Data set to be filtered [NW].
SvCharacter	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Chebychev
	3 : Critical damping
SvParameter	In Chebychev characteristic, the desired ripple (0..3) is set in dB. Otherwise set to 0.
SvOrder	Filter order. Bessel: Range 1..20, Chebychev 1..50, others 1..100.
SvCutOffFreq	Desired cut-off frequency in Hz.
Filtrate	
Filtrate	Filtered data set

Description:

The transferred data set is filtered with a low-pass filter.

The filter coefficients are calculated from the transferred parameters using bilinear transformation.

For accurate filtering, the cut-off frequency should lie below half of the sampling frequency of the output signal. When the frequency is too near to the sampling frequency, the amplitude response of the filter becomes more inaccurate.

Examples:

```
LowPassFiltered = FiltLP(signal, 2, 1, 10, 100)
```

The data set is filtered with a low-pass filter with Chebychev characteristic. The ripple should be 1dB. A filter of tenth order is used and the cut-off frequency should be 100Hz.

See also:

FiltBS, FiltBP, FiltHP, DFilt, FiltLpZ, SavitzkyGolay

FiltLpZ

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Low-pass filter without phase shift

Declaration:

```
FiltLpZ ( InputChannel, Characteristic, Parameter, Order, Cutoff frequency ) -> Result
```

Parameter:

InputChannel	The waveform to be filtered, time scaled in seconds.
Characteristic	Filter characteristic
	0 : Butterworth
	1 : Bessel
	2 : Tschebyschew
	3 : Critical damping
Parameter	With Tschebyschew characteristics, the ripple in dB (0..3). Otherwise set to 0.
Order	Filter order: 2, 4, 6, ... 20
Cutoff frequency	Upper cutoff frequency in Hz
Result	
Result	Filtered waveform

Description:

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter coefficients are computed from the parameters specified with bilinear transformation.

For filtering to be useful, the cutoff frequency should be significantly below one-half of the output signal's sampling frequency. The nearer the frequency is to the sampling frequency, the more precise the filter's amplitude response.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

```
f = FiltLpZ ( Vibration, 0, 0, 2, 150 )
```

A 2nd order Butterworth low-pass filter with an upper frequency of 150Hz is calculated.

See also:

FiltHpZ, FiltLP, SavitzkyGolay

Flag?

This function queries the status (on/off) of special waveform attributes.

Declaration:

```
Flag? ( Data, SvFlag ) -> SvOnOrOff
```

Parameter:

Data	Waveform whose attributes are to be queried.
SvFlag	Attribute selection
	0 : The current data format of the waveform is fixed, meaning that in subsequent processing, the data format (and for integer-formats, the scaling information) is retained if at all possible.
	1 : The values of the data set can be interpreted as color information for 1 pixel of an image. Only allowed for data formats '4 byte unsigned' (color information is encoded as RGB-value) or '1 byte unsigned' (color information is encoded as grey scale value in the range 0..255). By default, the attribute is only set by import filters for image files or special functions such as VpGetImages(). It is used by the curve window to optimize the display of image data.
SvOnOrOff	
SvOnOrOff	OnOrOff
	0 : Off
	1 : On

Description:

This function gets certain data set attributes, which can only take the boolean values [On] (or "True") or [Off] (or "False").

Examples:

A data set is loaded and smoothed. If the data set is recorded in an integer format and this format is fixed, the user is prompted to revoke the established data format before smoothing begins. Otherwise, the smoothing results would be recorded in an integer format whose resolution often isn't sufficient for the purpose.

```
FileLoad("test.dat", "", 0)
IF DataFormat?(test) > 1
   IF Flag?(test,0)
      IF BoxMessage("Question", "Cancel fixed data format?", "?4")
         SetFlag(test, 0, 0)
      END
   END
END
test = Smo5(test)
```

See also:

SetFlag, SetDataFormat, RGB, VpGetImages

Flipflop

Available in: Professional Edition and above

FlipFlop

Declaration:

```
Flipflop ( RJ, SK, Type ) -> Result
```

Parameter:

RJ	Input data R/J
SK	Input data S/K
Type	Type of Flip Flop
	"RS" : RS Flip Flop
	"JK" : JK Flip Flop
Result	
Result	Result

Description:

If the input data are zero, it is interpreted as logically zero (false, low). Else as logically 1 (true, high).

Both input channels must have the same time base, length and structure (segments and events).

RS Flip Flop

Returns a 1 for the state H and a 0 for the state L. Beginning with the state L, the state H is adopted if S is nonzero and R equals zero. If S equals zero and R is nonzero, then the state L is adopted. For the other two combinations of input values, the state remains intact.

JK Flip Flop

Returns a 1 for the state H and a 0 for the state L. Beginning with the state L, the state H is adopted if J is nonzero and K equals zero. If J equals zero and K is nonzero, the state L is adopted. If both J and K equal zero, the state remains intact. If both J and K are nonzero, then the state is switched.

Examples:

RS Flip Flop

```
RS = Flipflop ( R, S, "RS" )
; R = 0 1 1 0 1 0 1 0 1 0 1 0
; S = 0 0 1 1 1 0 0 0 1 1 1 0
;RS = 0 0 0 1 1 1 0 0 0 1 1 1
```

The LED is switched on if the signal exits the range from 0 to 8 and is only switched off again if the signal is within the range from 1 to 6.

```
LED = Flipflop( (K1 < 6) AND (K1 > 1), (K1 > 8) OR  (K1 < 0), "RS" )
```

JK Flip Flop

```
JK = Flipflop ( J, K, "JK" )
; J = 0 0 1 1 1 0 1 1 0
; K = 0 1 1 1 0 0 1 1 1
;JK = 0 0 1 0 1 1 0 1 0
```

The LED blinks as long as the signal is greater than 9.

```
GT = (Signal > 9)
LED = Flipflop( GT, GT, "JK" )
```

See also:

Monoflop

Floor

Next lower or equal integer

Declaration:

```
Floor ( Parameter ) -> Result
```

Parameter:

Parameter	Parameter. Allowed types: [ND],[XY].
Result	
Result	Integer potion of the parameter

Description:

The integer part of a number is the next smaller or equal integer. For example, 1.0, 1.1, 1.9 this would be the integer 1.0; for -3.1, -3.4, -4.0, this would be -4.0.

Remarks

- The x-coordinate(s) of the parameter and the result are the same.
- The parameter may be structured (events/segments).
- The unit remains unchanged.
- Please note that rounding errors may occur during calculations with real numbers. For example, Floor (Sqr (Sqrt (2))) returns the value 1, not 2 as expected. For more reliable (but not as exact!) use of the Floor function, add a small value to the parameter of Floor, e.g. Floor (1e-6 + Sqr (Sqrt (2))).
- The Floor() function is especially useful for various rounding operations. However these can often be more conveniently performed with the function Round().

Examples:

A number is rounded to the next higher integer:

```
rounded = Floor(number + 0.5)
Here, the values of a data set oare rounded to 2 decimal places:
```

```
NDrounded = 0.01 * Floor(0.5 + 100 * NDdata)
```

Simulates the discrete levels of an 8-bit AD-converter: 8-bits correspond to 256 levels; the converter has an analog input range of 10V:

```
NDdiscrete = Floor(NDdata* 256 / 10) * 10 / 256
```

See also:

Round

FOR

Counter-controlled loop

Declaration:

```
FOR SvCounter = SvStart TO SvEnd STEP SvStep
```

Parameter:

SvCounter	Counter variable
SvStart	The counter variable is set to this value when the loop is first entered.
SvEnd	The loop is aborted if the counter variable exceeds this value (counting forwards) or falls below it (counting backwards).
SvStep	After each iteration of the loop, the counter variable is changed by this amount. This specification is optional; the default is +1, i.e. the counter variable is increased by 1 upon each iteration.

Description

The end of the loop is denoted using the command END.

The increment can also be negative (backwards counting loop). The initial value must then be greater than or equal to the end value.

In the loop, it is possible to use the commands BREAK and CONTINUE in order to interrupt execution of the instructions prematurely.

The counter variable may be changed within the loop, but not deleted, nor have its type changed.

The construct

```
FOR i = start TO end STEP add
   ;...instructions
END
```

is equivalent to:

```
i = start
WHILE i <= end
   ;...instructions
   i = i + add
END
```

Examples:

Calculating the factorial of 5:

```
F = 1
FOR i = 2 TO 5
   F = F * i
END
```

In a data group, all channels are deleted whose maximum is less than 0.

```
n = GrChanNum?(group)
FOR i = n TO 1 STEP -1
   IF max(group:[i]) < 0
      GrChanDel(group, i)
   END
END
```

All files having the extension "*.dat" in a specified folder are identified and enumerated in a loop. If the file time is after a fixed cutoff date, the file is loaded and processed. After a maximum of 10 loaded files, processing is cancelled.

```
list  = FsFileListNew("c:\imc\dat", "*.dat", 0, 0, 0)
count = FsFileListGetCount(list)
loaded = 0
deadline = TimeJoin(1, 1, 2012, 0, 0, 0)
FOR i = 1 TO count
   time = FsFileListGetTime(list, i)
   IF time < deadline
      CONTINUE
   END
   ; load and process file
   TxName = FsFileListGetName(list, i)
   fh = FileOpenDSF(TxName, 0)
   ; ...
   loaded = loaded +1
   IF loaded = 10
```

```
        BREAK
    END
END
FsFileListClose(list)
```

See also:

FOREACH, WHILE, BREAK, CONTINUE

FOREACH

This command initializes a loop in which a data object's elements are enumerated. For each iteration, the respective current element is assigned to an iteration variable.

Declaration:

```
FOREACH ElementType IterationVariable IN EnumerationVariable
```

Parameter:

ElementType	Specifies the type of elements to be enumerated.
	SAMPLE : Enumerates all values belonging to a data set
	VALUE : Enumerates all numerical values belonging to a data set (unit and other characteristics are omitted). Faster than "SAMPLE". The loop variable may not be changed in the loop.
	SEGMENT : Enumerates all segments belonging to a data set
	EVENT : Enumerates all events belonging to a data set
	CHANNEL : Enumerates the data sets and text variables contained in a data group.
	ELEMENT : The text elements contained in a text array are enumerated.
IterationVariable	Upon each iteration, the current enumeration variable element is added to the iteration variable. Depending on the enumeration type, this can be either a single value, a segment or event of the enumeration variable, or a complete channel belonging to a data group.
EnumerationVariable	Variable whose elements are to be enumerated.

Description

The end of a FOREACH-loop is denoted by the command END.

The run through a FOREACH-loop can be interrupted prematurely by the command BREAK or CONTINUE.

The iteration variable may not be changed within the loop body.

The iteration variable may be changed within the loop body (except for "VALUE"), but the type must remain unchanged.

If the content of the iteration variable has changed, then at the end of a run of the loop (END/BREAK/CONTINUE), the new content is written back into the enumeration variable.

For a read-only enumeration of the numerical values of a data set, "VALUE" is more efficient than "SAMPLE".

The construct

```
FOREACH SAMPLE s IN data
   ; instructions
END
```

is equivalent to

```
i = 1
WHILE i<= Leng?(data)
   s = data[i]
   ; instructions
   data[i] = s ; only if s has changed
   i = i+1
END
```

The construct

```
FOREACH CHANNEL c IN group
   ; instructions
END
```

is equivalent to

```
i = 1
WHILE i<= GrChanNum?(group)
   c = group:[i]
   ; instructions
   group:[i] = c  ; Only if c has been changed.
   i = i+1
END
```

The enumeration variable type must be compatible with the enumeration type. For instance, if EVENT is specified as the type, the enumeration variable must be an event-based data set. The enumeration type CHANNEL, by contrast, requires a data group.

The enumeration can not be conducted across segment- or event boundaries. For instance, if SAMPLE is used and the enumeration variable is a segmented data set, then the segment desired must additionally be selected:

```
FOREACH SAMPLE s IN SegmentedData[12]
```

In order to enumerate all of a data set's values with events and segments, it is also possible to nest multiple loops:

```
FOREACH EVENT ev IN data
   FOREACH SEGMENT seg IN ev
      FOREACH SAMPLE s IN seg
         ; instructions
      END
   END
END
```

Use loops with caution. Many tasks for which the use of FOREACH SAMPLE enumerations would be convenient (e.g. searching the data set for particular criteria and possibly editing of located points of interest) can be handled much more efficiently by means of skillful application of mathematical and analytical functions in FAMOS.

Examples:

In a data set [data], interference is to beeliminated. These are identified by a value > 1000. These peaks are replaced by the respectively valid predecessor value.

```
lastValid = 0
FOREACH SAMPLE s in data
   IF s > 1000
      s = lastValid
   ELSE
      lastValid = s
   END
END
```

In an event-based data set, all events are identified whose maximum is > 0 , and these are copied to a new data set.

```
newdata = EMPTY
FOREACH EVENT ev IN data
   IF max(ev) > 0
      EventAppend(newData, ev, 0)
   END
END
```

All of a data group's channels whose standard deviation is higher than 2 are subjected to smoothing:

```
FOREACH CHANNEL c in group
   IF StDev(c) > 2
      c = Smo5(c)
   END
END
```

A text array [AllPathNames] contains the complete path name of files to be loaded:

```
FOREACH ELEMENT path in AllPathNames
   FileLoad(path,"", 0)
END
```

See also:

FOR, WHILE

FrequencyResponse

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

FRF, Frequency response function. Calculated by means of FFT.

Declaration:

```
FrequencyResponse ( InputChannel, OutputChannel, WindowWidth, WindowType, Overlapping, Type [, Base2] ) ->
Result
```

Parameter:

InputChannel	The reference channel. A system excitation. Input channel, scaled in seconds.
OutputChannel	The (delayed) output channel. A system response. InputChannel and OutputChannel have the same time base and are scaled in seconds.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Type	Method of calculating the frequency response function. (x: InputChannel, y: OutputChannel, G: PowerSpectrum, F: Spectrum)
	0 : H1: Gxy / Gxx: Measurement with noisy output signal, calculation by means of autospectrum or cross spectrum
	1 : H2: Gyy / Gyx: Measurement with noisy input signal, calculation by means of autospectrum or cross spectrum
	2 : HV: (Gxy / |Gxy |) * sqrt (Gyy / Gxx): Geometric mean of H1 and H2. Noisy input and output.
	3 : H: Fy / Fx: Not recommended!, OutputSpectrum / InputSpectrum.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is complex, i.e., it has a magnitude and a phase. The phase is given as being in the range -180 ... + 180.

Description:

The frequency response function is calculated by linear averaging of the power spectra.

The averaging is performed on the real and imaginary parts separately.

This calculation becomes significant only if there are many values to be averaged.

Examples:

```
FRF = FrequencyResponse ( Force, Movement, 1000, 0, 50, 0, 0 )
```

This calculates a sequence of 1000 point-power-spectra, which each overlap their neighbors by 50%. A force operates on a mechanical part. The part's movement is measured on the opposite end. The FRF of Type H1 is determined from the averaged power spectra.

See also:

CrossPowerDS, Coherence, PhaseContinuous

FsCopyFile

Copy files or folders

Declaration:

```
FsCopyFile ( TxSourceFile, TxTargetFile, Option, Depth ) -> Status
```

Parameter:

TxSourceFile	Source file name
TxTargetFile	Target file name
Option	Option
	0 : Don't overwrite existing files
	1 : Only copy newer files
	2 : Always overwrite existing files
Depth	Copy only folder contents or also subordinate folders?
	0 : Copy only specified folder
	1 : Include subordinate folders
Status	
Status	Function result status
	0 : Function was successful
	-1 : An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Description:

This function copies files or folders. The parameter Depth can be used to set whether to copy subordinate folders also.

Declaration of source file name

- The name may not be a relative path location.
- The name may contan filecards, if desired.
- If a unique folder or filename is specified, it must already exist.
- If a unique name designates a file, the parameter Depth doesn't matter.

Declaration of target filename

- The name may not be a relative path location.
- The name may not contain wildcards.
- If wildcards are used in the source name, the target name must be an already existing folder.
- If the source is a unique fileame and the target is an existing folder, the filename is adopted by the target.

Examples:

All files of the directory 'H:\new' having the extension *.raw are to be copied to the directory 'd:\archive\0003'. Only newer files are copied.

```
erg=FsCopyFile("H:\new\*.raw","d:\a\0003",1,0)
IF erg=-1
   error$=FsGetLastError()
END
```

The folder "c:\temdat\00001" and all its contents are copied to "g:\archiv\".

```
erg=FsCopyFile("c:\temdat\00001","g:\archiv\",1,1)
IF erg=-1
   error$=FsGetLastError()
END
```

See also:

FsMoveFile, FsGetLastError

FsCreateDirectory

Create folder

Declaration:

`FsCreateDirectory ( TxDirName ) -> Status`

Parameter:

TxDirName	Folder name
Status	
Status	Function result status
	0 : Function was successful
	-1 : An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Description:

This function creates folders, including folders with multiple subordinate folders. A complete path location must be supplied.

The declaration in TxDirName may not contain the wildcards '*' or '?'

The function can return the following values:

- 0 Folder creation successful
- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Examples:

In the existing folder 'L:\Temp', the subordinate folders 'Dat' and '00001' are to be created.

```
dir$="l:/temp/dat/00001"
result=FsCreateDirectory(dir$)
IF result<>0
   error$=FsGetLastError()
END
```

See also:

FsRemoveDirectory

FsDeleteFile

Delete file(s)

Declaration:

`FsDeleteFile ( TxFileName ) -> Amount`

Parameter:

TxFileName	Filename
Amount	
Amount	Amount of deleted files
	>=0 : Amount of deleted files
	-1 : Error. Use FsGetLastError() to obtain description.

Description:

The function deletes one or more files. The filename may contain the widcards '*' and '?' if desired.

Read-only files cannot be deleted using this function.

The fnction returns the numbr of files deleted.

The complete path location must be specified.

Any errors occurring can be investigated using the function FsGetLastError().

Examples:

All files from the folder c:\test having the extension .raw are deleted.

```
file$="c:\test\*.raw"
n=FsDeleteFile(file$)
error$=FsGetLastError()
```

See also:

FsMoveFile, FsGetLastError

FsDlgSelectDirectory

Folder selection dialog

Declaration:

```
FsDlgSelectDirectory ( TxTitle, TxStartDir, Option ) -> TxDirname
```

Parameter:

TxTitle	Dialog title bar
TxStartDir	Starting folders
Option	Option
	0 : Standard
	1 : With edit field for directory name
TxDirname	
TxDirname	Complete folder name

Description:

This function allows a folder to be selected via a dialog.

- If TxTitle is empty, "Choose folder" is used as the title.
- If the starting folder specified doesn't actually exist, the FAMOS data folder serves as the starting folder.
- The function returns the path location of the selected folder.
- If the dialog is aborted, the return value is empty.
- Position and size of the dialog can be set with the function SetBoxPos().

Examples:

```
dir$ = FsDlgSelectDirectory("Folder", "c:\copy", 0)
IF TLeng(dir$) > 0
   FileListID = FsFileListNew(dir$, "*.*", 2, 1, 0)
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
      file$=FsFileListGetName(FileListID,i)
      ;
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetName, FsFileListClose, FsGetLastError

FsDlgSelectFiles

File selection dialog

Declaration:

```
FsDlgSelectFiles ( TxTitle, TxStartDir, TxPattern, Reserved ) -> FileListID
```

Parameter:

TxTitle	Dialog title bar
TxStartDir	Starting folder
TxPattern	File filter
Reserved	Reserved parameter, always 0
FileListID	
FileListID	File list ID value
	>=1 : File list ID value
	-1 : Error. Use FsGetLastError() to obtain description.
	-2 : Dialog aborted.

Description:

This function allows multiple files to be selected via a dialog.

- If no string is entered for TxTitel, the dialog has a default title bar.
- If the entry TxStartDir for the starting folder is invalid or empty, the FAMOS data folder is used.
- If the parameter TxPattern is empty, then the filename is *.* and the file type is "All files" (*.*).
- If the TxPattern value is "*.raw", the filename is *.raw and the option for the file type control is "All files(*.*)".
- If the TxPattern value is "Measurement file | *.dat", the filename is *.raw and the option for the file type control is"Measurement data (*.dat)" of "All files(*.*)".

Examples:

```
FileListID=FsDlgSelectFiles("Auswahl","c:\copy","*.*",0)
n=FsFileListGetCount(FileListID)
i=1
WHILE i <= n
      file$=FsFileListGetName(FileListID,i)
      ;
      i=i+1
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetName, FsFileListClose, FsGetLastError

FsFileExists

Verifies whether the file or the folder exists

Declaration:

```
FsFileExists ( TxPathName ) -> Result
```

Parameter:

TxPathName	Pathname
Result	
Result	Result
	-1 : Error
	0 : File or folder does not exist
	1 : The pathname matches an existing file
	2 : The pathname matches an existing folder

Description:

With this function it is possible to test whether a file or a folder exists.

The specification in TxPathName may not contain global characers such as '*' or '?'

In case of error, the function returns the value -1. The error cause can be determined using the function FsGetLastError().

FsFileListClose

Deletes file list

Declaration:

```
FsFileListClose ( FileListID )
```

Parameter:

FileListID	File list ID

Description:

The function deletes either all file lists or a particular one. Any file lists no longer required should be disposed of using this function.

The File-Kit can administer a maximum of 20 file lists.

If -1 is entered as the FileListID parameter, all existing file lists are deleted.

Examples:

The example shows a typical application of the function FsFileListClose().

A file list is created using the function FsDlgSelectFiles().

An evaluation of the contents is performed in the program loop.

In the end, the file list is deleted.

```
FileListID = FsDlgSelectFiles("Select file", "c:\copy", "*.*", 0)
IF FileListID>0
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
      ro = FsFileListGetAttribute(FileListID, i, 1)
      :
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetName, FsFileListGetSize, FsFileListGetTime, FsFileListGetAttribute, FsDlgSelectFiles, FsGetLastError

FsFileListGetAttribute

Determines a file attribute of a file from the file list

Declaration:

```
FsFileListGetAttribute ( FileListID, Index, Attribute ) -> OnOff
```

Parameter:

FileListID	File list ID value
Index	Index of a file list entry. The index begins at 1 and ends with the number determined by the function FsFileListGetCount().
Attribute	Desired file attribute
	1 : Read-only
	2 : Hidden
	3 : Archive
OnOff	
OnOff	File attribute
	-1 : Error. Use FsGetLastError() to obtain description.
	0 : File attribute is not set
	1 : File attribute is set

Description:

This function can be used to determine individual file attributes of a file or folder from the file list.

If the list's ID value is invalid or if the entry's index is outside of the present range, -1 is returned. The function FsGetLastError() can be used to determine the error cause.

Examples:

In this example, the read-only attribute of all files and folders from the file list is polled.

Multiple files can be selected in a selection dialog. These files are internally read in to a file list and subsequently read out.

In this casem the read-only attribute is determined using the function FsFileListGetAttribute().

```
FileListID = FsDlgSelectFiles("Select file", "c:\copy", "*.*", 0)
IF FileListID>0
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
      ro = FsFileListGetAttribute(FileListID, i, 1)
      :
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetName, FsFileListGetSize, FsFileListGetTime, FsFileListClose, FsDlgSelectFiles, FsGetLastError

FsFileListGetCount

Determines number of file list entries

Declaration:

`FsFileListGetCount ( FileListID ) -> Amount`

Parameter:

FileListID	File list ID-value
Amount	
Amount	Amount of entries in file list
	>=0 : Number of entries in file list
	-1 : Error. Use FsGetLastError() to obtain description.

Description:

The function determies the number of entries in a file list. The file list was previously created by the function FsFileListNew(). The ID value returned by the function FsFileListNew() must be provided as the parameter. If the function value is -1, an error has occurred. The error cause can be determined using the function FsGetLastError().

Examples:

Read out and determine the amount of all files with the extension DAT from the folder 'c:\data'

```
FileListID= FsFileListNew("c:\data\", "*.dat", 2, 1, 1)
n=FsFileListGetCount(FileListID)
```

See also:

FsFileListNew, FsFileListGetName, FsFileListGetSize, FsFileListGetTime, FsFileListGetAttribute, FsFileListClose, FsGetLastError

FsFileListGetName

Determines file names in a file list

Declaration:

`FsFileListGetName ( FileListID, Index ) -> TxPathname`

Parameter:

FileListID	File list ID value
Index	Index of a file list entry. The index begins at 1 and ends with the number determined by the function FsFileListGetCount().
TxPathname	
TxPathname	Complete filename. The string is empty if an error occurs.

Description:

This function reads filenames out of a file list. The file list must previously be created using the function FsFileListNew() or FsDlgSelectFiles().

If the list's ID value is invalid or the entry's index is outside of the present range, an empty string is returned. The error cause can be determned with the help of the function FsGetLastError().

Examples:

Multiple files can be selected in a selection dialog.

These files are read in to a file list internally and subsequently read out.

The function FsFileListGetName() finds the filenames.

```
FileListID = FsDlgSelectFiles("Select file", "c:\copy", "*.*", 0)
IF FileListID>0
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
      file$=FsFileListGetName(FileListID,i)
      ;
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetSize, FsFileListGetTime, FsFileListGetAttribute, FsFileListClose, FsDlgSelectFiles, FsGetLastError

FsFileListGetSize

Return the size of a file from a file list

Declaration:

```
FsFileListGetSize ( FileListID, Index ) -> Size
```

Parameter:

FileListID	File list ID value
Index	Index of a file list entry. The index begins at 1 and ends with the number determined by the function FsFileListGetCount().
Size	
Size	File size in Bytes
	>=0 : GFile size in Bytes
	-1 : Error. Use FsGetLastError() to obtain description.
	-2 : The entry is a folder.

Description:

This function finds the size of a file from the file list. The file list must previously be created using either of the functions FsFileListNew() or FsDlgSelectFiles().

If the list ID value is invalid of if the entry's index is outside of the present range, -1 is returned. The function FsGetLastError() can be used to determne the error cause.

If the entry is for a folder, the return value is -2.

Examples:

Multiple files can be selected in a selection dialog.

These files are read in to a file list internally and subsequently read out.

The file size is found by the function FsFileListGetSize().

```
FileListID = FsDlgSelectFiles("Select file", "c:\copy", "*.*", 0)
IF FileListID>0
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
      size=FsFileListGetSize(FileListID,i)
      ;
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetName, FsFileListGetTime, FsFileListGetAttribute, FsFileListClose, FsDlgSelectFiles, FsGetLastError

FsFileListGetTime

Returns the time last modified for a file from the file list

Declaration:

```
FsFileListGetTime ( FileListID, Index ) -> Time
```

Parameter:

FileListID	File list ID value
Index	Index of a file list entry. The index begins at 1 and ends with the number determined by the function FsFileListGetCount().
Time	
Time	Time of last modification in FAMOS-format
	>=0 : File time
	-1 : Error. Use FsGetLastError() to obtain description.

Description:

The function returns the time a file from the file list was last modified. The file list must previously have been created using either of the functions FsFileListNew() or FsDlgSelectFiles().

If the list's ID value is invalid or if the entry's index is outside of the present range, -1 is returned. The function FsGetLastError() can be used to determine the error cause.

If the entry is for a folder, the folder's time of creation is returned.

Examples:

Multiple files can be selected in a selection dialog. These files are internally read in to a file list and subsequently read out.

The file time is determined using the function FsFileListGetTime(). This is available in FAMOS-format.

```
FileListID = FsDlgSelectFiles("Select file", "c:\copy", "*.*", 0)
IF FileListID>0
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
      time=FsFileListGetTime(FileListID,i)
      time$=TimeInText(time,3)
      ;
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListNew, FsFileListGetCount, FsFileListGetName, FsFileListGetSize, FsFileListGetAttribute, FsFileListClose, FsDlgSelectFiles, FsGetLastError

FsFileListNew

Read-in of files and folders

Declaration:

```
FsFileListNew ( TxPathname, TxFilePattern, Selection, Depth, Sort ) -> FileListID
```

Parameter:

TxPathname	Folder
TxFilePattern	File mask for read-in of files and folders
Selection	Selection of data to read in.
	0 : Only files
	1 : Only folders
	2 : Files and folders
Depth	Read-in folder only or also subordinate folders
	0 : Only specified folder
	1 : Include subordinate folders
Sort	Sorting criterion
	0 : Unsorted
	1 : Name (alphabetically)
	2 : Time
	3 : Size
	4 : Name (natural sort order)
FileListID	
FileListID	File list ID-value
	>=1 : Valid file list ID-value
	-1 : Error. Use FsGetLastError() to obtain description.

Description:

This function determines the files and/or subordinate folders belonging to a specified folder. The file and folder names are gathered into a file list. In this list, the entries are sorted by a variety of criteria. The function FsFileList* can be used to read out the list contents. If executed sucessfully, the function returns an ID-value for the file list. This value must be provided as the parameter to the FsFileList* functions.

The file kit can manage a maximum of 20 file lists at the same time. File lists that are no longer required should be closed with the FsFileListClose() function.

- With the TxFilemask, special files can be read in. If, for instance, "*.raw" is specified, only files with this extension are included. If folders are also to be included, enter "*.*" here.
- If the character "|" is added to the file mask, then all files and folders are returned, which do not match the mask.
- The depth is used to define whether only the files and folders belonging to the specified folder are to be found, or also those of all folders below it are to be included.
- Sorting can be performed according to name, size, or age. No matter which criterion is used, folders are listed first. If sorted by size, the files are listed from largest to smallest. If sorted by age, the files are listed from the most recent to the oldest.
- With 'natural sort order' (Sort = 4), digits in the names are considered as numerical value rather than text. E.g. 'Channel2' is ordered before 'Channel10'. The Windows Explorer (since Windows 7) uses the same sort order.
- Multithreading: The Filelist-ID returned by the function is valid globally (for all execution threads).

Examples:

The function FsDlgSelectDirectory() is used to set a folder in which all files and folders are to be registered. This also includes the subordinate folders. The entries found are ordered in the list alphabetically.

```
dir$ = FsDlgSelectDirectory("Folder", "c:\copy",0)
IF   TLeng(dir$) > 0
   FileListID= FsFileListNew(dir$, "*.*", 2, 1, 1)
   n=FsFileListGetCount(FileListID)
   i=1
   WHILE i <= n
```

```
      file$=FsFileListGetName(FileListID,i)
      ;
      i=i+1
   END
   FsFileListClose(FileListID)
END
```

See also:

FsFileListGetCount, FsFileListGetName, FsFileListGetSize, FsFileListGetTime, FsFileListGetAttribute, FsFileListClose, FsGetLastError

FsGetDiskFreeSpace

Determine free space on a drive

Declaration:

```
FsGetDiskFreeSpace ( TxDrive ) -> SvFreeSpace
```

Parameter:

TxDrive	Specification of drive
SvFreeSpace	
SvFreeSpace	Free space on drive

Description:

The functin determnes how much space is free on a drive.

Examples of valid declarations for the parameter TxDrive would be 'c', 'c:', 'c:\', 'c:\test', 'c:\test\file.dat'.

The function returns the following values:

- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.
- >=0 Free space in Bytes

Examples:

The free space on drive 'D:' is determined.

```
Freespace = FsGetDiskFreeSpace("d:")
IF Freespace=-1
   error$=FsGetLastError()
END
```

See also:

FsGetLogicalDrives, FsGetDriveType

FsGetDriveType

Determine a logical drive's type

Declaration:

`FsGetDriveType ( TxDrive ) -> SvType`

Parameter:

TxDrive	Declaration of drive
SvType	
SvType	Drive type
	-1 : Error
	2 : Floppy or ZIP drive
	3 : Hard drive
	4 : Network drive
	5 : CD-ROM drive
	6 : RAM drive

Description:

The function determinesa drive's type.

Examples of valid declarations for the parameter TxDrive would be: 'c', 'c:', 'c:\', 'c:\test', 'c:\test\file.dat'.

The function returns the following values:

- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.
- 2 Floppy or ZIP drive
- 3 Hard drive
- 4 Network drive
- 5 CD-ROM drive
- 6 RAM-drive

Examples:

The drive type of the logical drive 'F' is determined.

```
drivetyp = FsGetDriveType("f:\")
IF drivetyp=-1
   error$=FsGetLastError()
END
```

See also:

FsGetLogicalDrives, FsGetDiskFreeSpace

FsGetFileAttributes

Determine file attributes

Declaration:

```
FsGetFileAttributes ( TxFileName, Attribute ) -> OnOff
```

Parameter:

TxFileName	Filename
Attribute	File attribute
	1 : Read-only
	2 : Hidden
	3 : Archive
OnOff	
OnOff	File attribute
	-1 : Error
	0 : File attribute not set
	1 : File attribute is set

Description:

The function discloses whether certain file attributes are set.

The declaration in TxFileName may not contain the wildcards '*' or '?'

In case of error, the function returns -1. The function FsGetLastError() can be used to determine the error cause.

Examples:

The "Read-only" status of a file is determined and switched.

```
file$="\\server\copy\dirwal.h"
ro=FsGetFileAttributes(file$,1)
IF ro=1
   ro=0
ELSE
   ro=1
END
ok=FsSetFileAttributes(file$,1,ro)
```

FsGetFileNames

Read-in of files and folders.

Declaration:

```
FsGetFileNames ( TxPathname, TxFilePattern, Selection, Depth, Sort ) -> TxArrayFileNames
```

Parameter:

TxPathname	Folder
TxFilePattern	File mask for read-in of files and folders
Selection	Selection of data to read in.
	0 : Only files
	1 : Only folders
	2 : Files and folders
Depth	Read-in folder only or also subordinate folders
	0 : Only specified folder
	1 : Include subordinate folders
Sort	Sorting criterion
	0 : Unsorted
	1 : Name (alphabetically)
	2 : Time
	3 : Size
	4 : Name (natural sort order)
TxArrayFileNames	
TxArrayFileNames	Text-Array with the filenames and folder names identified

Description:

This function determines the files and/or subordinate folders belonging to a specified folder. The file and folder names are gathered into a file list. In this list, the entries are sorted by a variety of criteria. The return value is of the type text-array.

In case of error, an empty text-array is returned.

- With the TxFilemask, special files can be read in. If, for instance, "*.raw" is specified, only files with this extension are included. If folders are also to be included, enter "*.*" here.
- If the character "|" to the file mask, then all files and folders are returned, which do not match the mask.
- The depth is used to define whether only the files and folders belonging to the specified folder are to be found, or also those of all folders below it are to be included.
- Sorting can be performed according to name, size, or age. No matter which criterion is used, folders are listed first. If sorted by size, the files are listed from largest to smallest. If sorted by age, the files are listed from the most recent to the oldest.
- With 'natural sort order' (Sort = 4), digits in the names are considered as numerical value rather than text. E.g. 'Channel2' is ordered before 'Channel10'. The Windows Explorer (since Windows 7) uses the same sort order.

Examples:

From a given folder all files matching "*.dat" are loaded:

```
fileNames = FsGetFileNames("c:\imc\dat","a*.dat", 0, 0, 0)
FOREACH ELEMENT name in fileNames
   FileLoad( name, "", 0)
END
```

See also:

FsGetFileSize, FsGetFileTime, FsGetFileAttributes, FsGetLastError

FsGetFileSize

Determines the size of a file

Declaration:

```
FsGetFileSize ( TxFileName ) -> Size
```

Parameter:

TxFileName	Filename
Size	
Size	File size in Bytes

Description:

The function returns a file's size in Bytes.

The declaration in TxFileName may not contain the wildcards '*' or '?'

In case of error, the function returns the follwoing results:

- -1 Error. Use FsGetLastError() to obtain description.
- -2 Error. Object is folder, not file.

Examples:

The size of the file round.raw is determined.

```
file$="l:\temp\00284\round.raw"
size=FsGetFileSize(file$)
IF size=-1
   error$=FsGetLastError()
ELSE
   IF size=-2
      error$="Object is a folder"
   END
END
```

See also:

FsFileListGetSize

FsGetFileTime

Determines time of last modification of file, or time of creation of folder

Declaration:

```
FsGetFileTime ( TxFileName ) -> Time
```

Parameter:

TxFileName	Filename
Time	
Time	Time in FAMOS-format

Description:

If a file is concerned, the time of last modification is returned.

If TxFileName is a folder, the time it was created is returned.

The return value representing the time is in FAMOS-format.

The function returns -1 if an error occurs. The error cause can be determined using the function FsGetLastError().

The declaration in TxFileName may not contain the wildcards '*' or '?'

Examples:

The data of creation of the folder '00284' is determined and expressed as a text.

```
file$="l:\temp\00284\"
time=FsGetFileTime(file$)
time$=TimeInText(time,3)
```

The date last modified for the file 'sparn01.raw' is determined and expressed as a text.

```
file$="l:\temp\00284\sparn01.raw"
time=FsGetFileTime(file$)
IF time=-1
   error$=FsGetLastError()
ELSE
   time$=TimeInText(time,3)
END
```

FsGetLastError

Determines the last error occurring in an Fs*-function

Declaration:

`FsGetLastError ( ) -> TxError`

Parameter:

TxError	
TxError	Error text

Description:

The function returns an error text about the last error to occur.

All Fs*-functions returning the function value -1 indicate that an error occurred during execution. The exact error cause can be queried by this function.

The last error to occur is deleted upon calling an Fs*-function. The exceptions are the functions FsGetLastError() and FsGetLastErrorNumber().

For an error evaluation according to programming aspects, the function FsGetLastErrorNumber() is better adapted. It returns an error number.

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

See also:

FsGetLastErrorNumber

FsGetLastErrorNumber

Determine last error number of an Fs*-function

Declaration:

```
FsGetLastErrorNumber ( ) -> SvError Number
```

Parameter:

SvError Number	
SvError Number	Error number
	10001 : No filename specified
	10002 : Invalid file attribute
	10003 : Invalid time specification
	10004 : Invalid option
	10005 : Invalid selection
	10006 : Invalid sorting criterion
	10007 : Invalid file list ID
	10008 : Invalid file list index
	10010 : Too many file lists
	10011 : Cannot create new file list
	10012 : Folder is not empty
	10013 : Folder does not exist
	10014 : The current folder cannot be deleted
	10015 : For security reasons, root-folders cannot be deleted with this function.
	10016 : Relative path location not allowed
	10020 : File or folder name contains '*' or '?'
	10021 : File or folder doesn't exist
	10022 : Invalid specification for file or folder name
	10023 : Access to file denied (folder is read-only)
	10024 : Drive not ready
	10025 : Drive is full
	10026 : Invalid path location
	10027 : File exists already
	10028 : Drive doesn't exist
	10029 : Target name must be an existing folder.

Description:

The function returns the error number of last error to occur.

All Fs*-functions returning the function value -1 indicate that an error occurred during execution. The exact error cause can be queried by this function.

The last error to occur is deleted upon calling an Fs*-function. The exceptions are the functions FsGetLastError() and FsGetLastErrorNumber().

If a text is needed as an error desription, use the function FsGetLastError().

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

See also:

FsGetLastError

FsGetLogicalDrives

Determine logical drive

Declaration:

```
FsGetLogicalDrives ( ) -> TxDrives
```

Parameter:

TxDrives	
TxDrives	Listing of lettered drives

Description:

The function finds all logical drives present in the computer.

The corresponding letters denoting the drives are returned as strings.

Examples:

After carrying out the instruction, drives has the following content: 'ACDEFGHIJKLMN'.

```
drives = FsGetLogicalDrives()
```

See also:

FsGetDriveType, FsGetDiskFreeSpace

FsGetLongPathName

Conversion of a short filenam to long form

Declaration:

`FsGetLongPathName ( TxShortName ) -> TxLongName`

Parameter:

TxShortName	Short filename
TxLongName	
TxLongName	Long filename

Description:

The function converts a short file or folder name to the long form.

The function can only work if the file or folder actually exists.

The function returns an empty string if an error occurs. The function FsGetLastError() can be used to determine the error cause.

The declaration of the short filename may not contain the wildcards '*' or '?'.

A complete filename must be supplied.

Examples:

A short filename is converted to a long name.

Result: "e:\kit32\Filekit\filemanager.h".

```
file$="e:\kit32\filekit/filema~1.h"
longname$=FsGetLongPathName(file$)
```

See also:

FsGetShortPathName

FsGetParentDirectoryName

Determines a folder's parent folder

Declaration:

```
FsGetParentDirectoryName ( TxDirName ) -> TxParentDirName
```

Parameter:

TxDirName	Folder name
TxParentDirName	
TxParentDirName	Name of the parent folder

Description:

The function finds the parent folder of the folder whose name is specified.

The folder must already exist. In case of an error, an empty string is returned.

Examples:

The parent folder of the folder "d:\imc\dat\" is to be found.

```
parentdir$=FsGetParentDirectoryName("d:\imc\dat\")
```

The variable parentdir$ contains the string "d:\imc". The next call returns "d:\"

```
parentdir$=FsGetParentDirectoryName(parentdir$)
```

All further calls at this point would return "d:\".

See also:

FsGetShortPathName

FsGetShortPathName

Conversion of a long filename to a short one

Declaration:

`FsGetShortPathName ( TxLongFilename ) -> TxShortFilename`

Parameter:

TxLongFilename	Long filename
TxShortFilename	
TxShortFilename	Short filename

Description:

The function converts a long file or folder name to the corresponding short form of the name.

The short filename is in MSDOS 8.3 - format.

The function returns an empty string if an error occurs. The function FsGetLastError() can be used to determine the error cause.

The declaration of the long filename may not contain the wildcards '*' or '?'.

A complete filename must be supplied

Examples:

The long folder name "c:\multimedia files\music\" appears after conversion as "c:\multim~1\music\".

```
dir$="c:\multimedia files\music\"
short$=FsGetShortPathName(dir$)
```

FsMakePath

Compose complete filename

Declaration:

```
FsMakePath ( TxDrive, TxDirectory, TxName, TxExtension ) -> TxFilename
```

Parameter:

TxDrive	Specification of drive
TxDirectory	Specification of directory
TxName	Filename
TxExtension	File extension
TxFilename	
TxFilename	Complete filename

Description:

This function constructs a complete filename from filename components.

The following parameters in TxDrive produce the same result:

- 'c:\'
- 'c:'
- 'c'

In TxDirectory, it doesn't matter if the string begins or ends with a backslash.

The file extension can be entered with or without a preceding period.

Examples:

The filename 'c:\imc\bin\famos.exe' is to be constructed:

```
drive$="c:"
dir$="\mc\bin"
name$="famos"
ext$="exe"
file$=FsMakePath(drive$,dir$,name$,ext$)
```

See also:

FsSplitPath

FsMoveFile

Move files and folders

Declaration:

```
FsMoveFile ( TxSourceFile, TxtargetFile, Option, Depth ) -> Status
```

Parameter:

TxSourceFile	Source file name
TxtargetFile	Target file name
Option	Option
	0 : Don't overwrite existing files
	1 : Only move newer files
	2 : Always overwrite existing files
Depth	Move only folder contents or also subordinate folders?
	0 : Only move folder's contents
	1 : Include subordinate folders
Status	
Status	Function result status
	0 : Function was successful
	-1 : An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Description:

This function moves files or folders. The parameter Depth determines whether all subordinate folders are also to be moved.

Declaration of source file name

- The name may not be a relative path location.
- The name may contain wildcards.
- If a unique folder or filename is specified, it must be an existing one.
- If the unique name designates a file, the depth parameter doesn't matter.

Declaration of target file name

- The name may not be a relative path location.
- The name may not contain wildcards.
- If wildcards are used in the source name, the target name must be an already existing folder.
- If the source is a unique fileame and the target is an existing folder, the filename is adopted by the target.

Examples:

A file 'H:\Temp\2001.dat' is to be moved to 'c:\archiv\2001.raw' and renamed. If the file 'c:\archiv\2001.raw' already exists, no moving takes place.

```
erg=FsMoveFile("H:\Temp\2001.dat","c:\archiv\2001.raw",0,0)
IF erg=-1
   error$=FsGetLastError()
END
```

All files from the folder 'H:\new' with the extension *.raw are to be moved to the folder d:\a\0003. In this case, only relatively new files are moved.

```
erg=FsMoveFile("H:\new\*.raw","d:\a\0003",1,0)
IF erg=-1
   error$=FsGetLastError()
END
```

The folder 'H:\Temp\' including all its subordinate folders and files is moved to the folder e:\dat\0003.

```
erg=FsMoveFile("H:\Temp\","e:\dat\0003",0,1)
IF erg=-1
   error$=FsGetLastError()
END
```

See also:

FsCopyFile, FsRenameFile, FsGetLastError

FsPathCombine

Combines two strings to a file path

Declaration:

```
FsPathCombine ( TxPathName1, TxPathName2 ) -> TxPathName
```

Parameter:

TxPathName1	First pathname to be combined
TxPathName2	Second pathname to be combined
TxPathName	
TxPathName	The composite pathname

Description:

This function joins two path designations to one pathname.

If one of the two specified pathnames is an empty string, the function returns the other pathname.

If TxPathName2 contains an absolute pathname, the function returns TxPathName2.

TxPathName2 as relative path can start with an ".." or ".":

```
FsPathCombine("c:\folder1\folder2", ".\folder3")  => "c:\folder1\folder2\folder3"
FsPathCombine("c:\folder1\folder2", "..\folder3") => "c:\folder1\folder3"
```

If TxPathName2 starts with a "\", it is interpreted as root folder of the current drive (given by TxPathName1):

```
FsPathCombine("c:\folder1\folder2", "folder3")  => "c:\folder1\folder2\folder3"
FsPathCombine("c:\folder1\folder2", "\folder3") => "c:\folder3"
```

In case of error, the returned string is empty.

Examples:

FsPathCombine("","") => \

FsPathCombine("abc", "") => "abc"

FsPathCombine("", "abc") => "abc"

FsPathCombine("c:\abc\", "data\file.xml") => "c:\abc\data\file.xml"

FsPathCombine("c:\abc", ".xml") => "c:\abc\.xml"

FsPathCombine("\\ww\copy\", "\lib") => "\\ww\copy\lib"

FsPathCombine("\\ww\copy\something\else", "\lib") => "\\ww\copy\lib"

FsPathCombine("\\ww\copy\", "\\lib") => "\\lib"

FsPathCombine("lib","c:\ab\datei.exe")=> "c:\ab\datei.exe

See also:

FsMakePath

FsRemoveDirectory

Delete folder

Declaration:

```
FsRemoveDirectory ( TxDirName, Option ) -> Status
```

Parameter:

TxDirName	Folder name
Option	Option
	0 : Empty folder only
	1 : Folder complete with files and subordinate folders
Status	
Status	Function result status
	0 : Function was successful
	-1 : An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Description:

The specified folder is deleted.

The declaration in TxFileName may not contain the wildcards '*' or '?'.

The current folder may not be deleted.

Root-directories ("c:\") cannot be deleted on grounds of security.

With the option 0, the folder is only deleted if it is empty.

Option 1 deletes the folder contents including its subordinate folders and files. Even read-only files in this folder will be deleted. Finally, the folder itself is deleted.

The function returns the following values:

- 0 Folder deletion successful
- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Examples:

The folder '00110' is to be deleted. With Option 0 active, this only happens if this folder is empty.

```
dir$="\\server\imcan\temp\00110"
result=FsRemoveDirectory(dir$,0)
IF result<>0
   error$=FsGetLastError()
END
```

The folder '00110' and its entire contents are to be deleted. With Option 1 active, this happens even if the folder contains read-only files.

```
dir$="\\server\imcan\temp\00110"
result=FsRemoveDirectory(dir$,1)
IF result<>0
   error$=FsGetLastError()
END
```

See also:

FsCreateDirectory, FsGetLastError

FsRenameFile

REnames file or folder

Declaration:

```
FsRenameFile ( TxFileName, TxNewName ) -> Status
```

Parameter:

TxFileName	Current filename
TxNewName	New filename
Status	
Status	Function result status

Description:

An existing file or folder is renamed.

Folders can only be renamed.

Files can be moved at the same time.

The declaration of either may not contain the wildcards '*' or '?'.

The following rules apply to TxFileName:

- The file or folder must exist already.
- The path olcation may not be relative.

The following rule applies to TxNewName:

- A file or folder with this name may not already exist.
- If a filename is specified without a path location, the new file is created in the old file's folder.

The function returns the following values:

- 0 Renaming was successful
- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Examples:

Renaming a file in the FAMOS data folder.

```
existfile$="d:\imc\dat\b.dat"
newfile$="a.dat"
result=FsRenameFile(existfile$,newfile$)
IF result=-1
   error$=FsGetLastError()
END
```

Renaming and moving a file.

```
existfile$="c:\test\b.dat"
newfile$="e:\histroy\file28.dat"
result=FsRenameFile(existfile$,newfile$)
IF result=-1
   error$=FsGetLastError()
END
```

Renaming a folder.

```
dir$="c:\test"
newdir$="c:\histroy"
result=FsRenameFile(dir$,newdir$)
IF result=-1
   error$=FsGetLastError()
END
```

See also:

FsMoveFile, FsGetLastError

FsSetFileAttributes

Change file attributes

Declaration:

```
FsSetFileAttributes ( TxFileName, Attribute, OnOff ) -> Status
```

Parameter:

TxFileName	Filename
Attribute	File attribute
	1 : Read-only
	2 : Hidden
OnOff	Set or delete attribute
	0 : Delete
	1 : Set
Status	
Status	Function result status
	0 : Function was successful
	-1 : An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Description:

This function allows changing of the file attributes read-only and hidden.

The declaration in TxFileName may not contain the wildcards '*' or '?'.

The function returns the followinig values:

- 0 Attribute change successful
- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.

Examples:

The "Read-only" status of a file is determined and switched.

```
file$="\\server\copy\dirwal.h"
ro=FsGetFileAttributes(file$,1)
IF ro=1
   ro=0
ELSE
   ro=1
END
ok=FsSetFileAttributes(file$,1,ro)
```

FsSetFileTime

Change file date and time

Declaration:

```
FsSetFileTime ( TxFileName, Time ) -> Status
```

Parameter:

TxFileName	Filename
Time	Time in FAMOS-format
Status	
Status	Function result status
	0 : Function successful
	-1 : An error has occurred. The function FsGetLastError() can be used to determine the error cause.
	-2 : Specified object is a folder.

Description:

This function allows the time of a file's last modification to be altered.

The file age is changed, even if the file is write-protected.

The function can not be applied to folders.

The declaration in TxFileName may not contain the wildcards '*' or '?'

The function returns the following values:

- 0 Change of file age successful
- -1 An error has occurred. The function FsGetLastError() can be used to determine the error cause.
- -2 Specified object is a folder.

Examples:

The time last modification of the file 'bisch01.raw' is changed by one hour.

```
file$="l:\temp\00284\bisch01.raw"
time=FsGetFileTime(file$)
time=time+3600
ok=FsSetFileTime(file$,time)
IF ok<>0
   error$=FsGetLastError()
END
```

FsSplitPath

Decompose entire filename

Declaration:

```
FsSplitPath ( TxPathname, Option ) -> TxNamePart
```

Parameter:

TxPathname	Complete filename
Option	Part of name to determine.
	0 : Drive
	1 : Folder
	2 : Filename without extension
	3 : Extension
	4 : Filename + extension
	5 : Folder + filename + extension
	6 : Last part of folder entry
	7 : UNC-Share-Name
	8 : Drive + Folder
TxNamePart	
TxNamePart	Part of name

Description:

This function lets you extract particular parts of a string which represents a complete path location (comprising drive, folder and filename).

As an example, for 'c:\imc\bin\famos.exe' the various options return:

- 0 c:
- 1 \imc\bin\
- 2 famos
- 3 .exe
- 4 famos.exe
- 5 \imc\bin\famos.exe
- 6 bin
- 8 c:\imc\bin\

With Option 7, the UNC-Pfad of a local drive can be generated. So-called "UNC-Pfade" always are involved wherever resources in the network (on a "File-Server") are to be used. To keep things simple, a drive is often assigned to a Server-folder which gets lots of use (e.g. as a "workgroup-drive", in Example N:). Thus, one can access a logical drive N:\ instead of \\SERVER\TEMP\, as per UNC notation.

```
unc_name = FsSplitPath("n:\temp", 7)
unc_name contains \\SERVER\TEMP\TEMP\
```

Examples:

The following commands decompose the filename file$. The results are as shown above.

```
file$="c:\imc\bin\famos.exe"
drive$=FsSplitPath(file$,0)
dir$=FsSplitPath(file$,1)
name$=FsSplitPath(file$,2)
ext$=FsSplitPath(file$,3)
nameext$=FsSplitPath(file$,4)
path$=FsSplitPath(file$,5)
subdir$=FsSplitPath(file$,6)
```

See also:

FsMakePath

FsTempFileName

Composes temporary filename

Declaration:

```
FsTempFileName ( Option ) -> TxFileName
```

Parameter:

Option	Option
	0 : Complete path location
	1 : Filename + extension
	2 : Folder only
TxFileName	
TxFileName	Filename or folder name

Description:

This function generates temporary file or folder names. The examples show the results of the individual options.

With Option 0, a unique temporary filename is generated in the Windows Temporary folder.

Option 1 is identical to Option 0. But only the filename without path location is returned.

With option 0 or 1, the function creates an empty file by that name. This file will not be deleted automatically when the program ends; delete this file with FsDeleteFile() as necessary.

Option 2 returns the path location of the Windows Temporary folder.

If the returned string is empty, an error has occurred. The function FsGetLastError() can be used to determine the error cause.

Examples:

The options returned the following results:

```
tempfile$=FsTempFileName(0)
```

Result: C:\WINDOWS\TEMP\FAM2062.TMP

```
tempname$=FsTempFileName(1)
```

Result: FAM2063.TMP

```
tempdir$=FsTempFileName(2)
```

Result: C:\WINDOWS\TEMP\

See also:

GetLastError

Query last error

Declaration:

```
GetLastError ( ) -> TxError
```

Parameter:

TxError	
TxError	Last error

Description:

By using the command OnError(), the error handling in sequences can be configured so that execution of the sequence initially continues in spite of an error occurring. In that case the error cause is recorded and can be retrieved using this function.

Some functions, especially such which come from extension libraries (Kits), return a special return value to indicate non-critical errors. The associated error text can also be found using this function.

After GetLastError() has been called, the internal error memory is deleted.

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

The sequence 'DoSomething' is configured by an OnError("Return")-command in such a way that when an error occurs, the system skips to the end of the sequence. The calling sequence verifies whether an error has occurred and if so displays it in a message box.

```
OnError("ResumeNext")
SEQUENCE DoSomething
err = GetLastError()
IF err <> ""
   BoxMessage("Error in DoSomething", err, "!1")
END
```

A Panel-page is loaded by means of a Kit-function. If the file can not be loaded, a message appears which states the error cause.

```
ok = PnLoad("d:\templates\result.panel")
IF ok <> 0
   BoxMessage("Error", GetLastError(), "!1")
END
```

See also:

OnError, ThrowError

GetOption

Queries various current default settings, such as default folders and settings for math functions.

Declaration:

```
GetOption ( TxOptionName ) -> TxOptionValue
```

Parameter:

TxOptionName	Name of the desired option, see list.
	"Dir.DataFiles" : Default folder for loading/saving measurement data files. Used by the commands LOAD, SAVE and the like, as well as the functions FileOpenDSF() and FileOpenFAS().
	"Dir.Sequences" : Current sequence standard folder. Used by the command SEQUENCE. Upon starting FAMOS, initially set to the folder specified under "Options"/ "Folders". It can be modified using either the command MDIR or the function SetOption(). Otherwise it is the folder from which the last sequence has been loaded.
	"Dir.CurrentProject" : If a project is active, the associated project folder is returned, otherwise an empty text.
	"Dir.CurrentSequence" : While a sequence or dialog is running, contains the folder from which the sequence or dialog, respectively, was loaded. This corresponds to the current FAMOS working folder. Otherwise, an empty text is returned.
	"Dir.DefinitionFiles" : Returns the default folder for definitions files. These include, for example, ASCII-/EXCEL-export templates (*.aet), import filters created with the File-Assistant (*.fas) and definitions files for external DLL-functions (*.def).
	"Func.WarnLevel" : Determines which warning messages triggered during the execution of functions are displayed
	"Func.NoInfoMessages" : Some functions (Stat, LFit) print their results by default in the output window. Output can be suppressed with this option.
	"Func.FFT.Window" : Determines the FFT window function. Used by functions FFT(), Spec() and the like.
	"Func.FFT.Mode" : The FFT algorithm implemented requires the length of the input data set to be a power of 2. This option determines how to deal with other data set lengths.
	"Func.ResultFormat" : Determines the data format in which the functions return their results by default.
	"Func.ErrorBoxes" : Some functions have the choice of responding to an error by posting an error box or by indicating the error (silently) by their return values. To date only pertains to the file function group including FileOpenDSF() and the like.
	"DLLImport.DefinitionFile" : Filename with definitions for external DLL-functions. The functions listed here are imported to FAMOS via the general-purpose DLL-interface and can be used in sequences. The file needs to have been generated by means of the dialog 'Options'/'Register DLL-functions'.
	"Display.DecimalSeparator" : Sets the character displayed for separating real numbers' decimal places.
	"DDE.Text.NumFormat" : Determines the numerical format for sending data in text format by DDE.
	"DDE.Text.Delimeter" : Determines the separator(s) used between 2 numbers when sending data in text-format by DDE.
	"DDE.TimeOut" : Sets the maximum amount of time FAMOS waits for an answer in DDE communication with another application.
	"Units.Ctrl.Compatible" :
	"Units.Display.Greek" :
	"Units.Display.Ohm" :
	"Units.Create.Delim" :
	"Units.Create.Nm" :
	"Units.Create.Pow.1/2" :
	"Units.Create.Pow.2" :
	"Units.Create.Pow.3" :
	"Units.Create.Pow.Neg" :
	"Units.Create.u" :
	"Units.Create.Num.Space" :
	"Units.Create.1e3" :
	"Units.Create.1e-3" :
	"Units.Create./s" :

	"Units.Read.cal" :
	"Units.Read.Exp" :
	"Units.Read.g" :
	"Units.Read.Gs" :
	"Units.Read.hp" :
	"Units.Read.kt" :
	"Units.Read.L" :
	"Units.Read.lb" :
	"Units.Read.oz" :
	"Units.Read.pt" :
	"Units.Read.quot" :
	"Units.Read.s" :
	"Units.Read.ton" :
	"Units.Read.u" :
	"Units.Reduce.C" :
	"Units.Reduce.F" :
	"Units.Reduce.H" :
	"Units.Reduce.J" :
	"Units.Reduce.Ohm" :
	"Units.Reduce.Pa" :
	"Units.Reduce.S" :
	"Units.Reduce.T" :
	"Units.Reduce.Wb" :
TxOptionValue	
TxOptionValue	Value of the current setting. A list of the possible text constants, referenced to the respective specified parameter, is found in the description of the function SetOption().

Description:

This function enables querying of various default settings' values which are applied in the execution of commands and math functions.

If the setting wasn't made explicitly using the command SetOption(), the global default is used (dialogs "Options/Folders", "Options/Functions",...).

Examples:

The default data folder currently used is determined. A sub-folder "Results" of this folder is set as the default and files are saved to this folder.

```
folder = GetOption("Dir.DataFiles")
SetOption("Dir.DataFiles", folder + "\results")
;...saving results
SAVE result "result.dat"
```

See also:

SetOption, FFTOPTION, MDIR, LDIR, SDIR

GetScale

Available in: Professional Edition and above

Requests the scaling

Declaration:

```
GetScale ( Variable, Property ) -> Value
```

Parameter:

Variable	Variable
Property	Which property?
	"factor" : Scaling factor
	"offset" : Scaling offset
	"max" : Maximum displayable value
	"min" : Minimum displayable value
	"minEx" : Extended minimum displayable value
Value	
Value	Value

Description:

The scaling factor and scaling offset are governed by this formula:

[Physical value] = [integer number] * scaling factor + scaling offset

The scaling factor and scaling offset are only available with integer formats.

Minimum and Maximum are not values which are actually present in the data set, but represent the minimum/maximum which can be displayed by the data format used.

The scaling factor represents the value of 1 LSB (least significant bit). This is also referred to as the resolution.

To calculate the Minimum for signed integers, the negative of the maximum integer is used, so for example -127 for 1 Byte signed. Compatible with SetDatFormat().

To calculate the extended Minimum for signed integers, the negative of the maximum integer, minus 1 is used, so for example -128 for 1 Byte signed.

If a data set consisting of tqo components is specified, then for an XY data set, .Y is used, for RI accordingly .R and for BP accordingly .B. The user is able to personally select the component in the call, for example by appending .Y.

The numerical value is returned without any unit.

With Timestamp ASCII data, the information on the time stamps is issued.

For real numbers, -1e35 and 1e35 are returned as the minimum and maximum values.

There are data sets having multiple events in which the properties may differ for each event. E.g. the X values of XY data from imc DEVICES and imc STUDIO. In such cases, you can request the property for only one specific event (index).

Examples:

LSB of a measured channel whose format is 2 Byte signed integer

```
LSB = GetScale ( Channel_01, "factor")
```

Offset (shifted midpoint) of an XY-channel's Y-track

```
Offset = GetScale ( data.y, "offset")
```

See also:

SetDataFormat, DataFormat?

GetSystemInfo

Inquiry for information on the program and operating system.

Declaration:

```
GetSystemInfo ( TxInfo, TxParameter ) -> CurrentValue
```

Parameter:

TxInfo	Name of the information desired; see list.
	"Famos.Version" : Returns the current program version of FAMOS as an integer, where the major release is denoted by the hundreds decimal position and the minor release forms the last digits (with prefixed zero if needed), e.g. 603 for FAMOS 6.3. The revision number is not taken into account.
	"Famos.VersionString" : Returns the current FAMOS program version as text in the form "MAJORRELEASE.MINORRELEASE Rev. REVISIONNUMBER"; for instance "6.3 Rev. 2".
	"Famos.Edition" : Returns a number characterizing the edition currently being run. -1: Runtime, 0: Reader, 1: Standard, 2: Professional, 3: Enterprise.
	"Famos.IsX64" : Returns a 1 if the version is FAMOS x64. A 0 indicates that the version is x86 (32 bits).
	"Famos.IsKitAvailable" : Returns 1 when the FAMOS expansion library specified in the 2nd parameter is present and can be used; else 0. The search for the Kit can reference either its filename (without folder) or according to the following abbreviations: "CLS": Class-counting Kit. "SPC"; Spectral Analysis Kit. "OTR": Order-tracking Kit. "VPL": Video Player Kit. "ODS": ODS-Browser-Kit. "RKT": R-Kit. "RWY": Railway-Kit.
	"Famos.IsDLLFunctionAvailable" : Returns 1 if the function name specified in the 2nd parameter is an external DLL-function currently registered in FAMOS; else 0. Functions embedded in FAMOS via the general-purpose DLL-interface are registed by means of the dialog 'Options'/'Register DLL-Functions' or by means of the function SetOption("DLLImport.DefinitionFile",...).
	"Famos.IsSeqFunctionAvailable" : Returns 1 when the function name specified as the 2nd parameter is a sequence function currently registered in FAMOS; else 0. The complete function name (including a preceding '!') must be specified. See example #2. FAMOS looks for sequence libraries (file extension .sqf) in the folders specified under 'Options'/'Libraries' and registers all sequence functions it contains (these are displayed in the Functions list in the 'Libraries' folder).
	"Famos.Path.SampleProjects" : Provides the directory for the example projects supplied with the FAMOS installation. With a standard installation, this is the directory "C:\Users\Public\Documents\imc\imc FAMOS_Demo Projects". [Supported from V2023]
	"Famos.Path.SampleData" : Provides the directory for the sample measurement files supplied with the FAMOS installation. With a standard installation, this is the directory "C:\Users\Public\Documents\imc\imc FAMOS_Demo Projects_Demo Files\dat". [Supported from V2023]
	"System.Path.AppData" : Returns the Windows default folder for program-specific data. A typical path is "C:\Documents and Settings\USERNAME\Application Data" (Windows XP) or "C:\Users\USERNAME\AppData\Roaming" (Windows 7).
	"System.Path.CommonAppData" : Returns the Windows default folder for program-specific data which are available to all users. A typical path is "C:\Documents and Settings\All Users\Application Data" (Windows XP) or "C:\ProgramData" (Windows 7).
	"System.Path.Documents" : Returns the Windows default folder for user documents. A typical path is "C:\Documents and Settings\USERNAME\My Documents" (Windows XP) or "C:\Users\USERNAME\Documents" (Windows 7).
	"System.Path.CommonDocuments" : Returns the Windows default folder for documents which are shared with all users. A typical path is "C:\Documents and Settings\All Users\Documents" (Windows XP) or "C:\Users\Public\Documents" (Windows 7).
	"System.Net.HostName" : Returns the computer's default hostname
	"System.Net.IPAddr1" : Returns the computer's primary local IPv4-address
	"System.Net.IPAddr*" : Returns all of the computer's local IPv4-addresses, separated by '|'.You can use the function TxSplit() in order to isolate the individual addresses.
	"System.Path.TempFiles" : Returns the Windows default folder for temp files. A typical path is "C:\Documents and Settings\USERNAME\Local Settings\Temp" (Windows XP) or "C:\Users\USERNAME\AppData\Local\Temp" (Windows 7).
	"System.EnvVariable" : Returns the value of a Windows environment variable as text. Specify the name of the desired variable in the 2nd parameter of this function. Up to 256 characters are returned, longer text will be truncated. When the variable does not exist, an empty text is returned.
	"Device.LogicalProcessors" : Returns the count of logical processors (cores) in the system.
	"Screen.VirtualArea" : Returns the size of the virtual screen. This virtual screen is calculated as the cumulative size of all monitors present. For the second parameter, the text-constants "left", "top", "width" and "height" can be specified; the function then calculates the corresponding coordinate/dimension.
	"Screen.MonitorCount" : Returns the number of monitors present

	"Screen.PrimaryWorkArea" : Returns the coordinates of the primary display monitor's work area. The work area consists of the screen size minus the area required for the Windows taskbar (if displayed). For the second parameter, the text constants "left", "top", "width" 7 "height" can be specified; the function then calculates the resulting coordinates/dimension.
	"Screen.WorkArea#1" : Returns the coordinates of the 1st monitor's work area. The work area consists of the screen size minus the area required for the Windows taskbar (if displayed). For the second parameter, the text constants "left", "top", "width" 7 "height" can be specified; the function then calculates the resulting coordinates/dimension.
	"Screen.WorkArea#2" : Returns the coordinates of the 2nd monitor's work area. The work area consists of the screen size minus the area required for the Windows taskbar (if displayed). For the second parameter, the text constants "left", "top", "width" 7 "height" can be specified; the function then calculates the resulting coordinates/dimension.
	"Screen.WorkArea#3" : Returns the coordinates of the 3rd monitor's work area.
TxParameter	For "System.EnvVariable", the name of the desired variable; for "Screen.*", the designation of the desired quantity; else an empty text.
CurrentValue	
CurrentValue	Value of the requested information. Number or text.

Description:

The query of the system folder paths returns the actual names in the file system. In Windows-Explorer, these folders may either not be visible (hidden by default settings) or they are shown with a different (localized) name. E.g. Windows 7 (German): the folder "c:\Users" will be displayed with the localized name "c:\Anwender".

Examples:

Get the default folder for user documents:

```
MyDocFolder = GetSystemInfo("System.Path.Documents", "")
```

Get the content of the environment variable USERNAME, which contains the name of the current user account.

```
CurrentUser = GetSystemInfo("System.EnvVariable", "USERNAME")
```

A test of whether the Order-Tracking Kit functions can be used. Both calls are equivalent:

```
ok = GetSystemInfo("FAMOS.IsKitAvailable","OTR")
ok = GetSystemInfo("FAMOS.IsKitAvailable","im7otrk1.dll")
```

A sequence uses multiple sequence functions from the sequence library "MyFilters". At the beginning of the sequence, the system checks whether the functions are actually available at runtime, otherwise an appropriate error message is posted. Without this check, the formula interpreter would return a generic error message at the call line, taking the form "unknown object", which would not be helpful to the user.

```
ok = GetSystemInfo("Famos.IsSeqFunctionAvailable", "!MyFilters.HP_200Hz")
ok = ok AND GetSystemInfo("Famos.IsSeqFunctionAvailable", "!MyFilters.LP_100Hz")
IF NOT(ok)
   BoxMessage("Error", "The Sequence library 'MyFilters.sqf' is either not available or too old!", "!1")
   EXITSEQUENCE
END
```

See also:

GetOption

GrChanAppend

Add new channel to a group

Declaration:

```
GrChanAppend ( Group, NewChannel )
```

Parameter:

Group	Data group to which a channel is to be added
NewChannel	Data set or text which is to be added to the group

Description:

This function adds a copy of the specified data set or text to the given data group. The data set or text is duplicated and entered in the content list of the group.

In many cases, a direct assignment can be used instead of this function:

```
ExistingGroup:NewChannel
```

Examples:

A new data group is generated and a data set and a text are added:

```
newGroup = GrNew()
d1 = Ramp(0,1,100)
t1 = "Date: 1.1.1995")
GrChanAppend(newGroup, d1)
GrChanAppend(newGroup, t1)
; or also:
; newGroup:d1 = Ramp(0,1,100)
; newGroup:t1 = "Date: 1.1.1995"
```

See also:

GrChanDel, GrNew, GrChanNum?, GrChanName?, GrChanFind

GrChanDel

Deletes a channel belonging to a data group

Declaration:

```
GrChanDel ( Group, ChannelIdent )
```

Parameter:

Group	Data group from which a channel is to be deleted
ChannelIdent	Index (1..) or name of the channel to be deleted

Description:

The specified channel is deleted from the group. If the channel does not exist (invalid index, name not available), a warning message is generated.

The command DELETE can be used instead of this function.

Examples:

The data group Measurement_01 contains at its second position the channel c2 which is to be deleted. The following calls are equivalent:

```
GrChanDel(Measurement_01, 2)
GrChanDel(Measurement_01, "c2")
DELETE Measurement_01:c2
DELETE Measurement_01:[2]
```

All channels whose maximum is less than 100 are deleted from a data group.

```
i = GrChanNum?(MEASUREMENT_01)
WHILE i >= 1
   channelMax= max(MEASUREMENT_01:[i])
   IF channelMax < 100
      GrChanDel(MEASUREMENT_01, i)
   END
   i = i - 1
END
```

See also:

GrChanAppend, GrNew, GrChanNum?, GrChanName?, GrChanFind, DELETE

GrChanFind

Searches for a channel in a data group with a specified name

Declaration:

```
GrChanFind ( Group, TxChannelname ) -> SvChannelIndex
```

Parameter:

Group	Data group whose channels are to be searched
TxChannelname	Name of the data set or text to find
SvChannelIndex	
SvChannelIndex	Index of the channel in the group (1..) or 0, if not found

Description:

A data set or text with a specified name is searched for in a group. If the text or data set is found, its index is returned, otherwise a 0 is returned.

The index becomes invalid when the group structure is changed by inserting or deleting channels; the assignment channel name <-> index must be redefined.

Examples:

A channel with the name "MeasChan_123" is searched for in a data group. If found, the channel is displayed, otherwise an error message is generated.

```
Index = GrChanFind(Measurement011_A, "MeasChan_123")
IF Index = 0
   BoxMessage("Attention", "Channel does not exist!", "!1")
ELSE
   SHOW Measurement011_A:[Index]
   ;oder auch: SHOW Measurement011_A:MeasChan_123
END
```

See also:

GrChanAppend, GrChanDel, GrNew, GrChanNum?, GrChanName?

GrChanName?

Gets the name of a channel belonging to a data group

Declaration:

`GrChanName? ( Group, SvChannelIndex ) -> TxChannelName`

Parameter:

Group	Data group from which a channel's name is to be determined
SvChannelIndex	Index of the desired channel (1..).
TxChannelName	
TxChannelName	Name of the channel

Description:

The function returns the name of the specified data set or text in the group.

The first channel in the group is addressed by the index 1.

Alternatively, the more powerful function Name?() can be used.

Examples:

A data group is composed from an existing data group, in which the original channels as well as the data sets resulting from smoothing are added. The name of the smoothed data set is formed by adding a fixed ending to the original name.

```
channelCount = GrChanNum?(MEASUREMENT_01)
RESULT_01 = GrNew()
FOR i = 1 TO channelCount
   TxName = GrChanName?(MEASUREMENT_01, i)
   TxName2 = TxName + "_SMO"
   RESULT_01:<TxName> = MEASUREMENT_01:[i]
   ; oder auch GrChanAppend(RESULT_01, MEASUREMENT_01:[i])
   RESULT_01:<TxName2> = Smo(MEASUREMENT_01:[i], 10)
END
```

See also:

Name?, GrChanAppend, GrChanDel, GrNew, GrChanNum?, GrChanFind

GrChanNum?

Gets a data group's channel count

Declaration:

```
GrChanNum? ( Group ) -> SvChannelCount
```

Parameter:

Group	Data group whose channel count is to be determined
SvChannelCount	
SvChannelCount	Number of channels included

Description:

This function returns the number of channels is a data group; all data sets and text contained in the group are counted.

If the parameter does not represent a group the value -1 is returned.

A typical application is determining the channel count for subsequent enumeration (by means of FOR, WHILE) of the channels. Here, FOREACH CHANNEL-instruction can alternatively be used.

Examples:

All data sets with a maximum greater than 100 are displayed in a loop over all data groups.

```
channelCount = GrChanNum?(MEASUREMENT_01)
i = 1
WHILE i <= channelCount
   channelMax= max(MEASUREMENT_01:[i])
   IF channelMax > 100
      SHOW MEASUREMENT_01:[i]
   END
   i = i + 1
END
```

See also:

GrChanAppend, GrChanDel, GrNew, GrChanName?, GrChanFind, FOREACH

GrConcat

Joins the specified parameter variables to a new data group.

Declaration:

```
GrConcat ( P1 [, P2] [, P3] [, P4] [, P5] [, P6] [, P7] [, P8] [, P9] [, P10] [, P11] [, P12] [, P13] [, P14] [,
P15] ) -> Group
```

Parameter:

P1	Parameter #1
P2	Parameter #2 (optional)
P3	Parameter #3 (optional)
P4	Parameter #4 (optional)
P5	Parameter #5 (optional)
P6	Parameter #6 (optional)
P7	Parameter #7 (optional)
P8	Parameter #8 (optional)
P9	Parameter #9 (optional)
P10	Parameter #10 (optional)
P11	Parameter #11 (optional)
P12	Parameter #12 (optional)
P13	Parameter #13 (optional)
P14	Parameter #14 (optional)
P15	Parameter #15 (optional)
Group	
Group	New group. The elements it contains are copies of the specified parameters.

Description:

This function joins the parameters specified to a new data group. The new data group contains copies of the specified parameter variables; the respective variable-name is applies as the element-name.

As the parameters, any desired data types are allowed; the exception is data groups, where only directly specified variables are allowed (no indexing, no intermediate results). With data groups, the elements they contain are expanded and copies of them are adopted in the results group.

The order of the elements in the group created matches the oder of the parameters.

If the name of a variable to be appended already exists in the resulting group, that element of the resulting group will be overwritten while its position is retained. Thus, in case of duplicate names, the last parameter specified "wins". This means that this function can also be used in order to replace multiple elements of an existing group; see Example #3.

Examples:

Three additional channels are to be appended to an existing group:

```
a_fft = FFT(gr:a)
b_fft = FFT(gr:b)
c_fft = FFT(gr:c)
grNew = GrConcat(gr, a_fft, b_fft, c_fft)
```

For a group consisting of many channels, the FFT of each channel is to be calculated.

```
GrFFTs = FFT(GrInput, 0, 0)
```

This calculation is to be accelerated by means of parallel execution. For this purpose, the data group is subdivided into 4 parts which are computed in parallel. The 4 partial results are then re-joined into a data group of results.

```
count = GrChanNum?(GrInput)
chunksize = floor(count/4)
BEGIN_PARALLEL
   GrFFT1 = !MyFFT(GrPart(GrInput, 1,              chunksize))
   GrFFT2 = !MyFFT(GrPart(GrInput, 1+ chunksize,   chunksize))
   GrFFT3 = !MyFFT(GrPart(GrInput, 1+ chunksize*2, chunksize))
   GrFFT4 = !MyFFT(GrPart(GrInput, 1+ chunksize*3, count-3*chunksize))
END_PARALLEL
GrFFTs = GrConcat(GrFFT1, GrFFT2, GrFFT3, GrFFT4)
```

For this purpose, the sequence function !MyFFT is defined as follows:

```
; declaration:
; !MyFFT(Par [Data type: Normal]) => Result [Data type: Complex]
Result = FFT(Par, 0, 0)
```

Attention when parameters have duplicate names: the "last" one wins:

```
a = 1
g = GrConcat(a, a)
; the resulting group has only one channel: "a".
g:b = 1
b = 2
g2 = GrConcat(g, b)
; the resulting group has the channels "a" and "b", where "b" takes the value 2.
g3 = GrConcat(b, g)
; The resulting group has the channels "b" and "a", where "b" takes the value 1.

; exchange of multiple channels in an existing group:
g4:a = 2
g4:b = 1
g4:c = 1
g5:b = 2
g5:c = 2
g = GrConcat(g4, g5)
; the results group now contains three channels, all having the value 2.
```

See also:

GrNew, GrChanAppend, GrPart

Supported since:

Version 2022

GrExpand

The channels belonging to this data group are expanded.

Declaration:

```
GrExpand ( Group )
```

Parameter:

Group	Data group to be expanded

Description:

The channels belonging to a data group are expanded into individual variables and the group variable is subsequently deleted.

If variables with the same name already exist, they are overwritten.

The function's behavior is identical to that of the menu item "Variable"/"Expand".

Examples:

Multiple variables are to be subjected to similar calculations. To save writing effort, these variables are initially grouped together to a data group, then processed as desired and the group is subsequently dissolved.

```
Temp:channel1 = channel1
Temp:channel2 = channel2
Temp:channel3 = channel3
Temp = CutIndex(Temp, 100, 300)
Temp = Smo(Temp, 0.2)
; additional calculations...
GrExpand(Temp)
```

See also:

GrChanAppend, GrChanDel, GrChanName?, GrChanFind

GrJoin

Joins together the channels of a group

Declaration:

```
GrJoin ( Group, Zero ) -> Dataset
```

Parameter:

Group	Data group whose channels are to be joined
Zero	Reserved parameter. Always 0.
Dataset	
Dataset	Data set which consists of the joined channels in a group

Description:

Individual channels in a group are joined to form a new data set. Texts are skipped in concatenation. Characteristics and the data format of the result conform with the first data set in the group. All data sets in the group must have the same data type.

Examples:

Determining the absolute maximum across all of a group's channels:

```
AbsMaximum = max(GrJoin(max(myGroup), 0))
```

See also:

GrNew, GrChanAppend, GrChanNum?, GrChanName?, GrChanFind

GrNew

Creates an empty data group

Declaration:

```
GrNew ( ) -> NewGroup
```

Parameter:

NewGroup	
NewGroup	Empty data group

Description:

This function generates an empty group; additional channels can be assigned to this group.

It is also possible to generate a new group and the first channel at the same time using the following assignment:

```
NewGroup:FirstChannel = ...
```

Examples:

A new group is created and two channels are added to the group:

```
Measurement1 = GrNew()
GrChanAppend(Measurement1, Channel_1)
GrChanAppend(Measurement1, Channel_2)
```

The following formulas produce the same result:

```
Measurement1:Channel1_1 = Channel1_1
Measurement1:Channel1_2 = Channel1_2
```

See also:

GrChanAppend, GrChanDel, GrChanNum?, GrChanName?, GrChanFind

GrPart

One section of a data group is copied to a new data group.

Declaration:

```
GrPart ( GrSource, SvIndex, SvCount ) -> GrPartCopy
```

Parameter:

GrSource	Data group whose elements are to be copied
SvIndex	Index of the first element to be copied. The first element has the index 1.
SvCount	Count of elements to be copied. -1, if all elements down to the last are to be copied.
GrPartCopy	
GrPartCopy	Newly created group with copies of the specified elements

Description:

This function generates a new data group which is a section of the parameter data group.

When a -1 is specified for [SvIndex], or when ([SvIndex]+[SvCount]) is greater than the count of elements in the source data group, all elements are copied all the way to the end.

Examples:

From adata group having more than 4 elements, the two first and two last elements are copied to a new data group.

n = GrChanNum?(grSource)

grNew = GrConcat(GrPart(grSource, 1, 2), GrPart(grSource, n-1, 2))

For a group consisting of many channels, the FFT of each channel is to be calculated.

```
GrFFTs = FFT(GrInput, 0, 0)
```

This calculation is to be accelerated by means of parallel execution. For this purpose, the data group is subdivided into 4 parts which are computed in parallel. The 4 partial results are then re-joined into a data group of results.

```
count = GrChanNum?(GrInput)
chunksize = floor(count/4)
BEGIN_PARALLEL
   GrFFT1 = !MyFFT(GrPart(GrInput, 1,              chunksize))
   GrFFT2 = !MyFFT(GrPart(GrInput, 1+ chunksize,   chunksize))
   GrFFT3 = !MyFFT(GrPart(GrInput, 1+ chunksize*2, chunksize))
   GrFFT4 = !MyFFT(GrPart(GrInput, 1+ chunksize*3, count-3*chunksize))
END_PARALLEL
GrFFTs = GrConcat(GrFFT1, GrFFT2, GrFFT3, GrFFT4)
```

For this purpose, the sequence function !MyFFT is defined as follows:

```
; declaration:
; !MyFFT(Par [Data type: Normal]) => Result [Data type: Complex]
Result = FFT(Par, 0, 0)
```

See also:

GrConcat

Supported since:

Version 2022

Histo

Histogram with definable number of bars and bar width

Declaration:

```
Histo ( Data, SvWidth, SvCount ) -> Histogram
```

Parameter:

Data	Data set from which the histogram is to be calculated. Allowed data types: [ND],[XY]
SvWidth	Width of a histogram bin
SvCount	Number of histogram bins
Histogram	
Histogram	Histogram determined

Description:

The histogram function determines the number of occurrences of amplitude values (values of a data set). The possible range of values is divided into a number of bins with a specified width. Each value of the data set is sorted into one of these bins. The histogram indicates how many values were sorted into each bin.

The second parameter specifies the width of a bin. If the value is equal to zero, the width of the bins is determined automatically.

The third parameter specifies the number of bins. If a value is equal to zero, the number of bins is determined automatically.

Parameters specified as zero are determined automatically by selecting values appropriate for the range of the data set.

The edges of a bin are always an integer multiple of the bin width.

- The created data set has no y-unit and its x-unit corresponds to the y-unit of the specified data set.
- For completely automatic calculation of the histogram, the number of bins is generally set to approximately 100.
- The number of bins is theoretically limited to 10000. Usually values between ten and 100 are selected.
- When the bin width and number are specified manually, it may occur that some of the values in the data set cannot be sorted into any bins. The total number of values in the histogram will thus be less than the length of the given data set. To sort even these values beyond the normal range of the data set in the histogram, use the Clip function to limit the y-value range before applying the Histo function. With the Min and Max functions, the value range of the function can be determined; then the Floor function can be used to round these values. The bin width can be derived from this result.

Examples:

```
NDhisto = Histo(NDdata, 0, 0)
```

Fully automatic histogram calculation

```
NDhisto1 = Histo(NDcurrent1, 0.1 'A', 0)
NDhisto2 = Histo(NDcurrent2, 0.1 'A', 0)
```

The data sets NwCurr1 and NwCurr2 are given, with current values exhibiting maximum peaks of approximately 10A. The histograms of both data sets are to be compared. A bin width is selected to create approximately 100 bins; the exact number of bins should be determined automatically. The histograms can be compared directly where the value ranges of the data sets for current overlap.

```
NDhisto = Histo(NDdata, 1 'V', 200)
```

The data set's value range is known; it extends from approx. -50V to approx. +130V. A histogram is to be generated with 200 bins of width 1V.

See also:

ClsTimeAtLevel, Clip, Min, Max

HttpGetFile

Downloads the requested resource and saves it to a local file.

Declaration:

```
HttpGetFile ( TxURL, TxFileName ) -> EwStatus
```

Parameter:

TxURL	Identifier of the desired web resource
TxFileName	Complete filename under whch the resource is to be saved
EwStatus	
EwStatus	Status. 1 for success, 0 for error

Description:

This function retrieves the resource specified via its URL (Uniform Resource Locator) and saves it to a local file.

The URL generally consists of the following portions:

- Protocol (supported types: "http://" or "https://")
- Server address (e.g. "www.testserver.com")
- (Optional) path to the resource (e.g. "/weather")
- (Optional) query parameter ('query', typically specified as a list of value pairs taking the form "?Value1=Content@Value2=Content2@...")

To the specified server, an 'HTTP GET'-command is sent, which contains as its parameter the path and the query parameters.

The data received as the answer are saved unchanged in the file specified.

In case of error (return value = 0), the error cause can be queried using the function GetLastError().

Examples:

Downloads the current "Microsoft News"-page on Twitter and saves it as a local HTML-file:

```
uri ="https://twitter.com/MSFTnews"
status = HttpGetFile(uri, "c:\temp\LatestMSNews.html")
```

Downloads an image from the Hubble Telescope image archive:

```
uri = "https://cdn.spacetelescope.org/archives/images/screen/heic1501a.jpg"
status = HttpGetFile(uri, "c:\temp\_nebula.jpg")
```

See also:

HttpGetText, HttpOption

HttpGetText

Downloads the requested resource as a text.

Declaration:

```
HttpGetText ( TxURL ) -> TxReturn
```

Parameter:

TxURL	Identifier of the desired web resource
TxReturn	
TxReturn	Received text

Description:

This function retrieves the resource specified via its URL (Uniform Resource Locator).

The URL generally consists of the following portions:

- Protocol (supported types: "http://" or "https://")
- Server address (e.g. "www.testserver.com")
- (Optional) path to the resource (e.g. "/weather")
- (Optional) query parameter ('query', typically specified as a list of value pairs taking the form "?Value1=Content@Value2=Content2@...")

To the specified server, an 'HTTP GET'-command is sent, which contains as its parameter the path and the query parameters.

After downloading, the data received are converted to a text.

In case of error, an empty text is returned; the error cause can be queried using the function GetLastError().

Examples:

By means of the call below, the specified longitude and latitude values are used to return the associated address (reverse geocoding).

For this purpose, the search machine of OpenStreetMap is used. The parameter 'format' specifies the desired format for the answer (here XML; an alternative would be JSON); the parameters 'lat' (latitude) and 'lon' (longitude) specify the latitude and longitude.

```
answer = HttpGetText( "http://nominatim.openstreetmap.org/reverse?format=xml&lat=52.541861&lon=13.3869693")
```

The text returned includes the following section:

```
<road>Voltastraße</road>
<suburb>Gesundbrunnen</suburb>
<city_district>Mitte</city_district>
<state>Berlin></state>
<postcode>13355></postcode>
```

In order to extract individual fields from such answers, the function TxRegExMatch() can be recommended:

```
NameOfTheRoad = TxRegExMatch( answer, "<road>(.*?)</road>", ",", 0, 1)
; NameOfTheRoad now has the content 'Voltastraße'.
```

If the response is returned in JSON-format instead:

```
answer = HttpGetText( "http://nominatim.openstreetmap.org/reverse?format=json&lat=52.541861&lon=13.3869693")
;answer contains (besides other items): ...{... "road":"Voltastraße","suburb":"Gesundbrunnen", ...}
NameOfTheRoad = TxRegExMatch( answer, "~034road~034:~034(.*?)~034", ",", 0, 1)
```

See also:

HttpGetFile, HttpOption

HttpOption

This function sets options for subsequent calls of HttpGetText() or HttpGetFile().

Declaration:

```
HttpOption ( TxName, TxValue ) -> EwStatus
```

Parameter:

TxName	Option name
	"" : All options are reset to their default state. The 2nd parameter must also be empty.
	"Timeout" : Timeout for connection. The text specified for the 2nd parameter contains the value stated as a number of milliseconds. For "0", the default value, which is 20s, is restored.
	"User" : User name for web services with HTTP Basic access Authentication (RFC 2016)
	"Password" : Password for web services with HTTP Basic access Authentication (RFC 2016)
	"HEAD:???" : Sets a field in the HTTP Request header. '???' stands for the name of a field. The fields and their meanings are defined in accordace with RFC 2616. If the 2nd parameter is an empty text, the associated field is deleted.
TxValue	Value of the option. What content is permitted depends on the first parameter.
EwStatus	
EwStatus	Status. 1 for success, 0 for error

Description:

The settings made here remain in force until:

- explicit change by means of a new call of the function HttpOption()
- next new start of a top-level sequence
- Menu item 'Restart'

In case of error (return value = 0), the error cause can be queried using the function GetLastError().

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

The following command instructs the server on subsequent calls to HttpGetText() or HttpGetFile() to update the data or files before sending them, if necessary, and not to use server-side cached and potentially outdated data.

```
status = HttpOption("HEAD:Cache-Control", "no-store, max-age=0")
```

See also:

HttpGetText, HttpGetFile

Hyst

Applies a hysteresis to a data set

Declaration:

```
Hyst ( Data, SvWidth ) -> Filtrate
```

Parameter:

Data	Data set to be filtered [NW].
SvWidth	Hysteresis width (>= 0)
Filtrate	
Filtrate	Filtered data set

Description:

This function applies a hysteresis filter to the data set [Data]. The hysteresis filters is suited for removing small oscillations from a data set. The parameter SvWidth specifies the hysteresis width. The procedure used to apply the hysteresis is explained below:

The start value and the next function value of the data set [Data] are saved in the data set [Filtrate].

If the second value is greater/less than the previous value, the data has positive/negative slope, respectively. If the function values are equal, the next function value in [Data] is savedin the data set [Filtrate] and if appropriate the trend is determined, etc.

If the function is in an upward trend, then depending on the current function value of the data set [Data], one of the three following possibilities is implemented:

- The current function value of the data set [Data] is greater than the previous value of the data set [Filtrate]. So the current function value of [Data] is saved as the current function value in the data set [Filtrate].
- The current function value of the data set [Data] is less than or equal to the previous function value of the data set [Filtrate],but not less than the previous function value of the data set [Filtrate] minus the hysteresis width. The previous function value of the data set [Filtrate] is also saved as the current function value of the data set [Filtrate].
- The current function value of the data set [Data] is less than the previous function value of the data set [Filtrate] minus the hysteresis width. The current function value of the data set [Data] is saved as the current function value of the data set [Filtrate]. The trend is changed to an upward trend.

If the function is in an downward trend, then depending on the current function value of the data set [Data], one of the three following possibilities is implemented:

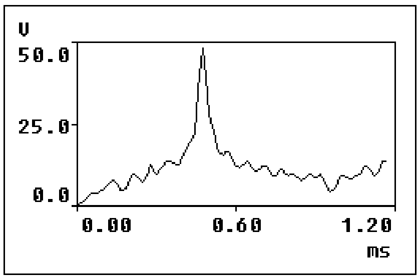
- The current function value of the data set [Data] is less than the previous function value of the data set [Filtrate]. So the current function value of [Data] is saved as the current function value in the data set [Filtrate].
- The current function value of the data set [Data] is greater than or equal to the previous function value of the data set [Filtrate],but not greater than the previous function value of the data set [Filtrate] plus the hysteresis width. The previous function value of the data set [Filtrate] is also saved as the current function value of the data set [Filtrate].
- The current function value of the data set [Data] is greater than the previous function value of the data set [Filtrate] plus the hysteresis width. The current function value of the data set [Data] is saved as the current function value of the data set [Filtrate]. The trend is changed to a downward trend.

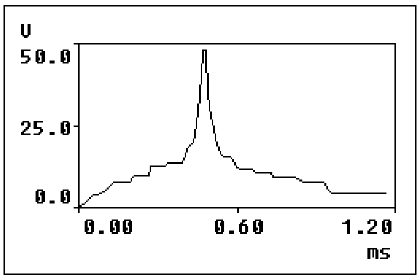
A hysteresis width of 0 returns the source data set [Data] as the target data set [Filtrate]. In contrast to the smoothing functions Smo(), Smo3()..., no frequency-dependent noise suppression is performed in this case but rather purely amplitude-dependent noise suppression.

Examples:

```
NDhyst = Hyst(NDdata, 10)
```

A hysteresis with width 10 is to be applied to the data set NDData in the left graph. The data set NDData consists of 118 values. The result of applying the hysteresis in imc FAMOS is outputted in the graph of the data set NDHyst, appearing on the right side. All oscillations smaller than the hysteresis width 10 have been filtered out:





See also:

Smo, Smo3, SlClip, STri, MInt

idB

Inverse function to dB

Declaration:

```
idB ( InputData ) -> Transformed
```

Parameter:

InputData	Data to be converted from dB the linear measure
Transformed	
Transformed	Resulting data set

Description:

This function is the inverse of the dB function, i.e. it makes a dB calculation retroactive. Complex data sets whose magnitude is expressed in dB are converted back to linear values.

Decibels (dB) mean twenty times the base 10 logarithm of a number. An reciprocal calculation to dB is thus equivalent to the term:

```
InverseDB = 10 ^ (number / 20)
```

- In calculating the reciprocal of dB, the unit is always erased.
- Complex data sets not of the type Dp cannot be processed with this function.
- The parameter may be structured (events/segments).
- With normal or XY-data sets the x-coordinate(s) of the parameter and the result are the same.
- The idB function cannot generate any negative values.

Examples:

```
linear = idB(dataInDB)
; This formula is equivalent to the formula:
linear = 10 ^ (dataInDB / 20)
; only the unit may be different.
```

See also:

dB, log

IF

Initializes a conditional branching. The subsequent instructions are only carried out if the expression here has a value > 0.

Declaration:

```
IF Condition
```

Parameter:

Condition	The following block's instructions are only carried out if the condition is met (evaluation returns a value >0).

Description

The end of the instructions belonging to this block is denoted by a subsequent ELSEIF, ELSE or END.

As the condition, it is possible to specify, for example, a single value variable or a complex expression using logical operators (AND, OR..) and/or comparison operators (<, =, ...).

An IF-block may contain any arbitrary amount of ELSEIF-branchings. If any ELSE-branching is also specified, it must appear at the last position of the chain.

Instead of IF/ELSEIF/ELSE, for many applications it is easier to use the SWITCH/CASE/DEFAULT-instruction.

Examples:

If a variable's maximum is greater than 0, it is displayed in a curve window.

```
Maxi = Max(data)
IF Maxi
   SHOW data
END
```

When a file is loaded, its return value is checked and an error message posted if appropriate.

```
fh = FileOpenDSF("Channel1.dat", 0)
IF fh = 0
   Pause ==> Can't load file <==
ELSE
 ; ...
   FileClose(fh)
END
```

A portion of an equidistantly sample data set [data] is to be cut out. The two boundaries [xmin] and [xmax] have previously been entered by the user and are now being checked for validity:

```
IF xmin < xoff?(data) OR xmin <= 0
   txError = "Illegal lower limit"
ELSEIF xmax > (xoff?(data)+(leng?(data)-1)*xdel?(data))
   txError = "Illegal upper limit"
ELSEIF xmin >= xmax
   txError = "Illegal limit relation"
ELSE
   result = Cut(data, xmin, xmax)
END
```

See also:

ELSEIF, ELSE, SWITCH

iFFT

Inverse FFT, transformation of a spectrum into a time function

Declaration:

```
iFFT ( Spectrum ) -> TimeData
```

Parameter:

Spectrum	Spectrum to be transformed. Allowed data types: [BP],[DP],[RI]
TimeData	
TimeData	Result of inverse FFT.

Description:

The iFFT function is the inverse of the Fast Fourier Transformation and negates the FFT function. The time function is reconstructed from the spectrum.

The spectrum may be present in any complex data type. The spectrum is expected in a form as the function FFT() would return using the Radix-2-procedure. Only one side of the spectrum is present. The first complex value of the spectrum represents the DC component, so it has zero imaginary component. Next come a power of 2 of complex spectral lines of which the last is real. Internally, this one side of the spectrum is extended into conjugated complex space so that the inverse FFT-algorithm can be used. A set of time domain data is generated, which always is a power of 2 in length.

- If the complex data set has 2^n + 1 points, then the time domain data set generated has 2^(n+1) points. If the complex data set's length is greater, then it is truncated to the next lower possible value.
- The maximum length which can be processed is 67108865 (2^26+1) points, which corresponds to 134.217.728 (2^27) time data. If the data set is too long, use the Leng function to shorten it.
- The iFFT function does not apply any window functions. To reverse the effects of a window function used for the FFT, perform the iFFT, create a new data set with the window function (see example in the FFT section) and divide the time data set by the new data set. Note that the values at the edges of the data set often will be fairly inaccurate.
- The x-unit of the time data set is always set to "s"; the y-unit is the y-unit of the magnitude or the real part of the complex data set.
- The iFFT requires temporary space in working memory for execution. If insufficient memory is available (especially in long data sets), an error message is generated.
- The x-offset of the transferred complex data set should be equal to zero; an error message is generated if it is not equal to zero. The x-offset is assumed to be zero. A positive x-offset can be considered by using the Cut function to extend the spectrum down to a frequency of zero.
- The iFFT() function inverses any FFT() function performed with a rectangular window. A spectrum created with the Spec() function cannot be inversed using iFFT().
- An FFT which was calculated with the Mixed-Radix-procedure can also generally not be reversed using iFFT(), since the procedure used here always returns a data set length which is a power of 2.

Examples:

```
NDdata = iFFT(BPspektr)
```

MpSpectr is a complex data set with 513 points; a time function with 1024 points is generated. The iFFT of a transfer function returns the weighting function of a transfer system.

```
NDdata = iFFT(Compl(BPspektr.M, 0))
```

The phase of the spectrum is set to zero; then the corresponding time function is calculated.

See also:

FFT, Spec

InDegr

Radians-to-degrees conversion factor = 57,297...

Description

Unit: "Deg"/"Rad"

Examples:

The constant PI as the measure of an angle in radians corresponds to 180 degrees:

```
Degree180 = PI * INDEGR
```

See also:

InRad, PI, PI2

InRad

Degrees-to-radians conversion factor = 0.01745...

Description

Unit: "Rad"/"Deg"

Examples:

An angle of 180 degrees corresponds to the circle constant PI (3.1415..) in radians:

```
_PI = 180'°' * INRAD
```

See also:

InDegr, PI, PI2

Int

Formation of the integral

Declaration:

```
Int ( Data ) -> Result
```

Parameter:

Data	Data set to be integrated. Permitted data types: [ND],[XY]
Result	
Result	Integration results

Description:

The transferred data set is integrated value for value using a simple, effective algorithm:

For normal, real data sets:

The sum of all sampling values of the data set is calculated up to (but not including) the current point and then multiplied by the sampling rate. Thus, integration can be reversed by differentiation.

For XY-data sets:

The product of the distance between two consecutive points and the mean of their y-values is summed up to the current point.

The integral indicates the area under the data set from the beginning up to the current point. Note that integration smoothes a curve.

- According to the algorithm used for integration, the first value of the generated data set is always zero. Thus, the generated data set will always be one value longer.
- The unit of the integrated data set is the product of the x- and y-units of the transferred data set.
- In contrast to the moving integral function mInt, the integration interval will grow from zero to the length of the data set.

Examples:

```
NDarea = Int(NDdata)
```

Simple calculation of the surface

```
NDarea = Red(Int(IPol(NDdata, 3)), 3)
```

Spline interpolation must be performed before integrating strongly curved data sets or data sets containing discontinuous "jumps. After integration, the amount of data can be reduced using the RSamp function.

```
NDint = Int(NDdata)
definiteInt = Value(NDint, 10) - Value(NDint, 5)
```

The definite integral of the data set NwData is to be calculated between the points x = 5 and x = 10. First the integration function NwInt is determined; the difference between the points yields the definite integral. Note that the definite integral can be interpreted as an area only if no zeros are present in this interval of the integration function.

See also:

IntEx, Diff, MInt, Sum, MvSum

IntervalFrom2Levels

Available in: Professional Edition and above

Intervals are derived from a signal, where the signal's crossing points through specified thresholds are given.

Declaration:

```
IntervalFrom2Levels ( input data, LevelBegin, SlopeBegin, LevelEnd, SlopeEnd [, Rounding] [, ResultFormat] ) ->
Result
```

Parameter:

input data	input data
LevelBegin	The signal must pass through this threshold, which represents a y-value or amplitude, in order to define the beginning of an interval.
SlopeBegin	Which type of slope is to mark the beginning of the interval, rising (positive) or falling (negative)?
	1 : Positive slope (departing from the point). At the transition, this formula applies: y[k] <= Level < y[k+1]
	2 : Positive slope (towards the point). At the transition, this formula applies: y[k] < Level <= y[k+1]
	3 : Negative slope (departing from the slope). At the transition, this formula applies: y[k] >= Level > y[k+1]
	4 : Negative slope (towards the point). At the transition, this formula applies: y[k] > Level >= y[k+1]
LevelEnd	The signal must pass through this threshold, which represents a y-value or amplitude, in order to define the end of an interval.
SlopeEnd	Which type of slope is to mark the end of the interval, rising (positive) or falling (negative)?
	1 : Positive slope (departing from the point). At the transition, this formula applies: y[k] <= Level < y[k+1]
	2 : Positive slope (towards the point). At the transition, this formula applies: y[k] < Level <= y[k+1]
	3 : Negative slope (departing from the slope). At the transition, this formula applies: y[k] >= Level > y[k+1]
	4 : Negative slope (towards the point). At the transition, this formula applies: y[k] > Level >= y[k+1]
Rounding	Do you wish to round the positions found? (optional , Default value: 0)
	0 : Automatic: Use precise value, but round up for digital signals.
	1 : Use precise value; the signal can be considered linearly interpolated between the sampling points.
	2 : The position found is rounded to the signal's sampling point lying closest to the interpolated x-value. For signals with discrete values.
	3 : The position found is rounded down to the signal's next sampling point which is <= the interpolated x-value.
	4 : The position found is rounded up to the signal's next sampling point which is >= the interpolated x-value. For digital signals.
ResultFormat	Format of the result (optional , Default value: "")
	"" : automatic: Generates an equidistant result for equidistant input data; an XY-result for XY-input data with a monotonic time track; else an interval code over parameter.
	"equi" : equidistant. Only for equidistant input data. Also for segmented data sets.
	"xy" : XY-pairs, interval-code with time stamp. Only for equidistant input data or XY-input data with monotonic time track. Not for data with segments. If the data have stair-steps, be aware that when retrieving a value later based on its time stamp (if it is at the sampling point), either the previous sample or the sample at the sampling point itself can accidently be used due to rounding errors.
	"para equi" : Equidistant data set with interval-code plotted over the parameter of the two-component data
	"para xy" : XY-data set with interval-code and value of the parameter of the two-component data. Not for data having segments.
Result	
Result	Interval data set

Description:

This function can be applied to input data having events or segments.

With XY-data which can take sometimes monotonic and sometimes non-monotonic forms, the automatic result format should be avoided.

If the input data are superimposed with noise, the function may find unintended level-crossings in the signal. The function Hyst() prevents small fluctuations and should be called beforehand. Smoothing generally does not provide a comprehensive solution for small fluctuations. Nevertheless, smoothing is still useful. But subsequently, the residual interfering fluctuations are eliminated with Hyst().

Both slope conditions can also be fulfilled by the same measured value. This leads to a point-shaped interval whose position is the first slope.

This function always searches for alternating slope conditions.

Interval format equidistant

A series of codes which indicate at which locations intervals begin and end

0: outside of an interval

1: within an interval

2+decimal digits: End of an interval. The decimal digits denote the relative position within the sampling interval

3+decimal digits: Beginning of an interval. The decimal digits denote the relative position within the sampling interval.

4+decimal digits: Transition between 2 adjacent intervals. The decimal digits denote the relative position within the sampling interval.

5+decimal digits: Point-shaped interval. The decimal digits denote the relative position within the sampling interval.

Interval format XY

The x-coordinate indicates the position; the y-coordinate has the following meaning:

2: End of an interval

3: Beginning of an interval

4: Transition between 2 adjacent intervals

5: Point-shaped interval

Examples:

Determines all intervals in which a signal rises to over 2.0 and at the end falls below 8.0. The signal is heavily distorted with noise.

```
input = 10*sin (Ramp(0,1e-3,400)*PI2*10+1.1) + Random ( 400, 2, 0, 0, 33 ) ; test data
input_smooth = Hyst( Smo(input,0.01), 0.1 )
ivl = IntervalFrom2Levels ( input_smooth, 2.0, 1, 8.0, 3, 0, "equi")
```

Determines all intervals in which a signal increases from 2.0 to 8.0. The signal is slightly distorted with noise.

```
input = 10*sin (Ramp(0,1e-4,4000)*PI2*10+1.1) + 0.2 * Random ( 4000, 2, 0, 0, 0 ) ; test data
input_smooth = Smo(input, 0.001)
input_smooth = Hyst ( input_smooth, 0.05)
ivl = IntervalFrom2Levels ( input_smooth, 2.0, 1, 8.0, 1, 0, "equi")
```

See also:

IntervalFromLevel, IntervalGetStatist, Hyst

IntervalFromLevel

Available in: Professional Edition and above

Intervals with immediately adjacent boundaries are generated from a signal; where the boundaries are defined as the points where the signal crosses a threshold. Only if the intervals do not meet the secondary conditions regarding amplitude and width, they are discarded so that there are gaps.

Declaration:

```
IntervalFromLevel ( input data, Level, LowerLevel, UpperLevel, Slope, MinWidth, MaxWidth [, Rounding] [, ResultFormat] ) -> Result
```

Parameter:

input data	input data
Level	The signal must pass through this threshold, which represents a y-value or amplitude, in order to determine the beginning or end of an interval.
LowerLevel	An interval is only recognized as valid, if at least one value within the interval is equally low or lower than this value. It is set to Level, if it is not to be checked.
UpperLevel	An interval is only recognized as valid, if at least one value within the interval is equally high or higher than this value. It is set to Level, if it is not to be checked.
Slope	Which slope does the system watch for; the rising (positive) or falling (negative)?
	1 : Positive slope (departing from the point). At the transition, this formula applies: y[k] <= Level < y[k+1]
	2 : Positive slope (towards the point). At the transition, this formula applies: y[k] < Level <= y[k+1]
	3 : Negative slope (departing from the slope). At the transition, this formula applies: y[k] >= Level > y[k+1]
	4 : Negative slope (towards the point). At the transition, this formula applies: y[k] > Level >= y[k+1]
	5 : Any slope (departing from the point). At the transition, the formula is: y[k] <= Level < y[k+1] or y[k] >= Level > y[k+1]
	6 : Any slope (towards the point). At the transition, the formula is: y[k] <= Level < y[k+1] or y[k] >= Level > y[k+1]
MinWidth	Minimum width of the interval in x-units. Shorter intervals are rejected. 0, if not to be checked.
MaxWidth	Maximum width of the interval in x-units. Longer intervals are rejected. 0, if not to be checked.
Rounding	Do you wish to round the positions found? (optional , Default value: 0)
	0 : Automatic: Use precise value, but round up for digital signals.
	1 : Use precise value; the signal can be considered linearly interpolated between the sampling points.
	2 : The position found is rounded to the signal's sampling point lying closest to the interpolated x-value. For signals with discrete values.
	3 : The position found is rounded down to the signal's next sampling point which is <= the interpolated x-value.
	4 : The position found is rounded up to the signal's next sampling point which is >= the interpolated x-value. For digital signals.
ResultFormat	Format of the result (optional , Default value: "")
	"" : automatic: Generates an equidistant result for equidistant input data; an XY-result for XY-input data with a monotonic time track; else an interval code over parameter.
	"equi" : equidistant. Only for equidistant input data. Also for segmented data sets.
	"xy" : XY-pairs, interval-code with time stamp. Only for equidistant input data or XY-input data with monotonic time track. Not for data with segments. If the data have stair-steps, be aware that when retrieving a value later based on its time stamp (if it is at the sampling point), either the previous sample or the sample at the sampling point itself can accidently be used due to rounding errors.
	"para equi" : Equidistant data set with interval-code plotted over the parameter of the two-component data
	"para xy" : XY-data set with interval-code and value of the parameter of the two-component data. Not for data having segments.
Result	
Result	Interval data set

Description:

This function can be applied to input data having events or segments.

With XY-data which can take sometimes monotonic and sometimes non-monotonic forms, the automatic result format should be avoided.

With general XY-data as well, the width is always determined as positive. The width is the absolute value of the difference between x at the end of and at the start of an interval.

With noisy signals, it can be useful to perform smoothing beforehand.

Behavior

This function detects the start and end of recurring cycles or transitions between cycles.

The underlying assumption is that noise of a known height is superimposed on the signals. The parameters UpperLevel and LowerLevel should have a distance from the level which is substantially greater than the noise.

When a positive slope is set, the signal is expected to rise to at least UpperLevel, and then fall back to at least LowerLevel, before there is once again a level-crossing of the respective slope set. The opposite applies accordingly for a negative slope setting.

A valid interval only is realized if the specified amplitude conditions are met as well as the specified time conditions regarding the interval width.

Signal periods which do not meet the conditions cause gaps between the intervals.

Algorithm

Initially the system looks for a first level crossing through the specified threshold, in the specified direction (slope). Next it looks for a second level-crossing.

The difference in the x-direction between the two is checked against both the minimum and maximum width. If it is outside of these bounds, the first level-crossing is discarded.

When a positive slope is set, then the signal must first reach UpperLevel, and only subsequently LowerLevel, otherwise the first leve-crossing is discarded. The opposite applies accordingly for a negative slope setting.

If the first level-crossing is not discarded, but both LowerLevel and UpperLevel have been reached, the second level-crossing is discarded. This prevents a premature end at the wrong slope.

If both level-crossings exist, they form an interval.

If the second level-crossing is not discarded, it is declared to be the first, otherwise the first remains intact. Next, the procedure is repeated from the beginning.

Interval format equidistant

A series of codes which indicate at which locations intervals begin and end

0: outside of an interval

1: within an interval

2+decimal digits: End of an interval. The decimal digits denote the relative position within the sampling interval

3+decimal digits: Beginning of an interval. The decimal digits denote the relative position within the sampling interval.

4+decimal digits: Transition between 2 adjacent intervals. The decimal digits denote the relative position within the sampling interval.

5+decimal digits: Point-shaped interval. The decimal digits denote the relative position within the sampling interval.

Interval format XY

The x-coordinate indicates the position; the y-coordinate has the following meaning:

2: End of an interval

3: Beginning of an interval

4: Transition between 2 adjacent intervals

5: Point-shaped interval

Examples:

Determines periodic intervals. Zero-crossing of a periodic signal whose measured value exceed -2.0 and +2.0 and whose period lies between 0.001 and 0.2.

```
ivl = IntervalFromLevel( input, 0.0, -2.0, 2.0, 1, 0.001, 0.2, 0, "equi")
```

Determines periodic intervals. The input signal is noisy.

```
input = 10*sin (Ramp(0,1e-3,400)*PI2*10+1.1) + Random ( 400, 2, 0, 0, 35 )
input_smooth = Smo(input, 0.01)
ivl = IntervalFromLevel( input_smooth, 0.0, -3.0, 3.0, 1, 0.01, 0.3, 0, "equi")
```

See also:

IntervalFrom2Levels, IntervalGetStatist

IntervalGetStatist

Available in: Professional Edition and above

For each interval within a data set, statistics such as Min, Max etc. are found.

Declaration:

```
IntervalGetStatist ( input data, Interval data, Interval format, Calculation [, Interpolation] [, ResultFormat]
) -> Result
```

Parameter:

input data	Input data; equidistant or XY with X monotonic
Interval data	Interval data
Interval format	Interpretation of the interval data: Are the intervals to be interpreted as X-positions or as parameters? The latter is important if the input data are of the type XY.
	"" : automatic: With equidistant input data and with XY-input data having a monotonic X-track, the interval data represent X-positions. Otherwise they are interpreted as parameters. Not recommended when the input data are XY, while the X-track can be sometimes monotonic and sometimes not.
	"equi" : equidistant. Only for equidistant input data
	"xy" : XY-pairs, interval-code with time stamp. Only for equidistante input data or XY-input data with monotonic time track.
	"para equi" : Equidistant data set with interval-code plotted over the parameter of the two-component data
	"para xy" : XY-data set with interval code and value of the 2-component-data's parameter
Calculation	Calculation
	"len" : Number of samples making up the width of the interval
	"width" : Width of the interval in x-units; difference between x at the end and x at the beginning
	"1/width" : Reciprocal of the interval width. If the interval represents a period, this is the frequency.
	"min" : Minimum
	"max" : Maximum
	"mean" : Mean value. In linear interpolation, the true mean value of the polygonal chain.
	"sum" : Sum of the y-values. Added proportionally at the interval edge. Signal always interpreted as stair-steps.
	"int" : Total integral of y over x
	"rms" : RMS-value, based on squared sum of y-values. Weighted according to distance between the samples. Edge values are counted proportionally. The signal is always interpreted as stair-steps.
	"stdev" : Standard deviation. All samples are weighted equally (as if equidistant). Edge values shorter than the sample distance are not counted. Signal always interpreted as stair-steps.
	"minposF" : Position of the minimum value. If the same value appears multiple times, the first occurrence is determined.
	"maxposF" : Position of the maximum value. If the same value appears multiple times, the first occurrence is determined.
	"minposL" : Position of the minimum value. If the same value appears multiple times, the last occurrence is determined.
	"maxposL" : Position of the maximum value. If the same value appears multiple times, the last occurrence is determined.
	"x begin" : x-value at beginning of the interval
	"x end" : x-value at end of the interval
	"y begin" : y-value at beginning of the interval
	"y end" : y-value at end of the interval
Interpolation	Interpolation (optional , Default value: 0)
	0 : Automatic: Linear interpolation, but like stair-steps for digital signals
	1 : The signal is imagined as linearly interpolated between the sampling points.
	2 : The signal is imagined as continuing at a constant value between the sampling points. Each value is retained as a stair step up to the next sampling point.
ResultFormat	Format of data set with results (optional , Default value: "list")

	"list" : List: Generates a simple list of result values; one result value per interval. Without time stamp and without any time information
	"equi" : equidistant. The result has the same length as the input data. Also for data having segments. The result value for each interval is outputted (i.e. repeated) during the entire interval. For the interval format "para", plotted equidistantly over the parameter of the two-component data.
	"xy" : XY-pairs; result value with time stamp. Not for data having segments. For the interval format "para", pairings of result value and parameter of the two-component data.
Result	
Result	Result

Description:

This function can be applied to input data having events or segments.

The input data and interval data do not need to have the same sampling interval. Instead the sampling interval and the x-offset are observed. The interval data determine the times at which the input data are applied. Any differing absolute time is ignored.

For incomplete intervals and intervals which lie (partially) outside of the input data, no result value is generated.

With equidistant results or intervals, only 1 beginning or end of an interval per sampling step can be specified.

The calculations extend from the interval begining to its end. If the beginning and the end do not exactly coincide with the positions of samples, partial samples are taken into account.

if the result contains a time track, the interval beginning is stated.

If the signal is interpolated in stair-step form and the interval booundaries lie at the signal's measurement points, then the smallest imprecision in calculation can cause unstable behavior: The measured reading is sometimes to the left and sometimes to the right of the interval boundary. For this reason, it sometimes belongs to the interval and sometimes not. This can have a pronounced effect in a Min/Max-search, for example. One remedy is to locate the interval boundaries significantly away from the measurement points beforehand, e.g. in the middle between the measurement points.

Examples:

Determines RMS-values of a signal's individual oscillations. The input signal is noisy.

```
input = 10*sin (Ramp(0,1e-3,400)*PI2*10+1.1) + Random ( 400, 2, 0, 0, 35 ) ; test data
input_smooth = Smo(input, 0.01)
ivl = IntervalFromLevel( input_smooth, 0.0, -3.0, 3.0, 1, 0.01, 0.3, 0, "equi")
st = IntervalGetStatist ( input, ivl, "", "rms", 1, "list")
```

See also:

IntervalFrom2Levels, IntervalFromLevel

IntEx

Available in: Professional Edition and above

Integration, formation of the integral

Declaration:

`IntEx ( Dataset [, Calculation] [, Format] [, Resetoption] [, ResetChannel] ) -> Integral`

Parameter:

Dataset	Data set to be integrated
Calculation	Rule to be applied for calculating the integral (optional , Default value: "rect")
	"rect" : Rectangle rule. Each sample is considered to be a rectangle with the width of a sampling interval.
	"trapez" : Trapezoid rule. The samples are considered to be interpolated linearly. Thus N samples will give N-1 trapezoids with a width of a sampling interval. The result is 1 sample shorter than with the rectangle rule.
	"trapez+" : Trapezoid rule. The samples are considered to be interpolated linearly. Thus N samples will give N-1 trapezoids with the width of a sampling interval. Additionally the last sample is interpreted as a rectangle with the width of a sampling interval. The result is the same size as with the rectangle rule.
Format	Format of result (optional , Default value: "zero")
	"zero" : Preceeding zero. The first value of the integral is a zero. This expresses that up to the beginning of measurement, there is no area yet. The resulting plot of the integral is delayed. It is 1 sample longer.
	"nozero" : No preceeding zero. The zero value that results from a starting integration is skipped. The result is no longer delayed. The result is distorted, which must be taken into account when interpreting the result.
	"total" : Total integral. Only the last value of the integral is returned. The result's size is 1.
Resetoption	Execute a reset? (optional , Default value: "no")
	"no" : Never reset. The parameter ResetChannel is not specified or is set to 0.
	"reset" : Hard reset. The integral is set to zero at defined positions. The ResetChannel must be specified.
	"soft" : Soft reset. The integral tapers smoothly toward zero at defined positions. The ResetChannel must be specified.
ResetChannel	ResetChannel. Usage depends on "Resetoption" (optional)
Integral	
Integral	Integral

Description:

The data set is integrated point by point.

The data set may contain events and segments, it must be equidistant.

The unit of the integrated data set is the product of the x- and y-units of the transferred data set.

The function Int() is used with XY data.

ResetChannel

If value = 0: Continue integration. If value <> 0: Reset of the integral to zero.

The resulting dataset is set to zero at those positions where the ResetChannel is <> 0. Depending to whether a preceeding zero is used, the result may have a different shape.

Both the data set and the ResetChannel must have the same structure regarding segments and events.

The ResetChannel has the same length as the resulting integral. E.g. with the rectangle rule and a preceeding zero, the calculated signal is 1 sample longer than the data set specified. The ResetChannel shall be able to reset any of the resulting samples. Thus it is 1 sample longer, too.

If the ResetChannel is too short, zeroes will be appended at the end to fill up the gap.

If both total integral and ResetChannel are used, then internal integration follows the algorithm without a preceeding zero.

Hard and soft reset

With the strong reset, the integral will be set to zero no matter what its value is. The result will show a jump to zero.

With soft reset a straight line will be calculated. That line connects the preceeding reset with the current value of the integral. That line will be subtracted from the integral. This leads to a smooth run toward zero. If there is no preceeding reset, the start of the data is used.

As an example, consider the sequence of values following the last reset: Input data=[7,3,17], ResetChannel=[0,0,1]. That results in a sum of input values: 7+3+17=27. With those 3 values, the mean is 27/3=9. The modified input sequence after subtraction of the mean value is [7-9,3-9,17-9] = [-2,-6,8]. Summing up leads to [-2,-8,0]. The last value is zero, in accordance with the last value of the ResetChannel.

The integral behind the last reset remains unchanged upon resetting.

The soft reset leads to an integration of a signal whose mean value has been subtracted from all values following the last reset until the current one.

All reset methods counteract interfering offset which cause the integral to drift away.

Examples:

Integration of acceleration a to get speed v.

```
v = IntEx ( a, "trapez")
```

Numerical examples to illustrate the effects of different calculations and formats

```
A=[3,5,9]
B=IntEx(A)
;B=[0,3,8,17]

B=IntEx(A,"rect", "zero")
;B=[0,3,8,17]

B=IntEx(A,"rect", "nozero")
;B=[3,8,17]

B=IntEx(A,"rect", "total")
;B=17

B=IntEx(A,"trapez", "zero")
;B=[0,4,11]

B=IntEx(A,"trapez", "nozero")
;B=[4,11]

B=IntEx(A,"trapez", "total")
;B=11

B=IntEx(A,"trapez+", "zero")
;B=[0,4,11,20]

B=IntEx(A,"trapez+", "nozero")
;B=[4,11,20]

B=IntEx(A,"trapez+", "total")
;B=20
```

Numerical examples illustrating effect of reset

```
A=    [ 7, 3,17, 9,11, 7, 3, 9]
Reset=[ 0, 0, 0, 1, 0, 0, 1, 0]
B=IntEx(A,"rect", "zero", "reset", Reset )
;B=   [ 0, 7,10, 0, 9,20, 0, 3,12]

B=IntEx(A,"rect", "nozero", "reset", Reset )
;B=   [ 7,10,27, 0,11,18, 0, 9]

B=IntEx(A,"rect", "zero", "soft", Reset )
;B=   [ 0,-2,-8, 0, 0, 2, 0, 3,12]

B=IntEx(A,"rect", "nozero", "soft", Reset )
;B=   [-2,-8, 0, 0, 4, 4, 0, 9]
```

See also:

Int, MInt

IPol

Interpolation with cubic splines

Declaration:

```
IPol ( Data, SvFactor ) -> Interpolated
```

Parameter:

Data	Data set to be interpolated with cubic splines. Allowed data types: [ND], [XY]
SvFactor	Factor by which to enlarge the data set
Interpolated	
Interpolated	Spline-interpolated data set

Description:

The transferred data set is interpolated with cubic splines. The second parameter specifies the factor by which the data set is to be enlarged.

This function lays a natural cubic spline through each point in theda ta set. A high interpolation factor yields a data set with correspondingly high point density. Interpolation with cubic splines produces a very smooth, rounded curve and no sharp angles. Cubic splines are designed to prepare curves with low point densities for high-quality graphic displays

Note that cubic splines tend to overshoot slightly.

Only intermediate values are calculated during interpolation; no extrapolation is performed.

- With XY-data, the X-track must be strictly monotonically increasing.
- The units remain unchanged.
- The sampling rate of the generated data set is reduced by the given factor.
- The transferred factor may only be an integer greater than 1 and less or equal then 8192.
- Since extrapolation is not performed, the resulting data set length will not be exactly equal to the product of the original length and the given factor. It will be shorter by (factor -1) values.
- In rectangular jumps, the overshoot of cubic splines will have a particularly distorting effect. Smooth the signal or divide it into sections, which are then interpolated.
- The interpolation of signals with high-frequency noise generally yield unsatisfactory results. It is then useful to suppress the noise by smoothing the curve before executing interpolation.
- With equidistantly sampled data, the function Red() reverses interpolation.

Examples:

The point density of a data set is increased by a factor of 10, to reconstruct the probable signal path with greater detail. The graphic display will have better results and subsequent mathematical operations, such as differentiation, will be more reliable.

```
data2 = IPol(data, 10)
```

See also:

Lip, MatrixIPol, Red, Leng

IsCplx

The function inquires whether the data set passed to it is complex.

Declaration:

`IsCplx ( Data ) -> SvResult`

Parameter:

Data	Data set to examine
SvResult	
SvResult	Result
	0 : Not a complex data set
	1 : Complex data set in Real-/Imaginary-part representation [RI]
	2 : Complex data set in Magnitude-/Phase-representation [BP].
	3 : Complex data set in Magnitude(decibel)-/Phase-representation [DP].

Description:

The function checks whether the data set passed to it is complex. If so, then the form of the data set is also determined.

Examples:

A file is loaded and examined to determined whether its first data set is complex:

```
idFile = FileOpenDSF("c:\dat\test.dat", 0)
IF idFile >= 1
   test = FileObjRead(idFile, 1)
   IF IsCplx(test) = 0
      BoxMessage("Attention", "Data set is complex", "!1")
   END
   ret = FileClose(idFile)
END
```

See also:

Cmp1, Cmp2, Compl

IsXY

The function verifies whether the data set is an XY-data set.

Declaration:

```
IsXY ( Data ) -> SvResult
```

Parameter:

Data	Data set to examine
SvResult	
SvResult	Result
	0 : Not an XY-data set
	1 : XY-data set

Examples:

A file is loaded and examined whether its first data set is of the type XY. If so, it is converted to an equidistantly sampled data setbefore being subjected to further processing:

```
idFile = FileOpenDSF("c:\dat\test.dat", 0)
IF idFile >= 1
   test = FileObjRead(idFile, 1)
   IF IsXY(test) = 0
      test = XYdt(test, 0.1)
   END
   ...
   ret = FileClose(idFile)
END
```

See also:

CmpX, CmpY, XYof

Join

Joins 2 data sets together
The function is obsolete and instead the more powerful function JoinEx() should be used.

Declaration:

```
Join ( DataFront, DataBack ) -> DataJoined
```

Parameter:

DataFront	First data set; allowed types: [ND]
DataBack	Second data set; allowed types: [ND]
DataJoined	
DataJoined	Results from joining the two data sets

Description:

A typical application of this function is to generate a data set from many single values. This concatenation can be started in a loop, simply by specifying an empty data set to which values are appended each time the loop is executed. The empty data set can be specified using the symbolic constant EMPTY, for example.

The EMPTY constant is particularly convenient for sequences which call the Join function in a loop.

- When two single values are joined to form a normal data set, the following standard values are selected for the data set: sampling interval = 1, x-offset = 0.
- When one single value and a normal data set are joined, the units of the normal data set are adopted. The unit of the single value should match the y-unit of the normal data set.
- When two normal data sets are joined, units, sampling interval and x-offset of the first data set are adopted. For useful application of the function, units and sampling intervals should match.
- You can use the function Append() to achieve time-/x-correct attaching or merging of data sets.

Beginning with Version 7.6, the more powerful function JoinEx() is available. Alternatively, the functionality of the Join()-function can be replicated with initialization lists. The following 3 calls are nearly equivalent:

```
concat = Join(front, back) ;(1)
concat = JoinEx(front, back) ;(2)
concat = [front, back] ;(3)
```

The only difference is that for (1), the result is always a Real data format, while for (2) and (3), the numerical format of the parameters is preserved if possible.

Examples:

A data set is to be created containing a list of several maxima:

```
NElist = EMPTY   ;empty data set is created
NElist = Join(NElist, SVmax1)
; A number (first maximum) is appended to the list; NsList becomes a normal single value.
NElist = Join(NElist, SVmax2)
; Another value (second maximum) is appended to the list; NsList becomes a normal data set with a length of 2 etc...
```

Data recorded before the trigger are joined with data after the trigger:

```
NDcomplete = Join(NDpreTrigger, NDpostTrigger)
```

See also:

JoinEx, Append, AppendLoop, Cut

JoinEx

Connects data sets to each other

Declaration:

```
JoinEx ( DataFront, DataBack [, DH2] [, DH3] [, DH4] [, DH5] [, DH6] [, DH7] [, DH8] [, DH9] [, DH10] [, DH11]
[, DH12] [, DH13] [, DH14] ) -> JoinedData
```

Parameter:

DataFront	First data set. Types allowed: [ND].
DataBack	Data set to be appended. Types allowed: [ND].
DH2	2nd data set (optional) to append (optional)
DH3	3rd data set (optional) to append (optional)
DH4	4th data set (optional) to append (optional)
DH5	5th data set (optional) to append (optional)
DH6	6th data set (optional) to append (optional)
DH7	7th data set (optional) to append (optional)
DH8	8th data set (optional) to append (optional)
DH9	9th data set (optional) to append (optional)
DH10	10th data set (optional) to append (optional)
DH11	11th data set (optional) to append (optional)
DH12	12th data set (optional) to append (optional)
DH13	13th data set (optional) to append (optional)
DH14	14th data set (optional) to append (optional)
JoinedData	
JoinedData	Result of der concatenating all parameters

Description:

Either single values or simple data sets (equidistantly sampled, no events, no segments) can be used as parameters.

- The new data set "inherits" the units and additional characteristic values (such as pretrigger/x-offset, sampling interval/x-delta) from the first parameter having a length >= 2. In order for the function to be used in a sensible way, the units and the sampling intervals of all parameters should match.
- When single values are specified, the units of [DataForw] are used. The result's sampling interval is 1.0 and its x-offset is 0.0.
- If all parameters have the same data format, it is also used for the result. Otherwise, it is converted to the numerical format "Real 8 Byte" (double).
- You can use the function Append() to join or merge data sets in correct accordance with their respective time or x-coordinates.
- If you intend to repeatedly append data to a data set (e.g. in a loop), you should use the function AppendLoop() which is optimized for this application case.

Instead of the function JoinEx(), in most cases you can also use an **initialization list** (comma-separated initializers in square brackets). The following two lines, for example, are equivalent:

```
Mean3Days = Mean(JoinEx(BeforeYesterday, Yesterday, Today))
Mean3Days = Mean([BeforeYesterday, Yesterday, Today])
```

Examples:

A data set is supplemented with a beginning and a final 0:

```
Data = JoinEx(0, Data, 0)
```

The data measured on 3 days in succession are joined to a single data set:

```
Data3Days = JoinEx(BeforeYesterday, Yesterday, Today)
```

In an event-based data set, the maxima of all events are found and copied to a new data set.

```
allMax = EMPTY
FOREACH EVENT ev IN data
```

```
   allMax = JoinEx(allMax, max(ev))
END
```

Note: For such applications it is often more efficient to use the function AppendLoop() since it is significantly faster performing frequent and repeated appending.

See also:

Join, Append, AppendLoop, Cut

KBT

KB-weighting and maximum-rate values

Declaration:

```
KBT ( Data, SvLowerFrequency, SvUpperFrequency, SvIntegrations, SvTimeRating, SvRate, SvNorm, SvMinimum, Zero )
-> KBData
```

Parameter:

Data	Data set to be analyzed
SvLowerFrequency	The lower cut-off frequency of the band-pass (>0!) should generally be set to 1 Hz. The program automatically selects a frequency 0.8 times smaller for the bandpass, i.e. 0.8Hz. If the specified upper and lower cut-off frequencies are less than zero, no filtering (band-pass and high-pass) or integration is performed.
SvUpperFrequency	The upper cut-off frequency of the band-pass (>0!), should generally be set to 80 for 80Hz. The program automatically selects a frequency 1/0.8 times larger, i.e. 100Hz.
SvIntegrations	Number of integrations to be executed. If you have an acceleration signal, for example, it must be integrated once to yield a velocity signal. The integrations and differentiations are executed only in combination with bandpass filtering.
	0 : No integration (default)
	1 : One integration
	2 : Double integral
	-1 : First derivative
	-2 : Second derivative
SvTimeRating	Averaging time for calculating the moving RMS (time rating).
	>0 : Free averaging duration
	0 : No RMS calculation
	-1 : FAST (0.125s)
SvRate	The rate at which the maximum rate values should be calculated
	0 : No calculation of maximum rate values.
	>0 : A period which you specify, e.g. 30 for 30 seconds. If the rate is less than the sampling rate, the latter is used as the rate. If the rate is longer than the duration of the source data set, the complete source data set is used. If the data set is longer than one interval and the last period extends beyond the data set, only the available measurement points are used. It is then up to the user to include the last maximum rate values.
SvNorm	This factor is used to multiply the result for normalization. (0 means no normalization. Multiplication is omitted.)
SvMinimum	Results below this value are set to zero. This excising of smaller values is only performed when a moving RMS is calculated. It makes no sense to use negative normalization! Since no negative values can occur after RMS calculation, [Minimum]=0 is used if no data are to be excised.
Zero	Reserved parameter; set to 0.
KBData	
KBData	Result of the evaluation

Description:

KB-weighting is generally performed to evaluate vibrations, especially in the frequency range of 1 Hz to 80 Hz. Vibrations in this range are usually perceived as unpleasant. KB-weighting is used prevalently in noise pollution protection for performing standards-compliant evaluation.

The KBT function provides you with the complete KB-weighting including determination of maximum rate values.

The KB-weighting is performed in accordance with DIN, in which the following steps are performed in succession:

- It is assumed that the measurement signal is unweighted (speed signal, velocity over time).
- This signal frequency range is clipped: 1 Hz and 80 Hz (normal); 1 Hz - 315 Hz (blasting vibrations) or 4 Hz - 80 Hz (railways). Part 1 of DIN 45669 specifies a Butterworth band-pass filter of fourth order, with the lower cut-off frequency reduced by a factor of 0.8 and the upper cut-off frequency increased by a factor of 1 / 0.8 (e.g. 0.8 Hz and 100 Hz).
- The clipped signal is filtered with a high-pass filter of first order and a cut-off frequency of 5.6 Hz. The frequency-weighted vibration signal is the result.
- The signal can then be integrated or differentiated to yield a velocity signal, since filtering has removed high-frequency noise (differentiation is thus more accurate) as well as eliminating the low-frequency drifts and offsets, improving the result of integration. This is not prescribed by DIN, but of great practical importance. Integration and differentiation are performed by extending the digital filter for the

bandpass with zeros or poles at frequency zero.
- The weighted vibration strength is extracted from the frequency-weighted signal by calculating the exponentially-weighted moving RMS. The moving RMS is calculated in the following way: The signal is squared, then exponentially averaged, then its square root is taken. Squaring, averaging and subsequent root extraction are characteristic of forming the RMS. With a regular RMS value, an evenly weighted average of all squares is calculated, while in this case for the moving RMS, time-weighting is performed. The exponential averaging used causes a sort of "forgetting". It can also be regarded as a 1st order low-pass filter. The time constant for the moving is 0.125s for the moving RMS according to DIN4150 Part 2.
- The next step is to find the maxima of the rated vibration strength (result of RMS calculation) In each interval of 30 s length, the maximum value is determined. These are then the maximum rate values.
- Now the values are normalized by multiplying the signal by a specified factor, for example to implement a defined scaling
- All maximum rate values below a defined threshold are set to zero. For any subsequent calculation of maximum rate values, only values greater than 0.1 are considered, the others are set to zero.

Application for determination only of the maximum rate values, e.g. to be calculated using 1/3-octave spectra. Set the following parameter values:

```
SvLowerFrequency = -1
SvUpperFrequency = -1
SvIntegrations = 0
SvTimeRating = 0
User's rate, e.g. 30s
SvNorm = 0
SvMinimum = 0
```

Application as a band-pass function (note that the high-pass with cut-off frequency 5.6 Hz is included in filtering!). Set the following parameters:

```
SvLowerFrequency = Lower cut-off frequency / * 0.8
SvUpperFrequency = Upper cut-off frequency / 0.8
SvIntegrations = 0
SvTimeRating = 0
SvRate = 0
SvNorm = 0
SvMinimum = 0
```

Application to calculate a moving RMS. Set the following parameters:

```
SvLowerFrequency = -1
SvUpperFrequency = -1
SvIntegrations = 0
SvTimeRating = User's averaging time constant or -1 for 0.125s
SvRate = 0
SvNorm = 0
SvMinimum = 0
```

The algorithms conform to:

- DIN 4150 Teil 2, Vibrations from construction sites, effect on people in buildings
- DIN 45669 Teil 1, Measurements of vibration emissions, vibration meters, requirements, testing
- The calculation of the maximum rate RMS value is not included in this function; use the RMS function instead.

Attention, time constant!

If you use the moving RMS-value formation, note that the effect is different than that of a simple low-pass. The time constant cannot be read directly from a graph of the filtered data because the root of the signal is computed after multiplication with the time constant. The time constant can however be made visible by squaring the results! The time constant can be read relatively well for a square wave input signal if no filtering is used.

Sampling rates

If the function performs filtering, a digital filter is used. The cutoff frequencies of this digital filter can only be less that half of the sampling frequency. Higher cutoff frequencies are not possible.

Filtering precision

The design of the digital filters used for the bandpass and high-pass filtering is only useful for frequencies significantly below half of the sampling frequency. In order to be able to use the bandpass at all, the sampling frequency should be at least 1kHz at an upper cutoff frequency of 80Hz.

Moving RMS value

The averaging time can be larger or smaller than the sampling rate. A zero-order approximation is performed (step signal).

Initial transient

When applying band-pass filters, remember the start-up signal, which can be approximated by calculating 1/frequency difference. The lower cut-off frequency for the start-up signal is significant in selecting the number of integrations. During start-up of the filter, no conclusions can be drawn from the result, so ignore the first values of the resulting data set. The start-up is not a problem specific to imc FAMOS or to digital filters, but applies to filters in general.

Range of values

Note the range of values permitted when selecting a normalizing factor. Any values in the resulting data set exceeding the permitted range are set to zero.

Unit

The x-unit of the source data set is expected in seconds. The time information specified with the function is expected in the same unit; frequencies in Hz. The result always has the y-unit of the source data, even in integration.

Data type

The type of the result is an equidistant data set. If a single value is calculated (one maximum rate value), a single value is returned.

Y-scaling

The calculated values are always expressed as linear values. If display in dB is desired, select the mathematical function dB (..) or the corresponding dB display in the curve window.

Examples:

The standard application of the function: a data set a with acceleration values is to be evaluated. Its sampling rate is 1 ms, and a frequency range of 1 Hz to 80 Hz is selected. Since the signal originates from an acceleration sensor, single integration is required to yield a velocity signal. The FAST evaluation (0.125 s) is used for calculation of the RMS value. Maximum rate values are calculated in intervals of 30 s. No further normalization is performed; maximum rate values less than 0.1 are set to 0. A last function call calculates the RMS of the maximum rate.

```
KBFTI = KBT(a, 1.0, 80.0, 1, -1, 30, 0, 0.1, 0)
KBFTM = RMS(KBFTI)
```

Intermediate results can be viewed by separating the compact KBT function call into separate steps: first the frequency rating, then the moving RMS calculation, and finally determination of the maximum rate values.

```
FrRating = KBT(a, 1.0, 80.0, 1, 0, 0, 0, 0, 0)
rms = KBT(FrRating, -1, -1, 0, 0.125, 0, 0, 0.1, 0)
KBFTI = KBT(rms, -1, -1, 0, 0, 30, 0, 0, 0)
KBFTM = RMS(KBFTI)
```

See also:

ExpoRMS, RMS, OctA, ABCRating, MInt

LAYOUT

Report Generator-Remote Control
The command is obsolete; instead of it, the more powerful commands of the Report Generator Kit such as RgDocOpen(), RgDocPrint() and RgCurveSet()/RgTextSet() should be used.

Declaration:

```
LAYOUT Task Name Variable
```

Parameter:

Task	Command to be executed
	PRINT : Prints the current report
	LOAD : Loads a Report-file
	OBJECT : A variable is transferred to a Report-object
Name	For LOAD and OBJECT: Filename or object title, respectively
Variable	For OBJECT: variable to transfer

Description

```
LAYOUT PRINT
The current report is printed out
```

```
LAYOUT LOAD
```

A Report-configuration file is loaded to the Report Generator. If you set placeholders in the Report Generator, use this command to restore all placeholders.

The Report Generator is started as an icon. Since sequences only require the Report Generator in conjunction with configurations to be loaded, it doesn't need to be called separately.

The specification for the filename may also be a complete path name, if the desired configuration file is not in the default folder. You can set the default folder for report either in the Report Generator itself or in the dialog "Options"/ "Folders".

```
LAYOUT OBJECT
```

A variable is passed to a Report Generator object. The desired object in the layout is selected by its title (name). Variables can be passed to text or curve objects. Before transfer of a curve object, the data set must be displayed in a curve window.

Examples:

```
LOAD Data.dat
SHOW Data
Txt = "Created by S.Smith"
LAYOUT LOAD c:\imc\drb\report1.drb
LAYOUT OBJECT Curve Data
LAYOUT OBJECT Name Txt
LAYOUT PRINT
```

A report is loaded. A curve and text object is transferred to the report, which is then printed.

See also:

RgDocOpen, RgDocPrint, RgCurveSet, RgTextSet

LDIR

Sets the folder for loading files

Declaration:

```
LDIR Folder
```

Parameter:

Folder	Complete pathname of the desired folder

Description

Instead of the command LDIR, the function SetOption() should be used in newly created sequences.

The folder for loading files is given a new setting.

Once this command is executed, this folder is used for loading files.

The folder name may also be expressedd in quotation marks. This is obligatory when the name contains spaces.

This command can also be called without any parameters (so without any specified fodler). Then the folder set under "Options/Folders" is again used as the default.

The folder elected here remains valid until:

- the command LDIR is called again
- the function SetOption("Dir.DataFiles",...) is called
- a file is loaded from a different folder using the dialog box "Load" in the menu "File"
- a new folder for loading files is set using the dialog under "Options"/ "Folders"

Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
LOAD DATA1
LDIR C:\TEST
LOAD DATA2
```

From the folder currently set, the file DATA1 is loaded in imc/FAMOS-format; subsequently the file C:\TEST\ DATA2.DAT.

```
LDIR "c:\My Tests on 12/1/98"
```

The pathname contains spaces and must therefore be written in quotation marks.

See also:

SetOption, LOAD, SDIR, MDIR, FileLoad, FileOpenDSF

Leng

Specifies a data set's number of values

Declaration:

```
Leng ( Data, SvLeng ) -> Result
```

Parameter:

Data	Data set whose length is to be changed
SvLeng	Length of the data set to be generated
Result	
Result	Data set with changed length

Description:

The length (data point count) of a data set is changed to a new value. If the new point count is less than the old one, the data set is truncated (shortened). If the new point count is higher, the data set is filled with zeroes up to the new length (for scalable integer data formats the unscaled raw values are set to zero)

- The second parameter should not have a unit, since it is a pure number.
- The data set's new point count can only be a whole number >= 0.
- The new length of the data set is a count of data points, not an x-coordinate/-range.

Examples:

The first 200 data points in a data set oare to form a new data set.

```
NDnew = Leng(NDold, 200)
```

A data set of 600 points in length is truncated to 512 points and subsequently filled with zeroes to a length of 1024 points. If this data set is used in an ACF or CCF, the signal is interpreted as non-recurring, i.e. not periodic.

```
NDHalfZeroes = Leng(Leng(NDdata, 512), 1024)
```

A data set with 1024 points is extended to 2048 points with a sin(x) / x - interpolator. This is achieved by lengthening the spectrum with zeroes and then re-transforming it. Use the rectangular window!

```
NDintpol = iFFT(Leng(FFT(NDdata), 1025))
```

See also:

Leng?, MatrixChangeDim, Cut, XDel, XOff, Red, IPol

Leng?

Finds a data set's length (value count)

Declaration:

```
Leng? ( Data ) -> SvLeng
```

Parameter:

Data	Data set whose length is to be found
SvLeng	
SvLeng	The parameter's length (point count)

Description:

A data set's length (data point count) is determined.

A count of data points is returned, not an x-coordinate differential.

Examples:

The duration of a data set is determined (product of sampling interval and point count):

```
duration = XDel?(NDdata) * Leng?(NDdata)
```

A histogram is standardized to 100% over the point count of the data set under examination:

```
NDnormhi= 100'%' * Histo(NDdata, 0, 0)/Leng?(NDdata)
```

The length of the single value is defined as 1. An empty data set's length is zero.

```
SvOne = Leng?(100)
SvZero = Leng?(EMPTY)
```

See also:

Leng, XDel?, XOff?, Time?, MatrixInfo

LFit

Linear regression; fitting to line

Declaration:

```
LFit ( Data ) -> Line
```

Parameter:

Data	Data set to which a line of best fit is to be found [ND],[XY]
Line	
Line	Regression line

Description:

A data set given by the equation

```
f(x) = A * x + B
```

determined, which best approximates the data set passed to the function. The coefficients A and B are determined accordingly. This function is based on the algorithm of linear regression, which determines an optimal straight line using the method of least squares.

Upon completion of the function, the equation is displayed in the output box in the imc FAMOS main window. The equation is displayed with units and is in the format:

```
 f(x) = 5.834 [Ohm] * x + 12.29 [V]
```

Calling the function with an empty text as the parameter returns the last coefficients calculated, A and B, in the form of a data set of two values.

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Normal data set [NW]:

The data set generated has the length and unit of the specified data set, with a maximum length of 100. If the length is shortened (truncated), the sampling time is changed so that the specified data set is defined for the same interval.

XY -data set [XY]

The generated data set has the length 2. It is defined over the same x-interval as the source data set.

Examples:

The linear regression for a data set calculated. Subsequently, the slope and offset of the regression line are computed.

```
RegressionLine = LFit(Daten)
coeff = LFit("")
factor= coeff[1]
offset = coeff[2]
```

The deviation from the best straight line is determined. The function RSamp is used first to align the sampling times before the difference is calculated:

```
NDerror = NDdata - RSamp(LFit(NDdata), NDdata)
```

See also:

eFit, RSampEx, Appro, ApproNonLin, Poly

Lip

Linear interpolation

Declaration:

```
Lip ( Data, SvFactor ) -> Interpolated
```

Parameter:

Data	Data set to be linearly interpolated [ND],[XY]
SvFactor	Factor by which to enlarge the data set
Interpolated	
Interpolated	Linearly interpolated data set

Description:

The data set passed is linearly interpolated. The 2nd parameter states the factor by which the data set is to be enlarged. With linear interpolation, it is imagined that the data set's points are connected by straight diagonal lines. Only intermediate values are calculated; there is no extrapolation.

- The units remain unchanged.
- With equidistantly sampled data [ND], the sampling interval is smaller by the factor specified. The interpolation can be reversed by the function Red().
- The factor specified may only be an integer greater than 1.
- Since extrapolation is not performed, the resulting data set length will not be exactly equal to the product of the original length and the given factor. It will be shorter by (factor -1) values. With graphical display of curves, there generally is linear interpolation already, so that this function doesn't need to be used for a graph.

Examples:

The point density of a data set is increased by a factor of 3:

```
data2 = Lip(data, 3)
```

See also:

IPol, MatrixIPol, Red, Leng

ln

Natural base e (Euler number) logarithm

Declaration:

```
ln ( Data ) -> Result
```

Parameter:

Data	Data; allowed types: [ND],[XY].
Result	
Result	The parameter's natural logarithm

Description:

The base e (Euler number e = 2.718....) logarithm is calculated. This logarithm is called the natural logarithm.

Remarks

- The x-coordinate(s) of the parameter and the result are the same.
- The parameter of the function should have no unit. But if it does have one, it is retained (a warning is issued).
- The parameter of the function ln() should always be greater than zero.
- The functions exp() and ln() are the inverse of each other.
- The parameter may be structured (events/segments).

Examples:

The natural logarithm of the predefined constant e is 1:

```
one = ln(e)
```

The following formula does not change the number. However, note the reduced definition range of the function exp():

```
number = ln(exp(number))
```

See also:

log, exp, ^(hoch)

LOAD

Open file

Declaration:

```
LOAD Filename VariableName
```

Parameter:

Filename	Name of the file to be loaded
VariableName	Name under which the variable is to be entered in the variable list

Description

This command is obsolete; instead of it you can use the more powerful and convenient function FileLoad().

The specified file is loaded in the currently set file format (FAMOS, BINARY or ASCII) and its data objects entered in the variable list. A complete file name, including the path, can be specified. If the second parameter is missing, the file name without directory and extension is used as the variable name.

In FAMOS files created with the version 3.0, the variable names are stored in the file; the variables are then created under the same name after being loaded in FAMOS. When the second parameter is specified and the file contains exactly one data object, then the object is stored under the specified name. If there are several data objects, all objects are read and the specified variable name is ignored.

Wildcards

Wildcards can also be specified in file names to load a series of files. The wildcard '?' stands for an exact character, '*' stands for an undefined number of any characters.

```
LOAD c:\imc\*.*
```

All files are loaded in the directory c:\imc.

```
LOAD a??.*
```

All files whose names begin with 'a' and consist of three characters are loaded in the current directory. Any file name extension can be used.

```
LOAD *a1*.dat
```

All files with the extension 'dat', whose names contain the string '1a' (at the beginning or end), are loaded.

Indirect calls and filenames with special characters

If the filename is in a text variable, the call is written in angle brackets:

```
MyFileName= "C:\Data\Signal.dat"
LOAD <MyFilename>
```

If the filename contains spaces, periods or other special characters, indirect calling by means of a text variable is obligatory. Additionally, the variable must be enclosed in more quotation marks:

```
MyFilename= """C:\Data\1.1 Signal at 20°C.dat"""
LOAD <MyFilename>
```

- If the FAMOS data format is set, a file name extension must not be specified. The extension ".DAT" is automatically appended.
- End a file name with a period if an extension is not desired.
- If no path is specified, the path set for loading files is used. Once FAMOS is started, this path is found in the standard directory in the dialog 'Options'/ 'Directories'. It can be changed using the command LDIR or the function SetOption().
- The filename may also be written within quotation marks. This can be necessary, if, for instance, the path contains spaces.
- In order to load files whos format you defined by means of the imc Fil Assistant, please use the function FileOpenFAS() or the command FASLOAD.
- The functions of the group FileOpen*() can also be used to load FAMOS files. These are much more powerful than the command LOAD, especially for loading files containing multiple data sets.

Examples:

```
FAMOS
LOAD c:\imc\dat\Data var1
```

The file WAVE.DAT is loaded in FAMOS format and the resulting variable is entered in the variable list under the name "var1".

```
ASCII
LOAD DBAS.ASC
```

The specified ASCII file is loaded and entered in the variable list under the name DBAS.

```
BINARY
LDIR c:\BINDATA
LOAD *.BIN
```

All files with the extension 'BIN' are loaded in binary format in the specified directory.

```
LOAD "c:\imc\My data files\Data"
```

The filename contains a space and therefore it must be written inside of quotation marks.

See also:

FileLoad, FileOpenDSF, LDIR, FAMOS, BINARY, ASCII

LOCAL

This instruction serves to declare local variables

Declaration:

```
LOCAL VariableName
```

Parameter:

VariableName	One or more variables' names (may include wildcard characters)

Description

Variables can be declared as 'local' within a sequence. Local variables are only valid during the running of the current sequence and can only be used within these; at the end of the sequence execution, such variables are deleted automatically.

The declaration can be made either directly or in the assignment:

```
LOCAL temp =  Ramp(0, 1, 100)
```

Or by forward declaration:

```
LOCAL temp
temp =  Ramp(0, 1, 100)
```

In the forward declaration, it is also possible to use wildcard characters ('*' - representing any amount of characters, '?' - exactly one arbitrary character):

```
LOCAL ?    ; all variables whose name is exactly one character long
LOCAL #*   ; all variables whose name starts with '#'
```

It is also possible to specify multiple name patterns, separated either by spaces or a comma:

```
LOCAL ?,#*
```

Designation

The internally used (complete) name of local variables is structured as follows: [Name]@![SequenceName] Applying the measurement design concept in FAMOS, the local variables are thus assignedd to a 'virtual' measurement, whose name is derived from the name of the current sequence. In the Variables list, they are also listed under their full name (e.g. when running the sequence in single-step mode or when sequence execution is cancelled in consequence of an error).

Name duplicates with permanent (global) variables:

- When generating local variables, any possibly already existing permanent variable having the same name is ignored.
- In the search for variables, the local variable 'wins' over any existing permanent variable of the same name.

```
X = 1  ; permanent variable
LOCAL X
X = 2  ; new local variable
Y = X  ; Y has the value 2 (like the local variable)
```

Special characteristics and limitatations of local variables:

- Local variables are ignored by the functions for querying the Variables list (VarGetInit, VarGetInit2, VarExist?).
- They can not be linked with Panel- or dialog elements.
- They are not visible for queries by means of DDE.
- They are not displayed in the Variables list/Measurement view.
- They are not saved along with the project.
- The declaration of a local variable only makes sense in the context of a sequence, in the 'Input'-window, the LOCAL-command is ignored for this reason.
- The forward declaration only applies to variables which are the direct result of an assignment. Variable which are the result of commands (e.g. LOAD, QUERY etc) are never local.

See also:

log

Base 10 logarithm

Declaration:

```
log ( Data ) -> Result
```

Parameter:

Data	Data; allowed types: [ND],[XY].
Result	
Result	Base 10 logarithm of the parameter

Description:

Computes the base 10 log

Remarks

- The x-coordinate(s) of the parameter and the result are the same.
- The parameter of the function should have no unit. But if it does have one, it is retained (a warning is issued).
- The parameter of the function log() should always be greater than zero.
- The parameter may be structured (events/segments).

Examples:

```
zero = log(1)
```

E.g.: log(0.1)= -1, log(1)= 0, log(10)= 1

See also:

ln, dB, ^(hoch)

LogSetup

Configures log output

Declaration:

```
LogSetup ( TxTarget [, SvReset] [, TxTimeStamp] [, TxOutputLevel] )
```

Parameter:

TxTarget	Determines the target for subsequent LogTrace()-output. If an empty string is specified, the messages are written in the output window of the FAMOS main window. Otherwise, a filename is expected here.
SvReset	Resets display/file (optional , Default value: 0)
	0 : Don't delete
	1 : Deletes output window/empties file.
TxTimeStamp	Outputs messages with a preceding time stamp. (optional , Default value: "")
	"" : No time stamp
	"hh:mm:ss" : Time (precision: 1 second)
	"hh:mm" : Time (precision: 1 minute)
	"hh:mm:ss,0" : Time (precision: 1/10 second)
	"d hh:mm:ss" : Date + Time (precision: 1 second)
	"d hh:mm" : Date + Time (precision: 1 minute)
	"d hh:mm:ss,0" : Date + Time (precision: 1/10 second)
TxOutputLevel	LogTrace()-commands configured with a lower importance are ignored. (optional , Default value: "*")
	"-" : Ignores all LogTrace()-outputs.
	"e" : Only output important errors.
	"e2" : Output all errors (e, e2).
	"w" : Outputs all errors and important warnings (e, e2, w).
	"w2" : Outputs all errors and all warnings (e, e2, w, w2).
	"i" : Output all errors, all warnings and important information (e, e2, w, w2, i).
	"*" : Output all

Description:

When a filename without a folder is specified for [TxTarget], the default sequence folder is used. If no file extension is specified, then ".log" is used.

The output file is written as a text file in the UTF-8 character set.

For output to a file, error messages and warnings either generated by FAMOS' Formula Interpreter or when executing functions are also logged.

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

A sequence loads all measurement files having a specified name pattern from a folder. For each file, a log entry is generated and appended to the log file. Each entry consists of a time stamp, filename and status (success/error) of the operation.

```
logLevel = "*"
LogSetup("c:\log\protocol.txt", 0, "d hh:mm:ss,0", logLevel)
SetOption("Func.ErrorBoxes", "No")  ; No error boxes for file functions!
fileNames = FsGetFileNames("c:\imc\dat","a*.dat", 0, 0, 0)
FOREACH ELEMENT name in fileNames
   IF FileLoad(name, "", 0) >= 0
      LogTrace("Load file " + name + " => Success")
   ELSE
      LogTrace("Load file " + name + " => Failure", "e")
   END
END
```

Note: As a centralized way to switch logging off completely, set the variable [logLevel] to "-". In order to only output the error messages, set [logLevel] to "e".

The log file generated in the previous example is later loaded again and displayed in the output window:

```
LogSetup("", 1)
LogTrace("--- Contents of the log file ---")
idFile = FileOpenASCII("c:\log\protocol.txt", 0)
TxLine = ""
ok = FileLineRead(idFile, TxZeile, 0)
WHILE ok = 0
   LogTrace(TxZeile)
   ok = FileLineRead(idFile, TxZeile, 0)
END
FileClose(idFile)
```

See also:

LogTrace, BoxOutput

LogTrace

Outputs a text to the output window or a file.

Declaration:

```
LogTrace ( TxLogText [, TxLevel] )
```

Parameter:

TxLogText	Text to be outputted
TxLevel	Characterizes the importance of the output; is checked against the output level specified in LogSetup() - if its [TxLevel] is lower (less important), the command is ignored. (optional , Default value: "i2")
	"e" : Error
	"e2" : Lower priority error
	"w" : Warning
	"w2" : Lower priority warning
	"i" : Information
	"i2" : Lower priority information

Description:

The function writes the specified text to the log. The output medium (text-file or output window) and output format (e.g. preceding time stamp) are configured by means of the function LogSetup().

For output to a file, the text is prefixed with an "(E)", "(W)" or "(I)", depending on [TxLevel]. "E" stands for error, "W" for warning, "I" for information. When logging data in the output window, the level is indicated by a corresponding symbol.

Examples:

A sequence loads all measurement files having a specified name pattern from a folder. For each file, a log entry is generated and appended to the log file. Each entry consists of a time stamp, filename and status (success/error) of the operation.

```
logLevel = "*"
LogSetup("c:\log\protocol.txt", 0, "d hh:mm:ss,0", logLevel)
SetOption("Func.ErrorBoxes", "No")  ; No error boxes for file functions!
fileNames = FsGetFileNames("c:\imc\dat","a*.dat", 0, 0, 0)
FOREACH ELEMENT name in fileNames
   IF FileLoad(name, "", 0) >= 0
      LogTrace("Load file " + name + " => Success")
   ELSE
      LogTrace("Load file " + name + " => Failure", "e")
   END
END
```

Note: As a centralized way to switch logging off completely, set the variable [logLevel] to "-". In order to only output the error messages, set [logLevel] to "e".

The log file generated in the previous example is later loaded again and displayed in the output window:

```
LogSetup("", 1)
LogTrace("--- Contents of the log file ---")
idFile = FileOpenASCII("c:\log\protocol.txt", 0)
TxLine = ""
status = FileLineRead(idFile, TxZeile, 0)
WHILE status = 0
   LogTrace(TxZeile)
   status = FileLineRead(idFile, TxZeile, 0)
END
FileClose(idFile)
```

See also:

LogSetup, BoxOutput, FileLineWrite

LostValueFill

Available in: Professional Edition and above

Fills the gaps between events, where the events result from the presence of Lost Value, Overflow, Not a Number, Overmodulation, Sensor Breakage.

Declaration:

```
LostValueFill ( input data [, Reference] [, ReplaceHow] [, Substitute value] [, Time_Option] [, MaxLen] ) ->
Result
```

Parameter:

input data	input data
Reference	Reference. If an empty text "", then there is no reference. Calculation of the start and end, and of the length of gaps is performed solely on the basis of x0 and the trigger times. When a reference channel is available, the start and end are determined by the reference channel. Thus, the actual start can be located before the beginning of the first event, and the end can be after the end of the last event. The reference channel must have the same sampling rate as the input data. (optional , Default value: "")
ReplaceHow	How to effect the replacement (optional , Default value: 0)
	0 : linear interpolation: between the last valid value before the gap and the first valid one after it
	1 : constant continuation
	2 : constant next: the next adjacent valid edge value is used
	3 : fixed value which is specified as the subsequent parameter ReplacementValue
Substitute value	The replacement value, if a fixed defined value is intended as the replacement value. Else 0. (optional , Default value: 0)
Time_Option	Time_Option (optional , Default value: 0)
	0 : The events' trigger times are not taken into account.
	1 : The events' trigger times are taken into account.
MaxLen	Maximum length of the result, in Samples = 0, if there is no limit. Setting limits is apporopriate if the input data might not originate from a proper measurement. In such cases, gigantic volumes of data can be generated due to unfavorable values for the event's trigger times or its x-offsets (x0). If the result is anticipated to become longer than the length defined here, the function cancels and posts an error message. (optional , Default value: 0)
Result	
Result	Result

Description:

Lost Values in a data set may have caused the data set to have been subdivided into multiple events. None of these individual events contain any Lost Values. The Lost Values can be considered to be located in the gaps between the events.

For some analyses it may be advantageous to append all the events back together to a single data set which correctly reflects the time relationships. The gaps between the events are replaced with new inserted values.

Lost Value

The Lost Value refers to a missing or invalid or "not actually available" value in a data set. For instance, a reading may go missing due to overflow of a buffer memory, or due to faulty transmission via a bus-system. The correct reading may not be possible to recover due to breakage of a sensor (ruptured thermocouple) or to overmodulation of a measurement amplifier. If an equidistantly sampled measurement channel needs a value, but none is available, a Lost Value is used instead.

With Real number formats, a Lost Value is frequently represented as 1e100; or as Not a Number, also represented as 1e100. With Integer number formats, usually a value at the end of the respective number range is selected. Thus for instance -32768 for 16-bit signed Integers.

In overmodulation of measurement amplifiers, readings are usually generated which lie (slightly) outside of the measurement range. They have typically been attenuated either by means of hardware or firmware.

In case of sensor (thermocouple) breakage, many measurement devices write a replacement value of approx. -2000 to the temperature channel. Thus, checking for values <=-1000 would detect these sensor breakage values.

Not a Number (NaN) is the binary code which the computer uses for the Real number data type when the value provided is either not a number or no numerical value is available for it. In imc FAMOS, the value is usually represented as 1e100. Not a Number can be used to represent a Lost Value. Not a Number can also result from arithmetical operations when the permitted numerical range has been violated.

Parameter

The input data are equidistant and are to comprise events. Input data without events are treated as data having one event.

The ReplaceHow option and the replacement value determine how the replacement is achieved.

Examples:

A test station program returned multiple events although measuring a continuous data stream, because data went missing due to (a few) network outages. All events are assigned the same trigger time; the offsets are expressed by the events' respective x0 values. The analysis software requires one single contiguous data set without any events. The overall measurement is anticipated to produce approx. 2e6 readings.

```
T = LostValueFill( T_Evn, "", 0, 0, 0, 1e7)
```

In a long-term test run, there are two temperature channels T1 and T2, sampled at the same rate during the entire measurement. However, T2 was subdivided into multiple events due to an overflow. For the subsequent analysis, both channels must have the same time frame.

```
T2_NoEvn = LostValueFill( T2, T1, 0, 0, 0, 1e7)
Delta = T2_NoEvn - T1
```

See also:

LostValueReplace, LostValueGaps, RSampEx

LostValueGaps

Available in: Professional Edition and above

Any instances of Lost Value, Overflow, Not a Number, Overmodulation, or Sensor Breakage are replaced with gaps. Gaps are represented as Events or XY-format.

Declaration:

```
LostValueGaps ( input data, ResultsFormat, FindHow [, Limit] ) -> Result
```

Parameter:

input data	input data
ResultsFormat	Format of the result: Should the gaps be represented by Events or by means of XY-values?
	0 : Events are generated. Each event comprises a compact sequence of values without Lost Values.
	1 : The data type XY is generated. Value pairs with Lost Values do not appear in the result.
FindHow	A condition specifying how Lost Values are found in the data set
	0 : =NaN (Not a Number)
	1 : >=Limit
	2 : <=Limit
	3 : >Limit
	4 : <Limit
	5 : >=Limit or <=-Limit
	6 : <Limit and >-Limit
Limit	Limit which appears in the condition FindHow (optional , Default value: 1e100)
Result	
Result	Result

Description:

If any Lost Values appear in a data set, the system must react to them for purposes of analysis and display. This is because the Lost Values contain important information. Otherwise, the correct values will be missing at these locations.

There are situations in which the Lost Values need to be simply excised so that gaps are left in the data. The application interprets the gaps accordingly.

The gaps can be represented by forcing the incorporation of an XY-format in the result. The XY-pairs containing a Lost Value do not appear.

It can be helpful to generate the XY-format no matter how often or randomly the LostValues occur. Display of the measured points in the curve window always clearly indicates at which locations there are genuine measured readings.

Alternatively, gaps can be represented by allocating sections of data to Events not having any LostValues. An Event then always comprises a compact sequence of valid values. The Lost Values would then be considered to be located in the gaps between the events.

Generating events is only appropriate when there are a few locations with bunched occurrence of LostValues. The opposite of this is when there are only a few contiguous regions in which there are no Lost Values. For reasons of efficiency it is worth avoiding the generation of too many brief events.

Another method of dealing with Lost Values makes use of the function LostValueReplace(). Which method is appropriate depends on the application.

Lost Value

The Lost Value refers to a missing or invalid or "not actually available" value in a data set. For instance, a reading may go missing due to overflow of a buffer memory, or due to faulty transmission via a bus-system. The correct reading may not be possible to recover due to breakage of a sensor (ruptured thermocouple) or to overmodulation of a measurement amplifier. If an equidistantly sampled measurement channel needs a value, but none is available, a Lost Value is used instead.

With Real number formats, a Lost Value is frequently represented as 1e100; or as Not a Number, also represented as 1e100. With Integer number formats, usually a value at the end of the respective number range is selected. Thus for instance -32768 for 16-bit signed Integers.

In overmodulation of measurement amplifiers, readings are usually generated which lie (slightly) outside of the measurement range. They have typically been attenuated either by means of hardware or firmware.

In case of sensor (thermocouple) breakage, many measurement devices write a replacement value of approx. -2000 to the temperature channel. Thus, checking for values <=-1000 would detect these sensor breakage values.

Not a Number (NaN) is the binary code which the computer uses for the Real number data type when the value provided is either not a number or no numerical value is available for it. In imc FAMOS, the value is usually represented as 1e100. Not a Number can be used to represent a Lost Value. Not a Number can also result from arithmetical operations when the permitted numerical range has been violated.

Parameter

The input data can be expressed either as equidistant or in XY-format with monotonically increasing x-axis. Events are possible, segments are not.

The FindHow condition and the comparison value Limit specify which values are to be replaced.

Examples:

A plot of temperature sampled at 10Hz which returns a value of -2000 in case of sensor breakage is corrected for purposes of subsequent analysis: An XY-format is generated, in which all value pairs with a sensor breakage do not appear.

```
T_xy = LostValueGaps(T, 0, 4, -1000)
```

The rotation speed has been measured in a long-term test on a test bench. There had been multiple network outages, so that for several minutes no readings were transmitted and the buffers overflowed. The data saving software inserted NaN (Not a Number) values at the corresponding locations in the measurement channels. A data set comprising multiple events is to be generated from this, in which all NaN are to be located in the gaps between the events.

```
Speed_Evn = LostValueGaps(Speed, 1, 0, 0)
```

See also:

LostValueReplace, LostValueFill

LostValueReplace

Available in: Professional Edition and above

Any values of Lost Value, Overflow, Not a Number, Overmodulation, Sensor Breakage are replaced in the data set.

Declaration:

`LostValueReplace ( input data, FindHow [, Limit] [, ReplaceHow] [, Substitute value] ) -> Result`

Parameter:

input data	input data
FindHow	A condition specifying how Lost Values are found in the data set
	0 : =NaN (Not a Number)
	1 : >=Limit
	2 : <=Limit
	3 : >Limit
	4 : <Limit
	5 : >=Limit or <=-Limit
	6 : <Limit and >-Limit
Limit	Limit which appears in the condition FindHow (optional , Default value: 1e100)
ReplaceHow	How to effect the replacement (optional , Default value: 0)
	0 : linear interpolation: between the last valid value before the Lost Values and the first valid one after them. If on the edge, continuation at constant value.
	1 : constant continuation
	2 : constant next: the next adjacent valid edge value is used
	3 : fixed value which is specified as the subsequent parameter ReplacementValue
Substitute value	The replacement value, if a fixed defined value is intended as the replacement value. Else 0. (optional , Default value: 0)
Result	
Result	Result

Description:

If any Lost Values appear in a data set, the system must react to them for purposes of analysis and display. This is because the Lost Values contain important information. Otherwise, the correct values will be missing at these locations.

There are situations in which only a few Lost Values appear and the data set is to be evaluated and displayed without much concern for the Lost Values. For such purposes, the Lost Values should be replaced with valid measurement readings from the neighboring region. This method can be acceptible if the signal changes only slowly. It is accomplished by determining an estimate of the signal's plot. The result thus comprises both correct values and some estimated ones.

Another method of handling Lost Values makes use of the function LostValueGaps(). Which method is appropriate depends on the application.

Lost Value

The Lost Value refers to a missing or invalid or "not actually available" value in a data set. For instance, a reading may go missing due to overflow of a buffer memory, or due to faulty transmission via a bus-system. The correct reading may not be possible to recover due to breakage of a sensor (ruptured thermocouple) or to overmodulation of a measurement amplifier. If an equidistantly sampled measurement channel needs a value, but none is available, a Lost Value is used instead.

With Real number formats, a Lost Value is frequently represented as 1e100; or as Not a Number, also represented as 1e100. With Integer number formats, usually a value at the end of the respective number range is selected. Thus for instance -32768 for 16-bit signed Integers.

In overmodulation of measurement amplifiers, readings are usually generated which lie (slightly) outside of the measurement range. They have typically been attenuated either by means of hardware or firmware.

In case of sensor (thermocouple) breakage, many measurement devices write a replacement value of approx. -2000 to the temperature channel. Thus, checking for values <=-1000 would detect these sensor breakage values.

Not a Number (NaN) is the binary code which the computer uses for the Real number data type when the value provided is either not a number or no numerical value is available for it. In imc FAMOS, the value is usually represented as 1e100. Not a Number can be used to represent a Lost Value. Not a Number can also result from arithmetical operations when the permitted numerical range has been violated.

Parameter

The input data can be equidistant or expressed in XY format. Segments and events are possible.

The FindHow condition and the comparison value Limit specify which values are to be replaced.

The ReplaceHow option and the replacement value determine how the replacement is achieved.

When ReplaceHow = constant, when there is a gap before the first value, the first value is extended to the front.

For XY input data, only constant continuation and a fixed value are possible.

For a replacement value which is not used, 0 must be specified.

Resumption of values and/or interpolation are only performed within an events or segment, never across the spaces in between.

Examples:

A plot of temperature sampled at 10Hz which returns a value of -2000 in case of sensor breakage is corrected for purposes of subsequent analysis: The sensor breakage values are replaced by means of linear interpolation of the surrounding valid values.

```
T = LostValueReplace(T_orig, 4, -1000, 0)
```

Import of an ASCII-file generates the rotation speed channel Rev, which contains NaN (Not a Number) values. These compromise the appearance of the channel's display. Since the channel contains no negative values, -1 is used as the replacement value.

```
Rev_ = LostValueReplace(Rev, 0, 0, 3, -1)
```

There is a measurement channel which is transmitted via CAN-Bus which indicates the gear of an automatic transmission. The firmware of the logging device reads a -3 if the gear could not be determined, and -4 if the transmission via CAN-Bus was faulty. The analysis always requires a valid value for the gear. Fault conditions are rare. The replacement value to be used is the valid value which is next adjacent to the time of the outage.

```
Gear = LostValueReplace(Gear_CAN, 2, -3, 2)
```

See also:

RangeSet, Set, LostValueGaps, LostValueFill

LoudnessLevel

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Either the loudness or the loudness level, as desired, is calculated for a specified spectrum of one-third octave bands as per DIN 45631/A1:2010-03 or ISO 532-1:2017. The method introduced by E. Zwicker is used.

Declaration:

```
LoudnessLevel ( ThirdSpectrumDB, SoundField, Unit ) -> Result
```

Parameter:

ThirdSpectrumDB	Input data set. A spectrum of one-third octaves, scaled in dB. Value range -60..+120dB. Any values beyond these range limits are reset to the limit values. The one-third octaves bands 25Hz through 12.5kHz should be included. Any not included are set to the minimum range value. The data set is scaled in the x-direction in one-third octaves, therefore: x=14 corresponds to 25Hz, x=15 corresponds to 31.5Hz, ... x=41 corresponds to 12.5kHz. If the data set specified is segmented, a sequence of values (a data set) is returned.
SoundField	SoundField
	2 : free-field, recorded outdoors
	3 : diffuse, recording in a (small) room or vehicle interior
Unit	Unit of result
	0 : phon (the loudness level LN is to be determined)
	1 : sone (the loudness N is to be determined)
Result	
Result	The result is expressed in one of the following units, depending on the sound field and type: "phonGF", "phonGD", "soneGF", "soneGD".

Description:

The algorithm handles only stationary noise, but not time variant noise.

These parameters are no longer to be used; they are only compatible with imc FAMOS 7.2 and below.

SoundField=0: even, frontal

SoundField=1: diffuse

The main difference is minor rounding suggested by DIN 45631:1991 in Appendix A, which no longer appears in new standards.

Examples:

The loudness or loudness level is to be determined from a spectrum of one-third octaves ThirdsDB.

```
phonGF = LoudnessLevel(ThirdsDB,2,0) ; phon, free field
soneGF = LoudnessLevel(ThirdsDB,2,1) ; sone, free field
phonGD = LoudnessLevel(ThirdOct,3,0) ; phon, diffuse
soneGD = LoudnessLevel(ThirdOct,3,1) ; sone, diffuse
```

The loudness of a sonic pressure waveform (a single value derived from the entire measurement) is to be determined.

Sample data set: a single tone of 1kHz, 70dB. These command lines can be used if no other sample data is available:

```
Microphone= 0.08944*sin ( ramp(0,1/40000, 50000)*PI2 * 1000 )
yUnit Microphone Pa
xUnit Microphone s
```

A one-third octave analysis

```
OctI(0, 0, 0, -2, 0, 0, 0)
ThirdsTotal = OctA(Microphone,25,12500)
```

The transient is intended to be disregarded. But the function OctA takes the RMS of the whole waveform including the transient. If the transient doesn't matter, Thirds = ThirdsTotal can be performed instead.

```
Settle = 0.5 's' / xdel?(Microphone) ; Assumption: 0.5s Duration of one-third octave filter transient
Length = leng? ( Microphone)
IF Length > Settle + 0.1
   ThirdsSettle = OctA(leng( Microphone, Settle ),25,12500)
   Thirds = SQRT ( ( Length * quad ( ThirdsTotal ) - Settle * quad ( ThirdsSettle ) ) / ( length - Settle ) )
   DEL ThirdsSettle
else
   Thirds = ThirdsTotal
```

```
end
DEL ThirdsTotal
DEL Length
DEL Settle
ThirdsDB = db ( Thirds / 2e-5'Pa' ) ; returns sonic pressure level
```

Calculation of loudness

```
soneGF = LoudnessLevel ( ThirdsDB, 2, 1 )
```

The time behaviour of the loudness is to be derived from the time behaviour of a one-third octave spectrum.

A time-dependent one-third octave analysis. The sonic pressure is recorded in the data set Microphone, scaled in Pa over s.

```
Interval = 0.1's' ; the time interval
OctI(0, 0.125, Interval/xdel?(Microphone), -2, 0, 0, 0)
Thirds = OctA(Microphone,25,12500)
ThirdsDB = db ( Thirds / 2e-5'Pa' )
SetSegLen( ThirdsDB, 28 ) ; segmenting
SetZoff(ThirdsDB,xoff? ( Microphone)) ; defining the z-coordinate
SetZDel(ThirdsDB,Interval)
SetUnit(ThirdsDB,Unit?(Microphone,0),2)
```

Calculation of loudness

```
soneGF = LoudnessLevel ( ThirdsDB, 2, 1)
```

See also:

LoudnessSpectrum, SoundIndex

LoudnessSpectrum

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

A spectrum of one-third octaves is used to plot the specific loudness over Barks as per DIN 45631/A1:2010-03 or ISO 532-1:2017. The specific loudness is designated N' and the abscissa values z are scaled in Barks. The method introduced by E. Zwicker is used.

Declaration:

```
LoudnessSpectrum ( ThirdSpectrumDB, SoundField, Type ) -> Result
```

Parameter:

ThirdSpectrumDB	Input data set. A spectrum of one-third octaves, scaled in dB. Value range -60..+120dB. Any values beyond these range limits are reset to the limit values. The one-third octaves bands 25Hz through 12.5kHz should be included. Any not included are set to the minimum range value. The data set is scaled in the x-direction in one-third octaves, therefore: x=14 corresponds to 25Hz, x=15 corresponds to 31.5Hz, ... x=41 corresponds to 12.5kHz. If the data set specified is segmented, a sequence of values (a data set) is returned.
SoundField	SoundField
	2 : free-field, recorded outdoors
	3 : diffuse, recording in a (small) room or vehicle interior
Type	Type
	0 : Slope loudness, which leads to the specific loudness
	1 : Core loudness. Only in special cases for the analysis of masking effects.
Result	
Result	The results' units are sone/Bark over Barks.

Description:

The result is defined over a range of 0 to 24 Barks with a resolution of 0.1 Bark.

The algorithm handles only stationary noise, but not time variant noise.

These parameters are no longer to be used; they are only compatible with imc FAMOS 7.2 and below.

SoundField=0: even, frontal

SoundField=1: diffuse

The main difference is a shift of the result by 1 value; formerly resulting in 241 values, now in 240 values.

Examples:

Specific loudness from a microphone signal "mic", measured in Pa over time. Recorded outdoors

```
Thd = SpecThirds_1( mic, 25, 12500, 0)
Thd_dB = dB ( Thd / 2e-5)
N = LoudnessSpectrum( Thd_dB, 2, 0 )
```

The core or slope loudness is to be determined from the one-third octave spectrum ThirdsDB:

```
N_FundF = LoudnessSpectrum( ThirdsDB,2,1) ; core loudness, free field
N_ContF = LoudnessSpectrum( ThirdsDB,2,0) ; slope loudness, free field
N_FundD = LoudnessSpectrum( ThirdsDB,3,1) ; core loudness, diffuse
N_ContD = LoudnessSpectrum( ThirdsDB,3,0) ; slope loudness, diffuse
```

A sonic pressure curve is used to derive the slope loudness (1 spectrum for the whole measurement). For sample data and info on eliminating the transient, see the example under LoudnessLevel().

A one-third octave analysis

```
OctI(0, 0, 0, -2, 0, 0, 0)
Thirds = OctA(Microphone,25,12500)
ThirdsDB = db ( Thirds / 2e-5'Pa' ) ; returns sonic pressure level
```

determining the slope loudness

```
N_ContF = LoudnessSpectrum( ThirdsDB,2,0)
```

The curve of a one-third octave spectrum over time is used to find the time-behaviour of the specific loudness over Barks.

A time-dependent one-third octave analysis. The sonic pressure is recorded in the data set Microphone, scaled in Pa over s.

```
Interval = 0.1's' ; the time interval
OctI(0, 0.125, Interval/xdel?(Microphone), -2, 0, 0, 0)
```

```
Thirds = OctA(Microphone,25,12500)
ThirdsDB = db ( Thirds / 2e-5'Pa' )
SetSegLen( ThirdsDB, 28 ) ; segmenting
SetZoff(ThirdsDB,xoff? ( Microphone)) ; defining the z-coordinate
SetZDel(ThirdsDB,Interval)
SetUnit(ThirdsDB,Unit?(Microphone,0),2)
```

determining the slope loudness

```
N_ContF = LoudnessSpectrum( ThirdsDB,2,0)
N_ContF_T = MatrixTranspose( N_ContF )
```

See also:

LoudnessLevel, OctI, SpecThirds, Sharpness

LowerValue

Returns the lower value of the two parameters.

Declaration:

```
LowerValue ( Parameter1, Parameter2 ) -> Result
```

Parameter:

Parameter1	First single value or data set to be compared.
Parameter2	Second single value or data set to be compared.
Result	
Result	The respective lower value of the two parameters.

Description:

The function has two practical applications. When the parameters are a data set/ single value combination, the effect is to set an upper limit on the data set values at the second parameter's value. If both parameters are data sets, the result is the lower envelope curve.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/segments); but the respective counterpart parameter must then either have exactly the same structure (same segment length, event-count and -length) or it must be a single value.

Examples:

The input channel is converted to decibels and limited to an upper limit of 100 dB.

```
Channel_01_dB = LowerValue(db(Channel_01), 100)
```

The lower and upper envelope curves of two data sets are determined.

```
Env_L = LowerValue(Channel_01, Channel_02)
Env_H = UpperValue(Channel_01, Channel_02)
```

See also:

UpperValue, <, RangeSet

MatrixAdd

Available in: Professional Edition and above

Addition and subtraction of two matrices or vectors, as well as their transposes

Declaration:

```
MatrixAdd ( Matrix A, Matrix B [, Calculation] ) -> Result
```

Parameter:

Matrix A	Matrix or Vector A
Matrix B	Matrix or Vector B
Calculation	Do you wish for the matrices to be transposed? (optional , Default value: "A+B")
	"A+B" : A + B
	"AT+B" : A transposed + B
	"A+BT" : A + B transposed
	"AT+BT" : A transposed + B transposed
	"A-B" : A - B
	"AT-B" : A transposed - B
	"A-BT" : A - B transposed
	"AT-BT" : A transposed - B transposed
Result	
Result	Result

Description:

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

The result of the function is generally a matrix and thus a segmented data set. If the result is a column vector, it is a data set without segments. If the result is a single number, it is interpreted as a column vector and thus also has no segments.

Subtraction by means of "A-B" etc. or by addition of negative matrix.

Units and sampling intervals are not taken into account.

In many cases, it is possible to apply the operators "+" or "-" for the calculation.

The addition can only be performed if the count of rows and columns match after any transposition.

Addition and subtraction of numbers for each element of the matrix with the operators "+" and "-", e.g. Matrix+1.

Examples:

Sum and difference of matrices

```
A = ramp(1, 1, 6 ) ; test data
setsegLen( A, 2)
; A:
; 1 3 5
; 2 4 6
B = MatrixInit ( 2, 3, "I" )
; B:
; 1 0 0
; 0 1 0
C_sum = MatrixAdd( A, B, "A+B" )
; C_sum:
; 2 3 5
; 2 5 6
C_diff = MatrixAdd( A, B, "A-B" )
; C_diff:
; 0 3 5
; 2 3 6
C_diff = MatrixAdd( A, -B, "A+B" )
```

```
; C_diff:
; 0 3 5
; 2 3 6
```

See also:

MatrixMult

MatrixChangeDim

Available in: Professional Edition and above

Insertion and deletion of rows and columns

Declaration:

```
MatrixChangeDim ( Matrix, Start, Count, Option ) -> Matrix
```

Parameter:

Matrix	Matrix
Start	As of this starting index, = 1 for the first element
Count	Count
Option	What change to make?
	"insert col" : Insert columns
	"insert row" : Insert rows
	"remove col" : Delete columns
	"remove row" : Delete rows
Matrix	
Matrix	Matrix

Description:

The result is a matrix of altered dimensions such as Matrix A.

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

The result of the function is generally a matrix and thus a segmented data set. If the result is a column vector, it is a data set without segments. If the result is a single number, it is interpreted as a column vector and thus also has no segments.

When inserting, zeroes (0.0) are inserted. However, the data format is preferentially retained. If for example, the 0.0 can not be displayed due to scaled integers, the closest value possible is used.

Insertion as of starting index=0 appends.

The matrix may not be empty. In order to bring a previosly empty matrix to a dimension > 0, MatrixInit() is used.

The result adopts the x- and z-coordinate.

Examples:

Insert 3 rows in front/on top

```
B = MatrixChangeDim ( A, 1, 3, "insert row" )
```

Append 2 rows

```
B = MatrixChangeDim ( A, 0, 2, "insert row" )
```

Delete 1st column

```
B = MatrixChangeDim ( A, 1, 1, "remove col" )
```

See also:

MatrixPart, MatrixInit, SetSegLen

MatrixCut

A straight cut is made through a matrix. The line of the cut is determined by 2 points.

Declaration:

```
MatrixCut ( Matrix, SvX1, SvZ1, SvX2, SvZ2, SvCalculation, SvScaleAxis, SvIncrement ) -> Result
```

Parameter:

Matrix	A section is made through this matrix.
SvX1	x-coordinate of the 1st point
SvZ1	z-coordinate of the 1st point
SvX2	x-coordinate of the 2nd point
SvZ2	z-coordinate of the 2nd point
SvCalculation	How are the intermediate values calculated?
	0 : Automatically
	1 : Linear interpolation
	2 : Constant extension
	3 : Nearest value
SvScaleAxis	Use x- or z-coordinate of matrix to label results?
	0 : Automatically
	1 : x-coordinates
	2 : Z-coordinates
SvIncrement	Increment (delta-X, dx) of result. Can be set automatically or established permanently
	0 : Automatically
	>0 : Permanent setting
Result	
Result	Section line

Description:

The matrix can be a simple matrix (segmented waveform) or a matrix with extended properties (waveform with events, description see below).

Coordinates:
Each segment is a column. The x-coordinates are arrayed along the matrix columns. Thus, the x-coordinates denote rows. The z-coordinates denote columns. In segmented data sets, the z-coordinate is defined by z0 (z-offset) and delta-Z (z-increment).

For a horizontal section, z1=z2.

For a vertical section x1=x2.

For a diagonal section, x1 and x2 are different, and z1, z2 are different.

The combination of input coordinates x1=x2 and z1=z2 is invalid. These coordinates don't specify a line, but rather a point.

Calculation:
"Automatic" and "Linear interpolation" interpolate between the auxiliary points (first between adjacent x-values, then between adjacent z-values). "Automatic" is recommended. The calculation "Constant extension" mimics stair-step behavior (cf stair-step representation in the curve window). In the direction is increasing indices, a value is kept constant until a new one appears. In the calculation "Nearest value" the matrix' value which is located nearest to the desired position is used.

Scaling axis:
For a diagonal section, the result can be scaled in x- or z-coordinates of the matrix (i.e. this is how the result's x-unit is formed). For a horizontal section the scaling can only follow the x-coordinate, for a vertical section only the z-coordinate. In automatic selection, the z-coordinate is favored; only if this isn't possible is the x-coordinate chosen.

Increment:
Automatic selection is recommended. In this case an increment is selected which doesn't cause substantial loss of information. By contrast, in fixed setting mode, points are only generated at whole multiples of the increment. The fixed increment should be selected in such a way as to prevent loss of information and too much data from accumulating. A fixed increment should only be used if the scaling axis is also permanently established.

Result:
The result is an equidistantly sampled data set. Depending on how the line slopes, an appropriate number of auxiliary points are selected, in order to prevent loss of information. If a different amount of auxiliary points is desired, the function Xydt can be used next.

Event-based input data:
If the matrix is a data set with events, observe the following: Each event is a row of the matrix. The matrix z-coordinate must increase strictly monotonously. All events must be equally long and have the same x-offset (x0) and same sampling interval delta-X (dx). With an automatic increment (=0), the result is an XY-data set. Since in events the z-coordinate usually doesn't increase at constant rate, the section line usually isn't sampled equidistantly. However, if a fixed (nonzero) increment is specified, an equidistantly samples data set is returned.

Examples:

An rpm-dependent spectrum "RotationSpectrum" is given. The frequency is plotted over the x-direction, and the rpms along the z-direction. The spectrum around the rpm-vlaue 2000 revs/min is to be extracted.

```
Spectrum2000 = MatrixCut(RotationSpectrum, 0, 2000, 1, 2000, 0, 1, 0)
```

For this purpose, a horizontal section is made. The result is to be scaled in Hz (corresponding to the matrix x-coordinate). The coordinates z1 and z2 don't matter but must be different.

An rpm-dependent spectrum "RotationSpectrum" is given. The frequency is plotted over the x-direction, and the rpms along the z-direction. The course of the 50Hz spectrum line is to be determined dependent of the rpms.

```
SpectrumLine = MatrixCut(Rotationspectrum, 50, 0, 50, 1, 0, 2, 0)
```

For this purpose, a vertical section is made. The result is to be scaled in revs/min (corresponding to the matrix z-coordinate). The coordinates z1 and z2 don't matter but must be different.

An rpm-dependent spectrum "RotationSpectrum" is given. The frequency is plotted over the x-direction, and the rpms along the z-direction. The course of the 3rd order is to be determined independently of the rpms. The 3rd order is the line for which: 3 * rpms [revs/min]= 60 * frequency [Hz]. The result should comprise one value for every 50 revs/min.

```
Order3 = MatrixCut(RotationSpectrum, 0, 0, 3, 60, 0, 2, 50)
```

For this purpose, a diagonal section is made. The first point is the origin. The second point determines the proportionality between the frequency and rpms, in other words the slope of the section line. The result is to be scaled in revs/min (corresponding to the matrix z-coordinate). A fixed increment of 50 revs/min (z-coordinate) is specified.

See also:

MatrixTranspose, MatrixSumLines

MatrixEigen

Available in: Professional Edition and above

Eigenvalues and eigenvectors of a matrix

Declaration:

`MatrixEigen ( Matrix [, Option] ) -> Result`

Parameter:

Matrix	Matrix
Option	Do you wish for eigenvalues and/or eigenvectors to be calculated? (optional , Default value: "val")
	"val" : Eigenvalues
	"vect" : Eigenvectors
	"val+vect" : Eigenvalues and eigenvectors
Result	
Result	Result

Description:

The function calculates the list of eigenvalues.

For a square matrix, the function calculates as many eigenvalues as there are rows. If there are duplicates of an eigenvalue, it appears multiple times in the result.

The eigenvalues and eigenvectors are determined as complex numbers in the form RealPart/ImaginaryPart. With a real number value, the imaginary part is 0.

Eigenvectors are normalized to make 1 their largest element. The eigenvectors are sorted the same way as their corresponding eigenvalues.

Only square matrices are supported.

The maximum supported dimension is 20*20.

The matrix must be real (not complex).

The right eigenvectors are determined: A*x = Eig*x, where A is the matrix, x an eigenvector and Eig the correpsonding eigenvalue.

If the function returns eigenvalues, then these are returned as a column vector.

If the function returns eigenvectors, then they are arrayed in a matrix. Every eigenvector is a column vector of the matrix.

If the function returns both eigenvalues and eigenvectors, then the result is a matrix whose frst column represents the eigenvalues. The other column vectors are the eigenvectors.

If there is an eigenvalue with multiplicity > 1, there can be both like and different eigenvectors for it.

Caution is needed in the interpretation, if due to the numerical effects, an eigenvalue having high multiplicity leads to multiple closely neighboring but different eigenvalues.

It is also necessary to exercise caution concerning the issue of whether an eigenvalue is real or complex. This is because due to numerical effects, a real eigenwert might sometimes receive a (small) complex component.

The function works with a QR-algorithm.

Examples:

Eigenvalues

```
A = leng(0,9) ; test data
setsegLen( A,3)
A[1,1] = 5
A[2,2] = 3
A[3,3] = 4
A[1,3] = 6 ; row 3, column 1
A[1,2] = 1 ; row 2, column 1
A[2,3] = 2 ; row 2, column 1
; A:
; 5 0 0
; 1 3 0
; 6 2 4
Eig = MatrixEigen ( A )
; Eig = [3, 4, 5]
Eig_r = Eig.r
```

Eigenvalues and vectors

```
A = leng(0,9) ; test data
setsegLen( A,3)
A[1,1] = 5
A[2,2] = 3
A[3,3] = 4
A[1,3] = 6 ; row 3, column 1
A[1,2] = 1 ; row 2, column 1
A[2,3] = 2 ; row 2, column 1
; A:
; 5 0 0
; 1 3 0
; 6 2 4
evv = MatrixEigen ( A, "val+vect" )
Eig = evv[1]
; Eig = [3, 4, 5]
ev1 = evv[2]
; ev1 = [ 0, -0.5, 1]
ev2 = evv[3]
; ev2 = [ 0, 0, 1]
ev3 = evv[4]
; ev3 = [ 1/7, 1/14, 1]
```

See also:

MatrixFromLine

Available in: Professional Edition and above

Formation of a matrix by forming a surface from a distribution of points (x,y,z) by interpolation.

Declaration:

```
MatrixFromLine ( MatrixRef, Amplitude, RowX, ColumnZ [, Substitution] ) -> Matrix
```

Parameter:

MatrixRef	Matrix used as reference
Amplitude	Amplitude, height, Y, stated in physical units
RowX	Row, data in X-direction, stated in physical units
ColumnZ	Column, data in Z-direction, stated in physical units
Substitution	Substitute value to which the range outside of the surface formed is set. (optional , Default value: 0)
Matrix	
Matrix	Matrix

Description:

Each triplet of row.column and amplitude is a sampling point of a surface y = y (x, z). The surface is interpolated linearly between sampling points.

The surface is interpreted as a mountain range, where a unique height is assigned to each point (element of the matrix).

If multiple of the original points are located at the same matrix element, the maximum among the amplitude values is given preference. This corresponds to a view of the surface from a bird's eye view.

The surface itself is a convex space.

The surface is generated by connection of adjacent points via lines which constitute the sides of triangles. The 3D display in the curve window proceeds in the exact same way.

In breaking down the surface into triangles, the resolution applied is 1e-5 in relation to the value range of RowX respectively ColumnZ.

The resulting surface tends not to be uniquely defined. This is easy to illustrate by imagining 4 points constituting the corners of a rectangle. Such a surface can be broken down as a pair of triangles, but since there are 2 diagonals, there are 2 different possible triangle pairs.

The triplets are not in a sorted arrangement. The function never interpolates between adjacent values of the data passed (RowX[i], RowX[i+1]), but only between values which are adjacent on the resulting surface.

The resulting continuous surface is sampled exactly at the matrix's sampling points, in other words at x0+i*dx for the row and at z0+k*dz for the column. Intermediate values of the surface are not taken into account. The resolution of the matrix must be set as fine enough to avoid the loss of significant information.

There is minimal rounding at the sampling points: the system rounds up/down in a region of 1e-5 of the sampling point distance (dx or dz) around a sampling point.

The normal data sets for rows, columns and amplitude all have the same length. They are evaluated point-by-point, regardless of their sampling interval.

The result's units are imported from the data sets Amplitude, RowX and ColumnZ.

The result is a matrix of the same dimensions as the reference matrix. The reference matrix determines x0, dx, z0, dz and the amount of rows and columns.

A matrix is a segmented data set. The segments are the columns.

The computational demands increase heavily with the length of the data set passed. A length of 10000 is already to be considered large.

Examples:

Forming a matrix from x, y, z data. Alpha (=Amplitude) is plotted over rotation speed N (in x-direction; range: 0 to 6000 RPM) and torque M (in z-direction, range: 0 to 500 Nm). Alpha, N and M are time-domain data sets with the same time base.

```
Matrix = MatrixInit ( 61, 51, "0", "", 0, 100, "", 0, 10, "")
Matrix = MatrixFromLine ( Matrix, Alpha, N, M, 0)
```

See also:

MatrixSet, Set, SetIndex

MatrixGet

Available in: Professional Edition and above

Linearisation by means of a 2-dimensional family of characteristic curves (map). Gets elements of a matrix.

Declaration:

```
MatrixGet ( Matrix, Row, Column [, Interpolation] [, Scaling] ) -> Result
```

Parameter:

Matrix	Matrix
Row	Row, addressed in the x-direction, in other words the n-th sample within a segment
Column	Column, addressed in the z-direction, in other words the n-th segment
Interpolation	How is interpolation performed? (optional , Default value: 4)
	0 : Constant from left. If the desired position lies between 2 elements, then the left-hand (earlier) one is used. Same effect as rounding down the index.
	1 : Linear: Between the values bordering on the desired position, linear interpolation is performed, and the value of this connecting straight line is read at the position. This is performed in the direction of the row; subsequently with this result in the column direction.
	3 : Constant from right. If the desired position lies between 2 elements, then the right-hand (later) one is used. Same effect as rounding up the index.
	4 : Constant next. If the desired position lies between 2 elements, then the respective closest one is used. Same effect as rounding the index.
Scaling	How are the row and column scaled? (optional , Default value: "index")
	"index" : Index beginning at 1
	"units" : Stated in physical units, so containing x0, dx or z0, dz
Result	
Result	Result

Description:

The result is a data set which contains an element from the matrix for each row-column pair.

The data sets for the rows and columns are of the same length and structure regarding segments and events.

A matrix is a segmented data set. The segments are the columns.

If the specified values for the row or column lie outside of the matrix, the boundary value is returned.

The function is suitable for correction of a data set by means of a 2-dimensional family of characteristic curves (map). In this context, the array of characteristic curves is the matrix, the data set are the rows and the other variable (e.g. temperature, for temperature dependence) is the column.

The result data set adopts the time base of the data set Row. The data set Row may be either equidistant or XY. If XY, the result adopts its time track. Regarding the data set Column, only the Y-values are taken into account, sample-by-sample.

With the constant interpolation types "constant from right" and "constant from left", an exact comparison of the position is performed. With the scaling in physical units in particular, the smalles rounding error of the real numbers can cause tipping over to the next adjacent value. The problems can be avoided by working with integer indices. If the measurement points themselves are addressed instead of demanding intermediate values, then "Constant next" is the proper choice.

In the syntax of the sequences, the element of a matrix can also be requested directly by means of square brackets, where the segment index (=column index) is stated first.

Examples:

Get element

```
A = ramp(0,1,12) ; test data
setsegLen( A,4)
; A:
; 0 4 8
; 1 5 9
; 2 6 10
; 3 7 11
B = MatrixGet ( A, 3, 2 )
; B = 6
alternatively:
```

```
B = A[2,3]
```

Correction of a temperature-dependent pressure value by means of a 2-dimensional family of characteristic curves

```
Pressure = MatrixGet ( Charakteristic, Pressure_Raw, Temperature, 1, "units" )
```

See also:

Chrct, MatrixSet, Value, Value2, ValueIndex, MatrixFromLine

MatrixInfo

Available in: Professional Edition and above

Query information on a matrix

Declaration:

```
MatrixInfo ( Matrix, Info ) -> Result
```

Parameter:

Matrix	Matrix
Info	What is to be queried?
	"rows" : Row count
	"columns" : Column count
	"diag" : Return diagonal (main diagonal) as a column vector
	"trace" : Trace, sum of the elements of a square matrix's main diagonal
Result	
Result	Result

Description:

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

Examples:

Diagonal

```
A=ramp(0,1,9) ; test data
setsegLen( A,3)
; A:
; 0 3 6
; 1 4 7
; 2 5 8
Diag = MatrixInfo ( A, "diag" )
; Diag = [0, 4, 8]
```

Row count, column count

```
A=ramp(0,1,12) ; test data
setsegLen( A,4)
; A:
; 0 4 8
; 1 5 9
; 2 6 10
; 3 7 11
rows = MatrixInfo ( A, "rows" )
; rows = 4
columns = MatrixInfo ( A, "columns" )
; columns = 3
```

See also:

MatrixMerge, SegLen?, Leng?

MatrixInit

Available in: Professional Edition and above

Creates a pre-initialized matrix

Declaration:

```
MatrixInit ( Rows, Columns [, Option] [, y-unit] [, x0] [, dx] [, x-unit] [, z0] [, dz] [, z-unit] ) -> Matrix
```

Parameter:

Rows	Count of rows in the new matrix > 0; in x-direction; length of a segment
Columns	Count of columns in the new matrix > 0; in z-direction; count of segments
Option	How is initialization to be performed? (optional , Default value: "0")
	"0" : All values 0.0
	"I" : Main diagonal 1.0, else 0.0
y-unit	y-unit, unit of the elements (optional , Default value: "")
x0	x0, offset along the row, in x-direction (optional , Default value: 0)
dx	dx, increment along the row, in x-direction, > 0 (optional , Default value: 1)
x-unit	x-unit, unit along the row, in x-direction (optional , Default value: "")
z0	z0, offset along the column, in z-direction (optional , Default value: 0)
dz	dz, increment along the column, in z-direction, > 0 (optional , Default value: 1)
z-unit	z-unit, unit along the column, in z-direction (optional , Default value: "")
Matrix	
Matrix	Matrix

Description:

The result is a matrix of the dimension [rows*columns].

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

The result of the function is generally a matrix and thus a segmented data set. If the result is a column vector, it is a data set without segments. If the result is a single number, it is interpreted as a column vector and thus also has no segments.

Examples:

Identity matrix

```
A = MatrixInit ( 3, 3, "I" )
; A:
; 1 0 0
; 0 1 0
; 0 0 1
```

Null matrix

```
A = MatrixInit ( 3, 4, "0" )
; A:
; 0 0 0 0
; 0 0 0 0
; 0 0 0 0
```

See also:

SetSegLen

MatrixInverse

Available in: Professional Edition and above

Determining the inverse matrix

Declaration:

`MatrixInverse ( Matrix [, Error handling] ) -> Result`

Parameter:

Matrix	Matrix
Error handling	Determines the system response to an error. (optional , Default value: 0)
	0 : Cancel and post error message
	1 : Return empty data set
Result	
Result	Inverse matrix

Description:

The result is a matrix having the same dimensions.

Only a unique solution is returned as a solution. If the solution is either not unique or there is none, the error handling takes effect.

When called with incorrect parameter values or insufficient memory, the system cancels the operation with the usual error message.

Units and sampling intervals are not taken into account.

The matrix must be square.

Examples:

Inversion of a matrix A

```
A = leng(0,9) ; test data
setsegLen( A,3)
A[3,1] = 0.25 ; row 1, column 3
A[1,2] = 1
A[2,3] = 2
; A:
; 0 0 0.25
; 1 0 0
; 0 2 0
C = MatrixInverse ( A )
; C:
; 0 1 0
; 0 0 0.5
; 4 0 0
```

See also:

SolveLinEq

MatrixIpol

Available in: Professional Edition and above

Interpolation of a matrix along rows and columns

Declaration:

```
MatrixIpol ( Matrix, FactorRowX, FactorColumnZ, Interpolation ) -> Result
```

Parameter:

Matrix	Matrix
FactorRowX	Factor in direction of the row (x). Interpolation within a segment
FactorColumnZ	Factor in direction of the column (z). Interpolation between segments by inserting additional segments
Interpolation	How is interpolation performed?
	0 : Constant. Data augmentation in which between 2 adjacent values, the left-hand value (former of the two) is reproduced. Also, replication of last value (extrapolation).
	1 : Linear. Data augmentation in which between 2 adjacent values, extra values are added by linear interpolation (straight connecting line).
	2 : Cubic spline. Data augmentation by imposing a cubic spline on the series of measured values. Interpolated values are derived from the spline.
	3 : Constant from right. Data augmentation in which between 2 adjacent values, the right-hand value (latter of the two) is reproduced. Also, replication of last value (extrapolation).
	4 : Constant next: Data augmentation in which between 2 adjacent values, the respective closer value is reproduced. Also replication of the last value (extrapolation).
	5 : Linear with last stair-step: Data augmentation in which between 2 adjacent values, values are added by linear interpolation (straight connecting line). The last value is replicated (extrapolation).
Result	
Result	Interpolated matrix

Description:

A matrix is a segmented data set. The segments are the columns.

The sampling interval of the data set generated is smaller by the specified factor FactorRowX, dz by the factor FactorColumnZ.

The factors mainly augment the data volume by means of interpolation. With the non-extrapolating interpolations, the data set is enlarged in the pertinent direction but not quite by the factor specified. There are (Factor -1) values missing in that direction.

If the input data set has no segments, it is a column vector and can only be interpolated in the x-direction by means of FactorRowX.

The function can also be applied to data having events but without segments, however this only performs interpolation in the x-direction.

The function can handle RGB-data in constant and linear interpolation.

Examples:

Resolution increase for a family of characteristic curves (map) by a factor of 10

```
Map = MatrixIpol( Map, 10, 10, 2 )
```

See also:

Ipol, Lip

MatrixMerge

Available in: Professional Edition and above

Copy a smaller matrix into a bigger one at a certain position

Declaration:

```
MatrixMerge ( MatrixA, MatrixB, Starting row, Starting column ) -> Matrix
```

Parameter:

MatrixA	Matrix A
MatrixB	Matrix B
Starting row	From this row index on, > 0; in x-direction; index of sample within a segment
Starting column	From this column index on , > 0; in z-direction; index of a segment
Matrix	
Matrix	Matrix

Description:

The result is a matrix of the same dimension as Matrix A.

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

The result of the function is generally a matrix and thus a segmented data set. If the result is a column vector, it is a data set without segments. If the result is a single number, it is interpreted as a column vector and thus also has no segments.

If submatrices are to be joined together to a larger matrix, then it is possible to initially fill the combined matrix with zeroes using MatrixInit(), for example. Subsequently, data are pasted in using multiple calls of MatrixMerge().

The diagonal (main diagonal) of a Matrix A is set if Matrix B is a vector (row- or column vector) and simultaneously StartingRow =-1 and StartingColumn =-1.

The result adopts the x- and z-coordinates of Matrix A.

Examples:

Set 1 row

```
A=ramp(0,1,12) ; test data
setsegLen( A,4)
; A:
; 0 4 8
; 1 5 9
; 2 6 10
; 3 7 11
B = TransposeMatrix([5, 5, 5])
C = MatrixMerge ( A, B, 3, 1 )
; C:
; 0 4 8
; 1 5 9
; 5 5 5
; 3 7 11
```

Set diagonal

```
A=ramp(0,1,9) ; test data
setsegLen( A,3)
; A:
; 0 3 6
; 1 4 7
; 2 5 8
Diag = [2, 2, 1]
C = MatrixMerge ( A, Diag, -1, -1 )
; C:
; 2 3 6
; 1 2 7
; 2 5 1
```

See also:

MatrixPart, ReplIndex, MatrixChangeDim, MatrixInfo

MatrixMult

Available in: Professional Edition and above

Multiplication of two matrices or vectors, as well as their transposes

Declaration:

```
MatrixMult ( Matrix A, Matrix B [, Calculation] ) -> Product
```

Parameter:

Matrix A	Matrix or Vector A
Matrix B	Matrix or Vector B
Calculation	Should the matrices be transposed before multiplication? (optional , Default value: "A*B")
	"A*B" : A * B
	"AT*B" : A transposed * B
	"A*BT" : A * B transposed
	"AT*BT" : A transposed * B transposed
	"cross" : Cross product of two 3-dimensional column vectors
Product	
Product	Product

Description:

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

The result of the function is generally a matrix and thus a segmented data set. If the result is a column vector, it is a data set without segments. If the result is a single number, it is interpreted as a column vector and thus also has no segments.

In a scalar product, a row vector is multiplied by a column vector. For example, if there are two column vectors, the formula "AT*B" is used.

In the dyadic product, a column vector is multiplied by a row vector. For example, if there are two column vectors, the formula "A*BT" is used.

This function is not used when the matrix is multiplied by a (scalar) factor. That is done by the operator "*". A simple factor of the type Single Value is interpreted as a scalar.

Addition and subtraction of matrices with the operators "+" and "-".

Adding and subtracting matrices and their transposes also with MatrixAdd().

In order to divide by a matrix, one multiplies by the inverse matrix.

Units and sampling intervals are not taken into account.

Examples:

Multiply matrix by column vector

```
A = leng(0,9)  ; test data
setsegLen( A,3)
A[3,1] = 4 ; row 1, column 3
A[1,2] = 1
A[2,3] = 2
; A:
; 0 0 4
; 1 0 0
; 0 2 0
b = [3, 2, -3]
; b:
; 3
; 2
; -2
c = MatrixMult( A, b, "A*B" )
; c:
; -12
; 3
; 4
```

See also:

MatrixAdd

MatrixPart

Available in: Professional Edition and above

Form part of a matrix

Declaration:

```
MatrixPart ( Matrix, Starting row, Rows, Starting column, Columns ) -> Submatrix
```

Parameter:

Matrix	Matrix
Starting row	From this row index on, > 0; in x-direction; index of sample within a segment
Rows	Count of rows to be adopted
Starting column	From this column index on , > 0; in z-direction; index of a segment
Columns	Count of columns to be adopted
Submatrix	
Submatrix	Submatrix

Description:

The result is a matrix of the dimension [rows*columns].

The function returns a submatrix. The portion to be exerpted, given by columns and rows, must fit completely within the matrix. For instance, if the matrix has 4 rows, then from the 1st row on, 4 rows may be excerpted, but from the 4th row, only one.

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A row vector is a segmented data set with a segment length of 1.

The result of the function is generally a matrix and thus a segmented data set. If the result is a column vector, it is a data set without segments. If the result is a single number, it is interpreted as a column vector and thus also has no segments.

The result adopts the x- and z-coordinate, but returns correspondingly shifted x- and z-offsets.

Examples:

Getting 1 row, namely the third

```
A=ramp(0,1,12) ; test data
setsegLen( A,4)
; A:
; 0 4 8
; 1 5 9
; 2 6 10
; 3 7 11
B = MatrixPart ( A, 3, 1, 1, 4 )
; B = [2, 6, 10]
```

Submatrix

```
A=ramp(-4,1,16) ; test data
setsegLen( A,4)
; A:
; -4 0 4 8
; -3 1 5 9
; -2 2 6 10
; -1 3 7 11
B = MatrixPart ( A, 2, 2, 3, 2 )
; B:
; 5 9
; 6 10
```

See also:

MatrixMerge, CutIndex, CutDt

MatrixSet

Available in: Professional Edition and above

Set elements of a matrix

Declaration:

```
MatrixSet ( Matrix, Row, Column, New [, Scaling] ) -> Matrix
```

Parameter:

Matrix	Matrix
Row	Row, addressed in the x-direction, in other words the n-th sample within a segment
Column	Column, addressed in the z-direction, in other words the n-th segment
New	New values
Scaling	How are the row and column scaled? (optional , Default value: "index")
	"index" : Index beginning at 1
	"units" : Stated in physical units, so containing x0, dx or z0, dz
Matrix	
Matrix	Matrix

Description:

Each row-column pair addresses one element in the matrix. This element is set to the new value.

The numerical values from the row and column are rounded to the next element. If an index lies outside of the matrix after rounding, no element of the matrix is changed.

The data sets for rows, columns and new values have the same length and structure regarding segments and events.

The result is a matrix of the same dimensions as the matrix passed.

The data sets for row and column may contain events and segments.

A matrix is a segmented data set. The segments are the columns.

In the syntax of the sequences, the element of a matrix can also be set directly by means of square brackets, where the segment index (=column index) is stated first.

Examples:

Set element

```
A = MatrixInit ( 3, 3, "I" )
; A:
; 1 0 0
; 0 1 0
; 0 0 1
B1 = MatrixSet ( A, 3, 2, 5 )
; B:
; 1 0 0
; 0 1 0
; 0 5 1
alternatively:
B2 = A
B2[2,3] = 5
```

See also:

MatrixGet, MatrixFromLine, Set, SetIndex

MatrixSumLines

The function finds the vector of the matrix row or column sums (segmented waveform).

Declaration:

`MatrixSumLines ( Matrix, SvOption ) -> Sum`

Parameter:

Matrix	Matrix (segmented waveform)
SvOption	Options parameter
	0 : The total of all values in a row. The first row consists of the 1st element from every segment.
	1 : The sum of a column's contents, in other words of a segment's values.
Sum	
Sum	Vector with column or row sums

Description:

The segmented waveform submitted is interpreted as a matrix (1 segment corresponds to a column) and the row or column sum is determined, depending on the value of the option parameter.

Examples:

A matrix (segmented waveform) is given as:

```
; 2 1 0
; 4 1 1
; 5 2 0
; with 2,4,5 = Segment 1
; Determining the column sum:
Matrix = [2,4,5,1,1,2,0,1,0]
SetSegLen(Matrix, 3)
columnSum = MatrixSumLines(Matrix, 1)
; The result is a data set with the values (11,4,1), each element being the sum of a matrix column.
rowSume = MatrixSumLines(Matrix, 0)
; which returns ( 3, 6, 7 ).
```

See also:

MatrixTranspose, MatrixCut

MatrixTranspose

A matrix (segmented waveform) is transposed (rows are switched with columns).

Declaration:

```
MatrixTranspose ( Matrix ) -> TransposedMatrix
```

Parameter:

Matrix	Matrix (segmented waveform)
TransposedMatrix	
TransposedMatrix	Resulting transposed matrix

Description:

The segmented waveform submitted is interpreted as a matrix (1 segment corresponds to a matrix column). The function transposes this matrix, i.e. the rows become columns and vice-versa.

Examples:

The function AmpSpectrumRMS() from the imc Spectrum package computes a series of spectra which are recorded as segments of the resulting data set. This result-matrix consists of amplitude values, plotted over the time and frequency. Next, the time and frequency axes are switched.

```
spectra = AmpSpectrumRMS(signal, 1024, 0, 50, 1, 0)
tm = MatrixTranspose(spectra)
```

See also:

MatrixSumLines, MatrixCut, MatrixMult

Max

Finds a data set's maximum value

Declaration:

```
Max ( Data ) -> SvMaximum
```

Parameter:

Data	Data set whose maximum value is to be found [ND],[XY]
SvMaximum	
SvMaximum	Data set maximum value

Description:

The function Max() returns a data set's the largest numerical value (y-value). The maximum value's position (x-coordinate) can be found using the function Pos().

Examples:

The maximum of a sinusoidal data set is its amplitude, also called its peak:

```
Amplitude = Max(sineWave)
```

This determines how often the most frequently used upper case letter occurs in the histogram of an ASCII text. Upper case letters have values ranging from 65 to 90.

```
thePeak = Max(Cut(HistoText, 65, 90))
```

..and the most frequent upper case letter itself:

```
letter = Pos(HistoText, thePeak)
```

See also:

Min, Mean, Pos, PosiEx, SearchLevel, MvMax, Stat

MDIR

Sets folder for loading and saving sequances and user-defined dialogs

Declaration:

```
MDIR Folder
```

Parameter:

Folder	Complete pathname of the desired folder

Description

Instead of the command MDIR, the function SetOption() should be used in newly created sequences.

The folder for loading and saving sequences/dialogs is given a new setting.

Once this command is executed, this folder is used for loading and saving sequences and user-defined dialogs.

This affects the command SEQUENCE, the function Dialog(), as well as the dialogs for manually loading and saving sequences and dialogs.

The folder name may also be expressedd in quotation marks. This is obligatory when the name contains spaces.

This command can also be called without any parameters (so without any specified fodler). Then the folder set under "Options/Folders" is again used as the default.

The folder elected here remains valid until:

- the command MDIR is called again
- the function SetOption("Dir.Sequences",...) is called
- a sequence- or dialog-file is manually loaded from a different folder or saved to a different folder
- a new folder for sequences/dialogs is designated using the dialog "Options"/ "Folders"

Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
SEQUENCE SEQ1
MDIR C:\MACRO
SEQUENCE SEQ2
```

From the currently set folder, the sequence SEQ1is called; next the sequence C:\MACRO\SEQ2.SEQ

```
MDIR "c:\My tests on 12/1/98"
```

The pathname contains spaces and must therefore be written in quotation marks.

See also:

SetOption, SEQUENCE, Dialog, LDIR

Mean

A data set's arithmetic mean

Declaration:

```
Mean ( Data ) -> SvMean
```

Parameter:

Data	Data from which the mean value is to be calculated [ND].
SvMean	
SvMean	Arithmetic mean of the data set

Description:

The arithmetic mean of a data set's numerical values (y-values) is found. This statistic is defined as the sum of all numerical values divided by their count.

Examples:

A pH value is to be determined. Since there is a strongly interfering rectifier near the measurement instrument, the measured variable was measured 100 times. The mean value over all measurements can be considered a substantially improved reading:

```
PH_mean = Mean(PH_allValues)
```

From an acceleration signal, the velocity is obtained through integration. For this purpose, the mean value of the signal must first be subtracted:

```
v = Int(a - Mean(a))
```

See also:

Max, Min, StDev, RMS, MvMean, Stat

MeasChanNames?

Returns the names of all channels which are assigned to a given measurement.

Declaration:

```
MeasChanNames? ( TxMeasurementName, TxFilter [, SvOption] ) -> TxChannelNames
```

Parameter:

TxMeasurementName	Name of the measurement queried
TxFilter	Empty text with which to get all channels. Otherwise, the wildcard characters '?' (meaning any arbitrary character) and '*' (any amount of arbitrary characters) can be used to define a selection filter.
SvOption	Option (optional)
	0 : Group channels are returned in the customary notation 'GroupName:ChannelName'. The group name itself is not contained int he result. (default)
	1 : For each group included, only the group name is returned.
TxChannelNames	
TxChannelNames	Text box with channel names

Description:

The function returns the name of all channels which are assigned to a given measurement. This corresponds to the content of the corresponding measurement node in the Variables list/Measurement view.

If the respective name does not comply with the rules governing variable names in sequences (e.g. it contains special characters or spaces), the name is additionally bracketed in **{...}**.

The concept of a variable's measurement association has previously mainly been used in the Data Source Browser. There, when a measurement value file is loaded, each variable generated is automatically assigned to a measurement name, in order to be able to distinguish among variables of the same name but from different data sources.

The names returned are sorted alphabetically.

Examples:

A Panel contains a curve window, which is configured for displaying the 1st channel of the 1st selected measurement ('Channel #1 @ Measurement #1') and a button 'Show'. The user initially loads all desired measurements manually. At the push of a button, then for each measurement, all channels starting with 'U_' are each displayed for 2 seconds.

Event-sequence 'Button pressed (Show)':

```
measurements = MeasNames?("*")
FOREACH ELEMENT name in measurements
   SelMeasListSetName(1, name, 1)
   channels = MeasChanNames?(name, "U_*")
   FOREACH ELEMENT chan in channels
      SelChanListSetName(1, chan, "", 1)
      SelListControl(0) ; refresh curve windows now
      Sleep(2)
   END
END
```

See also:

MeasNames?, SelMeasListSetName, SelChanListSetName, SetMeasurementName

MeasNames?

Returns the names of all currently loaded measurements.

Declaration:

```
MeasNames? ( TxFilter ) -> TxMeasNames
```

Parameter:

TxFilter	Empty text with which to get all measurements. Otherwise, the wildcard characters '?' (meaning any arbitrary character) and '*' (any amount of arbitrary characters) can be used to define a selection filter.
TxMeasNames	
TxMeasNames	Text box with measurement name

Description:

This function returns the names of all currently loaded measurements. This corresponds to the (unfiltered) content of the Variables list/Measurement view.

The concept of a variable's measurement association has previously mainly been used in the Data Source Browser. There, when a measurement value file is loaded, each variable generated is automatically assigned to a measurement name, in order to be able to distinguish among variables of the same name but from different data sources.

The names returned are sorted alphabetically.

Examples:

A panel contains a curve window which is configured for display of the 1st channel of the 1st selected measurement ('Channel #1 @ Measurement #1') and a button 'Show'. The user initially manually loads all measurements desired and selects a channel name (assigned the number #1) in the Variables list/Measurement View. At the push of a button, all currently available measurements are enumerated and each associated curve plot for the selected channel is displayed for 2 seconds.

Event-sequence 'Button pressed (Show)':

```
measurements = MeasNames?("")
FOREACH ELEMENT name IN measurements
   SelMeasListSetName(1, name)
   Sleep(2)
END
```

See also:

MeasChanNames?, SelMeasListSetName, SetMeasurementName

MeasurementName?

Finds the name of the measurement to which the variable is assigned.

Declaration:

```
MeasurementName? ( Variable ) -> TxMeasName
```

Parameter:

Variable	Variable to be queried
TxMeasName	
TxMeasName	Name of the assigned measurement

Description:

If no measurement is assigned to the variable, an empty string is returned.

The concept of a variable's measurement association has previously mainly been used in the Data Source Browser. There, when a measurement value file is loaded, each variable generated is automatically assigned to a measurement name, in order to be able to distinguish among variables of the same name but from different data sources.

Examples:

The measurements currently being edited contain the channels 'Voltage' and 'Current'. For the currently selected Measurement #1 (Variables List/Measurement View), the power is calculated and assigned a name reflecting the associated measurement. In consequence, the calculated variable is automatically displayed in the Channels list (Variables List/Measurement View).

```
IF SelUseMeasurement(1) = 0
   EXITSEQUENCE
END
TxMeasName = MeasurementName?(Voltage)
Power = Voltage * Current
SetMeasurementName(Power, TxMeasName)
```

See also:

SetMeasurementName

Median

Signal smoothing by means of median-filtering over a specified number of points.

Declaration:

```
Median ( Data, SvFilterWidth ) -> Filtrate
```

Parameter:

Data	Waveform to filter.
SvFilterWidth	Filter width in points
Filtrate	
Filtrate	Smoothed waveform

Description:

Median filtering proceeds according to the following algorithm (example for filter width of 5): For every input value at the location [n], one result value is found. For this purpose, the input values at the positions [n-2], [n-1], [n], [n+1] and [n+2] (total of 5) are ordered by their amplitude. The result value is the respective median element within the ordered list.

The filter width must lie within the range 3 to 99 and be uneven; sensible values would be 3 or 5, for example. If an even number is specified as the filter width, it is rounded up to the next uneven number.

Median-filtering can be used to suppress short, distinct disturbance peaks in the input signal. The filter width must be selected so as to be more than twice as wide as the width of the disturbances expected.

On the other hand, the signal can be significantly distorted by this function, especially if the signal oscillates strongly. But if the function is applied to mainly monotonic signals or signal segments, the distortion is only minimal.

The input waveform is assumed to extend at a respective constant level from its beginning and end. Thus, at the beginning and end of the waveform, transients subside within half of the filter width; disturbances usually can't be suppressed in these signal regions.

The median-filter is a non-linear filter; the input and output waveforms are not phase shifted.

Examples:

Median filtering over 5 points. Afterwards, the first two and last two values in the result are deleted.

```
filtered = Median(signal, 5)
filtered = CutIndex(filtered, 3, Leng?(filtered)-2)
```

See also:

Smo, MvMean, MvMin, MvMax, Sort

Min

Finds a data set's minimum value.

Declaration:

```
Min ( Data ) -> SvMinimum
```

Parameter:

Data	Data set whose minimum value is to be found [ND],[XY]
SvMinimum	
SvMinimum	Minimum of the data set

Description:

The Min function returns a y-value. The position of the minimum (x-coordinate) can be determined with the Pos function.

Examples:

A data set contains the measurement error of a device at various times: the error smallest in magnitude is to be determined:

```
mini = Min(Abs(error))
```

The supply voltage of a device was measured; the minimum value which occurred is to be determined and expressed as deviation in percent of the nominal value 12V.

```
deviation  = (Min(U_supply) -12'V') * 100'%' / 12'V'
```

See also:

Max, Mean, Pos, PosiEx, SearchLevel, MvMin, Stat

MInt

Moving integral over specified integration interval width

Declaration:

```
MInt ( Data, SvWidth ) -> Result
```

Parameter:

Data	Data set to be integrated; allowed data types: [ND]
SvWidth	Width of the integration interval
Result	
Result	Results of the moving integral

Description:

With moving integration, the integraton interval can have a permanent specified width. This constant integration interval is shifted across the entire data set. The concatenated values of this integral compose the resulting data set. Integration is performed by summing all values of the specified data set within the integration interval and then multiplying by the sampling rate.

The moving integral is computed as follows:

$$\mathrm{GInt}\,(t) = \int_{t-T}^{t} f(x)\,dx \qquad , t \geq T + x_0$$

Here T is the integration interval, f(x) is the function to be integrated, mInt(t) is the moving integral, and x0 is the x-offset of the function to be integrated. The first value of the moving integral is calculated for t = T + x0, i.e. the resulting data set starts somewhat later than the original data set, as illustrated by an xoffset increased by T.

Besides the multiplication with the sampling time involved, the moving integral works as a filter. It smoothes the function, just like a low-pass filter. The weighting function is rectangular. The increase of the x-offset suppresses start-up effects by not computing the first integral.

- The y-unit of the created data set is the product of the x- and y-units of the specified data set.
- The width of the integration interval must be less than the length of the data set to be integrated, so that the created data set has a length of at least 2. Otherwise, the Int() function should be used instead.

Examples:

```
NDp_real = MInt(NDp, 0.020 's') / 0.020 's'
```

A data set representing the instantaneous power p of a load is used to calculate the power consumption. The period of the power source is 20 ms. Use the Value function to obtain the value of the power consumption at a particular time.

See also:

Int, Sum, MvSum, Smo, Hyst

Mirror

Mirroring of a data set's values

Declaration:

```
Mirror ( Data ) -> Mirrored
```

Parameter:

Data	Data set to be mirrored; allowed types: [ND],[XY].
Mirrored	
Mirrored	Mirrored data set

Description:

The specified data set is mirrored. The mirror axis is located at the center of the data set. The reflected data set is generated as follows: the first value of the data set is exchanged with the last value, the second with the second-to-last, etc.

With XY-data, both the X- and the Y-values are mirrored, which means that only the order of the XYpairs in the data set changes, while the graphical display remains unchanged.

For "graphical" mirroring of an XY data set, you can use the following formula:

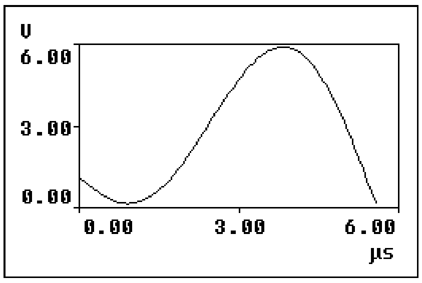
```
Source_mirror = XYof(min(Source.X)+ max(Source.X)-Source.X, Source.Y)
```

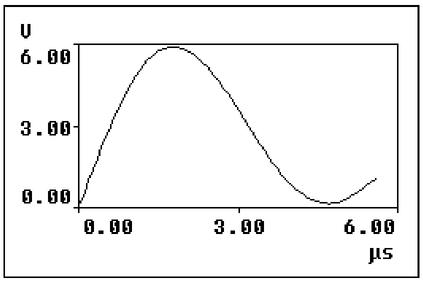
When the Mirror() function is applied to a mirrored data set, the original data set results.

Examples:

```
NDmirror = Mirror(NDdata)
```

The data set NwData displayed at the left is to be mirrored. The result of mirroring is illustrated in the graph on the right as the data set NwMirror:





See also:

Sort

Mod

Modulo (division remainder)

Declaration:

```
Mod ( Numerator, Denominator ) -> Result
```

Parameter:

Numerator	Numerator. Allowed types: [ND],[XY].
Denominator	Denominator. Allowed types: [ND].
Result	
Result	Division remainder (modulo) of numerator and denominator

Description:

The function Mod (modulus) determines the remainder of division. The numerator need not be an integer. The function works as follows:

The numerator and denominator values are made positive; then the denominator is subtracted from the numerator as many times as possible without producing a negative value. The sign of the result is then set to the sign of the numerator.

The modulus function is applied to every point in data sets. If both parameters of the Mod function are equidistant data sets, the elements are linked value for value, regardless of the corresponding xcoordinates. If the parameters are different data types, the single value is applied to each point of the normal data set.

Both parameters may be structured (events/ segments), however, in that case, the respective other parameter must have exactly the same structure (same segment length, event-count and -length), or be a single value.

Remarks

- The unit of the generated data is the quotient of the units of the first and second parameters.
- As in division, the second parameter may not be zero; a warning message is generated if a zero is specified.
- If the specified data sets have different lengths, the length of the result corresponds to that of the shorter data set.

```
SvRemainder = Mod(SvNumerator, SvDenominator)
```

Numerator	Denominator	Mod
17	2	1
18	2	0
17	-2	1
-17	2	-1
-17	-2	-1
8.5	3.5	1.5
7.5	3.5	0.5
7.0	3.5	0

Examples:

The phase of a 10th-order linear has the values -900° ... 0°. Since phasen between -360° and 0° are easier to imaging, but a phase is not changed by adding multiples of 360°, application of the function Mod() is useful:

```
NDphase360 = Mod(NDphase, 360)
```

See also:

/(Division), PhaseMod, Floor

Monoflop

Available in: Professional Edition and above

Monoflop

Declaration:

```
Monoflop ( Data, Impulse duration [, Type] [, Start] [, Direction] [, Logic] ) -> Result
```

Parameter:

Data	input data
Impulse duration	Impulse duration in x-units, rounded to integer multiples of the sampling interval
Type	Retriggerable? (optional , Default value: "no retrig")
	"no retrig" : Not retriggerable: Rising edges are only evaluated again after the end of an impulse.
	"retrig" : Retriggerable: Rising edges during the impulse generated prolong it.
	"1 retrig" : retriggerable by Value 1: Once a pule is generated, every 1 in the input signal prolongs the generated impulse.
Start	Interpretation of Start value =1? (optional , Default value: "const")
	"const" : no signal edge. Causes the values before the 1st measuremnet value to all appear to be 1.
	"0-1" : Transition from 0 to 1: Causes the values before the 1st measurement to all appear to be zero.
Direction	Direction, forwards or backwards (optional , Default value: "")
	"" : Regular: The signal is run in the regular direction forwards, meaning from front to back.
	"reverse" : Backwards: The signal is run in the reverse direction, meaning from back to front.
Logic	Logic, regular or inverted (optional , Default value: "")
	"" : Regular: Pulses consist of 1, empty spaces of zero.
	"invert" : Inverted: Pulses consist of zero, empty spaces of 1. The input is interpreted in inverted manner; the result is its inversion.
Result	
Result	Result

Description:

For a rising signal edge (transition from zero to non-zero), the Monoflop returns an impulse of defined duration (Value 1 during the impulse, else zero).

If the input data are zero, it is interpreted as logically zero (false, low). Else as logically 1 (true, high).

The Monoflop implements a dwell time.

In a simple typical application, the function is called with 2 parameters.

The input data can have events and segments.

When running through in reversed direction (direction = reverse), the signal end becomes the start. Pulses are not prolonged toward the end, but toward the beginning. The behavior is no longer causal. E.g. for applications in which pulses indicate certain events, and the intention is to delimit a region around these events, which is not always only after it, but sometimes also before it.

Examples:

Standard application: Not retriggerable Monoflop; impulse duration = 3s, sampling interval = 1s

```
MF = Monoflop ( data, 3 )
; in = 0 1 0 0 0
;out = 0 1 1 1 0

; in = 0 1 0 1 0
;out = 0 1 1 1 0

; in = 1 1 0 0 0
;out = 0 0 0 0 0
```

Retriggerable Monoflop; impulse duration = 3s, sampling interval = 1s

```
MF = Monoflop ( data, 3, "retrig" )
; in = 0 1 0 1 0 0 0 0
```

```
;out = 0 1 1 1 1 1 0 0

; in = 0 1 1 0 0
;out = 0 1 1 1 0

; in = 1 1 0 0 0
;out = 0 0 0 0 0
```

Monoflop retriggerable by State 1, impulse duration = 3s, sampling interval = 1s

```
MF = Monoflop ( data, 3, "1 retrig"  )
; in = 0 1 1 0 0 0 0 0
;out = 0 1 1 1 1 0 0 0

; in = 0 1 1 0 1 1 0 0 0
;out = 0 1 1 1 1 1 1 1 0

; in = 1 1 0 0 0
;out = 0 0 0 0 0
```

Non-retriggerable Monoflop; impulse duration = 3s, sampling interval = 1s. A 1 at the beginning is to be interpreted as a signal edge.

```
MF = Monoflop ( data, 3, "no retrig", "0-1" )
; in = 0 1 0 0 0
;out = 0 1 1 1 0

; in = 0 1 0 1 0
;out = 0 1 1 1 0

; in = 1 1 0 0 0
;out = 1 1 1 0 0
```

Non-retriggerable Monoflop; impulse duration = 3s, sampling interval = 1s. Inverse logic

```
MF = Monoflop ( data, 3, "no retrig", "const", "", "invert" )
; in = 1 0 1 1 1
;out = 1 0 0 0 1

; in = 1 0 1 0 1
;out = 1 0 0 0 1
```

Non-retriggerable Monoflop; impulse duration = 3s, sampling interval = 1s. Backwards

```
MF = Monoflop ( data, 3, "no retrig", "const", "reverse" )
; in = 0 0 1 1 0 0 0 0 1 0 1 0 1 0
;out = 0 1 1 1 0 0 1 1 1 0 1 1 1 0
```

See also:

Flipflop

MSave

Saving of multiple data sets as a multi-column Ascii-file.
This function is obsolete. For the purpose of exporting multi-column ASCII-files, the more convenient and flexible alternative would usually be to define an ASCII-export template and to use the functions FileSave() or FileOpenASCII2().

Declaration:

```
MSave ( SvTask, SvOption, Data, TxFormatOrFile ) -> SvError
```

Parameter:

SvTask	Task
	1 : Prepare new output. Must be performed once at the beginning of each data storing procedures. All other parameters must take the right types, but they are ignored.
	2 : Adds a data set to the file. [SvOption] must take the right type, but is ignored. [Data] is the data set to be added. [TxFormatOrDatei] describes the output format; see description. With this command, specify all the data sets in succession, which are to be saved in the file in the form of columns.
	3 : Subsequently the file is saved. [SvOption] selects various modes. [Data] must take the correct type, but will be ignored. [TxFormatOrFile] specifies the filename.
SvOption	Mode selection for [SvTask] = 3 (Dave file). Else, ignored, and should be set to 0, for example. See description.
Data	Variable to be saved. Data set or single value
TxFormatOrFile	For [SvTask] = 2: Default of the numerical format. For [SvTask] = 3: the filename to be used. Else, ignored and should be set to "", for example.
SvError	
SvError	Error code
	0 : The file has been saved successfully.
	1 : Insufficient working memory available
	2 : Error in creating the file. Please check the filename specified.
	3 : A maximum of 512 columns is allowed.
	4 : Unable to write a file. Available space on the data carrier may not be sufficient.
	5 : Incorrect specification in [TxFormatOrFile] for [SvTask] = 2.

Description:

Meaning of the parameter [TxFormatOrFile] for [SvTask] = 1:

"fV.N"	Floating point: V = decimal places (without sign), N= decimal places (Default = 6); total <= 15
"f0V.N"	like "fV.N", but with preceding zeroes before the decimal point
"eN"	Exponential: N = decimal places <= 15 (Default = 6)
"e0N"	like"eN", but with preceding zeroes in the 3-digit exponent
"xS"	Hexadezimal: S = digit count <= 8
"x0S"	like "xS", but with preceding zeroes
"aN"	automatic Floating Point/Exponential: N = decimal places (Default = 6)
"", " "	works in the same way as the former fixed format "f4.4"

Meaning of the parameter [SvOption] for [SvTask] = 2:

Add these:	
0:	Do not generate column with time
1:	The first column is the time of the first data set; in seconds
2:	like 1, but in milli-seconds
3:	like 1, but in micro-seconds
4:	like 1, but in nano-seconds
5:	The time is outputted in Hrs:Min:Sec; absolute clock time. With single values, the start time is the time the file was created.

6:	The time is outputted in Hrs:Min:Sec; Start at 00:00:00
7:	The time and date is expressed in the country format. With single values, the start time is the time the file was created.
8:	like 5, but start time represent the time of creation
9:	like 7, but start time and -date represent the time of creation
Add these:	
0:	The file will be newly created
10:	The data appended to an existing file. If the file does not exist, it will be created.
Add these:	
0:	The columns are separated by Tab-characters.
100:	The columns are separated by space characters.
900:	The column separation is specified by the system control.
Add these:	
0:	A dot is used as the decimal symbol.
1000:	A comma is used as the decimal symbol. (Excel)
9000:	The decimal symbol is specified by the system control.
Add these:	
0:	The data sets are equidistant.
10000:	The first data set contains X-coordinates for the rest of the columns.
Add these:	(only for formatted time output; Option + 5,6 or 7):
0:	Seconds without decimal places
100000:	Count of seconds' decimal places = 1
..	
900000:	Count of seconds' decimal places = 9

Unless a full filename is specified, the current default folder is used. Upon starting FAMOS, the current default folder is set to the preset (dialog: 'Options'/ 'Folders'). It can be changed using the function SetOption().

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

Saves the data sets a, b, c and d with a time expressed in milli-seconds. The columns are separated by Tab-characters. In the first line of the file, a brief description is presented. A decimal point is used.

The name of the file is "C:\Test.txt".

```
fh = FileOpenASCII("C:\Test.txt", 1)
IF fh > 0
   err = FileLineWrite(fh, "1.Zeit/ms, 2.U/mV, 3.I/mA", 0)
   err = FileClose(fh )
END
MSave(1, 0, 0, "")
MSave(2, 0, a,"f3.2")   ; z.B. " 123.12" oder "12345.12" oder "  1.12"
MSave(2, 0, b,"E3")      ; z.B. " 1.123E+001"
MSave(2, 0, c,"e5")      ; z.B. "-1.12345e+001"
MSave(2, 0, d,"A3")      ; z.B. "1.123E+005" or "-1123.456"
MSave(3, 12, 0, "C:\Test.txt") ;time in ms, append to file
```

See also:

FileOpenASCII2, FileSave, FileLineWrite

Mult

Time-/x-correct multiplication

Declaration:

```
Mult ( Factor1, Factor2, SvOption ) -> Product
```

Parameter:

Factor1	First parameter, factor; allowed types: [ND],[XY].
Factor2	Second parameter, factor; allowed types: [ND],[XY].
SvOption	Option
	0 : The trigger time of the two summands is ignored.
	1 : Time-correct superposition with regard to trigger-time
Product	
Product	Product; result of multiplication [XY]

Description:

Two data sets undergo time-correct or x-correct multiplication, meaning that the y-values for each shared x-value of time are multiplied.

The result is defined only within the x-range which is shared by both data sets. Within this range a resultvalue is determined for every point at which at least one of the data sets possesses a value. If no value exists for the other data set, one is determined by linear interpolation.

The x-tracks of both parameter data sets must be monotonous, i.e. the x-coordinates must increase continuously.

The operator * (multiplication) by contrast performs point-by-point multiplication of data sets' values.

Examples:

Two channels are measured; one between 11:00 and 13:00, and the other between. 12:00 and 14:00.

```
power12_13h = Mult(voltage11_13h, current12_14h, 1)
```

Time-correct multiplication of the two data sets with respect to the trigger time is performed. The result is defined for the time between 12:00 and 13:00 hours.

See also:

*(Multiplikation), Add, Sub, Div, Append

MvMax

Moving maximum with resampling

Declaration:

```
MvMax ( Data, SvWindow, SvReduction [, SvCompatibilty] ) -> MovingMax
```

Parameter:

Data	Data set to which function is applied [NW]
SvWindow	Width of function operation interval (in x-units)
SvReduction	Width of the reduction interval (in x-units)
SvCompatibilty	(optional , Default value: 0)
	0 : The maximum ratio of window width to reduction interval is limited to 10. The calculation is thus compatible with FAMOS-versions <=7.4.
	1 : The maximum ratio of window width to reduction interval is limited to 1000.
MovingMax	
MovingMax	Moving maximum

Description:

Each calculated value is the absolute maximum over the width [SvWindow].

[SvReduction] specifies the width of the reduction interval. Exactly one result value is calculated for each such interval. The x-coordinate associated with each value is the coordinate where the pertinent interval begins. For this reason, the "steps" representation of the resulting data set in the curve window is recommended.

The averaging interval and reduction interval must be integer multiples of the sampling interval. Otherwise rounding off occurs, since the calculations always apply to a whole set of measurement values.

Furthermore, there must be an integer ratio of 1:1, 2:1... **10**:1 (for SvCompatibility=0) or **1000**:1 (for SvCompatibility=1) between the count of measurement values in [SvWindow] and [SvReduction]. If this is not the case, the system will modify the two intervals accordingly.

If [SvReduction] and [SvWindow] are unequal, initially not enough values are available; in that case the respectively available input values are analyzed.

Examples:

Every 10 s, the maximum over the last 10 s is calculated:

```
resultMax = MvMax(timeData, 10, 10)
```

Every 10 s, the maximum over the last 20 s is calculated:

```
resultMax = MvMax(timeData, 20, 10)
```

For every 5th point, the maximum over the last 10 points is calculated:

```
_xdelta = xdel?(timeData)
resultMax = MvMax(timeData, 10*_xdelta, 5*_xdelta)
```

See also:

Max, MvMin, MvMean, MvRMS

MvMean

Moving mean value with resampling

Declaration:

```
MvMean ( Data, SvWindow, SvReduction [, SvCompatibilty] ) -> MovingMean
```

Parameter:

Data	Data set to which function is applied [NW]
SvWindow	Width of function operation interval (in x-units)
SvReduction	Width of the reduction interval (in x-units)
SvCompatibilty	(optional , Default value: 0)
	0 : The maximum ratio of window width to reduction interval is limited to 10. The calculation is thus compatible with FAMOS-versions <=7.4.
	1 : The maximum ratio of window width to reduction interval is limited to 1000.
MovingMean	
MovingMean	Moving mean

Description:

Each calculated value is the arithmetical mean over the width [SvWindow].

[SvReduction] specifies the width of the reduction interval. Exactly one result value is calculated for each such interval. The x-coordinate associated with each value is the coordinate where the pertinent interval begins. For this reason, the "steps" representation of the resulting data set in the curve window is recommended.

The averaging interval and reduction interval must be integer multiples of the sampling interval. Otherwise rounding off occurs, since the calculations always apply to a whole set of measurement values.

Furthermore, there must be an integer ratio of 1:1, 2:1... **10**:1 (for SvCompatibility=0) or **1000**:1 (for SvCompatibility=1) between the count of measurement values in [SvWindow] and [SvReduction]. If this is not the case, the system will modify the two intervals accordingly.

If [SvReduction] and [SvWindow] are unequal, initially not enough values are available; in that case the respectively available input values are analyzed.

Examples:

Every 10 s, the mean value over the last 10 s is calculated:

```
resultMean = MvMean(timeData, 10, 10)
```

Every 10 s, the mean valud over the last 20 s is calculated:

```
resultMean = MvMean(timeData, 20, 10)
```

For every 5th point, the mean valud of the last 10 points is calculated:

```
_xdelta = xdel?(timeData)
resultMean = MvMean(timeData, 10*_xdelta, 5*_xdelta)
```

See also:

Mean, MvMin, MvMax, MvRMS, Median

MvMin

Moving minimum with resampling

Declaration:

```
MvMin ( Data, SvWindow, SvReduction [, SvCompatibilty] ) -> MovingMin
```

Parameter:

Data	Data set to which function is applied [NW]
SvWindow	Width of function operation interval (in x-units)
SvReduction	Width of the reduction interval (in x-units)
SvCompatibilty	(optional , Default value: 0)
	0 : The maximum ratio of window width to reduction interval is limited to 10. The calculation is thus compatible with FAMOS-versions <=7.4.
	1 : The maximum ratio of window width to reduction interval is limited to 1000.
MovingMin	
MovingMin	Moving minimum

Description:

Each calculated value is the absolute minimum over the width [SvWindow].

[SvReduction] specifies the width of the reduction interval. Exactly one result value is calculated for each such interval. The x-coordinate associated with each value is the coordinate where the pertinent interval begins. For this reason, the "steps" representation of the resulting data set in the curve window is recommended.

The averaging interval and reduction interval must be integer multiples of the sampling interval. Otherwise rounding off occurs, since the calculations always apply to a whole set of measurement values.

Furthermore, there must be an integer ratio of 1:1, 2:1... **10**:1 (for SvCompatibility=0) or **1000**:1 (for SvCompatibility=1) between the count of measurement values in [SvWindow] and [SvReduction]. If this is not the case, the system will modify the two intervals accordingly.

If [SvReduction] and [SvWindow] are unequal, initially not enough values are available; in that case the respectively available input values are analyzed.

Examples:

Every 10 s, the minimum over the last 10 s is calculated:

```
resultMin = MvMin(timeData, 10, 10)
```

Every 10 s, the minimum over the last 20 s is calculated:

```
resultMin = MvMin(timeData, 20, 10)
```

For every 5th point, the minimum over the last 10 points is calculated:

```
_xdelta = xdel?(timeData)
resultMin = MvMin(timeData, 10*_xdelta, 5*_xdelta)
```

See also:

Min, MvMax, MvMean, MvRMS

MvRMS

Moving RMS-value with equally weighted averaging

Declaration:

```
MvRMS ( Data, SvWindow, SvReduction [, SvCompatibilty] ) -> MovingRMS
```

Parameter:

Data	Data set to which function is applied [NW]
SvWindow	Width of function operation interval (in x-units)
SvReduction	Width of the reduction interval (in x-units)
SvCompatibilty	(optional , Default value: 0)
	0 : The maximum ratio of window width to reduction interval is limited to 10. The calculation is thus compatible with FAMOS-versions <=7.4.
	1 : The maximum ratio of window width to reduction interval is limited to 1000.
MovingRMS	
MovingRMS	Moving RMS-value

Description:

The moving RMS-value withequally weighted averaging and subsequent resampling is calculated.

Each calculated value is the root-mean-square over the width [SvWindow].

[SvReduction] specifies the width of the reduction interval. Exactly one result value is calculated for each such interval. The x-coordinate associated with each value is the coordinate where the pertinent interval begins. For this reason, the "steps" representation of the resulting data set in the curve window is recommended.

The averaging interval and reduction interval must be integer multiples of the sampling interval. Otherwise rounding off occurs, since the calculations always apply to a whole set of measurement values.

Furthermore, there must be an integer ratio of 1:1, 2:1... **10**:1 (for SvCompatibility=0) or **1000**:1 (for SvCompatibility=1) between the count of measurement values in [SvWindow] and [SvReduction]. If this is not the case, the system will modify the two intervals accordingly.

If [SvReduction] and [SvWindow] are unequal, initially not enough values are available; in that case the respectively available input values are analyzed.

Examples:

Every 10 s, the RMS-value over the last 10 s is calculated:

```
resultRMS = MvRMS(timeData, 10, 10)
```

Every 10 s, the RMS-value over the last 20 s is calculated:

```
resultRMS = MvRMS(timeData, 20, 10)
```

For every 5th point, the RMS over the last 10 points is calculated:

```
_xdelta = xdel?(timeData)
resultRMS = MvRMS(timeData, 10*_xdelta, 5*_xdelta)
```

See also:

RMS, ExpoRMS, MvMin, MvMax, MvStDev

MvStDev

Moving standard deviation with resampling

Declaration:

```
MvStDev ( Data, SvWindow, SvReduction [, SvCompatibilty] ) -> MovingStDev
```

Parameter:

Data	Data set to which function is applied [NW]
SvWindow	Width of the calculation interval (in x-units)
SvReduction	Width of the reduction interval (in x-units)
SvCompatibilty	(optional , Default value: 0)
	0 : The maximum ratio of window width to reduction interval is limited to 10. The calculation is thus compatible with FAMOS-versions <=7.4.
	1 : The maximum ratio of window width to reduction interval is limited to 1000.
MovingStDev	
MovingStDev	Moving standard deviation

Description:

Each calculated value is the standard deviation over the width [SvWindow].

[SvReduction] specifies the width of the reduction interval. Exactly one result value is calculated for each such interval. The x-coordinate associated with each value is the coordinate where the pertinent interval begins. For this reason, the "steps" representation of the resulting data set in the curve window is recommended.

The calculation interval and reduction interval must be integer multiples of the sampling interval. Otherwise rounding off occurs, since the calculations must always apply to a whole set of measurement values.

Furthermore, there must be an integer ratio of 1:1, 2:1... **10**:1 (for SvCompatibility=0) or **1000**:1 (for SvCompatibility=1) between the count of measurement values in [SvWindow] and [SvReduction]. If this is not the case, the system will modify the two intervals accordingly.

If [SvReduction] and [SvWindow] are unequal, initially not enough values are available; in that case the respectively available input values are analyzed.

Examples:

Every 10 s, the standard deviation over the last 10 s is calculated:

```
resultStDev = MvStDev(timeData, 10, 10)
```

Every 10 s, the standard deviation over the last 20 s is calculated:

```
resultStDev = MvStDev(timeData, 20, 10)
```

For every 5th point, the standard deviation over the last 10 points is calculated:

```
_xdelta = xdel?(timeData)
resultStDev = MvStDev(timeData, 10*_xdelta, 5*_xdelta)
```

See also:

StDev, MvMean, MvRMS, Stat

MvSum

Moving additive totaling with resampling

Declaration:

```
MvSum ( Data, SvWindow, SvReduction [, SvCompatibilty] ) -> MovingSum
```

Parameter:

Data	Data set to be totalled [ND]
SvWindow	Width of the calculation interval (in x-units)
SvReduction	Width of the reduction interval (in x-units)
SvCompatibilty	(optional , Default value: 0)
	0 : The maximum ratio of window width to reduction interval is limited to 10. The calculation is thus compatible with FAMOS-versions <=7.4.
	1 : The maximum ratio of window width to reduction interval is limited to 1000.
MovingSum	
MovingSum	Moving sum

Description:

Each calculated value is the sum over the width [SvWindow].

[SvReduction] specifies the width of the reduction interval. Exactly one result value is calculated for each such interval. The x-coordinate associated with each value is the coordinate where the pertinent interval begins. For this reason, the "steps" representation of the resulting data set in the curve window is recommended.

The calculation interval and reduction interval must be integer multiples of the sampling interval. Otherwise rounding off occurs, since the calculations must always apply to a whole set of measurement values.

Furthermore, there must be an integer ratio of 1:1, 2:1... **10**:1 (for SvCompatibility=0) or **1000**:1 (for SvCompatibility=1) between the count of measurement values in [SvWindow] and [SvReduction]. If this is not the case, the system will modify the two intervals accordingly.

If [SvReduction] and [SvWindow] are unequal, initially not enough values are available; in that case the respectively available input values are analyzed.

Examples:

Every 10 s, the sum over the last 10 s is calculated:

```
resultSum = MvSum(timeData, 10, 10)
```

Every 10 s, the sum over the last 20 s is calculated:

```
resultSum = MvSum(timeData, 20, 10)
```

For every 5th point, the sum over the last 10 points is calculated:

```
_xdelta = xdel?(timeData)
resultSum = MvSum(timeData, 10*_xdelta, 5*_xdelta)
```

See also:

Sum, MvRMS, Int, MInt

Name?

Gets a variable's name

Declaration:

```
Name? ( Variable [, Structure] [, Format] ) -> TxName
```

Parameter:

Variable	Variable whose name is to be found. The data type is arbitrary.
Structure	Structure of the name found (optional , Default value: 0)
	0 : Complete identifier, with group name and measurement if applicable
	1 : Identifier with group name (if applicable), without measurement
	2 : Name without group name, without measurement
Format	Formatting of the name found (optional , Default value: 0)
	0 : Standard
	1 : Advanced syntax: To make it possible for a variable name to be specified in formulas, it must obey certain rules (e.g. no spaces, first character not a digit, etc.). If this condition is not met, the name must additionally be bracketed in curly brackets {...}. With this option, the name returned is supplemented with these curly brackets, if necessary.
TxName	
TxName	Name

Description:

Gets the name of the variable passed. Typical application cases for this function are:

- Sequence functions with parameters passed by means of a reference: Here, the local parameter is not a true variable but instead an alias for the variable passed when the function is called. Applying the function to such a reference parameter then returns the name of the referenced variable.
- Classic sequences: On the basis of the placeholder-name (e.g. PA1), it is possible to get the name of the variable which was passed when the sequence was called.
- Data groups: A channel name is determined from the channel index. This is a more powerful alternative to the function GrChanName?().

The first parameter must be a variable specified directly, without additional indexing (such as component-, event-index). With groups, specifying the channel index is allowed.

Examples:

The following sequence function displays the data set passed to it in a free-floating curve window. The data set's name is displayed in the header.

```
!ShowCurve(par)
   ; par: [ND, Reference]
   CwNewWindow(par, "show")
   CwNewChannel("append new axis", par)
   CwDisplaySet("header.count", 1)
   CwDisplaySet("header.text", Name?(par))
```

In a data group [input], all channels are enumerated. All channels whose maximum is greater than 2 are copied to a new data group [output].

```
limit = 2
count = GrChanNum?(input)
FOR i = 1 TO count
   IF max(input:[i]) > limit
      name = Name?(input:[i], 2, 1)
      output:<name> = input:[i]
   END
END
```

See also:

Comm?, MeasurementName?, GrChanName?, VarGetName?

NorthCorrection

Available in: Professional Edition and above

Correction of angle reading within a window, in order to allow sensible averaging of compass readings or angles/phases.

Declaration:

```
NorthCorrection ( Data, Window ) -> Result
```

Parameter:

Data	input data
Window	Width of the averaging interval expressed in x-units; rounded to whole multiples of the sampling interval
Result	
Result	Result

Description:

The function NorthCorrection() perfoms angle reading correction according to the addition method.

It prevents a discontinuity at due North, i.e. a jump from 360° to 0°, within the averaging interval. For values fluctuating around 360°, and which thus can sometimes lie near 0°, an incorrect mean value of 180° is thus avoided.

The result of averaging may lie beyond the range of compass readings 0°...360° , e.g. 365°. The function PhaseMod() converts the result back to the compass range (0°...360°), e.g. from 365° to 5°.

A condition for this procedure is that the wind direction does not change by more than 180 degrees within the averaging interval.

Any appropriate usage of NorthCorrection() will only be in conjunction with averaging or similar functions, and subsequent use of PhaseMod().

The input data are expressed in the unit Degrees and are typically in the value range -360 to 360 degrees or even somewhat higher.

The function adds/subtracts 360 degrees at the locations necessary.

The input data are equidistant.

Examples:

Calculation of the moving wind direction averaged over 10s.

```
NC = NorthCorrection( Channel, 10 )
NC_Mean = MvMean( NC, 10, 10 )
Wind = PhaseMod( NC_Mean )
```

See also:

PhaseMod, PhaseContinuous

NOT

Logical inversion

Declaration:

```
NOT ( Operand ) -> ZeroOrOne
```

Parameter:

Operand	Single value or data set to be inverted.
ZeroOrOne	
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description:

Logical inversion of a number. The result is 1 if the operand is 0. Otherwise the result is 0.

The function can be applied to single values and data sets. With data sets, the function is carried out data point by data point.

Examples:

The following two expressions are equivalent:

```
NOT(A) AND NOT(B)
NOT(A OR B)
```

Inversion of a digital data set

```
Result = NOT(DigChannel1)
```

See also:

AND, OR, XOR

OctA

1/3-octave analysis

Declaration:

```
OctA ( Data, SvLowerFrequency, SvUpperFrequency ) -> OctavData
```

Parameter:

Data	The data set to be subjected to 1/3-octave analysis
SvLowerFrequency	The lower endo of the frequency range
SvUpperFrequency	The upper end of the frequency range
OctavData	
OctavData	Results of filtering

Description:

This function performs the common frequency-band analyses: 1/3-octave (third bands), 1/12 octave (semitone bands) and 1/24 octave (quartertone bands). Frequency rating curves are also provided. All filtering is executed using digital filters - an FFT is not used.

The analysis is performed as follows:

Digital filters are created corresponding to the sampling rate and the desired bandwidth and frequency ranges.

The data set is filtered with these digital filters.

The filtered data are subjected to time-weighting. With time-weighting, the moving RMS-value based on exponential averaging is formed. The time constant can be specified. The moving RMS-value is calculatedd as follows: The signal is squared, then exponentially averaged, then the square root is taken. Squaring, averaging and subsequent root extraction are characteristic of forming the RMS. With a regular RMS value, an evenly weighted average of all squares is calculated, while in this case for the moving RMS, time-weighting is performed. The exponential averaging used causes a sort of "forgetting". It can be thought of as 1st order lowpass filtering.

The time-weighted is now resampled once again. Only every n-th value is processed. If rigorous averaging is used with the time weighting, it is generally not necessary to use every value, since adjacent values are barely different from each other.

It is possible to normalize the results again by multiplying with a reference value. The refernece value is a value corresponding to 0 dB, for example.

Next, frequency-weighting is performed. The weighting options are linear, A, B, C and D.

The function can return three different varieties of results:

The time plot of a frequency band is determined. You obtain a data set plotted over time.

Multiple frequency bands are determined, with averaging over the entire data set. You obtain a data set plotted over frequency.

Multiple frequency bands can be averaged over time. For example, when the frequencies f1, f2 and f3 are averaged over time, the resulting data set has the following form: the values f1, f2 and f3 at the first time, then the values f1, f2, f3 at the second time, etc. The data set is scaled for the first set of frequencies, (f1..f3). The time between frequency samples and the number of frequencies is not contained in the data set. This data type is complicated to work with and thus reserved for special applications! Refer to the section on periodic comparison in the Curve Window manual. This data set can only be displayed meaningfully using the periodic comparison representation. A single period can be extracted using the mathematical functions. The data set can be thought of as a matrix, with each line as a period and all lines listed consecutively.

To convert acceleration data to velocity data or vice versa, the signal can be integrated or differentiated. It is usually convenient to perform these operations at this point, since high frequency noise has already been removed by the band-pass filtering (making differentiation easier) and since low-frequency drifts and offsets have been eliminated (simplifying integration). This is not described in the DIN standards, but we mention it because it is of practical significance. Integration and differentiation are executed by expanding the digital band-pass filter with poles or zeros at the frequency 0.

The time shift of the amplitude in a frequency band is generally a strongly varying signal, which must first be rectified and averaged. The method used here is one for moving RMS calculation. The following time ratings can be selected:

=0	No time weighting
> 0	Time constant is defined by the user; it must be smaller or greater than the sampling rate.
-1 FAST	The time constant is 0.125s.
-2 SLOW	The time constant is 1s.
-3 Impulse	In increasing amplitude, the time constant is 35ms; in decreasing amplitude, the time constant is 1.5s. Impulse shaped signals are recorded quickly and decaying signals slowly.
-4 Peak	Extreme display for very short impulses, with guaranteed display of the peak value. In increasing amplitude, the time constant is zero (can be achieved exactly in a computer, but only approximately in analog circuits). In decreasing amplitude, the time constant is 3s.

Because the sensitivity of the human ear is frequency-dependent, frequency weighting curves are available for "adapting" the measured data.

Below, the available frequncy weighting selections are listed. Linear weighting means no weighting. Frequencies between the ones specified are linearly interpolated, meaning that the dB-value is interpolated linearly. beyond the end of the table, the last weighting is held constant.

Weighting types

Frequenz [Hz]	**A [dB]**	**B [dB]**	**C [dB]**	**D [dB]**
10	-70.4	-38.2	-14.3	-26.5
12.5	-63.4	-33.2	-11.2	-24.5
16	-56.7	-28.5	-8.5	-22.5
20	-50.5	-24.2	-6.2	-20.5
25	-44.7	-20.4	-4.4	-18.5
31.5	-39.4	-17.1	-3.0	-16.5
40	-34.6	-14.2	-2.0	-14.5
50	-30.2	-11.6	-1.3	-12.5
63	-26.2	-9.3	-0.8	-11.0
80	-22.5	-7.4	-0.5	-9.0
100	-19.1	-5.6	-0.3	-7.5
125	-16.1	-4.2	-0.2	-6.0
160	-13.4	-3.0	-0.1	-4.5
200	-10.9	-2.0	0.0	-3.0
250	-8.6	-1.3	0.0	-2.0
315	-6.6	-0.8	0.0	-1.0
400	-4.8	-0.5	0.0	-0.5
500	-3.2	-0.3	0.0	0.0
630	-1.9	-0.1	0.0	0.0
800	-0.8	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0
1250	0.6	0.0	0.0	2.0
1600	1.0	0.0	-0.1	5.5
2000	1.2	-0.1	-0.2	8.0
2500	1.3	-0.2	-0.3	10.0
3150	1.2	-0.4	-0.5	11.0
4000	1.0	-0.7	-0.8	11.0
5000	0.5	-1.2	-1.3	10.0
6300	-0.1	-1.9	-2.0	8.5
8000	-1.1	-2.9	-3.0	6.0
10000	-2.5	-4.3	-4.4	3.0
12500	-4.3	-6.1	-6.2	0.0
16000	-6.6	-8.4	-8.5	-4.0
20000	-9.3	-11.1	-11.2	-7.5

The following nominal pass ranges apply to octave-filters, where the numerical values of the frequencies repeated every 3 decades. Thus the list can be extended forward and backward.

Octaves

Center frequency [Hz]	Lower limit [Hz]	Upper limit [Hz]
16	11.2	22.4
31.5	22.4	45
63	45	90
125	90	180

250	180	355
500	355	710
1000	710	1400
2000	1400	2800
4000	2800	5600
8000	5600	11200
16000	11200	22400

The following frequencies apply to 1/3-octaves, where the numerical values repeat every decade, so that the table can be extended forwards and backwards.

1/3-octaves

Center frequency [Hz]	Lower limit [Hz]	Upper limit [Hz
1000	900	1120
1250	1120	1400
1600	1400	1800
2000	1800	2240
2500	2240	2800
3150	2800	3550
4000	3550	4500
5000	4500	5600
6300	5600	7100
8000	7100	9000
10000	9000	11200

The center frequencies of the 1/12- and 1/24-octave bands are found at the center frequencies of the 1/3-octave and at intermediate values calculated logarithmically at regular intervals. Any additional frequency samples use the edges of the 1/3 octave range.

Example of 1/12-octaves:

1/12 octaves

Center frequency [Hz]	Lower limit [Hz]	Upper limit [Hz]
1000	974	1029
1058	1029	1089
1120	1089	1151
1183	1151	1216
1250	1216	1286
1323	1286	1361
1400	1361	1448
1497	1448	1547
..	..	..

Example of 1/24-octaves:

1/24 octaves

Center frequency [Hz]	Lower limit [Hz]	Upper limit [Hz]
1000	987	1014
1029	1014	1043
1058	1043	1073
1089	1073	1104
1120	1104	1135
1151	1135	1167
1183	1167	1200

1216	1200	1233
1250	1233	1268
1286	1268	1304
1323	1304	1342
..	..	..

When data sets are created in the frequency range, the x-axis (frequency) is scaled in a particular way. The frequency bands are at logarithmic distances, so that it is convenient to lable the x-axis with the logarithm of the frequency. When displaying frequency data sets in curve windows, you can use "1/3-octave labeling" for the x-axis. The logarithm is then expanded and the frequencies are written along the axis in accordance with DIN. The following table shows how the x-scaling of the data is related to the frequency bands, based on the following rule: calculate the log base ten of the center frequency and then round this value.

x-scaling

x-scaling	Center-frequency [Hz]
..	..
-3	0.5
-2	0.63
-1	0.8
0	1
1	1.25
2	1.6
3	2
4	2.5
5	3.15
6	4
7	5
8	6.3
9	8
10	10
11	12.5
..	..
20	100
30	1000
40	10000
41	12500
42	16000
43	20000
..	..

The 1/3-octaves are located at the x-positions 0, 1, 2..., the octaves at the positions 0, 3, 6, 9, 12..., 1/12-octaves at the positions 0, 0.25, 0.5, 0.75, 1, 1.25..., and 1/24-octaves are located at all multiples of 1/8.

This means the various bandwidths are expressed as multiples of a 1/3-octave. Accordingly, for frequency-scaling, the Delta-X of the results data set is set to the following values:

Bandwidth	Delta-X
Octave	3
1/3-octave	1
1/12 octave	0.25
1/24 octave	0.125

The function OctA() itself has only 3 arguments. All other settings are made using the function OctI(). The arguments belonging to the function OctI() are explained here also.

Parameter	Meaning

FrequencyLower	One end of the frequency range of interest. if a value of zero is specified, the value is set to [FrequencyTop]. You always specify the center of a frequency range. The center frequency does not need to be met exactly. But it is important that the frequency be within the pass-through range of the desired band. Then the exact frequency is used automatically. At least one of the two specified frequencies must be above zero. No values below zero may be specified.
FrequencyUpper	Like [FrequencyLower], but the other end of the frequency range
...Integrations	Amount of integrations to perform 0 No integration 1 One integration 2 Double integration -1 First derivative -2 Second derivative For instance, if you have a measurement from an accelerometer but you want to know the travel distance, select double integration.
...TimeWeight	Averaging duration, time weighting > 0 your averaging duration 0 no averaging -1 FAST -2 SLOW -3 Impulse -4 Peak
...Reduction	Reduction factor. The amount of data points in the result is reduced from the source data point count by this factor. A factor of 2 meanss that every 2nd value is adopted. A factor of 1 means that each value is adopted, without any reduction. A reduction factor of 0 indicates that the factor is to be the same as the length of the source data set, thus generating exactly one resulting value. For a reduction factor = 0, an RMS-value for the entire data set is generated. This RMS-value is the true (equally-weighted) RMS-value, as is also calculated by the mathematics function RMS(). The constant specified for the time weighting is ignored.
...Width	Width of frequency bands -1 octave -2 1/3-octave -3 1/12-octave -4 1/24-octave
...FreqWeight	Frequency weighting 0 linear 1 A 2 B 3 C 4 D
...reference	Reference value for normalization; multiplication by this value is performed. For a reference value of zero, no normalization is performed.

The algorithms conform to:

- DIN 45651, Octave filters for electro-acoustical measurements
- DIN 45652, third-octave filters for electro-acoustical measurements
- DIN IEC 651, Sound level meters
- Filtering is performed with 8th order band-pass filters with Butterworth characteristic and meet the requirements of the standards DIN EN 61260 and IEC 1260, Class 0.

Attention, time constant!

Please note that applying calculation of moving RMS, the effect is different from using simple a low-pass. The time constant cannot be read directly from a graph of the filtered data because the root of the signal is computed after multiplication with the time constant. The time constant can however be made visible by squaring the results!

Sampling rates

++++
When the function performs filtering, a digital filter is used. This digital filter's cutoff frequencies may only be less than half of the sampling frequency. Higher cutoff frequencies are not possible. The frequencies of the band edges are relevant, not the center frequncies.

Filtering precision

>
The design of the digital filters used for the bandpass is only good for frequeciessignificantly less than half of the sampling frequency. To genuinely benefit from the band-pass filter, this means that the sampling frequency should be significantly larger than twice the highest frequency to be evaluated.

Moving RMS value

The averaging time can be larger or smaller than the sampling rate. The calculation is performed with a zero-th order approximation (stair-step-shaped signal). With a reduction factor of zero, the equally-weighted true RMS is calculated (see RMS function). This is the best reflection of the entire data set's spectrum.

Initial transient

When applying band-pass filters, remember the start-up signal, which can be approximated by calculating 1/frequency difference. The first values of the resulting data set can be ignored. Start-up effects are not specific to imc FAMOS or to digital filters, they apply to filters in general.

Range of values

Please note the range of values permitted when selecting a reference value. Any values in the resulting data set outside the valid range are set to zero.

Unit

The x-unit of the source data set is expected in seconds. Time information specified for the function is expected in the same unit. The frequency specified is expected to be expressed in Hz. The result always has the y-unit of the source data, even in integrations. The x-unit of the source data set is used for a time display; for a frequency display, "Thirds" is used. This unit indicates that the number displayed counts the number of 1/3 octaves (1/3 octave number). A display with proper labeling can be selected in the Curve Window.

Y-scaling

The calculated values are always specified as linear. Select the mathematical function dB or the relevant display mode in the Curve Window to activate a display in dB.

Reduction

Note that during data reduction, the time-wighed data set is merely resampled. With a reduction factor of 3 for example, only the third, sixth, ninth, etc. are actually saved, while the first and second are discarded. For this reason, you should not set the samling significantly slower than the time constant. 3 samples per time constant are recommended; any more would lie approximately in line with these. A reductions factor of zero means that the equally-weighted true RMS is calculated for all data - the time constant for the time wighting is ignored.

Examples:

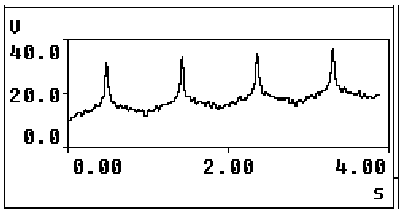
A test data set is obtained from the sample data set SLOPE. Here, the 1/3-octave with the center frequency 10Hz is plotted over time. Thus the intention is to determine how the frequency components change in the range 10Hz over time.

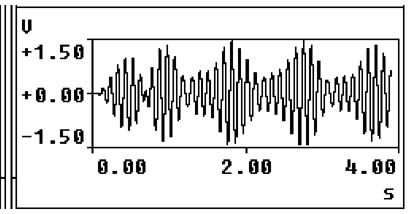
First the data set b is created by interpolation and changing the sampling rate, giving it a duration of approximately 4 s and a sampling rate of 1 ms.

```
b = ipol((xdel(slope, 0.01), 10)
```

The frequency band analysis is now initialized without integration, no time wieghting and no data reduction. A third-octave filtering is performed without frequency weighting and the data are not normalized. This is the most simple settings configuration of the function OctI(). Next, the OctA function is used to create the data set t_10, which displays the band-pass filtered signal at the 1/3 octave 10Hz.

```
OctI(0, 0, 1, -2, 0, 0, 0)
t_10 = OctA(b, 10.0, 0)
```

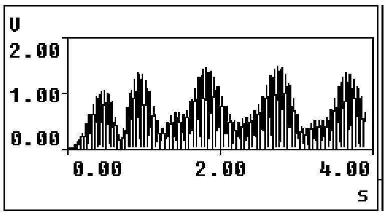


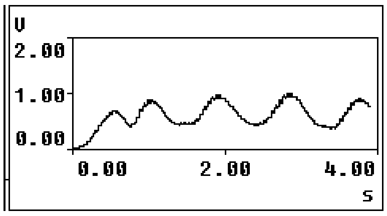


To view the amplitudes of this signal over time, rectification is necessary. This is performed using time weighting. This is performed first using a very small time constant (=1e-20 s), then with the standard FAST time constant. Using small time constants correspond to rectifying the signal conventionally.

```
OctI(0, 1e-20, 1, -2, 0, 0, 0)
tgl_10 = OctA(b, 10.0, 0)
OctI(0, -1, 1, -2, 0, 0, 0)
tf_10 = OctA(b, 10.0, 0)
```

With averaging, amplitude trend is much more apparent:



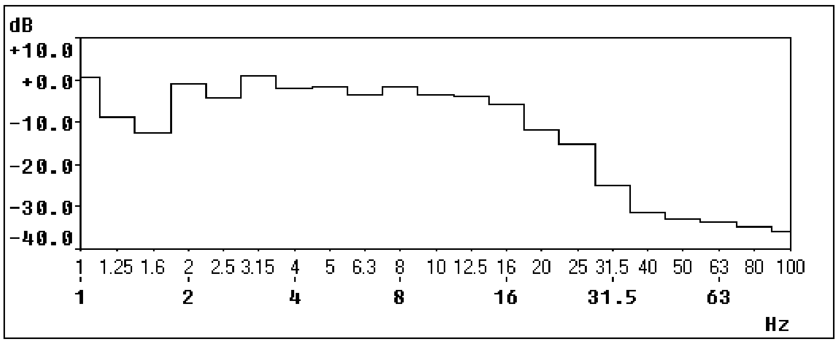


Now several third-octave bands are determined simultaneously (once) for the entire data set. This time, the reduction value is set to 0, for which reason the time wighting is effectively deactivated. The 1/3-octave spectrum is to be determined for the band of 1 Hz to 100 Hz. The function returns the amplitudes of the specified third-octave calculated from or averaged over the entire data set, since for a reduction factor of 0, one value pwer data set length is generated.

```
OctI(0, 0, 0, -2, 0, 0, 0)
tz = OctA(b, 1, 100)
```

In this graphical display, the x-axis is labeled in 1/3-octaves and octaves. This display can be selected via the curve window's Display menu and by setitng the x-axis accordingly.

Additionally, the y-axis is scaled in dB and step display mode is selected.

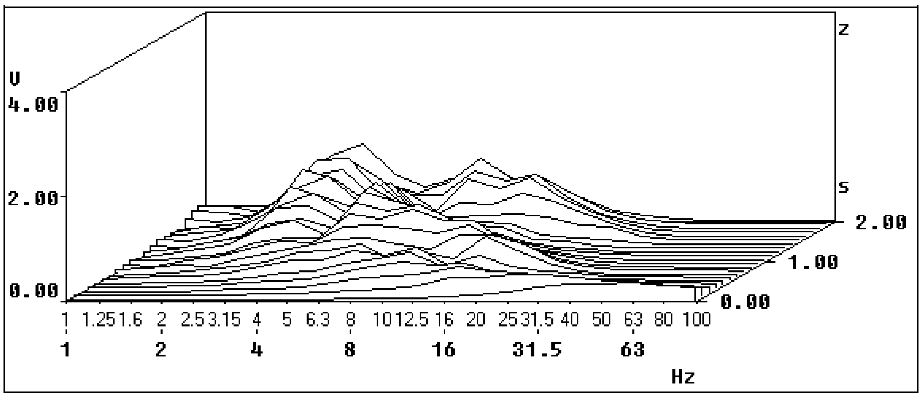


Now the most complicated procedure is applied. All frequencies from the specified third band (1Hz to 100Hz) are determined over time and displayed in a waterfall diagram. A reduction factor of 100 and the FAST time wighting are to be used - all other settings are the same as in the previous example. The OctA function creates a data set which must be processed and displayed in a very specific way.

```
OctI(0, -1, 100, -2, 0, 0, 0)
tz_21 = OctA(b, 1, 100)
```

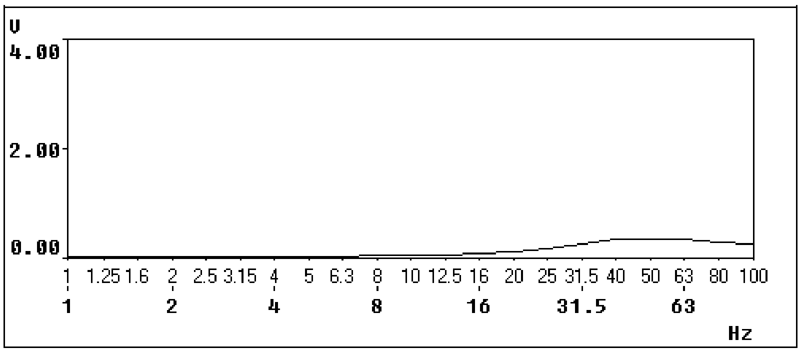
To display the data set as a waterfall diagram, first display the data set in a curve window, then select waterfall and period comparison display modes and third-octave labeling for the x-axis.

A period of 21 is specified for the selection of further curves. This is the number of 1/3 octaves which lie between 1Hz and 100Hz. The result of the OctA function can be thought of as a multi-period data set. The number of periods to be compared can be specified, in this example, there are 21 (this number 21 has nothing to do with the number of 1/3 octaves, it is the quotient of (2sec. / 0.1s + 1)). With the sampling rate of 1 ms for the source data set and a reduction factor of 100, a complete 1/3 octave spectrum is created every 0.1s. Use the options for 3D-display to scale the z-axis accordingly with dz=0.1s. Thus, the z-axis is labeled from 0s to 2s.



Now individual spectra, namely the first three, are excised from the data set, which can be regarded as a matrix. For this purpose, the function Perio() is used. Here, too the length of a spectrum (21 1/3-octaves) must be specified.

```
t0 = Perio(tz_21, 21, 0)
t1 = Perio(tz_21, 21, 1)
t2 = Perio(tz_21, 21, 2)
```



See also:

OctI, SpecThirds, SpecThirds_1, OtrRpmThirds, ExpoRMS, ABCRating

OctI

Initialization of the 1/3-octave analysis function OctA().

Declaration:

```
OctI ( SvIntegrations, SvTimeRating, SvReduction, SvWidth, SvFrequencyRating, SvNorm, Zero )
```

Parameter:

SvIntegrations	Number of intgrations to perform
	0 : No integration (default)
	1 : One integration
	2 : Double integral
	-1 : First derivative
	-2 : Second derivative
SvTimeRating	Averaging duration, time weighting
	>0 : Free averaging duration
	0 : No averaging
	-1 : FAST (default)
	-2 : SLOW
	-3 : Impulse
	-4 : Peak
SvReduction	Reduction factor (default = 0)
SvWidth	Width of the frequency bands
	-1 : Octave
	-2 : 1/3-octave (default)
	-3 : 1/12 octave
	-4 : 1/24 octave
SvFrequencyRating	Frequency weighting
	0 : Linear (default)
	1 : A
	2 : B
	3 : C
	4 : D
SvNorm	Reference value for normalization; multiplication by this value is performed. The default is 0 - no normalization.
Zero	Reserved parameter; set to 0.

Description:

This function initializes a 1/3-octave filter analysis. Several parameters can be initialized at once, which later do not need to be entered again when repeating the use of the 1/3-octave analysis.

The effects of the parameters are described in detail in conjunction with the 1/3-octave analysis function OctA().

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

See also:

OctA, SpecThirds, SpecThirds_1, OtrRpmThirds

OdsGetLastErrorCode

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Requests the error code of the ODS function error which occurred most recently.

Declaration:

```
OdsGetLastErrorCode ( ) -> EwError
```

Parameter:

EwError	
EwError	Error code

Description:

Calling this function deletes the internal error record.

See also:

OdsGetLastErrorTxt

OdsGetLastErrorTxt

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Requests the error description of the ODS function error which occurred most recently.

Declaration:

```
OdsGetLastErrorTxt ( ) -> TxError
```

Parameter:

TxError	
TxError	Error text

Description:

Calling this function deletes the internal error record.

See also:

OdsGetLastErrorCode

OdsIEAddAttribute

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Adds a numeric instance attribute.

Declaration:

```
OdsIEAddAttribute ( InstanceID, TxName, Data, Zero ) -> Status
```

Parameter:

InstanceID	ID of the instance element
TxName	Name of the new attribute.
Data	Data set or single value with the content of the new attribute.
Zero	Reserved parameter. Always set to 0.
Status	
Status	Function status
	0 : Error
	1 : Function executed successfully.

Description:

A new attribute is added to the specified instance element.

Such a locally defined attribute (also called an instance attribute) is private for the instance element and not defined by the application model.

The ODS data type of the new attribute is automatically determined from the data type of the data set which is passed in.

In case of error, a 0 is returned; if the function is successful, a 1. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

Two private attributes are assigned to each instance element which is currently selected in the ODS-Browser's tree diagram.

```
SessionID = OdsPluginSessionConnect( 0 )
IF SessionID <> 0
   IDList = OdsPluginListSelItems(0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      OdsIEAddAttribute( IDList[i], "Status", 1, 0)
      OdsIEAddAttributeTxt( IDList[i], "Inspector", "J. Doe", 0)
      i = i+1
   END
END
```

See also:

OdsIEGetAttribute, OdsIESetAttributeTxt, OdsIEAddAttributeTxt

OdsIEAddAttributeTxt

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Adds an instance attribut (text).

Declaration:

```
OdsIEAddAttributeTxt ( InstanceID, TxName, TxContent, Zero ) -> Status
```

Parameter:

InstanceID	ID of the instance element
TxName	Name of the new attribute.
TxContent	Content of the new attribute.
Zero	Reserved parameter. Always set to 0.
Status	
Status	Function success
	1 : Function executed successfully
	0 : Error.

Description:

A new text attribute is added to the specified instance element.

Such a locally defined attribute (also called an instance attribute) is private for the instance element and not defined by the application model.

The ODS data type of the new attribute is [DT_STRING].

In case of error, a 0 is returned; if the function is successful, a 1. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

Two private attributes are assigned to each instance element which is currently selected in the ODS-Browser's tree diagram.

```
SessionID = OdsPluginSessionConnect( 0 )
IF SessionID <> 0
   IDList = OdsPluginListSelItems(0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      OdsIEAddAttribute( IDList[i], "Status", 1, 0)
      OdsIEAddAttributeTxt( IDList[i], "Inspector", "J. Doe", 0)
      i = i+1
   END
END
```

See also:

OdsIEGetAttribute, OdsIESetAttributeTxt, OdsIEAddAttributeTxt

OdsIEBuildVarName

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

A valid FAMOS variable name is derived from an instance element's ASAM-path.

Declaration:

```
OdsIEBuildVarName ( InstanceID, Option ) -> TxVarName
```

Parameter:

InstanceID	ID of the instance element
Option	Option for forming a name.
	0 : Name of the instance element only
	1 : Prefixes the name of the parent element.
	2 : Appends the name of the parent element.
TxVarName	
TxVarName	Valid FAMOS variable name.

Description:

A valid FAMOS variable name is derived from a given instance element. For this purpose, the name of the instance element (attribute "name" in the ODS base model) and further ASAM-path elements which depend on the option (2nd parameter) are given consideration.

Invalid characters in the resulting name are replaced by an underline '_'.

In case of error, an empty string is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to an ODS-server named "ODSTest" is set up and all measurement quantities whose names begin with "Channel1_" are imported to FAMOS. The FAMOS variable name is formed from the name of measurement to which it belongs and the channel name.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurementQuantity" ,"Channel1_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxVarName = OdsIEBuildVarName( IDList[i], 1)
       <TxVarName> = OdsIEGetChannel( IDList[i], "", 0)
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetChannel, OdsIEGetMeasurement

OdsIECreateElement

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Creates a new instance element

Declaration:

```
OdsIECreateElement ( Typ, InstanceID, Option, Zero ) -> IDNewInstance
```

Parameter:

Typ	ODS type designator for the instance to be created.
InstanceID	ID of the instance element used. Its meaning depends on the next parameter.
Option	Relation of the new instance to [InstanceID].
	0 : The new instance element is created as the child of [InstanceID].
	1 : The new instance-element is created as a copy of [InstanceID]. Both elements then have the same father.
	2 : The new instance element is a TopLevel-element, so it has no father. [InstanceID] must be 0.
Zero	Reserved parameter; always set to 0.
IDNewInstance	
IDNewInstance	If successful, ID of the newly created instance, else 0.

Description:

The new instance's attributes are initialized with default values.

The function can **not** be used to create instances of the type "AoMeasurementQuantity", "AoLocalColumn" or "AoSubMatrix". Use the function OdsIEImportData() for this purpose.

Examples:

A new TopLevel-instance of the type "AoTest" is created and various attributes are set.

```
SessionID = OdsSessionCreate( "TEST", "","", 0 )
IF SessionID > 0
   id = OdsIECreateElement("#AoTest", 0, 2, 0)
   IF id > 0
      OdsIESetAttributeTxt( id, "#name", "Test_123", 0)
      OdsIESetAttributeTxt( id, "#description", "Test only", 0)
      OdsIESetAttribute( id, "Charge", 1205, 0)
      OdsIESetAttribute( id, "#version_date", ZeitSystem?(), 0)
      OdsSessionClose(0)
   END
END
```

See also:

OdsIEDeleteElement, OdsIEImportData

OdsIEDeleteElement

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Deletes an instance element

Declaration:

```
OdsIEDeleteElement ( InstanceID, Zero ) -> Status
```

Parameter:

InstanceID	ID of the instance element to be deleted.
Zero	Reserved parameter. Always set to 0.
Status	
Status	Success of the function
	0 : Error
	1 : Function performed successfully.

Description:

Deletion of instances cannot be reversed, therefore use this command with all due caution.

Deleting an instance element deletes all of the element's children, in other words all the elements in the branch of the tree diagram to which it belongs.

No check is made of whether deleting the instances causes the data storage to become inconsistent. Therefore it can occur that other instances contain relations which refer to the instances deleted here. These then become invalid, of course.

Examples:

All instances of the type "AoTest" whose names start with "T_" are listed.

Each instance found is deleted (after a confirmation by the user).

```
SessionID = OdsSessionCreate( "Test", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoTest", "T_*", 0)
   i = 1
   Count = Leng?( IDList)
   WHILE i >= Count
      TxName = OdsIEGetAttributeTxt( IDList[i], "#name", 0)
      IF BoxMessage( "Delete instance?", TxName, "?4") = 1
         OdsIEDeleteElement( IDList[i], 0)
      END
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsIECreateElement

OdsIEGetAttribute

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Inquires the value of an instance element's numerical attribute.

Declaration:

```
OdsIEGetAttribute ( InstanceID, TxAttributeName, Zero ) -> Values
```

Parameter:

InstanceID	ID of the instance element
TxAttributeName	Attribute name. Either the corresponding name according to the data basis' application model or, if it exists, the name according to the ODS base model.
Zero	Reserved parameter. Always set to 0.
Values	
Values	Value(s) of the specified attribute.

Description:

As the attribute designator (2nd parameter), either the name according to the application model or, alternatively, the name from the ODS base model can be entered. For base model names, a "#" must be prefixed to the actual name.

The attribute's contents must be convertible either to a numerical value or to a vector of numerical values.

If the attribute is a statement of time/date (data type DT_DATE), it is converted to the internal FAMOS time format (double).

In case of error, an empty data set (length = 0) is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

Every ODS-element must, as a obligatory attribut, possess a unique ID; this attribute's name in the base model is "id". A connection to an ODS-server named "ODSTest" is opened and the ID of a measurement specified by its ASAM-path is read out.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   InstanceID = OdsIEListByAsamPath("/[Test]T_2;/[SubTest]SV_5;/[Measurement]N1_14;", 0)
   IF InstanceID > 0
      ID = OdsIEGetAttribute( InstanceID, "#id", 0)
      ; Supposing the corresponding attribute name in the
      ; application model is "MeasurementID", then the call
      ; ID = OdsIEGetAttribute( InstanceID, "MeasurementID", 0)
      ; is equivalent
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetAttributes, OdsIEGetAttributeTxt, OdsIEGetPropertyTxt

OdsIEGetAttributes

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Requests all of an instance element's attributes

Declaration:

```
OdsIEGetAttributes ( InstanceID, Zero ) -> GrAttribute
```

Parameter:

InstanceID	ID of the instance element
Zero	Reserved parameter. Always set to 0.
GrAttribute	
GrAttribute	Group of all attributes of an instance element.

Description:

The function reads out all of a given instance element's attributes returns them in the form of a data group. The data group's channels contain the name of the respective attributes; the respective resulting data type (data set or text) is determined automatically according to the attribute's ODS data type.

In case of error, an empty data group is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to the ODS-server "ODSTest" is opened and all measurements whose names begin with "Channel1_" are imported to FAMOS. One group per measurement group is created whose name is formed from the same of the measurement to which it belongs and from the name of the measurement group. Each group contains the attributes of the measurement and the actual measured data.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurementQuantity" ,"Channel1_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxVarName = OdsIEBuildVarName( IDList[i], 1)
       <TxVarName> = OdsIEGetAttributes( IDList[i], 0)
       <TxVarName>:Daten = OdsIEGetChannel( IDList[i], "", 0)
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetAttribute, OdsIEGetAttributeTxt, OdsIEGetPropertyTxt, OdsIEGetChannel

OdsIEGetAttributeTxt

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Requests the contents (as text) of an instance element attribute.

Declaration:

```
OdsIEGetAttributeTxt ( InstaneID, TxAttributeName, Zero ) -> TxContents
```

Parameter:

InstaneID	ID of the instance element
TxAttributeName	Name of the attribute to find
Zero	Reserved parameter. Always set to 0.
TxContents	
TxContents	Contents of the specified attribute

Description:

As the attribute designator (2nd parameter), either the name according to the application model or, alternatively, the name from the ODS base model can be entered. For base model names, a "#" must be prefixed to the actual name.

The attribute's content must be convertible to text, so it must not contain, for example, a vector of numeric values.

In case of error, an empty text is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to an ODS-server named "ODSTest" is opened and the names and starting dates of all messages [AoMeasurement] it contains are listed.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurement" ,"", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxName = OdsIEGetAttributeTxt( IDList[i], "#name", 0)
      TxDate = OdsIEGetAttributeTxt( IDList[i], "#measurement_begin", 0)
      BoxOutput( TxName + " " + TxDate, LEER, "", 1)
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetAttribute, OdsIEGetAttributes, OdsIEGetPropertyTxt

OdsIEGetChannel

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

For a given measurement quantity, the measured data are read out and converted to a FAMOS data set.

Declaration:

```
OdsIEGetChannel ( InstanceID, TxChannelName, Zero ) -> Measured data
```

Parameter:

InstanceID	ID of the instance element whose measured data are to be read out. The instance must be either of the type Measurement [AoMeasurement] or Measurement Quantity [AoMeasurementQuantity].
TxChannelName	If the first parameter refers to a measurement, the name of the desired measurement quantity must be entered here. If the first parameter refers directly to a measurement quantity, an empty text must be passed in.
Zero	Reserved parameter. Always set to 0.
Measured data	
Measured data	Data set with the measured data from the selected measurement quantity.

Description:

The function reads out the measured data for a given measurement quantity [AoMeasurementQuantity].

The instance specified in the first parameter must belong to one of the following types:

Measurement [AoMeasurement]: The first parameter determines the measurement, and the channel select from it is determined by the name specified in the 2nd parameter.

Measurement quantity [AoMeasurementQuantity]: The first parameter directly determines the desired measurement quantity. Then an empty text is passed in for the 2nd parameter.

In case of error, an empty data set (length = 0) is returned. The error cause can then be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to the ODS-server "ODSTest" is set up. Then all measurements are found whose names begin with "TEST2_". All channels belonging to these measurements and whose names begin with "T2" are read out.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurement" ,"TEST2_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      chan = OdsIEGetChannel( IDList[i],"T2", 1, 0)
      TxName = OdsIEBuildVarName( IDList[i],0)
      TxName = TxName + "_T2"
      RENAME chan <TxName>
   END
   OdsSessionClose(0)
END
```

And, in order to read out all measurement quantities whose names begin with "T2", regardless of what measurement they belong to:

```
...
 IDList = OdsIEListByType("#AoMeasurementQuantity" ,"T2", 0)
 Count = Leng?( IDList)
 i = 1
 WHILE i <= Count
      chan = OdsIEGetChannel( IDList[i],"", 1, 0)
      TxName = OdsIEBuildVarName( IDList[i],1)
      RENAME chan <TxName>
 END
...
```

See also:

OdsIEGetMeasurement

OdsIEGetMeasurement

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

The channels contained in a measurement are read out and converted to a FAMOS data group.

Declaration:

```
OdsIEGetMeasurement ( InstanceID, TxNamePattern, AttributeOption, Null ) -> GrMeasurement
```

Parameter:

InstanceID	InstanceID of the measurement. Type: AoMeasurement
TxNamePattern	Pattern for channel names
AttributeOption	Transfer the measurement's attributes, too?
	0 : No attributes
	1 : All attributes
Null	Reserved parameter. Always set to 0.
GrMeasurement	
GrMeasurement	Group of all channels specified and measurement attributes, if appropriate.

Description:

The function reads out the measurement channels [AoMeasurementQuantity] of the measurements selected and stores them as a data group. The instanceID passed in must refer to an instance element of the type measurement [AoMeasurement]. Optionally, measurement's attributes are read out too.

The 2nd parameter (TxNamePattern) can be used to limit the channels to be included by entering a search pattern to narrow down the search. The usual wildcard symbols '*' and '?' are allowed. If all channels are to be read, enter here either an empty text or "*".

In the case of error, an empty group is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to the ODS-server "ODSTest" is set up. Then all measurements are found whose names begin with "TEST2_". All channels belonging to these measurements and whose names begin with "T2" are read out.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurement" ,"TEST2_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILEE i <= Count
      GrMeas = OdsIEGetMeasurement( IDList[i],"T2*", 1, 0)
      TxName = OdsIEBuildVarName( IDList[i],0)
      RENAME GrMeas <TxName>
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetChannel

OdsIEGetPropertyTxt

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Queries pre-defined properties of an instance element.

Declaration:

```
OdsIEGetPropertyTxt ( InstanceID, TxName, Zero ) -> TxPropertyValue
```

Parameter:

InstanceID	ID of the instance element
TxName	Selection of the property to find.
	"Path" : Complete Asam path of the element.
	"AppName" : Name of the element in the application model.
	"BaseName" : Name of the element in the ODS base model.
	"ParentName" : Name of ther parent instance.
Zero	Reserved parameter. Always set to 0.
TxPropertyValue	
TxPropertyValue	Queried property

Description:

Along with the attributes defined by the base and application models, this instance element also possesses certain permanent properties which are mainly derived from the element's type and its position in the data source's ODS tree structure.

These include, for example, the instance element's unique ASAM-path, as well as the name of the corresponding element in the application and base model.

In case of error, an empty text is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to the ODS-server "ODSTest" is set up and the ASAM-pats of all measurements [AoMeasurement] it contains whose names start with "Test2_" are listed.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurement" ,"TEST2_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxPath = OdsIEGetPropertyTxt( IDList[i], "Path", 0)
      BoxOutput( TxPath, EMPTY, "", 1)
      ; The following call
      ; TxBaseName = OdsIEGetPropertyTxt( IDList[i], "BaseName", 0)
      ; always returns "AoMeasurement".
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetAttribute, OdsIEGetAttributes, OdsIEGetAttributeTxt

OdsIEImportData

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Adds measured data to the ODS data storage

Declaration:

```
OdsIEImportData ( InstanceID, Data, Zero ) -> IDNewInstance
```

Parameter:

InstanceID	ID of the instance element to which the data are to be added. Depending on the data type (data set or data group), the instance must be either of the type Measurement [AoMeasurement] or Test [AoTest/AoSubTest].
Data	Data set or data group with the measured data.
Zero	Reserved parameter. Always set to 0.
IDNewInstance	
IDNewInstance	ID of the newly created instance, if successful, else 0.

Description:

The measured data passed in as parameters are accepted in the data storage as a new instance.

Either a single data set or a data group can be imported.

[Data] is of the type Normal Waveform: The data are added to an existing measurement as a new channel. [InstanceID] must refer to an element of the type Measurement [AoMeasurement], and the newly created instance is of the type [AoMeasurementQuantity].

[Data] is of the type Data Group: the data are added to an existing test as a new measurement. [InstanceID] must refer to an element of the type Test [AoTest or AoSubTest], and the newly created instance is of the type [AoMeasurement].

Examples:

In the data storage, a new measurement named "M_212" is created as the child of the sub-test "SV7". The two FAMOS data sets "SO21" and "SO22" are imported to this measurement as measurement quantities:

Option 1:

```
SessionID = OdsSessionCreate( "Test", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoSubTest", "SV_7", 0)
   IF leng?(IDList) = 1
      idTest = IDList[1]
      idmea = OdsIECreateElement("#AoMeasurement", idTest, 0, 0)
      ok = OdsIESetAttributeTxt( idmea, "#name", "M_212", 0)
      idmeq1 = OdsIEImportData( idmea, SO21, 0)
      idmeq2 = OdsIEImportData( idmea, SO22, 0)
   END
   OdsSessionClose(0)
END
```

Option 2:

```
SessionID = OdsSessionCreate( "Test", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoSubTest", "SV_7", 0)
   IF leng?(IDList) = 1
      M_212:SO21 = SO21
      M_212:SO22 = SO22
      idmea  = OdsIEImportData( idTest, M_212, 0)
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEGetMeasurement, OdsIEGetChannel

OdsIEListByAsamPath

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Determines the instance element for a given ASAM path.

Declaration:

```
OdsIEListByAsamPath ( TxAsamPath, Zero ) -> InstanceID
```

Parameter:

TxAsamPath	Complete ASAM path for the desired instance element.
Zero	Reserved parameter. Always set to 0.
InstanceID	
InstanceID	ID of the instance element in case of success. This ID be supplied as a parameter for all instance-specific Kit functions.
	> 0 : ID of the instance element
	0 : Error

Description:

The ASAM path describes the position of the instance within the hierarchically-structured data storage and is unique for each instance. In terms of function and structure, it is approximately comparable to the path for a file in a file system.

Each partial element of the ASAM path is defined by the name of the describing application element (in square brackets), the name of the instance and, if available, the instance version, e.g.:

```
"/[Test]T_2;/[SubTest]SV_5;/[Measurement]N1_14;"
```

In case of error, 0 is returned. The error cause can be found using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to the ODS server named "ODSTest" is opened and the measurement data specified by their ASAM-path are read into FAMOS.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   InstanceID = OdsIEListByAsamPath("/[Test]T_2;/[SubTest]SV_5;/[Measurement]N1_14;", 0)
   IF InstanceID > 0
      TxVarName = OdsIEBuildVarName( InstanceID, 0)
       <TxVarName> = OdsIEGetChannel( InstanceID, "", 0)
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEListByType, OdsIEListChildren, OdsPluginListSelItems

OdsIEListByType

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Determines all instance elements of a given type. The search can be limited by specifying a name pattern (using wildcard symbols).

Declaration:

```
OdsIEListByType ( TxOdsType, TxNamePattern, Zero ) -> InstanceElements
```

Parameter:

TxOdsType	Designation of an ODS element type. The type name can be derived from either the data source's application model or the ODS base model.
TxNamePattern	All instances whose names match the pattern (using wildcard symbols) are returned
Zero	Reserved parameter. Always set to 0. Exception for Namepattern ="#", see below.
InstanceElements	
InstanceElements	IDs of all instance elements which correspond to the specifications.

Description:

As the type-designator (1st parameter), either the name according to the application model or, alternatively, the name from the ODS base model can be entered. For base model names, a "#" must be prefixed to the actual name.

In the name pattern (2nd parameter), the wildcard characters "?" and "*" are used in the usual way ("?" stands for any one character, "*" for any string of characters).

To find every instance of the desired type independently, enter here an empty text or "*".

Special case: If the 2nd parameter has the content "#", then the 3rd parameter is interpreted as ODS instance id. This corresponds to the value of the base attribute "id" of the instance element. The result has than either the length 0 (element not found) or 1.

The data set returned contains the IDs of all instance elements which correspond to specifications. To determine the number of elements found, Leng?() can be used. To access a particular element, use the '[ ]' operator (see example).

In case of error, an empty data set (length = 0) is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A connection to the ODS-server named "ODSTest" is opened and all measurement quantities whose name starts with "Channel1_" are imported to FAMOS.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurementQuantity" ,"Channel1_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxVarName = OdsIEBuildVarName( IDList[i], 1)
       <TxVarName> = OdsIEGetChannel( IDList[i], "", 0)
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsIEListByAsamPath, OdsIEListChildren, OdsPluginListSelItems

OdsIEListChildren

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Determines the direct child-instances (descendants) of an instance element.

Declaration:

```
OdsIEListChildren ( InstanceID, TxOdsType, TxNamePattern, Zero ) -> InstanceElements
```

Parameter:

InstanceID	ID of the instance-element whose children are to be determined.
TxOdsType	Designation of an ODS element type. Only instances of the specified type are returned. The type name can be derived either from the data source's application model or from the ODS base model. Empty string if the type doesn't matter.
TxNamePattern	Only instances whose name corresponds to the pattern entered here (using wildcard symbols) are returned. Empty text or "*", if the name doesn't matter.
Zero	Reserved parameter. Always set to 0.
InstanceElements	
InstanceElements	IDs of all instance elements which match the specifications.

Description:

As the type-designator (2nd parameter), either the name according to the application model or, alternatively, the name from the ODS base model can be entered. For base model names, a "#" must be prefixed to the actual name.

If all children are to be listed regardless of the type, an empty string can be entered here.

In the name pattern (3rd parameter), the wildcard symbols '?' and '*' can be used in the usual way ('?' stands for any one character, '*' for any string of characters).

To find all instances regardless of their name, enter here an empty string or "*".

The data set returned contains the IDs of all instance elementswhich correspond to the specifications. To determine the number of element found, Leng?() can be used; and the operator '[ ]' can be used to access a particular element.

In case of error, an empty data set (length = 0) is returned. The error cause can be determined using either of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

An ODS-server named "ODSTest" is opened and a search is done from all measurements whose name begins with "TestA_". For each measurement found, the names of the measurement quantities it contains are listed.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType( "#AoMeasurement, "TestA_*", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i<= Count
      TxName = OdsIEGetAttributeTxt( IDList[i], "#name", 0)
      BoxOutput( TxName, EMPTY, "", 1)
      IDChildList = OdsIEListChildren( IDList[i], "#AoMeasurementQuantity", "", 0)
      ChildCount = Leng?( IDChildList)
      j = 1
      WHILE j <= ChildCount
         TxName = ".." + OdsIEGetAttributeTxt( IDChildList[j], "#name", 0)
         BoxOutput( TxName, EMPTY, "", 1)
         j = j+1
      END
      i = i+1
   END
   OdsSessionClose(0)
END
```

Note: According to the ODS base model, the base element [AoMeasurementQuantity] is a direct child of [AoMeasurement].

See also:

OdsIEListByAsamPath, OdsIEListByType, OdsPluginListSelItems

OdsIERemoveAttribute

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Deletes an instance attribute

Declaration:

```
OdsIERemoveAttribute ( InstanceID, TxName, Zero ) -> Status
```

Parameter:

InstanceID	ID of the instance element
TxName	Name of the attribute to be deleted.
Zero	Reserved parameter. Always set to 0.
Status	
Status	Function success
	1 : Function executed successfully
	0 : Error.

Description:

The specified attribute is deleted.

The attribute to delete must be a local instance attribute. Attributes defined by the application model cannot be deleted.

In case of error, a 0 is returned; if the function is successful, a 1. The error cause can be determined using one of the funcitons OdsGetLastErrorTxt() or OdsGetLastErrorCode().

See also:

OdsIEAddAttribute, OdsIEAddAttributeTxt

OdsIESetAttribute

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Changes an instance element's numeric attribute.

Declaration:

```
OdsIESetAttribute ( InstanceID, TxAttributeName, Data, Zero ) -> ErrorCode
```

Parameter:

InstanceID	ID of the instance element
TxAttributeName	Name of the attribute. Either the corresponding name according to the application model of the data basis or the name according to the ODS base model, if defined.
Data	Data set or single value with the attribute's new content.
Zero	Reserved parameter. Always set to 0.
ErrorCode	
ErrorCode	Success of the function
	1 : Function executed successfully
	0 : Error

Description:

As the attribute designator (2nd parameter), either the name according to the application model or, alternatively, the name from the ODS base model can be entered. For base model names, a "#" must be prefixed to the actual name.

The data type of the attribute addressed must be compatible to the data parameter specified. For numeric attributes, a single value is expected; for attributes with a vector data type, a data set of an appropriate type (e.g. simple real or complex) must be specified.

In case of error, a 0 is returned, and if the function is executed successfully, a 1. the error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A new TopLevel instance of the type "AoTest" is created and various attributes are set.

```
SessionID = OdsSessionCreate( "TEST", "","", 0 )
IF SessionID > 0
   id = OdsIECreateElement("#AoTest", 0, 2, 0)
   IF id > 0
      OdsIESetAttributeTxt( id, "#name", "Test_123", 0)
      OdsIESetAttributeTxt( id, "#description", "Test only", 0)
      OdsIESetAttribute( id, "Charge", 1205, 0)
      OdsIESetAttribute( id, "#version_date", TimeSystem?(), 0)
      OdsSessionClose(0)
   END
END
```

See also:

OdsIEGetAttribute, OdsIESetAttributeTxt

OdsIESetAttributeTxt

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Changes the content (text) of an instance element's attribute.

Declaration:

`OdsIESetAttributeTxt ( InstanceID, TxAttributeName, TxContent, Zero ) -> ErrorCode`

Parameter:

InstanceID	ID of the instance element
TxAttributeName	Name of the attribute. Either the corresponding name according to the application model of the data basis or, the name according to the ODS base model, if defined.
TxContent	New content of the addressed attribute.
Zero	Reserved parameter. Always set to 0.
ErrorCode	
ErrorCode	Success of the function
	1 : Function executed successfully
	0 : Error.

Description:

As the attribute designator (2nd parameter), either the name according to the application model or, alternatively, the name from the ODS base model can be entered. For base model names, a "#" must be prefixed to the actual name.

The addressed attribute's data type must be <Text>.

In case of error, a 0 is returned; if the function is successful, a 1. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

A new TopLevel instance of the type "AoTest" is created and various attributes are set.

```
SessionID = OdsSessionCreate( "TEST", "","", 0 )
IF SessionID > 0
   id = OdsIECreateElement("#AoTest", 0, 2, 0)
   IF id > 0
      OdsIESetAttributeTxt( id, "#name", "Test_123", 0)
      OdsIESetAttributeTxt( id, "#description", "Test only", 0)
      OdsIESetAttribute( id, "Charge", 1205, 0)
      OdsIESetAttribute( id, "#version_date", TimeSystem?(), 0)
      OdsSessionClose(0)
   END
END
```

See also:

OdsIEGetAttributeTxt, OdsIESetAttribute

OdsInitialize

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Initialization of the ODS-system. The necessary parameters for the (re-)initialization of the CORBA runtime environment are defined.

Declaration:

```
OdsInitialize ( TxNSAddress, TxORBParameter, TxPrivateParameter, Zero ) -> Error code
```

Parameter:

TxNSAddress	Address of the CORBA naming service in the form "PCAddress:PortNumber"
TxORBParameter	Additional parameters for the initialization of the CORBA-ORB. Usually empty.
TxPrivateParameter	Additional initialization parameters for the ODS-browser. Usually empty.
Zero	Reserved parameter. Always set to 0.
Error code	
Error code	Success of the function
	1 : Function executed successfully
	0 : Error

Description:

The CORBA-kernel used by the Browser is (re)started, for which certain necessary initialization parameters are to be specified.

The initialization parameters to be specified depend mainly on the properties of the ODS-server to be connected. These especially include the specification of the "CORBA naming service" used by the server. The latter is absolutely necessary for the Browser to find the desired server.

If this function is not called, the settings set in the ODS plug-in are used (dialog "Presettings/System").

It usually only makes sense to call the function in a sequence before all other ODS access functions. This achieves a precisely defined system response regardless of the current plug-in settings.

Calling this function closes all connections currently open! The CORBA-kernel is then stopped and restarted with the parameters supplied.

In case of error, a 0 is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Multithreading: All functions of the ODS kit may only be called in the Standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

A connection to an ODS-server is opened and the names of all measurements [AoMeasurement] it contains are listed. The desired ODS-server signed in under the name "ODSTest" at the ODS naming service running at Port 900 of the local PC.

```
OdsInitialize("localhost:900","","",0)
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurement" ,"", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxName = OdsIEGetAttributeTxt( IDList[i], "#name", 0)
      BoxOutput( TxName, LEER, "", 1)
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsSessionCreate, OdsPluginSessionConnect

OdsPluginListSelItems

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Determines which instance elements in the ODS plug-in's active session are selected.

Declaration:

```
OdsPluginListSelItems ( Zero ) -> InstanceElementList
```

Parameter:

Zero	Reserved parameter. Always set to 0.
InstanceElementList	
InstanceElementList	ID's of all instance elements which are currently selected.

Description:

The data set returned contains the IDs of all selected instance elements. To find out the number of elements found, the function Leng?() can be used; and for accessing a particular element, the '[ ]' operator (see example).

The active session for the Kit must previously have been set to a plug-in session using OdsPluginSessionConnect() or OdsSessionSelect().

In case of error, an empty data set (length = 0) is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

All instance elements which are currently selected in the ODS-Browser's tree diagram are found and exported to FAMOS along with all their attributes.

```
SessionID = OdsPluginSessionConnect( 0 )
IF SessionID <> 0
   IDList = OdsPluginListSelItems(0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxVarName = OdsIEBuildVarName( IDList[i], 0)
       <TxVarName> = OdsIEGetAttributes( IDList[i], 0)
      i = i+1
   END
END
```

See also:

OdsPluginSessionConnect, OdsSessionSelect

OdsPluginSessionConnect

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

The session currently displayed by the ODS plug-in becomes the kit's new active session.

Declaration:

`OdsPluginSessionConnect ( Zero ) -> SessionID`

Parameter:

Zero	Reserved parameter. Always set to 0.
SessionID	
SessionID	ID of the session
	< 0 : Session-ID
	0 : Error.

Description:

This function is used in order to subsequently be able to use the ODS-Kit's functions in the session currently open in the ODS-Browser plug-in.

A session activated with this function may **not** be closed using OdsSessionClose().

In case of error, a 0 is returned. The error cause can be determined using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Examples:

All instance elements that are curently selected in the ODS-Browser's tree diagram are found and exported together with all their attributes to FAMOS.

```
SessionID = OdsPluginSessionConnect( 0 )
IF SessionID <> 0
   IDList = OdsPluginListSelItems(0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxVarName = OdsIEBuildVarName( IDList[i], 0)
       <TxVarName> = OdsIEGetAttributes( IDList[i], 0)
      i = i+1
   END
END
```

See also:

OdsPluginListSelItems, OdsSessionCreate, OdsSessionSelect

OdsSessionClose

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Closes the connection to an ODS-server.

Declaration:

```
OdsSessionClose ( Option ) -> ErrorCode
```

Parameter:

Option	Selection
	0 : Closes the active session
	-1 : Closes all sessions which were opened by the Kit.
ErrorCode	
ErrorCode	Success of the function
	1 : Function performed successfully
	0 : Error

Description:

[Option] = 0: The active session is closed. The session must have been opened beforehand using the function OdsSessionCreate(). The function may **not** be used on sessions that were activated using the function OdsPluginSessionConnect().

If multiple sessions are open at once, then once this session has been successfully closed, the oldest one left is activated. If you wish to continue working with a different one instead, use the function OdsSessionSelect() afterwards.

[Option] = -1: All sessions are closed which were previously opened by the Kit using OdsSessionCreate().

In case of error, a 0 is returned. The error cause can be found by using one of the the functions OdsGetLastErrorTxt() and OdsGetLastErrorCode().

See also:

OdsSessionCreate, OdsSessionSelect, OdsPluginSessionConnect

OdsSessionCreate

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Sets up a connection to an ODS data source (ODS-server).

Declaration:

```
OdsSessionCreate ( TxServerName, TxServerOptions, TxSessionParameter, Null ) -> SessionID
```

Parameter:

TxServerName	Name of the ODS data source (ODS-server).
TxServerOptions	Server-specific settings needed for establishing a connection. Usually empty.
TxSessionParameter	Options, passed to the server when the session is opened, e.g. user name and password.
Null	Reserved parameter. Always set to 0.
SessionID	
SessionID	ID of the open connection or 0 if the connection could not be made.
	> 0 : Session-ID
	0 : Error.

Description:

The function serves to set up a connection (session) with an ODS-server. Afterwards, the kit's other functions are used to access the data storage.

If connection is successfully established, the session opened here becomes the kit's active session. Almost all of the ODS-Kit's functions refer to the currently active session.

To switch between multiple sessions open at the same time, use the function OdsSessionSelect(). This contains the session-ID returned here as a parameter.

The first parameter (name of the ODS-service) must be entered in the same form as the service was registered by the server in the CORBA naming service used. In the rare case that two servers have the same name and differ only by the additional parameter "kind" (see output field "Kind" in the dialog "Connect to server"), the server must be specified by "name.kind".

For special applications in mixed environments, it is also possible to specify a file which contains the IOR (Interoperable Object Reference) of an existing CORBA-connection to an ODS-server. In this way, the Kit can hook up to an existing connection. The syntax for [TxServerName] is then "IOR_FILE=<filename>", where <filename> refers to a file containing a valid IOR.

The options for opening the session (3rd parameter) are entered in the form "NAME=VAUE" and separated by commas.

The following parameters, among others for authentication, are defined by ODS; whether they are actually used depends on the server.

Parameter	Definition/Content
USER	User name
PASSWORD	Password
OPENMODE	E.g.: "read","write"

In addition to these pre-defined parameters, further server-specific options can be set here, e.g.:

```
SessionID = OdsSessionCreate( "ODSTest", "","USER=hans, PASSWORD=fgasr, PROJECT=foo", 0 )
```

If you don't need a connection any longer, you absolutely should close it explicity using the function OdsSessionClose().

If you wish to work on a session using the kit which is currently opened with the ODS-Browser plug-in, use instead of OdsSessionCreate() the function OdsPluginSessionConnect().

In case of error, a 0 is returned. The cause of the error can be found using either of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

Multithreading: All functions of the ODS kit may only be called in the Standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

A connection to an ODS-server named "ODSTest" is opened and the names of all measurements [AoMeasurement] it contains are listed.

```
SessionID = OdsSessionCreate( "ODSTest", "","", 0 )
IF SessionID > 0
   IDList = OdsIEListByType("#AoMeasurement" ,"", 0)
   Count = Leng?( IDList)
   i = 1
   WHILE i <= Count
      TxName = OdsIEGetAttributeTxt( IDList[i], "#name", 0)
      BoxOutput( TxName, EMPTY, "", 1)
      i = i+1
   END
   OdsSessionClose(0)
END
```

See also:

OdsSessionSelect, OdsSessionClose, OdsPluginSessionConnect

OdsSessionSelect

Scope: ASAM-ODS Browser

Available in: Enterprise Edition and above (ODS-Browser-Kit)

Switches to a different session

Declaration:

```
OdsSessionSelect ( SessionID ) -> ErrorCode
```

Parameter:

SessionID	ID of the session to be activated
ErrorCode	
ErrorCode	Success of the function
	1 : Function performed successfully
	0 : Error

Description:

Nearly all Kit-functions affect the active session. If multiple sessions are open at once, this function can be used to switch the active session. Subsequent calls to session-specific kit functions then pertain to the session activated here.

The first parameter [SessionID] must previously have been returned by a call to OdsSessionCreate() or OdsPluginSessionConnect().

In case of an error, 0 is returned. The error cause can be found using one of the functions OdsGetLastErrorTxt() or OdsGetLastErrorCode().

See also:

OdsSessionCreate, OdsPluginSessionConnect

OnError

This function specifies how FAMOS responds in cases of an error occurring in the sequence execution.

Declaration:

```
OnError ( TxMode [, TxErrorMessage] [, TxErrorVariableName] [, ErrorValue] )
```

Parameter:

TxMode	Set whether/how to resume sequence execution after any subsequently occurring errors.
	"" : Default response. An error is displayed in a message box and the sequence execution is stopped. This corresponds to the behavior in older FAMOS-versions up to and including Version 7.4.
	"Default" : Same meaning as "" (default)
	"Return" : No error message box; skip to sequence end. Any existing caller (sequence/dialog/panel) is resumed. If applicable, the calling sequence can query the error using GetLastError().
	"ReturnFail" : No error message box; skip to sequence end. The error code is returned to the caller.
	"ResumeNext" : No error message box; resume at next command line. GetLastError() can be used to query the error code.
	"ResumeEnd" : No error message box; skip to next END-command. Specially well suited for resuming errors in FOR- or FOREACH-loops at the next loop iteration. If there is no subsequent END-command, then skip to the end of the sequence. GetLastError() can be used to query the error code.
TxErrorMessage	Error message to be outputted. If not specified (or for empty text), the original error message (syntax error of the Formula Interpreter or runtime error when running a function) is used. (optional)
TxErrorVariableName	When specified, then at fault condition, the value specified in the next parameter is written to the variable having this name. (optional)
ErrorValue	At fault condition, the variable specified in the previous parameter is set to the value specified here (allowed: single value or text). (optional)

Description:

While running sequences, errors can occur which are caused either by the Formula Interpreter ("Syntax error", e.g. unknown function name, wrong number of parameters) or when performing functions ("runtime error", e.g. incorrect type or illegal function parameter content). By default, these errors are indicated in a message box and futher execution is cancelled. This behavior can be reconfigured by means of OnError().

It is recommended to apply the advanced error modes defined here with care and after extensive testing. Unexpended, possible critical errors may become "latent". The skip through the sequence which occurs at fault condition suspends normal processing of the routine, so that for example any cleanup activity (deleting variables, closing files etc.) may not be performed. In general, it is more sensible to test any possible error causes and to respond accordingly to these in the routine. Thus for instance, the function VerifyVar() can be tested for whether the data type and a variable's structure meets the requirements of the subsequent analysis.

- When creating and testing sequences and sequence functions, in some cases it may be helpful to interrupt the routine whenever an error occurs, even if the current setting of OnError() specifies a different response. To achieve this, you can activate the option "Always break on error" under the menu item "Sequence"/"Execute".
- The settings specified here only apply to the current sequence, so they are not even "inherited" by sub-sequences or sequence functions called by this sequence. At the end of the current sequence, the previously valid settings become applicable again.
- Sequence functions with [TxMode]="ReturnFail": At the end of the sequence function, any return value is not generated, In/Out-parameters are not updated.
- Sequence functions with [TxMode]="Return": The sequence function responds as if no errorhad occurred - th ereurn value is generated, In/Out-parameters are updated. With sequence functions having this mode and return value, it is necessary to ensure that the Return-variable is created already before occurrence of potential errors and is set to a default value then, for example.
  For instance, this can be assured by generating the Return-variables at the beginning of the sequence or by using the Return-variables as [TxErrorVariable] in this function (see Example #3).
- When this function is called, the internal error memory queried by GetLastError() is deleted.
- In evey mode, the error message is also displayed in the FAMOS output window.

Examples:

A sequence loads all files from a folder in a loop. Each file is expected to contain a channel having the name 'channel1'; this channel is smoothed and saved in a different folder. If any file can not be loaded or contains no channel named 'channel1', the system initially ignores this and proceeds with the next file. At the end of the sequence, a note is outputted if appropriate indicating that not all files could be processed.

```
error = 0
inPath = "c:\inbox"
outPath = "c:\outbox"
filenames = FsGetFileNames(inPath, "*.dat", 0, 0, 0)
```

```
SetOption("Func.ErrorBoxes", "Yes")  ; File functions show error boxes!
OnError("ResumeEnd", "", "error", 1)
FOR i = 1 TO TxArrayGetSize(filenames)
   FileLoad(fileNames[i], "", 0)
   channel1 = Smo(channel1, 0.5)
   FileSave(outPath + FsSplitPath(fileNames[i], 4),"", 0, channel1)
   DELETE channel1
END
IF error
   BoxMessage("Error", "Could not process all files!", "!1")
END
```

FAMOS' function for calculating the square root, Sqrt(), returns 0 for a negative inut value and posts a corresponding warning. The following sequence function instead returns an error.

```
; Declaration: !Sqrt_Strict(Par) => Result
OnError("ReturnFail")
IF min(Par) < 0
   ThrowError("The parameter contains negative values!")
END
Result = Sqrt(par)
```

The following sequence function checks whether a data set is located within the tolerance band defined by 2 other data sets. If a function cannot be run due to inappropriate parameters, there is a special return value which indicates this.

```
; Declaration: !ToleranceBand(TestData, Lower, Upper) => Result
; Result = 0:  OK
; Result = 1:  top of tolerance band violated
; Result = 2:  bottom of tolerance band violated
; Result = 3:  top and bottom of tolerance band violated
; Result = -1: Error: Input data do not match (in terms of x-axis) or have events/segments

OnError("Return", "", "Result", -1)
Verify(Leng?(TestData)=Leng?(Lower) AND Leng?(TestData)=Leng?(Upper))
Verify(xdel?(TestData)=xdel?(Lower) AND xdel?(TestData)=xdel?(Upper))
Verify(xoff?(TestData)=xoff?(Lower) AND xoff?(TestData)=xoff?(Upper))

TooLarge = Max(TestData- Upper) > 0   ; e.g. error with segmented data
TooSmall = (Max(Lower - TestData) > 0) * 2
Result = TooLarge + TooSmall
```

A sequence calls 3 sequence functions in succesion, which each perform an independent evaluation of long duration. If any of the evaluations fails, the subsequent evaluations should still be performed anyway. At the end, a message box is displayed if not all evaluations were not successful. The output window provides the user with specific error mesages which indicate which evalations failed. (The sequence functions are written to include OnError("ReturnFail",...) in such a way that they return critical errors to the caller, but do not interrupt the routine.)

```
error =  0
OnError("ResumeNext", "Error in #1", "error", 1)
!Evaluation_1()

OnError("ResumeNext", "Error in #2", "error", 1)
!Evaluation_2()

OnError("ResumeNext", "Error in #3", "error", 1)
!Evaluation_3()

IF error
   BoxMessage("Error", "At least 1 evaluation could not be executed!", "!1")
END
```

See also:

Verify, VerifyVar, ThrowError, GetLastError

OR

Logical "OR"-operator

Declaration:

```
Operand1 OR Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

"OR"-operation on two numbers. The result is 1 if either of the operands is 1. Otherwise the result is 0.

The operator can be applied to single values or data sets. With data sets, the operation is applied data point by data point..

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/ segments); but then the respective other parameter must either have the exactly same structure (same segment length, event count and length), or be a single value.

Examples:

A data set's maximum value is determined and checked for constancy within a tolerance band.

```
Channel1 = ...
Maximum = max(Channel1)
IF Maximum < 21 OR Maximum > 34
   BoxMessage("Attention", "Value outside tolerance", "!1")
END
```

The disjunction operation is applied to two digital data sets. The result data set's value is 1 everywhere that at least one operand data set's value is 1.

```
Result = (DigChannel1 OR DigChannel2)
```

See also:

AND, NOT, XOR

OtrEncoderRevs01

Available in: Enterprise Edition and above (OrderTracking-Kit)

A rectangular signal sampled by an incremental encoder is used to find the rotation speed. The signal consists of the sampled rectangle signal which consist of only ones and zeroes.

Declaration:

```
OtrEncoderRevs01 ( Encoder signal, PulsePerRevolution, Zero ) -> Time-history of rotation speed
```

Parameter:

Encoder signal	The rectangular encoder signal
PulsePerRevolution	The amount of pulses (marks) per revolution. >= 1. This need not be an integer.
Zero	Always zero
Time-history of rotation speed	
Time-history of rotation speed	

Description:

The encoder is assumed to have turned by one increment each time the signal makes a transition from "Less than or equal to 0.0" to "Greater than 0.0". If the signal were to contain the sequence of values { 0, 0, 1, 1, 1, 0, 0, 1, 1, 0 }, then 2 pulse would have been detected. The function returns a rotation speed time-history, scaled in RPM. If the measurement data do not correspond to the appropriate levels, the signal can be re-shaped using, for example, the Schmitt-Trigger function stri().

This function's effects are not influenced by OtrTachoMode ().

Note: For other tacho signaltypes, only OtrTachoMode() and OtrTachoToSpeed() are suitable.

Examples:

Only a binary input (digital IN, taking only the values 1 or 0) is available for measuring a rotation speed. The rectangular signal of the encoder used, Rectangle_01, is sampled at a fixed sampling rate. The encoder emits 100 pulses per revolution and was sampled at the very fast rate of 0.1 ms. The rotation speed ranges between 10RPM and 2000RPM. The frequency of the rectangles ranges from 17Hz to 3.3kHz, which can just be sampled at a rate of 10kHz.

```
speed = OtrEncoderRevs01 ( Rectangle_01, 100, 0 )
```

In the following example, the encoder is sampled with an analog input. The captured voltage "VoltageInput" is 0.8V at LOW, and at HIGH 4.7V. A Schmitt-Trigger is used to condition the signal. The encoder has 128 markings.

```
Rectangle_01 = stri ( VoltageInput,  3, 2)
speed = OtrEncoderRevs01 ( Rectangle_01, 128, 0 )
```

See also:

OtrTachoMode, OtrTachoToSpeed, OtrTachoToDist

OtrFrequLine

Available in: Enterprise Edition and above (OrderTracking-Kit)

Frequency line calculation: The magnitude or phase of a periodic sinusoidal signal is determined.

Declaration:

```
OtrFrequLine ( Oscillation signal, PeriodLength, Number of Periods, Option ) -> Result
```

Parameter:

Oscillation signal	Signal with periodic oscillation
PeriodLength	Number of samples in a period, >=2
Number of Periods	Averaging over this many periods, >=1
Option	What to calculate
	0 : Determine magnitude (root mean square value)
	1 : Determine phase (in degrees)
Result	
Result	Calculated magnitude or phase time plot

Description:

The function determines the value of either magnitude or phase of the oscillation for each interval of length "Period length * PeriodAmount". Each of such intervals is exactly the size of "PeriodAmount" oscillations. The oscillation duration must be constant. Its length need not be an integer number of samples. It is specified by "PeriodLength", the period duration divided by the sampling interval.

The function determines one line of the discrete Fourier spectrum (DFT) with Rectangle-windowing.

If the signal contains other frequency components to a significant extent, a large number of periods should be specified in order to reduce their distorting influence. It may be worth band-pass filtering the signal beforehand.

The product of PeriodLength and PeriodAmount may not exceed 2e9 .

The phase is determined within the range -180 to +180 degrees. The value of the phase is 0 degrees for a cos-oscillation, -90 degrees for a sine oscillation.

If PeriodLength is not an integer number, the product of PeriodLength and PeriodAmount must be an integer number of samples.

Examples:

An oscillation signal is sampled over the rotation angle (vib_revs), so that all oscillation components up to the 8th order are included. the signal thus has 16 points per revolution. The phase of the 1st order is to be determined. One value for every 5 revolutions is desired.

```
Phase = OtrFrequLine ( vib_revs, 16, 5, 1 )
```

See also:

OtrOrderSpecFromFFT

Available in: Enterprise Edition and above (OrderTracking-Kit)

The order spectrum, referenced to the rotation speed, is calculated from the FFT-spectrum referenced to the rotation speed. The FFT-spectrum is given as a root mean square (RMS)-value spectrum.

Declaration:

```
OtrOrderSpecFromFFT ( FFT_Spectrum, Resolution, OrderMax, Calculation ) -> Result
```

Parameter:

FFT_Spectrum	FFT-Spectrum, referenced to rotation speed. A segmented data set is given, where every segment represents a spectrum.
Resolution	The resolution of the order spectrum, e.g. 0.1, if 0.1 orders is the spacing between order spectrum lines. The resolution is generally an integer fraction of 1.0, such as [1, 1/2, 1/3, 1/4, ..]
OrderMax	The highest order (line) in the order spectrum
Calculation	Calculation method by which a single order line is condensed from multiple FFT spectrum lines.
	0 : The maximum is used
	1 : Power remains unchanged
	2 : Sampling (interpolation)
Result	
Result	Order spectrum, referenced to rotation speed

Description:

The function finds an order spectrum. The order spectrum can be determined from the FFT spectrum (in approximation) if the rotation speed is known. In this context, a frequency of (rotation speed / 60.0) corresponds to the 1st order.

The data set containing the FFT spectrum is segmented. One segment equals one FFT. One rotation speed value is valid for each segment. The rotation speed should not be 0.0 and must be scaled in RPM. The data set's z-coordinate determines what the rotation speed is.

The function either stretches out or compresses the FFT-spectrum. Therefore it is not worth it to make the resolution as small as possible or the maximum order very high. That only increases the calculation time and memory requirements without providing better results.

The first line in the FFT-spectrum (that is the offset or DC-component) must be f = 0.0 Hz. The resulting order spectrum has the same DC-component.

The FFT spectrum should reflect the magnitude (RMS-value) of the frequency lines.

Calculation method

- With the calculation method "Maximum", the maximum of all the FFT-spectrum lines contributing to an order line is formed. With the calculation method "Power" the resulting order line is determined so that its power is the sum of the power in every contributing lines in the FFT-spectrum. This means that the spectral power density remains intact. With the calculation method "Sampling", all the spectral lines of the FFT-spectrum are considered as connected by straight lines (i.e. linear interpolation). This polygonal trace is sampled at the appropriate positions.

Note:

The algorithm is simple and inexact. Precise and rigorous calculation of the order spectrum is performed by the function OtrOrderSpec(), which uses a tracking filter and resampling over the rotation angle.

Examples:

First, the rotation speed-referenced FFT-spectrum is determined from the signals "Vibration" and "speed". Then, to save calculation time, the order spectrum is formed from the already determined FFT-spectrum. The biggest peaks in the FFT-spectrum are to remain intact (meaning: appear in the order spectrum with the same amplitude).

```
FFT_Spectrum = OtrRpmSpectrum ( Vibration, speed, 1000, 6000, 100, 1024, 0, 1, 0)
Resolution = 0.1 ; of the spectrum, i.e. 1/10 orders
OrderMax = 6.0 ; display lines up to here in the order spectrum.
Calc = 0 ; Default. Use maximum. Peak values remain intact
OrderSpec = OtrOrderSpecFromFFT( FFT_Spectrum, Resolution, OrderMax, Calc)
```

See also:

OtrRpmSpectrum, OtrOrderSpec

OtrOrderSpectrum

Available in: Enterprise Edition and above (OrderTracking-Kit)

Order spectrum related to the instantaneous rotation speed is determined from the time-history of vibration and tachometer signals. The desired rotation speed range is divided into classes of equal width.

Declaration:

```
OtrOrderSpectrum ( Vibration, Tachometer signal, RPM_Min, RPM_Max, RPM_class_width, Resolution, OrderMax,
WindowType, AveragingType ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Tachometer signal	Rotation speed signal interpreted as per OtrTachoMode(). Default is time-history of rotation speed, scaled in rpm's.
RPM_Min	Lower limit of desired rotation speed range. Scaled in rpm's.
RPM_Max	Upper limit of desired rotation speed range. Scaled in rpm's.
RPM_class_width	Width of rot.-speed class (interval). Scaled in rpm's.
Resolution	Resolution of the order spectrum. 0.1, if 0.1 orders is the distance between oder spectrum lines. The resolution must be an integral fraction of 1.0, thus: [1, 1/2, 1/3, 1/4, ..]
OrderMax	The highest order (line) displayed in the order spectrum.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
AveragingType	The method of summarizing all spectra of a single rpm-class.
	0 : arithmet. mean
	1 : arithmet. mean, 50% overlap
	2 : arithmet. mean, 75% overlap
	3 : Maximum
	4 : Maximum, 50% overlap
	5 : Maximum, 75% overlap
	6 : Minimum
	7 : Minimum, 50% overlap
	8 : Minimum, 75% overlap
	9 : first
	10 : arithmet. mean, strict
	11 : arithmet. mean, strict, 50% overlap
	12 : arithmet. mean, strict, 75% overlap
	13 : Maximum, strict
	14 : Maximum, strict, 50% overlap
	15 : Maximum, strict, 75% overlap
	16 : Minimum, strict
	17 : Minimum, strict, 50% overlap
	18 : Minimum, strict, 75% overlap
	19 : first, strict

	20 : RMS
	21 : RMS, 50% overlap
	22 : RMS, 75% overlap
	23 : RMS, 90% overlap
	24 : RMS, 95% overlap
	25 : RMS, 98% overlap
	26 : RMS, 99% overlap
Result	
Result	Order spectrum in relation to rotation speed

Description:

For sampling, the absolute value of the RPMs is used; for assignment to an RPM-class, the original RPM value.

The inverse of the resolution states how many revolutions contribute to an order spectrum. For instance, with a resolution of 0.1 each spectrum is determined from 10 revolutions.

The spectral lines are stated as RMS (root mean square) values.

Since the FFT is calculated internally from a larger number of data (a power of two), some spectral lines are cut off. The visible spectrum thus no longer reflects the whole power of the signal. A rectangular window for the FFT is highly recommended, if high resolution of the frequency is desired.

The averaging is based only on the amplitude spectrum.

Strict performance of averaging:

- A spectrum is only calculated if the instantaneous RPM signal remains within the bounds of one class for a set time interval. Thus, the RPM may only change slowly. If you wish to compute spectra when the RPM signal changes quickly, select a correspondingly large RPM_class_width!

Non-strict performance of averaging:

- Class association is determined by the mean RPM-value during the set time interval for spectrum calculation. Precision is sacrificed if the RPM signal changes quickly.

A tracking Butterworth low-pass filter is used as the anti-aliasing filter. The signal thus filtered is sampled over the rotation angle. Then an FFT is calculated. This procedure causes the signal to temporarily contain much higher order lines than in the spectrum which is ultimately returned. The AAF's 3dB-order is far behind the order lines in the resulting spectrum.

The function works well for:

- OrderMax < 10 / (SamplingTime_Vibration * max (rotation speed)),

where SamplingTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value.

- The instantaneous rotation speed may change only slowly. The speed my not fall far below 1% of capacity. If either the rotation speed or OrderMax is much too large, the calculation time increases very much without providing better results.
- The rotation angle is determined by integration (summing up) of the tachometer signal. If the tacho signal is itself a rotation speed signal (and not an impulse signal), the rotation speed must be available at good precision.
- The Tacho signal's sampling interval must be either equal to the vibration's sampling time or an integer multiple of it.
- If no spectrum is found for a certain RPM-class, this spectrum is filled with zeroes.
- The result is a segmented waveform. Each segment is an order spectrum.
- Before calling this function, OtrTachoMode() should be called at least once.

Examples:

The order spectrum related to the rotation speed is to be calculated from the time-history of the rotation speed and of the vibration, vib. The time-based signals are sampled every 0.2 ms.

```
OtrTachoMode ( 0, 0, 0, 0)
rpm_Min = 1000.0 ; minimum of rotation speed range
rpm_Max = 6000.0 ; maximum of rotation speed range
rpm_Delta = 100.0 ; width of individual rotation speed classes
Resolution = 0.1 ; spectrum resolution, every 1/10 order is shown, calculated for 10 revolutions
OrderMax = 6.0 ; display spectral lines up to this order.
Windowtype = 0 ; 0 default (rectangle)
AvgType = 0 ; 0 (arithmet. mean)
OSpectrum = OtrOrderSpectrum ( vib, speed, rpm_Min, rpm_Max, rpm_Delta, Resolution, OrderMax, Windowtype, AvgType )
```

Here we have: OrderMax = 6.0 < 10 / (0.0002 * 6000) = 8.3

OtrTachoMode() is first used to set the tacho signal type used.

See also:

OtrTachoMode, OtrRpmOrder, OtrRpmSpectrum, OtrTimeOrderSpectrum, OtrOrderSpecFromFFT

OtrResample

Available in: Enterprise Edition and above (OrderTracking-Kit)

Sampling of a vibration signal over the rotation angle, given the tacho signal.

Declaration:

```
OtrResample ( Vibration, Tachometer signal, OrderMax, Interpolation ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Tachometer signal	Rotation speed signal interpreted as per OtrTachoMode(). Default is time-history of rotation speed, scaled in rpm's.
OrderMax	Maximum order line in the result
Interpolation	Specifies how intermediate values are found
	0 : Constant interpolation (stair-steps)
	1 : Linear interpolation
	2 : Cubic interpolation
Result	
Result	Vibration signal sampled over the rotation angle

Description:

The resulting angle-referenced signal is scaled in the x-direction by recording revolutions made. The x-coordinate starts at 0, is 0.5 after half a revolution, after a whole revolution 1.0, after two it is 2.0 etc.

- Sampling interval of result: 0.5 / OrderMax
- The rotation speed's absolute value is used.
- The function has no anti-aliasing filter. A preceding call of OtrTrackingLowPass() is necessary.

The function works well for:

OrderMax <= 30 / (SampleTime_Vibration * max (rotation speed)),

where SamplingTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value.

- The rotation speed should change only slowly and should not fall far below 1% of capacity. If the rotation speed becomes far too high, the calculation time required increases enormously without providing better results.
- The rotation angle is obtained by integration (summation) of the tachometer signal. If the tacho signal itself is a rotation speed time-history (not an impulse signal), the rotation speed must be available at good precision. Note that even an only slightly incorrect or imprecise rotation speed signal when integrated can lead to a substantial deviation of the angle, thus in turn causing a creeping phase deviation.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- Before calling this function, OtrTachoMode() should be called at least once.

Examples:

A vibration signal "vib" is sampled every 0.5 ms. The rotation speed can reach 3000 RPM. The vibration is to be sampled over the rotation angle. All components up to the 15th order are to be included.

```
OtrTachoMode ( 0, 0, 0, 0)
tlp = OtrTrackingLowPass ( vib, speed, 8.0, 4 ) ; anti-aliasing filter
omax = 15.0
res = OtrResample ( tlp, speed, omax, 2 )
```

We have: OrderMax = 15.0 <= 30 / (0.0005 * 3000) = 20.0.

The result has a resolution of 0.5 / OrderMax = 0.0333 revolutions. There are 30 samples per revolution. The anti-aliasing filter is designed to dampen by 3 dB at the 8th order and by 20 dB at the 14.4th order. At the 6.1th order the amplitude error is already less than 5%. A higher order interpolation is used.

OtrTachoMode() is first used to set the tacho signal type used.

See also:

OtrResampleAAF, OtrTrackingLowPass, OtrTachoMode

OtrResampleAAF

Available in: Enterprise Edition and above (OrderTracking-Kit)

Sampling of a vibration signal over the rotation angle, given the tacho signal and using a tracking anti-aliasing filter.

Declaration:

```
OtrResampleAAF ( Vibration, Tachometer signal, OrderMax, Order3dB, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Tachometer signal	Rotation speed signal interpreted as per OtrTachoMode(). Default is time-history of rotation speed, scaled in rpm's.
OrderMax	Maximum order line in the result
Order3dB	Order (line), for which the low-pass filter dampens the signal by 3dB.
FilterOrder	The filter order of the low-pass filter (1 ... 10)
Result	
Result	Vibration signal sampled over the rotation angle

Description:

The resulting angle-referenced signal is scaled in the x-direction by recording revolutions made. The x-coordinate starts at 0, is 0.5 after half a revolution, after a whole revolution 1.0, after two it is 2.0 etc.

- Sampling interval of result: 0.5 / OrderMax
- The rotation speed's absolute value is used.
- A Butterworth low-pass filter is used as the anti-aliasing filter.
- 3rd order interpolation is used to find intermediate values.

The function works well for:

OrderMax < 24 / (SampleTime_Vibration * max (rotation speed)),

where SampleTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) the maximum occurring RPM-value. Note that the high-frequency components in the resulting signal are already strongly dampened.

Order3dB << OrderMax

<< indicates: much smaller.

- The rotation speed should change only slowly and should not fall far below 1% of capacity. If the rotation speed becomes far too high, the calculation time required increases enormously without providing better results.
- The rotation angle is obtained by integration (summation) of the tachometer signal. If the tacho signal itself is a rotation speed time-history (not an impulse signal), the rotation speed must be available at good precision. Note that even an only slightly incorrect or imprecise rotation speed signal when integrated can lead to a substantial deviation of the angle, thus in turn causing a creeping phase deviation.
- The upper cutoff frequency of the low-pass filter must always be far below half of the vibration signals sampling rate. Above approx. (0.4 * sampling rate), filtering cannot be performed.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- The function can be replicated by calling OtrTrackingLowPass() and OtrResample().
- Before calling this function, OtrTachoMode() should be called at least once.

Examples:

A vibration signal is sampled every 1 ms. The rotation speed can reach 4500 RPM. The vibration is to be sampled over the rotation angle. All components up to the 5th order are to be included.

```
OtrTachoMode ( 0, 0, 0, 0)
omax = 5.0
o3dB = 2.7 ; The low-pass dampens by 3 dB for this order
fo = 8 ; an 8th order low-pass filter is used
res = OtrResampleAAF ( vib, speed, omax, o3dB, fo )
```

We have: OrderMax = 5.0 <= 24 / (0.001 * 4500) = 5.33

The result has a resolution of 0.5 / OrderMax = 0.1 revolutions. There are 10 samples per revolution. the anti-aliasing filter is designed to dampen the 2.7th order by 3 dB, and by 60 dB at the 5th order. At the 2.3th order the amplitude error is already below 5%.

OtrTachoMode() is first used to set the tacho signal type used.

See also:

OtrTrackingLowPass, OtrResample, OtrTachoMode

OtrRpmOrder

Available in: Enterprise Edition and above (OrderTracking-Kit)

Finds the RMS (root mean square) value of an order line related to the rotation speed. The desired rotation speed range is divided into classes of equal width. Calculation is preformed with the help of a tracking band-pass filter.

Declaration:

```
OtrRpmOrder ( Vibration, Rotation speed, RPM_Min, RPM_Max, RPM_class_width, OrderCenter, WidthPercent,
FilterOrder, Interpolation ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
RPM_Min	Lower limit of desired rotation speed range. Scaled in rpm's.
RPM_Max	Upper limit of desired rotation speed range. Scaled in rpm's.
RPM_class_width	Width of rot.-speed class (interval). Scaled in rpm's.
OrderCenter	Order (line) at which the band-pass' center frequency is located.
WidthPercent	Overall width in percent. Recommended 10%...100%. Input range [0.01 ... 10000.0]. For example, at 30% width the ratio of the upper to the lower cutoff frequency of the bandpass is 1.30.
FilterOrder	The bandpass filter's order (2, 4, 6, 8, 10)
Interpolation	How is the result interpolated?
	0 : No interpolation
	1 : Constant interpolation (centered at measured values)
	2 : Linear interpolation
Result	
Result	RMS value of an order line related to the rotation speed

Description:

The rotation speed range starts at RPM_Min, the resolution is always RPM_class_width. The parameter RPM_Max is only used to determine the amount of values in the result.

Every RPM-class (of width RPM_class_width) must contain a sufficient number of data points in the vibration signal. If there are no values at all, the result in that RPM-class remains zero. Only if a nonzero interpolation is selected the unfilled RPM-classes are filled by interpolation of adjacent values. If interpolation is selected, unfilled classes at the ends of the range are also filled by level linear extension.

The internally selected center frequency for the filter is

center frequency = OrderCenter* (Current_RPM / 60)

The upper cutoff frequency is above the center frequency and depends on the filter's width. The upper cutoff frequency is always significantly below half of the vibration signal's sampling frequency. Above approx. 1 / (0.48 * sampling interval), no band-pass filtering can be performed.

The function works well for:

OrderMax << 28 / (SampleTime_Vibration * max (rotation speed)),

where SamplingTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value.

OrderMax = OrderCenter * sqrt (1 + WidthPercent / 100)

<< indicates that the order specified should be much smaller.

- The rotation speed should change only slowly.
- The resulting center frequency should not fall much below 0.1 percent of capacity. If either the rotation speed or OrderCenter becomes much too big, the calculation time increases greatly without improving the results.
- If the resulting center frequency falls too low, the result can become less precise.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- Note that band-passes need a certain amount of time for transients to subside. This time increases markedly for narrow-band filters. A band-pass of width 1% is very narrow in this sense. A width of 25% corresponds to a 1/3 octave filter, a width of 100% to a 1/1 octave filter.

Examples:

We wish to use the 2.5th order line to obtain the RMS-value of this order line in relation to the rotation speed. We are given the vibration "vib" with a sampling interval of 0.0005 ms and the rotation speed "speed".

```
rpm_Min = 1000.0 ; minimum of rotation speed range
rpm_Max = 6000.0 ; maximum of rotation speed range
rpm_Delta = 100.0 ; width of individual rotation speed classes
om = 2.5 ;  the 2.5th order is selected.
width = 30 ; 30% overall width
fo = 6 ; a 6th order band-pass filter is used
Interpolation = 0 ; 0 default (none)
OLine = OtrRpmOrder ( vib, speed, rpm_Min, rpm_Max, rpm_Delta, om, width, fo, Interpolation )
```

We have: OrderMax = 2.5 * sqrt (1 + 30 / 100) = 2.9

and thus: OrderMax = 2.9 << 28 / (0.0005 * 6000) = 9.3

See also:

OtrRpmPresentation

Available in: Enterprise Edition and above (OrderTracking-Kit)

From the time-histories of a signal and the rotation speed, the signal referenced to the rotation speed is constructed. The desired RPM-range is divided into evenly spaced classes.

Declaration:

```
OtrRpmPresentation ( Vibration, Rotation speed, RPM_Min, RPM_Max, RPM_class_width, Calculation, Interpolation,
EndOpen ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed.
RPM_Min	Lower limit of desired RPM-range
RPM_Max	Upper limit of desired RPM-range.
RPM_class_width	Width of an RPM-class (interval).
Calculation	How all the values from a single RPM-class are summarized
	0 : Root mean square (RMS) value (default)
	1 : Arithmetical mean
	2 : Minimum
	3 : Maximum
	4 : Minimum of absolute values
	5 : Maximum of absolute values
	6 : Mean of absolute values
	7 : Sum
	8 : Standard deviation, sqrt (1/N * ...)
	9 : First value
	10 : Last value
	11 : Number of vibration signal data points
	12 : Number of revolutions ("Vibration"-parameter must also contain rotation speed data! Rotation speed scaled in RPM!)
Interpolation	How is the result interpolated?
	0 : No interpolation
	1 : Constant interpolation (centered at measured values)
	2 : Linear interpolation
EndOpen	Are the outer ends of the range considered open?
	0 : Closed ends. Default. If the rotation speed is outside the desired range, vibration values are ignored.
	1 : Open ends. Only for histogram applications. Everything < RPM_Min goes into the same class as RPM_Min, everything > RPM_Max goes into the same class as RPM_Max
Result	
Result	Signal plotted in reference to rotation speed

Description:

The RPM-range always starts at RPM_Min, the resolution is always RPM_Class_width. The specification of RPM_Max is only used to find the number of values in the result. The result partly has the character of a histogram, so that a bar graph or stair-step display is often useful.

The vibration and rotation speed may either both be time-based or angle-based.

The rotation speed need not be scaled in RPM. However, the rotation speed, RPM_Min, RPM_Max and RPM_Class_width must all have the same scaling (y-unit).

Every RPM-class (of width RPM_class_width) must contain a sufficient number of data points in the vibration signal. If there are no values at all, the result in that RPM-class remains zero. Only if a nonzero interpolation is selected the unfilled RPM-classes are filled by interpolation of adjacent values. If interpolation is selected, unfilled classes at the ends of the range are also filled by level linear extension.

Examples:

From the time-history of an order line we wish to obtain a representation of this order line's rms value referenced to the rotation speed. We are given the vibration signal "vib" and the rotation signal "speed". Time-history of the 1.5th order line from the vibration

```
tbp = OtrTrackingBandPass ( vib, speed, 1.5, 30, 4 )
rpm_Min = 1000.0 ; minimum of rotation speed range
rpm_Max = 6000.0 ; maximum of rotation speed range
rpm_Delta = 100.0  ; width of the RPM-classes
Calc = 0 ; 0 = rms value
Interpolation = 0 ; 0 default (none)
EndOpen = 0 ; 0 Default (closed)
Order_rpm = OtrRpmPresentation ( tbp, speed, rpm_Min, rpm_Max, rpm_Delta, Calc, Interpolation, EndOpen)
```

See also:

OtrRpmPresentVector

OtrRpmPresentFast

Available in: Enterprise Edition and above (OrderTracking-Kit)

From the time-histories of a spectrum and the rotation speed, the spectrum is referenced to the rotation speed. Especially for fast run-ups and run-downs. The result is generally not evenly distributed along the RPM-axis.

Declaration:

```
OtrRpmPresentFast ( Spectrum sequence, Rotation speed, RPM-change, RPM-resolution, Calculation ) -> Result
```

Parameter:

Spectrum sequence	Spectrum time-history
Rotation speed	Time-history of (rapidly changing) rotation speed.
RPM-change	Calculation only takes place if the rotation speed changes in the desired direction. Meaning: spectra are only counted either for rising or for falling RPMs.
	1 : Only rising speed
	-1 : Only falling speed
RPM-resolution	If the RPM-values of consecutive spectra are very close to each other, the different spectra can melt into one, and in graphical display they would be indistinguishable. The RPM resolution specifies how far a spectrum's rotation speed must deviate before a separate one is made.
Calculation	How are spectra of almost equal rotation speed summarized? If the rotation speeds of adjacent spectra are not further apart than specified in the parameter RPM-resolution, they are summarized in one of the following ways. This returns a representative spectrum.
	0 : Root mean square (RMS) value (default)
	1 : Arithmetical mean
	2 : Minimum
	3 : Maximum
	4 : Minimum of absolute values
	5 : Maximum of absolute values
	6 : Mean of absolute values
	7 : Sum
	9 : First value
	10 : Last value
Result	
Result	Spectra referenced to the rotation speed

Description:

The function is for analysis of a run-up or run-down of a machine.

As input data, the function takes a sequence of spectra (the time-history of a spectrum) and a matching sequence of RPM-values (The rotation speed's time-history). An RPM value is thus associated with each spectrum. Each spectrum-RPM pair is written to the result-matrix. The result-matrix contains a spectrum for every RPM.

The sequence of input data is a segmented data set where each segment represents a spectrum. Such data can be produced using, for instance, the functions belonging to the Spectrum Kit, which return a time-sequence of frequency spectra. A separate data set of the corresponding rotation speed values must also exist. The rotation speed is resampled appropriately, e.g. by a moving mean value with the function MvMean().

The sampling interval for the rotation speed signal is ignored. The function assumes that every spectrum pairs up with one RPM-value.

The result is a data set with Events (a matrix with additional properties). Each event represents a spectrum and is valid for a particular rotation speed. One rotation speed is assigned to each event. The events are always sorted in order of increasing rotation speed.

The rotation speed need not be scaled in RPM. But the rotation speed and its resolution must take the same scaling (y-unit).

The rotation speed resolution can also be 0.0. For fast run-ups (or fast run-downs) of a machine, the RPM resolution is unimportant. In that case 0.0 is the best choice. Only in cases where the rotation speed changes very slowly or remains constant does the RPM-resolution play a role. There are rotation speeds so close together that they can be considered as one in terms of measurement precision or for post analysis purposes. In a run-up from 1000 RPM to 6000 RPM, it isn't relevant for distinguishing spectra whether the rotation speed is 3345 RPM or 3350 RPM. In that case the resolution can be set to 5 RPM. If the resolution is very small, slow rot. speed changes can lead to data glut.

If the rotation speed doesn't change from one spectrum to the next (or is less than the specified RPM-resolution), the spectra are summarized. That means that a better or more representative spectrum can be found for such a rotation speed, for instance, by means of averaging.

The parameter "Calculation" is only important if the change in rotation speed from one spectrum to the next is less than the resolution. The calculation determines how different individual spectra of the same rotation speed are summarized. The method specified may be, for instance, a mean value or maximum.

For slow changing rot. speed, the function OtrRpmPresentVector() is to be preferred. There, the matrix is subdivided evenly along the RPM-axis, which usually aids in display and analysis.

Run-up or run-down: The parameter RPM-change specifies whether to take either rising or falling rotation speed changes into account. For example, if rising RPM is chosen and then the RPM falls, the resulting spectra are ignored and don't enter the results. For instance, if during a run-up the RPM dips shortly before resuming to rise, only the spectra from the dip are ignored. When the RPM-values are again above the last value in the rising region, the spectra count once more. The results are filled only once and in one direction. If, for instance, another run-up occurs it is ignored. In such cases the function OtrRpmPresentVector() is to be preferred.

Examples:

Using a function from the Spectrum Kit, the time-history of a spectrum derived from the channel "Vibration" is determined. This channel is sampled at intervals of 1ms. We wish to find spectra of length 1000 points with 75% overlap. The rotation speed is simultaneously sampled at the same rate. We take a measurement of a fast run-up of a motor.

```
Spectra = AmpSpectrumRMS ( Vibration, 1000, 0, 75, 1, 0, 0 ) ; from the Spectrum Kit
speed1 = MvMean ( speed,  1.0, 0.25 )
UpDown = 1 ; 1 run-up, -1 run-down
NResolution =0.0  ; RPM-resolution
Calc = 1 ; 1 = mean
RpmSpectrum = OtrRpmPresentFast (Spectra, speed1, UpDown, NResolution, Calc)
```

After every 250 values in the vibration channel, a spectrum is found, in other words after 250ms. Thus the rotation speed must be acquired every 250ms. This is to be averaged over 1000ms.

See also:

OtrRpmPresentVector

OtrRpmPresentVector

Available in: Enterprise Edition and above (OrderTracking-Kit)

From the time-histories of a spectrum and the rotation speed, the spectrum referenced to the rotation speed is constructed. The desired RPM-range is divided into evenly spaced classes.

Declaration:

```
OtrRpmPresentVector ( Spectrum sequence, Rotation speed, RPM_Min, RPM_Max, RPM_class_width, Calculation,
Interpolation, EndOpen, RPM-change ) -> Result
```

Parameter:

Spectrum sequence	Spectrum time-history
Rotation speed	Time-history of rotation speed
RPM_Min	Lower limit of desired RPM-range
RPM_Max	Upper limit of desired RPM-range.
RPM_class_width	Width of an RPM-class (interval).
Calculation	How all the values from a single RPM-class are summarized
	0 : Root mean square (RMS) value (default)
	1 : Arithmetical mean
	2 : Minimum
	3 : Maximum
	4 : Minimum of absolute values
	5 : Maximum of absolute values
	6 : Mean of absolute values
	7 : Sum
	9 : First value
	10 : Last value
Interpolation	How is the result interpolated?
	0 : No interpolation
	1 : Constant interpolation (centered at measured values)
	2 : Linear interpolation
EndOpen	Are the outer ends of the range considered open?
	0 : Closed ends. Default. If the rotation speed is outside the desired range, vibration values are ignored.
	1 : Open ends. Only for histogram applications. Everything < RPM_Min goes into the same class as RPM_Min, everything > RPM_Max goes into the same class as RPM_Max
RPM-change	Calculation only proceeds if the rotation speed changes in the correct direction; either rising or falling
	0 : No matter
	1 : Only rising speed
	-1 : Only falling speed
Result	
Result	Spectra referenced to the rotation speed

Description:

As input data, the function takes a sequence of spectra (the time-history of a spectrum) and a matching sequence of RPM-values (The rotation speed's time-history). An RPM value is thus associated with each spectrum. Each spectrum-RPM pair is written to the result-matrix. The result-matrix contains a spectrum for every RPM-range.

The sequence of input data is a segmented data set where each segment represents a spectrum. Such data can be produced using, for instance, the functions belonging to the Spectrum Kit, which return a time-sequence of frequency spectra. A separate data set of the corresponding rotation speed values must also exist. The rotation speed is resampled appropriately, e.g. by a moving mean value with the function MvMean().

The sampling interval for the rotation speed signal is ignored. The function assumes that every spectrum pairs up with one RPM-value.

The RPM-range always starts at RPM_Min, the resolution is always RPM_Class_width. The specification of RPM_Max is used only to determine the number of values in the result. The result partly has the character of a histogram, so that a bar graph or stair-step display is often useful.

The result is a segmented data set (matrix). Each segment represents one spectrum and is valid for a particular rotation speed.

The rotation speed need not be scaled in RPM. However, the rotation speed, RPM_Min, RPM_Max and RPM_Class_width must all have the same scaling (y-unit).

Every RPM-class (of width RPM_class_width) must contain at least one spectrum. If there is no spectrum at all, the result in that RPM-class remains zero. Only if a nonzero interpolation is selected the unfilled RPM-classes are filled by interpolation of adjacent values. If interpolation is selected, unfilled classes at the ends of the range are also filled by level linear extension.

For very fast run-ups and coast-downs, this function is not well adapted since many RPM-regions of the matrix are unfilled or else a huge class width must be used for the rotation speed. In such cases, the function OtrRpmPresentFast() is preferable.

Examples:

The time-history of the spectrum from the vibration signal "Vibration" is determined with the help of a Spectrum Kit function. The vibration channel as a sampling interval of 1ms. Spectra with a of length of 1000 points with 50% overlap are to be computed. A rotation speed signal "speed" was captured simultaneously at a sampling interval of 10ms.

```
Spectra = AmpSpectrumRMS ( Vibration, 1000, 0, 50, 1, 0 ) ; from the Spectrum Kit
speed1 = MvMean ( speed,  1.0, 0.5 )
rpm_Min = 1000.0 ; minimum of rotation speed range
rpm_Max = 6000.0 ; maximum of rotation speed range
rpm_Delta = 100.0  ; width of the RPM-classes
Calc = 0 ; 0 = rms value
Interpolation = 0 ; 0 default (none)
Bounds = 0 ; 0 Default (closed)
UpDown = 0 ; Default (no matter what change)
RpmSpectrum = OtrRpmPresentVector (Spectra, speed1, rpm_Min, rpm_Max, rpm_Delta, Calc, Interpolation, Bounds, UpDown)
```

A spectrum is computed after every 500 data points in the vibration channel, in other words, after 500ms. Thus, a rotation speed is needed every 500ms. This is to be averaged over 1000ms.

See also:

OtrRpmPresentation, OtrRpmPresentFast

OtrRpmSpectrum

Available in: Enterprise Edition and above (OrderTracking-Kit)

The FFT spectrum (RMS-values!) referenced to a rotation speed is determined from the time-histories of the vibration and the rotation speed. The desired RPM-range is divided into equally spaced classes.

Declaration:

```
OtrRpmSpectrum ( Vibration, Rotation speed, RPM_Min, RPM_Max, RPM_class_width, WindowWidth, WindowType,
AveragingType [, Base2] ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
RPM_Min	Lower limit of desired rotation speed range. Scaled in rpm's.
RPM_Max	Upper limit of desired rotation speed range. Scaled in rpm's.
RPM_class_width	Width of rot.-speed class (interval). Scaled in rpm's.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
AveragingType	The method of summarizing all spectra of a single rpm-class.
	0 : arithmet. mean
	1 : arithmet. mean, 50% overlap
	2 : arithmet. mean, 75% overlap
	3 : Maximum
	4 : Maximum, 50% overlap
	5 : Maximum, 75% overlap
	6 : Minimum
	7 : Minimum, 50% overlap
	8 : Minimum, 75% overlap
	9 : first
	10 : arithmet. mean, strict
	11 : arithmet. mean, strict, 50% overlap
	12 : arithmet. mean, strict, 75% overlap
	13 : Maximum, strict
	14 : Maximum, strict, 50% overlap
	15 : Maximum, strict, 75% overlap
	16 : Minimum, strict
	17 : Minimum, strict, 50% overlap
	18 : Minimum, strict, 75% overlap
	19 : first, strict
	20 : RMS

	21 : RMS, 50% overlap
	22 : RMS, 75% overlap
	23 : RMS, 90% overlap
	24 : RMS, 95% overlap
	25 : RMS, 98% overlap
	26 : RMS, 99% overlap
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	FFT spectrum referenced to rotation speed

Description:

The window width need not be a power of 2. A window width of 500 or 1000, for example, would produce proper frequency line spacing.

- The spectral lines are stated as RMS (root mean square) values.
- The averaging is based only on the amplitude spectrum.

Strict performance of averaging:

- A spectrum is only calculated if the instantaneous RPM signal remains within the bounds of one class for a set time interval. Thus, the RPM may only change slowly. If you wish to compute spectra when the RPM signal changes quickly, select a correspondingly large RPM_class_width!

Non-strict performance of averaging:

- Class association is determined by the mean RPM-value during the set time interval for spectrum calculation. Precision is sacrificed if the RPM signal changes quickly.

The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.

- If no spectrum is found for a certain RPM-class, this spectrum is filled with zeroes.
- The result is a segmented data set. Each segment is a spectrum.

Examples:

From the time history of a vibration "vib" and the rotation speed "speed", the spectrum referenced to the rotation speed is to be found.

```
rpm_Min = 1000.0 ; minimum of rotation speed range
rpm_Max = 6000.0 ; maximum of rotation speed range
rpm_Delta = 100.0 ; width of individual rotation speed classes
WindowSize = 1000 ; Width of window for FFT, in number of data points
WindowType = 0; 0 Rectangle
AvgType = 0 ; 0 (arithmet. mean)
RpmSpectrum = OtrRpmSpectrum ( vib, speed, rpm_Min, rpm_Max, rpm_Delta, WindowSize, WindowType, AvgType, 0 )
```

At a sampling interval of 0.5ms and a window width of 1000 points, a spectrum with the frequency line spacing of 2 Hz is calculated.

See also:

OtrRpmPresentVector, OtrRpmPresentFast, OtrRpmSpectrumFast

OtrRpmSpectrumFast

Available in: Enterprise Edition and above (OrderTracking-Kit)

The FFT-spectrum (root mean square values!), referenced to rotation speed, is determined from the time-histories of a vibration and rotation speed. Especially suited to fast run-ups and run-downs. The result is generally not equally distributed along the RPM-axis.

Declaration:

```
OtrRpmSpectrumFast ( Vibration, Rotation speed, WindowWidth, WindowType, Overlapping, RPM-change, RPM-resolution,
Calculation [, Base2] ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of (rapidly changing) rotation speed.
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	Overlap in %, e.g. 0.0, 50 or 75. The individual time windows, from which the spectra are taken, overlap each other by this amount.
	0 : no overlapping
	50 : 50% overlap, or 1/2
	75 : 75% overlap, or 3/4
RPM-change	Calculation only takes place if the rotation speed changes in the desired direction. Meaning: spectra are only counted either for rising or for falling RPMs.
	1 : Only rising speed
	-1 : Only falling speed
RPM-resolution	If the RPM-values of consecutive spectra are very close to each other, the different spectra can melt into one, and in graphical display they would be indistinguishable. The RPM resolution specifies how far a spectrum's rotation speed must deviate before a separate one is made.
Calculation	How are spectra of almost equal rotation speed summarized? If the rotation speeds of adjacent spectra are not further apart than specified in the parameter RPM-resolution, they are summarized in one of the following ways. This returns a representative spectrum.
	0 : Root mean square (RMS) value (default)
	1 : Arithmetical mean
	2 : Minimum
	3 : Maximum
	4 : Minimum of absolute values
	5 : Maximum of absolute values
	6 : Mean of absolute values
	7 : Sum
	9 : First value
	10 : Last value
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	

Result	Spectra referenced to the rotation speed

Description:

The function is for analysis of a (rapid) run-up or run-down of a machine.

The function's input data are the time-histories of a vibration signal and a rotation speed signal. The spectra are determined from the vibration signal. For each spectrum, the average rotation speed is found. A matrix is formed from the spectrum-rotation speed pairs. This matrix represents the spectrum referenced to the rotation speed.

The result is a data set with Events (a matrix with additional properties). Each event represents a spectrum and is valid for a particular rotation speed. One rotation speed is assigned to each event. The events are always sorted in order of increasing rotation speed.

The rotation speed need not be scaled in RPM. But the rotation speed and its resolution must take the same scaling (y-unit).

The rotation speed resolution can also be 0.0. For fast run-ups (or fast run-downs) of a machine, the RPM resolution is unimportant. In that case 0.0 is the best choice. Only in cases where the rotation speed changes very slowly or remains constant does the RPM-resolution play a role. There are rotation speeds so close together that they can be considered as one in terms of measurement precision or for post analysis purposes. In a run-up from 1000 RPM to 6000 RPM, it isn't relevant for distinguishing spectra whether the rotation speed is 3345 RPM or 3350 RPM. In that case the resolution can be set to 5 RPM. If the resolution is very small, slow rot. speed changes can lead to data glut.

If the rotation speed doesn't change from one spectrum to the next (or is less than the specified RPM-resolution), the spectra are summarized. That means that a better or more representative spectrum can be found for such a rotation speed, for instance, by means of averaging.

The parameter "Calculation" is only important if the change in rotation speed from one spectrum to the next is less than the resolution. The calculation determines how different individual spectra of the same rotation speed are summarized. The method specified may be, for instance, a mean value or maximum.

For slow changing rot. speed, the function OtrRpmSpectrum() is to be preferred. There, the matrix is subdivided evenly along the RPM-axis, which usually aids in display and analysis.

Run-up or run-down:

- The parameter RPM-change specifies whether to take either rising or falling rotation speed changes into account. For example, if rising RPM is chosen and then the RPM falls, the resulting spectra are ignored and don't enter the results. For instance, if during a run-up the RPM dips shortly before resuming to rise, only the spectra from the dip are ignored. When the RPM-values are again above the last value in the rising region, the spectra count once more. The results are filled only once and in one direction. If, for instance, another run-up occurs it is ignored. In such cases the function OtrRpmSpectrum() is to be preferred.

The channels Vibration and Speed may also have different sampling rates.

Examples:

The FFT spectrum of a fast motor run-up is to be determined in reference to the rotation speed. The measurement signals "Vibration" and "speed" (rotation speed) are given. Calculate FFTs of width 1000 points.

```
WindowSize = 1000 ; Width of window for FFT, in number of data points
WindowType = 0; 0 Rectangle
Overlap = 75 ; Overlap in %
UpDown = 1 ; 1 run-up, -1 run-down
NResolution =0.0  ; RPM-resolution
Calc = 1 ; 1 = mean
FFT_Spectrum = OtrRpmSpectrumFast ( Vibration, speed, WindowSize, WindowType, Overlap, UpDown, NResolution, Calc, 0)
```

A substantial time window overlap is selected for the FFT. This provides a larger number of spectra even for rapid run-ups.

See also:

OtrRpmPresentVector, OtrRpmPresentFast

OtrRpmThirds

Available in: Enterprise Edition and above (OrderTracking-Kit)

From the time-histories of a vibration and rotation speed, the 1/3 octave spectrum referenced to the rotation speed is determined. The desired RPM-range is divided into equally spaced classes.

Declaration:

```
OtrRpmThirds ( Vibration, Rotation speed, RPM_Min, RPM_Max, RPM_class_width, f1, f2, Frequency weighting ) ->
Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
RPM_Min	Lower limit of desired rotation speed range. Scaled in rpm's.
RPM_Max	Upper limit of desired rotation speed range. Scaled in rpm's.
RPM_class_width	Width of rot.-speed class (interval). Scaled in rpm's.
f1	Center frequency of lowest 1/3 octave in Hz
f2	Center frequency of highest 1/3 octave in Hz
Frequency weighting	Frequency weighting for the result
	0 : linear
	1 : A-weighting
	2 : B-weighting
	3 : C-weighting
	4 : D-weighting
Result	
Result	1/3 octave spectrum referenced to rotation speed

Description:

The two frequency limits f1 and f2 are to be given as the 1/3-octave center frequencies, e.g. f1 = 8 Hz and f2 = 12500 Hz. f1 < f2. The upper 1/3-oct. with its frequency band must lie entirely within half of the sampling frequency.

The individual 1/3-octave values are stated as root mean square (RMS) values.

While the transients in the individual 1/3-octave (band-pass) filters are subsiding at the start of the measurement, the input signal's values are ignored. The transient time for the 1kHz 1/3-octave is is assumed to be 35ms. This time interval is inversely proportional to the 1/3-octave frequency. For very low 1/3-octaves, this time interval becomes considerable. A correspondingly long measurement must then be expected.

The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.

If there are no values in a certain RPM-class, this 1/3-octave spectrum is filled with zeroes. For sensible operation, every 1/3-octave class should have enough measurement values. The lowest 1/3-octave bands deserve special attention in this regard.

The result is a segmented data set. Each segment contains one 1/3 octave spectrum. The x-coordinate of the result counts the 1/3-octave bands (just like the Famos-function OctA). For good representation in the curve window, 1/3-octave labeling should be selected.

The 1/3-octave filter and analyses are in accordance with IEC 651 (sound level measurement), DIN 45652 (1/3-octave filter for electro-acoustic measurements) and EN61260-1:2014 or IEC61260-1:2014 (band-pass filter for octaves and fractions of octaves, filter Class 1).

Examples:

From the time-histories of a vibration signal "vib" and of the rotational speed, the 1/3-octave spectrum referenced to the rotation speed is to be found. The sampling interval for the vibration signal is 0.025 ms.

```
rpm_Min = 1000.0 ; minimum of rotation speed range
rpm_Max = 6000.0 ; maximum of rotation speed range
rpm_Delta = 100.0 ; width of individual rotation speed classes
fEval = 1 ; 0 (linear) 1 (A-weighting)
f1 = 10
f2 = 12500
Thirds = OtrRpmThirds ( vib, speed, rpm_Min, rpm_Max, rpm_Delta, f1, f2, fEval )
```

See also:

OtrRpmSpectrum, SpecThirds, OctI

OtrTachoMode

Available in: Enterprise Edition and above (OrderTracking-Kit)

Is the tacho signal a rotation speed or impulses from an encoder? The interpretation of the signal is specified for the purposes of other Kit functions which use the tacho signal.

Declaration:

```
OtrTachoMode ( Signal type, Encoder type, Encoder pulses, MinRPM )
```

Parameter:

Signal type	Tacho signal type
	0 : Rotation speed in RPM
	1 : Number of impulses per sample step
	2 : Cronos Mode Pulse instant
	3 : Sampled rectangle signal
	4 : Sinusoidal signal
Encoder type	Encoder type. Does the encoder have all cogs?
	0 : Default
	1 : 1 cog missing
	2 : 2 cogs missing
Encoder pulses	Number of encoder pulses. The encoder emits this many pulses per revolution. >= 1. Need not be an integer.
MinRPM	Minimum rotation speed, scaled in RPM. >= 0. For encoders with missing teeth, the rotation speed at which the gap is still just detectable.

Description:

What is the tacho signal's type, i.e., is it already a rotation speed, or pulses from an encoder? This sets the interpretation of the tacho signal for other functions of the Kit for which it is a parameter. Thus, calling this function doesn't cause any action to be take per se, rather, the Kit's memory receives the instruction how to treat the tacho signal in future function calls. Only such functions are affected which explicitly have a parameter designated Tacho. Other functions which take a rotation speed signal as a parameter, expect a parameter scaled in RPM.

Signal type 0: Rotation speed in RPM

- The tacho signal is a rotation speed signal permanently scaled in RPM.

Signal type 1: number of pulses per sampling step

- The tacho signal contains a sequence of (whole) numbers. Each number represents the number of pulses already counted within the current sampling step. Counting of the pulses has already been performed. It is assumed that the rotation speed is directly proportional to the number of pulses counted. The function smoothes the rotation speed signal such as to provide an improved estimate of the rotation speed. If the signal were to contain the sequence of values { ...., 3, 4, 4, 4, 3, 4, 4, 4, 3... }, then for this portion of the signal, a rotation speed of approx. 3.75 is estimated. The function interprets each of the signal's values as a number of pulses. If the signal were to contain the sequence of values { 0, 0, 1, 0, 1, 2, 1 }, then 5 pulses would have been detected. Note: from such a signal as this, the rotation speed can often not be determined well.

Signal type 2: Cronos Mode Pulse instant

- This signal type is only available if you are working with the measurement device imc-Cronos, in which case the device's incremental encoder input must be set to "Mode/ Pulse instant". A very precise pulse acquisition is performed.

Signal type 3: Sampled rectangular signal

- The encoder is assumed to have turned by one increment each time the signal makes a transition from "Less than or equal to 0.0" to "Greater than 0.0". If the signal were to contain the sequence of values { 0, 0, 1, 1, 1, 0, 0, 1, 1, 0 }, then 2 pulses would have been detected. The function returns a rotation speed time-history, scaled in RPM. If the measurement data do not correspond to the appropriate levels, the signal can be re-shaped using, for example, the Schmitt-Trigger function stri().

Signal type 4: Sinusoidal signal

- Sinusoidal signals or other having a clear zero crossing in a positive edge can be processed. If the signal contains noise, it may be necessary to perform smoothing first and then use a Schmitt-trigger. The encoder is assumed to emit a pulse upon a zero-crossing in the positive direction (rising edge). If the signal were to contain the sequence of values { -3.0, -1.0, +1.5, +2.8, +1.3, +0.1, -0.6 }, then a pulse would be detected upon the transition from -1.0 to +1.5.

Special features of encoders with missing pulses:

- The encoder type = 0 is the default. Only in case the signal type is 2 (Cronos Mode Pulse instant), a different encoder type can be selected. The number of encoder pulses must always be entered including the missing tooth and must be integer. For instance, for an encoder with emits a pulse for every 10 degrees, and which should then actually have 36 cogs, 36 must be entered for the number of encoder pulses. But the encoder only emits 35 pulses since one is missing. Another typical encoder has a pulse for every 6 degrees. Here; we enter 60 cogs although 2 are missing so that there are really only 58. The first cog after the gap is taken to be the zero-pulse (which isn't relevant to this function). Encoders with missing cogs can only be used in measurements with Cronos. Recognition of the missing cogs is only possible if the rotation speed is more or less constant around the gap. Particularly with low rotation speeds, this can't be assured. Since recognition of the gap is then not certain, the minimum rotation speed must be set to a non-zero value. For higher rotation speeds, the gap is usually clearly recognized due to mechanical inertia. The function tries to re-synchronize itself after an error in the pulse sequence (or after a pulse sequence interpreted apparently incorrectly). Nevertheless, incorrect values for the rotation speed can have occured now and then.

Minimum rotation speed:

- The value can be 0.0 most of the time. The minimum rotation speed is important in order to distinguish between a standstill and crawling rotation. Otherwise, in situations in which the drive stops, a tiny rotation greater than zero could seem to exist still. By setting the minimum rotation speed to a value above zero, the rotation signal registers a definite zero if the rotation dips below this minimum. In general, it's no longer possible to distinguish between standstill and very slow continuous motion if the distance between pulses is large. The minimum rotation speed states what speed is the least which still matters. If the distance between pulses is greater than for this minimum rotation, a rotation speed of 0.0 is assumed. For encoders with missing teeth, the minimum rotation speed must be greater than 0.0. Note: this artificial setting of the rotation as zero changes (distorts) the integral over the rotation speed. Therefore, such a rotation speed signal should not be subsequently used to find a rotation angle through integration.

Note:

- Before functions which take "Tacho signal" as a parameter can be called, e.g. OtrTachoToSpeed(), it is necessary for the function OtrTachoMode() to have been called at least once, so that the tacho signal's type is clearly defined. If OtrTachoMode() is not called, it is assumed that Tacho represents a rotation speed in RPM. The respective most recent setting in OtrTachoMode() remains in effect until OtrTachoMode() is next called. Note that the record in memory remains valid for the duratoin of the application. Therefore, to make a sequence of commands (a routine) immune to historical artifacts (the effects of whatever the system did before), a call to OtrTachoMode() should come at the start.

Examples:

The rotation speed of a cogwheel is to be determined. For this purpose, a rectangular voltage signal was generated by an inductive encoder. This voltage was captured at a constant sampling rate of 1kHz and is located in the channel named "Signal". The voltage is the approx. 0V..1V in the gap, and approx. 18V..22V when the cog's tip is near. The cogwheel has 8 cogs.

```
OtrTachoMode ( 3, 0, 8, 10.0)
EncoderPulses = stri ( Signal, 15, 5 )
Speed =OtrTachoToSpeed ( EncoderPulses )
```

Since the voltage doesn't come in an appropriate form, it is altered to provide cleaner zero-crossings. If the signal is noisy, it may need to be smoothed and/or subjected to a Schmitt-Trigger.

OtrTachoMode () is called once to set the tacho signal type. Afterwards, only OtrTachoToSpeed () neeeds to be called for various measurements and channels.

The drive runs at rotation speeds in the range 500...3000 RPM. This returns a pulse frequency of up to 400Hz(8*50Hz). Rotation speeds under 10 RPM are not to be displayed; instead they are to be indicated by a straight zero level.

See also:

OtrOrderSpectrum, OtrResampleAAF, OtrResample, OtrTachoToSpeed, OtrTachoToDist, OtrEncoderRevs01

OtrTachoToDist

Available in: Enterprise Edition and above (OrderTracking-Kit)

From a tacho signal, provided as encoder pulses, for example, a data set recording the number of revolutions is calculated. Beforehand, the function OtrTachoMode () must be used to set the tacho signal type.

Declaration:

```
OtrTachoToDist ( Tachometer signal ) -> Time-history of revolutions
```

Parameter:

Tachometer signal	Tacho signal, interpreted as per previous call of OtrTachoMode().
Time-history of revolutions	
Time-history of revolutions	

Description:

This function integrates the encoder pulses. The result thus contains the cumulative sum of all pulses counted by then. The result is scaled as a count of the drive's revolutions. If the result has a value of 1.0 somewhere, it means that one whole revolution was completed by then since the beginning. If the value is later 2.0, then 2 whole revolutions have occurred 2. The function can return not only whole revolutions but also fractions.

Integration generally begins at the first pulse. If the encoder is missing cogs, integration starts at the first pulse after the gap. This produces a phase-correct rotation angle signal.

Integration of the rotation speed returned by OtrTachoToSpeed() will not (!) produce equivalent results.

To obtain a display in degrees, the result can be multiplied by 360 degrees.

Before calling this function, OtrTachoMode() should be called at least once.

Examples:

A rotor angle of a cogwheel on a camshaft is to be determined, which has 36 cogs with one gap. The rotation speed ranges between 800 and 7000 RPM. The data are captured using Cronos PL in Pulse instant mode, which provides a precise time for every single edge in the encoder signal.

```
OtrTachoMode ( 2, 1, 36, 500)
Revolutions =OtrTachoToDist ( Tacho )
```

The result Revolutions states the revolutions counted. The tacho signal type must previously be set using OtrTachoMode ().

See also:

OtrOrderSpectrum, OtrResampleAAF, OtrResample, OtrTachoMode, OtrTachoToSpeed, OtrEncoderRevs01

OtrTachoToSpeed

Available in: Enterprise Edition and above (OrderTracking-Kit)

From a tacho signal which may be available as encoder pulses, a rotation speed signal scaled in RPM is calculated. Before calling this function, the function OtrTachoMode() must be used to state the tacho signal type.

Declaration:

```
OtrTachoToSpeed ( Tachometer signal ) -> Time-history of rotation speed
```

Parameter:

Tachometer signal	Tacho signal, interpreted as per previous call of OtrTachoMode().
Time-history of rotation speed	
Time-history of rotation speed	

Description:

Before calling this function, OtrTachoMode() should be called at least once.

The rotation history obtained with this function should not be integrated. If the integral (consisting of rotation angle or revolutions) is needed, the function OtrTachoToDist() should be used. The purely order analysis functions which sample over the rotation angle , e.g. OtrOrderSpectrum(), should also not receive their rotation data from OtrTachoToSpeed(). Instead, OtrTachoMode() should be set appropriately and the tacho signal directly passed to these functions. Other functions, by contrast, only need the rotation speed, not the angle, e.g. OtrTrackingLowPass(). For these functions, the results of OtrTachoToSpeed() are well adapted.

Examples:

The rotation speed at a single cog is to be found. For this purpose, a rectangular voltage signal is generated by an inductive encoder. This voltage was sampled at a constant rate of 100Hz and is exists on a channel named "Signal". The voltage is approx. 0V..1V in the gap, approx. 4V..4V at the cog's tip. The cogwheel has 12 cogs.

```
OtrTachoMode ( 3, 0, 12, 0.0)
EncoderPulses = stri ( Signal, 3.5, 1.5 )
speed =OtrTachoToSpeed ( EncoderPulses )
```

Since the voltage is not available in apprpriate form, it must be modified to achieve definite zero-crossings. Beforehand, the function OtrTachoMode () must be used to set the tacho signal type.

See also:

OtrOrderSpectrum, OtrResampleAAF, OtrResample, OtrTachoMode, OtrTachoToDist, OtrEncoderRevs01, OtrTachoToSpeedX, PulseDuration

OtrTachoToSpeedX

Available in: Enterprise Edition and above (OrderTracking-Kit)

The rotation speed signal is calculated from the tacho signal. The result gets the data type XY and is scaled in RPM. Before calling this function, the function OtrTachoMode() must be used to state the tacho signal type.

Declaration:

```
OtrTachoToSpeedX ( Tachometer signal ) -> XY time plot of rotation speed
```

Parameter:

Tachometer signal	Tacho signal, interpreted as per previous call of OtrTachoMode().
XY time plot of rotation speed	
XY time plot of rotation speed	

Description:

Before calling this function, OtrTachoMode() should be called at least once.

Only signal types 'Cronos Mode Pulse instant' and 'Sinusoidal signal' are supported.

Encoders with missing teeth are not supported.

With signal types 'Cronos Mode Pulse instant' only one pulse per sampling interval may occur. If there is more than one pulse, the function returns -1 at the corresponding positions.

The result returned is an XY channel with time-stamps. For each pulse in the input data, an XY pair is generated.

The x-coordinate of the result provides all precise time-stamps of the pulses for further calculations.

The y-coordinate contains the speed values that are calculated from the distance to the next pulse. If the signal is displayed in steps, then each RPM-value represents the speed for the complete interval. Since no speed can be calculated for the very last pulse, the speed value from the preceeding pulse will be used.

Examples:

The rotation speed of a cogwheel is to be determined. The input channels contains 'Cronos Pulse Time' data.

```
OtrTachoMode ( 2, 0, 1, 0)
speed =OtrTachoToSpeedX ( CronosPulseTimes )
pulse_times = speed.x
```

See also:

OtrTachoMode, OtrTachoToSpeed

OtrTimeOrderSpectrum

Available in: Enterprise Edition and above (OrderTracking-Kit)

The order spectrum is determined from the plots of vibration and the tachometer signal over time.

Declaration:

```
OtrTimeOrderSpectrum ( Vibration, Tachometer signal, dx, Resolution, OrderMax, WindowType, Speed_Low ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Tachometer signal	Rotation speed signal interpreted as per OtrTachoMode(). Default is time-history of rotation speed, scaled in rpm's.
dx	Sampling interval/time resolution of the result. Order spectra are entered into the result at this distance from each other.
Resolution	Resolution of the order spectrum. 0.1, if 0.1 orders is the distance between oder spectrum lines. The resolution must be an integral fraction of 1.0, thus: [1, 1/2, 1/3, 1/4, ..]
OrderMax	The highest order (line) displayed in the order spectrum.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
	6 : Hamming (RMS=1)
	7 : Hanning (RMS=1)
	8 : Blackman (RMS=1)
	9 : Blackman / Harris (RMS=1)
	10 : Flat Top (RMS=1)
Speed_Low	Lower boundary of the rotation speed, expressed in revs/min. If the absolute value of the rotation speed is <= this value, the order spectrum is set to zero.
Result	
Result	Order spectrum in relation to time

Description:

This function determines an order spectrum for each time point in the result. The first one is assigned the time x0 of the vibration signal. Subsequent ones follow at intervals given by the parameter dx.

The inverse of the resolution states how many revolutions contribute to an order spectrum. For instance, with a resolution of 0.1 each spectrum is determined from 10 revolutions.

The spectral lines are stated as RMS (root mean square) values.

Since the FFT is calculated internally from a larger number of data (a power of two), some spectral lines are cut off. The visible spectrum thus no longer reflects the whole power of the signal. A rectangular window for the FFT is highly recommended, if high resolution of the frequency is desired.

A tracking Butterworth low-pass filter is used as the anti-aliasing filter. The signal thus filtered is sampled over the rotation angle. Then an FFT is calculated. This procedure causes the signal to temporarily contain much higher order lines than in the spectrum which is ultimately returned. The AAF's 3dB-order is far behind the order lines in the resulting spectrum.

The function works well for:

- OrderMax < 10 / (SamplingTime_Vibration * max (rotation speed)),

where SamplingTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value.

A calculated order spectrum results from a certain time period of input data. Typically, there are multiple revolutions (corresponding to the RPM-signal) within this time period. At low RPM-values, this time period may be long. The point in time at which an entry into the result is made is the center of this time period.

The parameter dx is typically selected such that at the highest expected RPM-value, there is a certain resulting overlap of order spectra. The typical overlap is between 50% and 90%. The shortest time duration for an order spectrum is 60 / (Maximum_RPMs * Resolution). At 50% overlap, this duration is divided by 2; at 90% overlap, by 10. If the value of dx is too small, one obtains many order spectra which are (almost) the same. If too large, it is possible for important order spectra to go missing.

dx is rounded to integer multiples of the vibration signal's sampling interval.

The parameter Speed_Low ensures that with long-term data sets, even low RPM-values or even standstills (RPM = 0) are treated. Since in such time regions, there is no sampling over the angle, no new order spectra are generated. Old spectra will then no longer fill up this region. Instead, spectra with zeroes result.

If there is at least one RPM-value captured by Speed_Low within an order spectrum's time period, the entire order spectrum is set to zero.

At the beginning or the end of the entire time range, it can happen that there are no order spectra at the edge. In that case, the region up to the edge is filled with the next existing order spectrum.

If there are no new order spectra for middle time points of the result, the preceding one is used.

The moving antialiasing-filter has a certain settling time. The results of this filtering are ignored as long as the filter is still (strongly) affected by transient oscillations. For this reason, the initial measured values do not affect the first order spectra.

The window function for the FFT used generally has 1 as its mean value. Only such which are denoted by RMS=1 have a root-mean-square value of 1.

In contrast to mean value=1, with RMS=1 the result is divided by sqrt(ENBW=Equivalent noise bandwidth) according to the window type used. E.g. division by sqrt(1.5) in the case of a Hanning window.

- The rotation speed should change only slowly. If the rotation speed becomes far too high, the calculation time required increases enormously without providing better results.
- The rotation angle is determined by integration (summing up) of the tachometer signal. If the tacho signal is itself a rotation speed signal (and not an impulse signal), the rotation speed must be available at good precision.
- The Tacho signal's sampling interval must be either equal to the vibration's sampling time or an integer multiple of it.
- The result is a segmented waveform. Each segment is an order spectrum.
- Before calling this function, OtrTachoMode() should be called at least once.
- For the sampling, the absolute value of the rotation speed is used.

Examples:

An order spectrum's plot over time is to be calculated from the time-history of the rotation speed and of the vibration, vib. The time-based signals are sampled every 0.2 ms.

The maximum rotation speed present is 6000 revs/min.

```
OtrTachoMode ( 0, 0, 0, 0)
dx = 0.05 ; After the elapse of this many seconds, a new order spectrum appears in the result.
Resolution = 0.1 ; spectrum resolution, every 1/10 order is shown, calculated for 10 revolutions
OrderMax = 6.0 ; display spectral lines up to this order.
Windowtype = 0 ; 0 default (rectangle)
SpeedLow = 100 ; If the rotation speed decreases to this value or lower, this needs to be interpreted as a standstill. The result is set to zero within this time region.
OTime = OtrTimeOrderSpectrum ( vib, speed, dx, Resolution, OrderMax, Windowtype, SpeedLow )
```

Here we have: OrderMax = 6.0 < 10 / (0.0002 * 6000) = 8.3

For resolution reasons, an order spectrum is calculated over 10 revolutions. At maximum rotation speed, these 10 revolutions take this many seconds: 10/(6000/60) = 0.1. For an assumed overlap of 50%, it is necessary to select dx = 0.05 [seconds].

OtrTachoMode() is first used to set the tacho signal type used.

See also:

OtrOrderSpectrum, OtrTachoMode

OtrTrackingBandPass

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking band-pass filter. A vibration signal is band-pass filtered, and the filter's center frequency depends on the rotation speed. The time-history of an order is determined.

Declaration:

```
OtrTrackingBandPass ( Vibration, Rotation speed, OrderCenter, WidthPercent, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs.
OrderCenter	Order (line) at which the band-pass' center frequency is located.
WidthPercent	Overall width in percent. Recommended 10%...100%. Input range [0.01 ... 10000.0]. For example, at 30% width the ratio of the upper to the lower cutoff frequency of the bandpass is 1.30.
FilterOrder	The bandpass filter's order (2, 4, 6, 8, 10)
Result	
Result	Time-signal with band-pass filtered signal

Description:

The function finds the time-history of the signal component belonging to a particular order. By means of a narrow band-pass, the time history of, for instance, a fractional order can be found in this way. It may be appropriate to use the function OtrTrackingExpoRms() afterwards.

The rotation speed's absolute value is used.

The internally selected center frequency for the filter is

center frequency = OrderCenter* (Current_RPM / 60)

The upper cutoff frequency is above the center frequency and depends on the filter's width. The upper cutoff frequency is always significantly below half of the vibration signal's sampling frequency. Above approx. 1 / (0.48 * sampling interval), no band-pass filtering can be performed.

The function works well for:

OrderMax << 28 / (SampleTime_Vibration * max (rotation speed)),

where SamplingTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value.

OrderMax = OrderCenter * sqrt (1 + WidthPercent / 100)

<< indicates that the order specified should be much smaller.

- The rotation speed may change only slowly.
- The resulting center frequency may not fall much below 0.1 percent of capacity. The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- If the resulting center frequency falls too low, the result can become less precise.
- Note that band-passes need a certain amount of time for transients to subside. This time increases markedly for narrow-band filters. A band-pass of width 1% is very narrow in this sense. A width of 25% corresponds to a 1/3 octave filter, a width of 100% to a 1/1 octave filter.
- The filter has Butterworth characteristics. For a constant rotation speed, it works comparably to the function FiltBp().

Examples:

A vibration signal "vib" is sampled at an interval of 0.5 ms. The rotation speed can reach 8000 RPM. Find the time-history of the 1.5th order.

```
om = 1.5 ; the 1.5th order is selected.
fo = 6 ; a 6th order band-pass filter is used
width = 30 ; 30% overall width
tbp = OtrTrackingBandPass ( vib, speed, om, width, fo )
```

We have: OrderMax = 1.5 * sqrt (1 + 30 / 100) = 1.7

and thus: OrderMax = 1.7 << 28 / (0.0005 * 8000) = 7.0

Finding the moving root mean square (RMS)-value:

```
bp_rms = OtrTrackingExpoRms ( tbp, rpm, 3.0 )
```

See also:

OtrTrackingLowPass, OtrTrackingBandStop, OtrTrackingExpoRms, OtrRpmOrder, OtrFrequLine, FiltBp

OtrTrackingBandPassZ

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking band-pass filter without phase shift. A vibration signal is band-pass filtered, and the filter's center frequency depends on the rotation speed.

Declaration:

```
OtrTrackingBandPassZ ( Vibration, Rotation speed, OrderCenter, WidthPercent, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs.
OrderCenter	Order (line) at which the band-pass' center frequency is located.
WidthPercent	Overall width in percent. Recommended 10%...100%. Input range [0.01 ... 10000.0]. For example, at 30% width the ratio of the upper to the lower cutoff frequency of the bandpass is 1.30.
FilterOrder	The bandstop filter's order (4, 8, 12, 16, 20)
Result	
Result	Time-signal with band-pass filtered signal

Description:

The function finds the time-history of the signal component belonging to a particular order. By means of a narrow band-pass, the time history of, for instance, a fractional order can be found in this way. It may be appropriate to use the function OtrTrackingExpoRms() afterwards.

The rotation speed's absolute value is used.

The internally selected center frequency for the filter is

center frequency = OrderCenter* (Current_RPM / 60)

The upper cutoff frequency is above the center frequency and depends on the filter's width. The upper cutoff frequency is always significantly below half of the vibration signal's sampling frequency. Above approx. 1 / (0.48 * sampling interval), no band-pass filtering can be performed.

The function works well for:

OrderMax << 28 / (SampleTime_Vibration * max (rotation speed)),

where SamplingTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value.

OrderMax = OrderCenter * sqrt (1 + WidthPercent / 100)

<< indicates that the order specified should be much smaller.

- The rotation speed may change only slowly.
- The resulting center frequency may not fall much below 0.1 percent of capacity. The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- If the resulting center frequency falls too low, the result can become less precise.
- Note that band-passes need a certain amount of time for transients to subside. This time increases markedly for narrow-band filters. A band-pass of width 1% is very narrow in this sense. A width of 25% corresponds to a 1/3 octave filter, a width of 100% to a 1/1 octave filter.
- The filter has Butterworth characteristics. At a constant rotation speed it operates comparably to the function FiltBpZ().

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

A vibration signal "vib" is sampled at an interval of 0.5 ms. The rotation speed can reach 8000 RPM. Find the time-history of the 1.5th order.

The filter must not cause a delay or phase shift.

```
om = 1.5 ; the 1.5th order is selected.
fo = 6 ; a 6th order band-pass filter is used
width = 30 ; 30% overall width
tbp = OtrTrackingBandPassZ ( vib, speed, om, width, fo )
```

We have: OrderMax = 1.5 * sqrt (1 + 30 / 100) = 1.7

and thus: OrderMax = 1.7 << 28 / (0.0005 * 8000) = 7.0

Finding the moving root mean square (RMS)-value:

```
bp_rms = OtrTrackingExpoRms ( tbp, rpm, 3.0 )
```

See also:

OtrTrackingLowPass, OtrTrackingBandStop, OtrTrackingExpoRms, OtrRpmOrder, OtrFrequLine, FiltBp

OtrTrackingBandStop

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking band-stop filter. A vibration signal is band-stop filtered, and the filter's center frequency depends on the rotation speed.

Declaration:

```
OtrTrackingBandStop ( Vibration, Rotation speed, OrderCenter, WidthPercent, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs.
OrderCenter	Order (line) at which the band-stop filter's center frequency is located.
WidthPercent	Overall width in percent. Recommended: 10%...100%. Input range [0.01 ... 10000.0]. For example, at 30% width the ratio of upper to lower cutoff frequency of the band stop is 1.30.
FilterOrder	The band-stop filter's filter order (2, 4, 6, 8, 10)
Result	
Result	Band-stop filtered time-based signal

Description:

The function finds the time-history of signal components which do not belong to a particular order. In this way a narrow band-stop can be used, for instance, to eliminate any desired order from the signal.

The rotation speed's absolute value is used.

The internally selected center frequency for the filter is

center frequency = OrderCenter* (Current_RPM / 60)

The upper cuttoff frequency is above the center frequency and depends on the filter's width. The upper cutoff frequency must always be far below the half of the sampling frequency of the vibration signal. Above approx. 1 / (0.48 * sampling interval), band-stop filtering cannot be performed.

The function works well for:

OrderMax << 28 / (SampleTime_Vibration * max (rotation speed)),

where SampleTime_Vibration is the vibration signal's sampling interval and max (rotation speed) is the maximum occurring RPM value.

OrderMax = OrderCenter * sqrt (1 + WidthPercent / 100)

<< indicates that the order specified should be much lower.

- The rotation speed may change only slowly.
- The resulting center frequency should not fall far below 0.1 percent of capacity.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- Note that the band-stop filter needs a certain amount of time for transients to subside. This time is especially long for narrow filters. A band-stop filter of width 1% is an extremely narrow filter in this sense.
- The filter has Butterworth characteristics. At a constant rotation speed, it works comparably to the function FiltBs(), but with a different choice of initial values.

Examples:

A vibration signal "vib" is sampled at intervals of 1 ms. The rotation speed can reach 8000 RPM. The 2.5th order is to be eliminated.

```
om = 2.5 ;  the 2.5th order is selected.
fo = 4 ; a 4th order band-stop filter is used.
width = 30 ; 30% overall width
tbs = OtrTrackingBandStop ( vib, speed, om, width, fo )
```

We have: OrderMax = 2.5 * sqrt (1 + 30 / 100) = 2.9

and thus: OrderMax = 2.9 << 28 / (0.001 * 8000) = 3.5

See also:

OtrTrackingLowPass, OtrTrackingBandPass, OtrTrackingExpoRms, OtrRpmOrder, OtrFrequLine, FiltBs

OtrTrackingBandStopZ

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking band-stop filter without phase shift. A vibration signal is band-stop filtered, and the filter's center frequency depends on the rotation speed.

Declaration:

```
OtrTrackingBandStopZ ( Vibration, Rotation speed, OrderCenter, WidthPercent, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs.
OrderCenter	Order (line) at which the band-stop filter's center frequency is located.
WidthPercent	Overall width in percent. Recommended: 10%...100%. Input range [0.01 ... 10000.0]. For example, at 30% width the ratio of upper to lower cutoff frequency of the band stop is 1.30.
FilterOrder	The band-stop filter's filter order (4, 8, 12, 16, 20)
Result	
Result	Band-stop filtered time-based signal

Description:

The function finds the time-history of signal components which do not belong to a particular order. In this way a narrow band-stop can be used, for instance, to eliminate any desired order from the signal.

The rotation speed's absolute value is used.

The internally selected center frequency for the filter is

center frequency = OrderCenter* (Current_RPM / 60)

The upper cuttoff frequency is above the center frequency and depends on the filter's width. The upper cutoff frequency must always be far below the half of the sampling frequency of the vibration signal. Above approx. 1 / (0.48 * sampling interval), band-stop filtering cannot be performed.

The function works well for:

OrderMax << 28 / (SampleTime_Vibration * max (rotation speed)),

where SampleTime_Vibration is the vibration signal's sampling interval and max (rotation speed) is the maximum occurring RPM value.

OrderMax = OrderCenter * sqrt (1 + WidthPercent / 100)

<< indicates that the order specified should be much lower.

- The rotation speed may change only slowly.
- The resulting center frequency should not fall far below 0.1 percent of capacity.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- Note that the band-stop filter needs a certain amount of time for transients to subside. This time is especially long for narrow filters. A band-stop filter of width 1% is an extremely narrow filter in this sense.
- The filter has Butterworth characteristics. At a constant rotation speed it operates comparably to the function FiltBsZ(), but with a different choice of initial values.

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

A vibration signal "vib" is sampled at intervals of 1 ms. The rotation speed can reach 8000 RPM. The 2.5th order is to be eliminated.

The filter must not cause a delay or phase shift.

```
om = 2.5 ;  the 2.5th order is selected.
fo = 4 ; a 4th order band-stop filter is used.
width = 30 ; 30% overall width
tbs = OtrTrackingBandStopZ ( vib, speed, om, width, fo )
```

We have: OrderMax = 2.5 * sqrt (1 + 30 / 100) = 2.9

and thus: OrderMax = 2.9 << 28 / (0.001 * 8000) = 3.5

See also:

OtrTrackingLowPass, OtrTrackingBandPass, OtrTrackingExpoRms, OtrRpmOrder, OtrFrequLine, FiltBs

OtrTrackingExpoRms

Available in: Enterprise Edition and above (OrderTracking-Kit)

Calculation of the moving RMS value with exponentially weighted averaging, where the time constant depends on the rotation speed.

Declaration:

```
OtrTrackingExpoRms ( Vibration, Rotation speed, SmoothingRevs ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs.
SmoothingRevs	The time constant corresponds to this many revolutions.
Result	
Result	Smoothed time-history

Description:

If a time constant is specified for finding the exponential RMS-value with the function ExpoRms (), it is the time constant for the exponential function. In the function OtrTrackingExpoRms (), the constant is not specified as a time interval but as a number of revolutions. The time constant is then found based on the prevalent rotation speed. The resulting time constant is inversely proportional to the rotation speed.

Time constant = SmoothingRevs * Current_RotaSpeed / 60

SmoothingRevs is a real number.

The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.

At a constant rotation speed, the functions operates comparably to the function ExpoRms(), but with a different choice of initial values.

Examples:

By means of band-pass filtering, the vibration time-history of an order line is to be obtained. This time-history is to be represented as a root mean square (RMS) value referenced to the time. We are given a vibration "vib" and a rotation speed "speed".

```
tbp = OtrTrackingBandPass ( vib, speed, 1.5, 30, 4 )
_Smooth = 3.0 ; number of revolutions
bp_rms = OtrTrackingExpoRms ( tbp, speed, _Smooth )
```

For obtaining the moving RMS-value, the time constant used is 3 revolutions.

See also:

OtrTrackingBandPass, OtrTrackingBandStop, ExpoRms

OtrTrackingHighPass

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking high-pass filter. A vibration signal is high-pass filtered, where the filter's cutoff frequency depends on the rotation speed.

Declaration:

```
OtrTrackingHighPass ( Vibration, Rotation speed, Order3dB, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
Order3dB	Order (line), for which the high-pass filter dampens the signal by 3dB.
FilterOrder	The highpass filter's order (1 ... 10)
Result	
Result	Filtered vibration signal.

Description:

The rotation speed's absolute value is used.

The filter's internally selected cutoff frequency is

Cutoff frequency = Order3dB * (current rotation speed / 60)

The cutoff frequency must always be much less than one half of the vibration signal's sampling frequency in order to achieve a filtering effect. The function works well for:

Order3dB <<24 / (SampleTime_Vibration * max (rotation speed))

where SampleTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value. << indicates that the order specified must be much smaller. Conversely, it means that the maximum rotation speed may not become too high.

- The rotation speed should change only slowly and may not fall much below 1% of capacity.
- The high-pass filter's upper cutoff frequency must always be far less than the half of the vibration signal's sampling frequency. Above approx. (0.4 * sampling frequency) filtering can no longer be performed.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- The filter has Butterworth characteristics. At a constant rotation speed it operates comparably to the function FiltHp(), but with a different choice of initial values.

Examples:

A vibration signal "vib" is sampled at a rate of 0.2 ms. The rotation speed "speed" can reach 6000 RPM. Components below the 10th order are to be suppressed.

```
o3dB = 10.0 ; the high-pass filter dampens by 3dB for this order
fo = 2 ; a 2nd order high-pass filter is used
tlp = OtrTrackingHighPass ( vib, speed, o3dB, fo )
```

We have: OrderMax = 10.0 << 24 / (0.0002 * 6000) = 20.0.

The filter is dimensioned to dampen by 3 dB at the 10th order.

See also:

OtrTrackingBandPass, OtrTrackingLowPass, OtrResampleAAF, OtrRpmPresentation, FiltHp

OtrTrackingHighPassZ

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking high-pass filter without phase shift. A vibration signal is low-pass filtered, where the filter's cutoff frequency depends on the rotation speed.

Declaration:

```
OtrTrackingHighPassZ ( Vibration, Rotation speed, Order3dB, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
Order3dB	Order (line), for which the high-pass filter dampens the signal by 3dB.
FilterOrder	The highpass filter's order (2, 4, 6, .. 20)
Result	
Result	Filtered vibration signal.

Description:

The rotation speed's absolute value is used.

The filter's internally selected cutoff frequency is

Cutoff frequency = Order3dB * (current rotation speed / 60)

The cutoff frequency must always be much less than one half of the vibration signal's sampling frequency in order to achieve a filtering effect. The function works well for:

Order3dB <<24 / (SampleTime_Vibration * max (rotation speed))

where SampleTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value. << indicates that the order specified must be much smaller. Conversely, it means that the maximum rotation speed may not become too high.

- The rotation speed should change only slowly and may not fall much below 1% of capacity.
- The high-pass filter's upper cutoff frequency must always be far less than the half of the vibration signal's sampling frequency. Above approx. (0.4 * sampling frequency) filtering can no longer be performed.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- At a constant rotation speed the filter operates comparably to the function FiltHpZ(), but with a different choice of initial values.

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

A vibration signal "vib" is sampled at a rate of 0.2 ms. The rotation speed "speed" can reach 6000 RPM. Components below the 10th order are to be suppressed.

The filter must not cause a delay or phase shift.

```
o3dB = 10.0 ; the high-pass filter dampens by 3dB for this order
fo = 2 ; a 2nd order high-pass filter is used
thp = OtrTrackingHighPassZ ( vib, speed, o3dB, fo )
```

We have: OrderMax = 10.0 << 24 / (0.0002 * 6000) = 20.0.

The filter is dimensioned to dampen by 3 dB at the 10th order.

See also:

OtrTrackingHighPass, OtrTrackingLowPassZ, OtrResampleAAF, OtrRpmPresentation, FiltHpZ, FiltHp

OtrTrackingLowPass

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking low-pass filter. A vibration signal is low-pass filtered, where the filter's cutoff frequency depends on the rotation speed.

Declaration:

```
OtrTrackingLowPass ( Vibration, Rotation speed, Order3dB, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
Order3dB	Order (line), for which the low-pass filter dampens the signal by 3dB
FilterOrder	The filter order of the low-pass filter (1 ... 10)
Result	
Result	Filtered vibration signal.

Description:

The rotation speed's absolute value is used.

The filter's internally selected cutoff frequency is

Cutoff frequency = Order3dB * (current rotation speed / 60)

The cutoff frequency must always be much less than one half of the vibration signal's sampling frequency in order to achieve a filtering effect. The function works well for:

Order3dB <<24 / (SampleTime_Vibration * max (rotation speed))

where SampleTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value. << indicates that the order specified must be much smaller. Conversely, it means that the maximum rotation speed may not become too high.

- The rotation speed should change only slowly and may not fall much below 1% of capacity.
- The upper cutoff frequency of the low-pass filter must always be far below half of the vibration signals sampling rate. Above approx. (0.4 * sampling rate), filtering cannot be performed.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- The filter has Butterworth characteristics. At a constant rotation speed it operates comparably to the function FiltLp(), but with a different choice of initial values.

Examples:

A vibration signal "vib" is sampled at a rate of 0.2 ms. The rotation speed "speed" can reach 6000 RPM. Components above the 10th order are to be suppressed.

```
o3dB = 10.0 ; the low-pass filter dampens by 3dB for this order
fo = 6 ; a 6th order low-pass filter is used
tlp = OtrTrackingLowPass ( vib, speed, o3dB, fo )
```

We have: OrderMax = 10.0 << 24 / (0.0002 * 6000) = 20.0.

The low-pass filter is dimensioned to dampen by 3 dB at the 10th order, by 40 dB at the 22nd order. Below the 8.3th order, the amplitude error is already below 5%.

See also:

OtrTrackingBandPass, OtrResampleAAF, OtrRpmPresentation, FiltLP

OtrTrackingLowPassZ

Available in: Enterprise Edition and above (OrderTracking-Kit)

Tracking low-pass filter without phase shift. A vibration signal is low-pass filtered, where the filter's cutoff frequency depends on the rotation speed.

Declaration:

```
OtrTrackingLowPassZ ( Vibration, Rotation speed, Order3dB, FilterOrder ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal
Rotation speed	Time-history of rotation speed signal. Scaled in RPMs
Order3dB	Order (line), for which the low-pass filter dampens the signal by 3dB
FilterOrder	The lowpass filter's order (2, 4, 6, .. 20)
Result	
Result	Filtered vibration signal.

Description:

The rotation speed's absolute value is used.

The filter's internally selected cutoff frequency is

Cutoff frequency = Order3dB * (current rotation speed / 60)

The cutoff frequency must always be much less than one half of the vibration signal's sampling frequency in order to achieve a filtering effect. The function works well for:

Order3dB <<24 / (SampleTime_Vibration * max (rotation speed))

where SampleTime_Vibration is the sampling interval for the vibration signal and max (rotation speed) is the maximum occurring RPM-value. << indicates that the order specified must be much smaller. Conversely, it means that the maximum rotation speed may not become too high.

- The rotation speed should change only slowly and may not fall much below 1% of capacity.
- The upper cutoff frequency of the low-pass filter must always be far below half of the vibration signals sampling rate. Above approx. (0.4 * sampling rate), filtering cannot be performed.
- The sampling interval for the rotation speed must be either the same as the vibration's sampling interval, or an integer multiple of it.
- At a constant rotation speed the filter operates comparably to the function FiltLpZ(), but with a different choice of initial values.

The waveform is filtered once forwards and once backwards. But for this purpose a filter is used which has the characteristics specified but whose order is only one-half of that stated.

The amplitude-frequency response is standardized so that the damping is 3dB at the cutoff frequency.

The filter is not causal. For this reason, the chronological order between cause and effect no longer applies.

Transisnt effects in both directions exist, in particular not only at the beginning, but also at the end of the waveform.

Ideally, the value of the phase is zero. In practice, however, this only applies to the state after all transients have subsided, not to border areas.

Examples:

A vibration signal "vib" is sampled at a rate of 0.2 ms. The rotation speed "speed" can reach 6000 RPM. Components above the 10th order are to be suppressed.

The filter must not cause a delay or phase shift.

```
o3dB = 10.0 ; the low-pass filter dampens by 3dB for this order
fo = 6 ; a 6th order low-pass filter is used
tlp = OtrTrackingLowPassZ ( vib, speed, o3dB, fo )
```

We have: OrderMax = 10.0 << 24 / (0.0002 * 6000) = 20.0.

The filter is dimensioned to dampen by 3 dB at the 10th order.

See also:

OtrTrackingLowPass, OtrTrackingBandPassZ, OtrResampleAAF, OtrRpmPresentation, FiltLpZ, FiltLP

ParametersPassed?

The function returns the number of parameters that were passed when calling a sequence or a sequence function.

Declaration:

`ParametersPassed? ( ) -> Number`

Parameter:

Number	
Number	The number of parameters passed when calling the sequence or sequence function.

Description:

In a sequence functions with optional parameters: before using an an optional parameter in the code you should first check whether this parameter was passed by the caller and thus the corresponding variable even exists. The number of parameters actually passed can be determined using the ParametersPassed?() function. The missing parameters can then, for example, either be created yourself with standard values or their use can be prevented by the conditional execution of code branches. The first example demonstrates relevant techniques.

Even when calling classic sub-sequences with the SEQUENCE command, the number of parameters actually passed is only known at runtime. ParametersPassed?() also provides the number of parameters passed and thus information about how many of the formal parameters PA1 .. PA20 are valid.

The function returns -1 if it is not called within a sequence function or in a sequence started with the SEQUENCE command.

Examples:

A sequence function "CalcSum" is defined with 4 parameters, the last 2 being optional. The function should calculate the sum over all specified parameters.

Declaration:

```
!CalcSum( summand1, summand2 [, summand3] [, summand4]) => totalSum
```

The code of the sequence function must therefore take into account that [summand3] and/or [summand4] were not passed by the caller. Accessing these two parameters in the code would then lead to an error because the variables are then not known at runtime.

Variant #1: Conditional execution of code branches, depending on the number of parameters passed:

```
parCount = ParametersPassed?()
; parCount has the value 2, 3 or 4

totalSum = summand1 + summand2
SWITCH parCount
   CASE 3
      ; ; the 4th parameter is missing
      totalSum = summand1 + summand2 + summand3
   CASE 4
      ; all parameters available
      totalSum = summand1 + summand2 + summand3 + summand4
END
```

Variant #2: Create parameters that were not transferred yourself and initialize them with standard values:

```
parCount = ParametersPassed?()  ; 2, 3 or 4

IF parCount < 3
   ; the 3rd parameter is missing, initialize with default value
   summand3 = 0
END
IF parCount < 4
   ; the 4th parameter is missing, initialize with default value
   summand4 = 0
END

totalSum = summand1 + summand2 + summand3 + summand4
```

Variant #3: The parameters passed are processed in a loop. Requires that the parameter naming follows the "FixedName" + sequential number scheme and can be useful for a larger number of optional parameters.

```
parCount = ParametersPassed?()  ; 2, 3 or 4
totalSum = summand1 + summand2

FOR i = 3 TO parCount
   txParName = "summand"+ TForm(i, "")
   totalSum = totalSum + <txParName>
END
```

The following subsequence applies a low-pass filter (Butterworth, 4th order) to the first parameter passed. The second parameter specifies the cutoff frequency. If this is omitted, the default value of 10Hz is used. If no parameters or more than 2 parameters were passed, the sequence terminates with an error message.

```
OnError("ReturnFail")
parCount = ParametersPassed?()

SWITCH parCount
   CASE 1
      f_cutoff = 10 ; default value
   CASE 2
      f_cutoff = PA2
   DEFAULT
      ThrowError("Unexpected number of passed parameters!")
END
PA1 = FiltLP(PA1, 0, 0, 4, f_cutoff)
```

See also:

SEQUENCE, BoxVarSelector, VarGetInit

Supported since:

Version 2024

PAUSE

Execution of the sequence is interrupted and a message is displayed which must be acknowledged by the user.

Declaration:

`PAUSE OutputText`

Parameter:

OutputText	Arbitrary text, which is displayed

Description

The sequence is halted and a dialog box with the buttons "OK" and "Cancel", as well as the specified text is displayed. You can then view intermediate resuls within the sequence or intervene in the sequence in other ways. If you select the button "OK", the sequence is executed further. If you select the button "Cancel", execution of the sequence is cancelled.

- Alternatively, you can also use the function BoxOutput() in order to display information for the user. For entering texts or values by the user, the functions BoxValue?() and BoxText?() are available. For simple decisions ('Yes'/'No'-prompts), you can use the function BoxMessage().
- In order to interrupt execution of the sequence for a specified delay time, you can use the function Sleep().
- Multithreading: The function may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

Examples:

```
SHOW x
PAUSE First Step
x = x + 1
PAUSE Continue calculation?
x = x + 2
```

The original data set x and the first intermediate results can be viewed at leisure.

See also:

BoxOutput, BoxMessage, BoxValue?, Sleep, EXITSEQUENCE

Peaks

Returns the number of peaks (oscillations) in the data set.

Declaration:

```
Peaks ( Data ) -> SvCount
```

Parameter:

Data	Data set under investigation. Permitted data types: [ND]
SvCount	
SvCount	Count of peaks (oscillations)

Description:

The Peaks() function counts the number of peaks in a data set. A peak begins when the values of a data set become greater than zero and ends when the values become smaller than or equal to zero. Therefore, they are counted with a half-peak resolution.

A data set with the values 0, 0, 0, 1, 0, 0 would then contain one peak, as would the data set with the values 1, 2, 1, -1, -1, 2.

A data set with the values 0, 1, 0, 1 would contain 1.5 peaks, a data set with the values -1, 1, -1, 1, -1exactly 2 peaks.

Before applying the Peaks() function, the signal should be conditioned appropriately by subtracting its offset and smoothing or by using the Schmitt trigger function.

Examples:

```
three = Peaks(cos(Ramp(0, 0.1, 200)))
```

A cosine-shaped data set contains 3 full periods of the cosine oscillation.

```
_peaks = Peaks(Smo5(NDdata) - 5)
```

A data set whose peaks are combined with a y-offset of 5, is smoothed and the y-offset is cleared before the peaks are counted. Smoothing should prevent the function from counting distortions as peaks.

```
_peaks = Peaks(sTri(NDdata, 5, 10))
```

The peak counting is especially reliable when the peaks are shaped with the Schmitt trigger function before counting.

```
frequ = Peaks(NDdata)/(Leng?(NDdata) * xDel?(NDdata))
```

The peak frequency is determined by dividing the number of peaks by the duration.

See also:

STri, PulseDuration, OtrTachoToSpeed, Smo

Perio

This is a versatile function which can be used to compute any of the following properties in a periodic data set: mean values, standard deviation, upper/ lower envelope curve across all the periods; or a particular period in the data set is outputted.

Declaration:

```
Perio ( Data, SvPeriodLength, SvOption ) -> Result
```

Parameter:

Data	Periodic data set to be processed; allowed data types: [ND]
SvPeriodLength	Number of values in a period
SvOption	Defines the calculation type
	-1 : Output of the mean values over all periods
	-2 : Output of the standard deviations over all periods
	-3 : Output of the upper envelope curve over all periods (maxima)
	-4 : Output of the lower envelope curve over all periods (Minima)
	>=0 : Output of the period entered
Result	
Result	Result corresponding to [SvOption]

Description:

The data set NDData is divided into periods. A period contains the number of values specified as the parameter [SvPeriodLength]. This is also the number of values contained in the data set NDPeriod, which will be returned. Depending on the specification of the parameter SvOption, the mean values, standard deviation, the upper or lower envelope curve is calculated over all periods or a special period is returned.

When -1 is selected for the parameter SvOption, the mean values over all periods are calculated and returned. The total number of mean values determined is the same number of values contained in a period. The first mean value returned is the mean value of all first values in all periods; the second mean value returned is the mean value of all second values in all periods, etc.

When -2 is specified for the parameter [SvOption], the standard deviation over all periods is calculated and returned. The number of standard deviation values determined is the same number of values contained in a period. The first standard deviation value returned is the standard deviation for all first values in all periods; the second standard deviation value is the standard deviation for all second values in all periods, etc.

By selecting -3 for the parameter [SvOption], the upper envelope curve over all periods is calculated and returned. The total number of upper envelope curve values (maxima) is the same as the number of values contained in a period. The first upper envelope curve value returned is the maximum of all first values in all periods; the second upper envelope curve value returned is the maximum of all second values in all periods, etc.

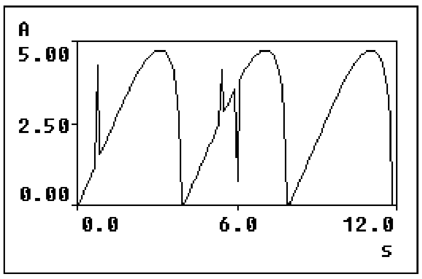
Selecting -4 for the parameter [SvOption] calculates and returns the lower envelope curve over all periods. The total number of lower envelope curve values is the same as the number of values in a period. The first lower envelope curve value returned is the minimum of all first values in all periods; the second lower envelope curve value returned is the minimum for all second values in all periods, etc.

When an integer greater than or equal to zero is selected for the parameter [SvOption], the period corresponding to this number is returned. For example, when 0 is selected, the first period is returned; when 1 is selected, the second period is returned, etc.

- If the period length if greater than the length of the data set, selecting -2 for the parameter SvChoice sets the standard deviation value to 0; when -1, -3 , -4 or 0 is selected for the parameter SvChoice, the resulting data set NDPeriod is extended with zeros.
- Even if the length of the data set is not a multiple of the period length, entries of -1, -2, -3 or -4 for the parameter SvPeriod consider all values occurring in the data set for calculation; when the value corresponding to the last period is specified (>= 0), the resulting data set NDPeriod is extended with zeros.

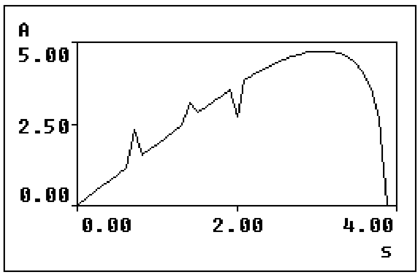
Examples:

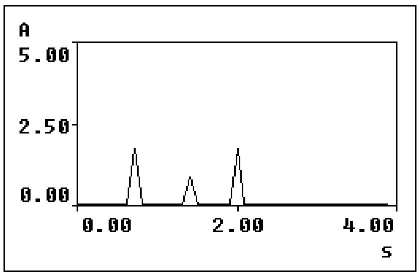
The data set NDData illustrated below is to be processed using the Perio function. The data set consists of 120 values; a period is specified as consisting of 40 values:



```
NwMean = Perio(NwData, 40, -1)
NwStDev = Perio(NwData, 40, -2)
```

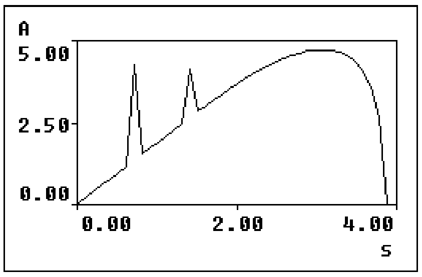
The result of applying the Perio function with a [SvOption] parameter of -1 (calculation of mean values) is shown in the graph on the left, as the data set NDMean. The selection of -2 for the parameter [SvOption] (calculation of standard deviation) is returned in imc FAMOS in the graph on the right, as the data set NDStDev:

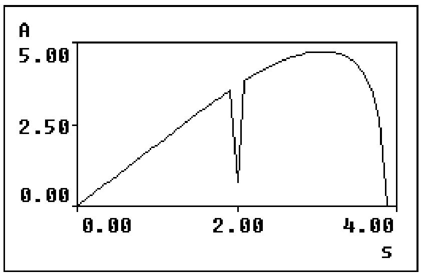




```
NwTop  = Perio(NwData, 40, -3)
NwBottom = Perio(NwData, 40, -4)
```

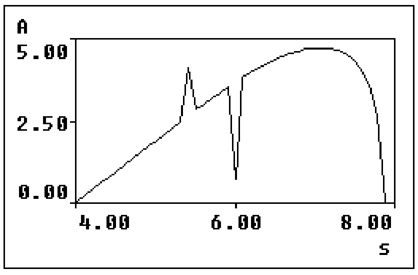
The result of applying the Perio function with the [SvOption] parameter set to -3 (calculation of the upper envelope curve) is returned in imc FAMOS as the data set NDTop in the left graph; selection of -4 for the parameter [SvOption] (calculation of the lower envelope curve) is returned in imc FAMOS as the data set NDBot in the right graph:





```
Nw2nd = Perio(NwData, 40, 1)
```

The result of applying the function Perio with a value of 1 for the parameter [SvOption] (returns the second period) is returned in the following graph as the data set NDSpeci:



PhaseContinuous

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

The course of a signal phase is made continuous. This function eliminates all jumps of 360 degrees.

Declaration:

```
PhaseContinuous ( Phase ) -> Result
```

Parameter:

Phase	The function can be applied to complex waveforms which are represented in magnitude/phase format. In that case, only the phase is processed. Segmented waveforms in particular are also allowed. The function can also be applied directly to phase plots (not only to complex waveforms).
Result	
Result	Corrected phase

Description:

Since FFT calculations and certain other analytical functions only can determine the phase as multiples of 360 degrees, phase jumps seem to occur, e.g. from -180 degrees to + 180 degrees. But it should be possible to specify an angle beyond -180 degrees, e.g. -270 degrees.

This function eliminates all jumps of more than 180 degrees by adding multiples of 360 degrees. the phase's y-unit is used as the indication of whether the correction is made in terms of degrees (360 degrees) or arc (2*PI).

The phase of the first data point, however, is ignored. This is because at a frequency of 0 this is the steady component (DC), of which the phase is either 0 or 180 degrees. Therefore, the jump between the first and second values remains uncorrected.

Examples:

```
Phase = PhaseContinuous ( Phase )
```

Here, the phase plot is recorded in a waveform. The 360 degree jumps are eliminated.

```
frf = FrequencyResponse ( in, out, 1000, 0, 50, 0 )
frf = PhaseContinuous ( frf )
```

Here, the phase of an FRF function is made continuous.

See also:

PhaseMod, NorthCorrection, mod, FrequencyResponse

PhaseMod

Available in: Professional Edition and above

Wind directions, angles, or phases are transformed into a customary value range, e.g. 0 .. 360 degrees

Declaration:

```
PhaseMod ( Data [, Range] ) -> Result
```

Parameter:

Data	input data
Range	Range (optional , Default value: "360")
	"360" : 0..360 degrees
	"180" : -180..+180 degrees
	"2pi" : 0..2*pi
	"pi" : -pi..pi
Result	
Result	Result

Description:

Angles are transformed back into the desired value range by addition of an appropriate multiple of 360°, without changing the angle's actual content.

For the range 0..360 degrees, for example, when the angle is above 360 degrees, 360 is subtracted, and for angles below 0, the number 360 is added.

The function replaces a suitable modulo calculation

If the phase of a complex data set is corrected, its component .P must be specified as an inpunt data set.

Only equidistant input data are supported. If the input data have a time track, the function must be applied to .Y of the data set.

Examples:

Transforms phase after a calculation into the range -180 to +180 degrees

```
Phase = PhaseMod( Phase, "180" )
```

Calculation of the moving wind direction averaged over 10s.

```
NC = NorthCorrection( Channel, 10 )
NC_Mean = MvMean( NC, 10, 10 )
Wind = PhaseMod( NC_Mean, "360" )
```

See also:

NorthCorrection, mod, PhaseContinuous

PI

Circle constant PI = 3.1415...

Examples:

The constant PI as the measure of an angle in radians corresponds to 180 degrees:

```
Degree180 = PI * INDEGR
```

Generating a data set showing exactly the first half cycle of a sine function:

```
HalfWave = sin(Ramp(0, PI/100, 100))
```

See also:

PI2

PI2

2 * circle constant PI = 6.2831...

Examples:

The constante PI2 as a measure for an angle expressed in radians corresponds to 360 degrees:

```
Degree360 = PI2 * INDEGR
```

Generating a data set showing exactly the first cycle of a sin function:

```
wave = sin(Ramp(0, PI2/100, 100))
```

See also:

PI

PnClose

Closes the active Panel

Declaration:

```
PnClose ( SvOption )
```

Parameter:

SvOption	Options parameter or return value of a Panel dialog

Description:

Closes the active Panel or all Panels.

The meaning of the Options parameter depends on the call's context:

Call within an event sequence of a Panel-dialog which had been started by the function Dialog():

The Panel-dialog is closed and the parameter passed is used as the return value of the Dialog()-command. The function thus behaves analogously to the function DlgCloseDialog() for user-defined dialogs.

Else:

A value of 0 signifies that the active Panel is to be closed. A 1 closes all open panels.

The Panels will not be saved; any canges will be lost.

This function was introduced with FAMOS V2022 and replaces the function DbClosePanel() of older versions.

Examples:

A Panel file is opened. A variety of updates are performed, after which the Panel is printed and then closed again.

```
err = PnLoad("d:\templates\result.panel")
IF err <> 0
   BoxMessage("Error", GetLastError(), "!1")
ELSE
   ; various updates
   ; ...
   PnPrint(0)
   PnClose(0)
END
```

A Panel 'InputValue.panel' consists of, among other things, an input box "input" for entering a positive numerical value, as well as 2 buttons 'OK' and 'Cancel'. The Dialog()-command returns the entered value, or -1 to cancel.

Event-sequence 'Button pressed' for the 'OK'-button:

```
value = PnGetValue("input")
PnClose(value)
```

Event-sequence 'Button pressed' for the 'Cancle'-button

```
PnClose(-1)
```

Event-sequence 'Close' (user utilizes system menu to close):

```
; Same behavior as for the 'Cancel'-button
PnClose(-1)
```

Calling the dialog:

```
value = Dialog("InputValue.panel", "", 0)
IF value < 0
   EXITSEQUENCE 0
END
;Continue with sequence...
...
```

See also:

PnLoad, DbShow, Dialog

Supported since:

Version 2022

PnDeleteItem

Scope: Panels

Deletes one or all entries (listbox, droplist etc.)

Declaration:

```
PnDeleteItem ( TxElementName, Index )
```

Parameter:

TxElementName	Name of the element to be changed
Index	Index of the entry to be deleted. The first entry has the index 1. To delete all entries, enter a 0.

Applies to:

Listbox, Droplist, Combobox

Examples:

In a list box with multi-selection, all selected entries are deleted:

```
Count = PnGetItemCount("list1")
i = Count
WHILE i > 0
   IF PnIsItemSelected("list1", i)
      PnDeleteItem("list1", i)
   END
   i = i - 1
END
```

See also:

PnGetItemCount, PnGetItemText, PnSetItemText, PnInsertItem, PnFindItem

PnEnable

Scope: Panels

The specified element is disabled (and thus not operable by the user), or enabled again

Declaration:

`PnEnable ( TxElementName, Task )`

Parameter:

TxElementName	Name of the Panel element to be displayed.
Task	Task
	0 : Disable element
	1 : Enable element

Description:

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel page with the name 'Filter' contains a button 'Execute', with which a currently selected channel (1st measurement/1st channell) is to be filtered. If no such channel is selected in either the Measurement or Channel lists, the button is disabled.

Event sequence 'Data selection changed''

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   PnEnable("Filter.Execute", 1)
ELSE
   PnEnable("Filter.Execute", 0)
END
```

See also:

PnShow

PnExportGraphics

Scope: Panels

A page of the active Panel is exported to a selectable graphics format.

Declaration:

```
PnExportGraphics ( TxFileName, PageSelection, Format, Resolution, ColorCount, Zero ) -> Success
```

Parameter:

TxFileName	Filename under which to save the exported file
PageSelection	Selection of the page to be exported.
	-1 : The active page is exported.
	>=1 : Page number; only the specified page is exported.
Format	Graphics-Format
	0 : Portable Networks Graphic-FileFormat (*.png)
	1 : JPEG-FileFormat (*.jpg)
	2 : Windows Bitmap (*.bmp)
	3 : Windows Enhanced Metafile (*.emf)
Resolution	Specifies the resolution to be selected for the Bitmap-formats. The unit is 'dpi' (Dots per Inch). Typical values include 150dpi or 300dpi. The value must lie within the range 72 - 1200dpi. For [Format] = 3 (metafile), this value should be set to 0.
ColorCount	Specifies the color type to be generated for the Bitmap-formats. For [Format] = 3 (metafile), this should be set to 0.
	0 : Not used (EMF)
	1 : Truecolor
	2 : 256 colors
Zero	Reserved, always set to 0
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

A page of the active Panel is exported under the specified filename to the selected graphics format.

If the specified filename does not have any extension, the default extension for the selected file format is used.

If no full pathname is provided, the if there is an active project the current project folder is used. Otherwise, the default folder for Panel files set in the FAMOS presettings is used.

Examples:

A Panel page contains a button 'Export'. When the button is pressed by the user, then in the associated event-sequence a text box on the same page is first filled with the current date and time, and next this page is exported in the PNG-format. Resolution: 300dpi, 24 *10^6 colors).

Event-sequence 'Pressed' of the 'Export'-button

```
TxDate = TimeToText( TimeSystem?(), 3)
PnSetText( "Date", TxDate)
PnExportGraphics("d:\exports\panel.png", -1, 0, 300, 1, 0)
```

See also:

PnPrint, PnExportPDF

PnExportPDF

Scope: Panels

The active Panel is exported as a PDF document.

Declaration:

```
PnExportPDF ( TxFilename, PageSelection, Option [, Method] ) -> Success
```

Parameter:

TxFilename	Name of the file to be created
PageSelection	Selection of the page(s) to be exported
	-1 : The active page is exported.
	0 : The entire Panel (all pages) are exported.
	>=1 : Page number; only the specified page is exported.
Option	Option parameter
	0 : If the file already exists, it is overwritten.
	1 : If the file already exists, new pages are appended.
Method	Export method (optional , Default value: 0)
	0 : The global default setting ("Options"/"File export"/"PDF") is used.
	1 : Automatic selection of the procedure which ensures that the resulting file's size is as small as possible.
	2 : Bitmap: The entire page is saved as a bitmap and embedded in the PDF-document. Compatible with FAMOS-versions <= 7.2
	3 : Vector graphic preferred: The individual graphics objects embedded individually in the PDF-document as much as possible. Provides best quality.
	4 : Vector graphic preferred. Alternative export method with the virtual printer driver "Win2PDF". This provides improvements in speed in many scenarios and can reduce the size of the PDF-file generated. The driver needs to have been installed prviously (https://www.win2pdf.com). A free 30-day trial version is available. The setting "Printout quality" in "Control Panel"->"Printers & Scanners"->"Win2PDF" is applied. 300 DPI are normally a good compromise between file size and quality.
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

The active Panel is exported under the specified filename to the PDF format.

If the specified filename has no name extension, '.pdf' is appended.

If no complete pathname is specified, then if there is an active project the current project folder is used. Otherwise, the default folder for Panel files specified in the FAMOS presettings is used.

Examples:

A Panel page contains a button 'Generate Report'. When the buton is pressed by the user, then in the associated event-sequence a text box on the same page is first filled with the current date and time, and next all pages of this Panel are exported to a PDF file.

Event-sequence 'Pressed' of the 'Generate Report'-button

```
TxDate = TimeToText( TimeSystem?(), 3)
PnSetText( "Date", TxDate)
PnExportPDF("d:\reports\report.pdf", 0, 0)
```

See also:

PnPrint, PnExportGraphics

PnFindItem

Scope: Panels

Finds a (list-) entry with the specified contents.

Declaration:

```
PnFindItem ( TxElementName, TxContents ) -> Index
```

Parameter:

TxElementName	Name of the element in question
TxContents	Text for the entry for which to search
Index	
Index	Index of the entry to be found; (>=0) if found. 0 otherwise.

Description:

The function is not case-sensitive

Applies to:

Listbox, Droplist, Combobox, CCV Selector

Examples:

Looks for an entry in a list box (with single-selection). If the entry exists, it is selected and scrolled into view, if necessary.

```
i = PnFindItem("list", "100.0")
IF i > 0
   PnSelectItem("list", i)
END
```

See also:

PnGetItemCount, PnGetItemText, PnSetItemText, PnInsertItem, PnDeleteItem

PnGetActivePage

Scope: Panels

The page number of the currently active (visible) page in the current Panel is determined.

Declaration:

```
PnGetActivePage ( ) -> PageNumber
```

Parameter:

PageNumber	
PageNumber	Current page number

Description:

Examples:

A Panel contains multiple pages with a button '>>'. Upon clicking on the button, the respective next page is to be turned.

```
Page = PnGetActivePage()
IF Page < PnGetPageCount()
   PnSetActivePage(Page +1)
END
```

See also:

PnSetActivePage

PnGetFileSelection

Scope: Panels

The current selection in a File Explorer element's file list is requested.

Declaration:

```
PnGetFileSelection ( TxElementName, ReturnFormat ) -> Selection
```

Parameter:

TxElementName	Name of the File Explorer element requested
ReturnFormat	Governs in which format the selected entries are returned.
	0 : Complete pathname
	1 : Filename + extension
	2 : Display name in the file list
Selection	
Selection	The currently selected file (data type: Text; for file lists with single selection) of the selected files (data type. Text array; for lists with multi-selection). Empty text/text array of length 0, if no selection can be found.

Description:

With this function, the current selection in the file list is requested.

Folder names are returned without concluding '\'.

Special case: If the file list is not visible (mode: "Directory tree only") and the tree diagram is configured for display of files, the selection in the tree is requested. This can be either a folder or a file.

Path-/filename vs. displayed name: The name of afile or folder may deviate from the name displayed, for example for special folders in which a localized name is displayed (e.g. Windows with input language "German", displayed name "c:\Programme", actual path: "c:\Program files") or with files in which the file extension mayhave been omitted (depending on the Windows-Explorer setting "Hide file extensions for known [data] types").

Examples:

The currently selected data files in a FileExplorer widget are loaded:

Widget with single selection:

```
TxFileName = PnGetFileSelection( "FileExplorer1", 0)
FileLoad( TxFileName, "", 0)
```

Widget with multiple selection:

```
SelectedFiles = PnGetFileSelection( "FileExplorer1", 0)
FOREACH ELEMENT TxFileName in SelectedFiles
   FileLoad( TxFileName, "", 0)
END
```

See also:

PnGetFolder, PnSetFolder, PnSetFileSelection

PnGetFolder

Scope: Panels

Gets the File Explorer's current folder.

Declaration:

`PnGetFolder ( TxElementName ) -> Folder`

Parameter:

TxElementName	Name of the File Explorer element requested
Folder	
Folder	Current folder

Description:

The actual path in the file system is always returned. This may deviate from the name displayed, for example for special folders in which a localized name is displayed (e.g. Windows with input language "German", displayed name "c:\Programme", actual path: "c:\Program files").

The folder name is returned with a concluding '\'.

If a node is selected which does not correspond to an actual folder (e.g. "This PC" or "Network (places)"), an empty text is returned.

Examples:

The currently choosen folder of a FileExplorer widget will be used as new standard folder for data files:

```
TxFolder = PnGetFolder( "FileExplorer1")
SetOption("Dir.DataFiles", TxFolder)
```

See also:

PnSetFolder, PnGetFileSelection, PnSetFileSelection

PnGetItemCount

Scope: Panels

Determines the number of entries in the specified panel element (listbox, droplist etc.)

Declaration:

```
PnGetItemCount ( TxElementName ) -> Count
```

Parameter:

TxElementName	Name of the element to be queried.
Count	
Count	Number of entries

Applies to:

Listbox, Droplist, Combobox, CCV Selector, Radiogroup

Examples:

In a list box with multi-selection, all selected entries are deleted:

```
Count = PnGetItemCount("list1")
i = Count
WHILE i > 0
   IF PnIsItemSelected("list1", i)
      PnDeleteItem("list1", i)
   END
   i = i - 1
END
```

See also:

PnGetItemText, PnSetItemText, PnInsertItem, PnFindItem, PnDeleteItem

PnGetItemText

Scope: Panels

Returns the text for a list item.

Declaration:

`PnGetItemText ( TxElementName, Index ) -> TxContents`

Parameter:

TxElementName	Name of the element to be queried.
Index	Index of the entry to be queried. The first entry has the index 1.
TxContents	
TxContents	Text for the specified entry

Applies to:

Listbox, Droplist, Combobox, CCV Selector, Radiogroup

Examples:

The currently selected entry is read out of a list containing names of measurement files in FAMOS-format, and the corresponding file is opened.

```
i = PnGetSelectedItem("listFiles")
IF i > 0
   FileName$ = PnGetItemText("listFiles", i)
   fh = FileOpenDSF( FileName$,0)
   IF fh > 0
      ;...
      FileClose(fh)
   END
END
```

See also:

PnGetItemCount, PnSetItemText, PnInsertItem, PnFindItem, PnDeleteItem

PnGetPageCount

Scope: Panels

Determines how many pages there are in the active Panel.

Declaration:

```
PnGetPageCount ( ) -> Pagecount
```

Parameter:

Pagecount	
Pagecount	Page count

Description:

Examples:

The current Panel contains multiple pages which all contain a text box named 'Date' in the text box. All these boxes are updated with the current date.

```
Page = PnGetPageCount()
TxDate = TimeToText( TimeSystem?(), 3)
WHILE Page > 0
   PnSetActivePage(Page)
   PnSetText( "Date", TxDate)
   Page = Page-1
END
```

See also:

PnSetActivePage

PnGetPageIndex

Scope: Panels

For the active panel, the associated page number is determined for a page name.

Declaration:

```
PnGetPageIndex ( PageName ) -> PageNumber
```

Parameter:

PageName	Name (title) of the page to be queried.
PageNumber	
PageNumber	The number (>= 1) of the corresponding page. 0 if no page with the specified name exists.

Description:

Page names in the panel are case-sensitive. The page name must therefore be specified exactly in this regard.

Examples:

In the active panel, the page named "Final report" is exported to a PDF file.

```
PnExportPDF("d:\reports\report.pdf", PnGetPageIndex("Final report"), 0)
```

See also:

PnGetPageName

Supported since:

Version 2022

PnGetPageName

Scope: Panels

For the active panel, the associated page name is determined for a page number.

Declaration:

```
PnGetPageName ( PageNumber ) -> PageName
```

Parameter:

PageNumber	Number of the page to be queried (>= 1).
PageName	
PageName	The name (title) of the page.

Description:

Examples:

Before printing the active page, the user is asked whether he really wants to print the selected page.

```
PageIndex = PnGetActivePage()
PageName = PnGetPageName(PageIndex)
ok = BoxMessage("Print", "Do you really want to print the page [" + PageName + "]?","?2")
IF ok = 1
   PnPrint(PageIndex)
END
```

See also:

PnGetPageIndex

Supported since:

Version 2022

PnGetPosition

Scope: Panels

Queries the specified element's position.

Declaration:

```
PnGetPosition ( TxElementName, Position ) -> Position
```

Parameter:

TxElementName	Name of the Panel element to be queried
Position	Positionsparameter
	0 : X-coodinate of left upper corner
	1 : Y-coodinate of left upper corner
	2 : Width
	3 : Height
Position	
Position	Current position [in millimeter] of the Panel element.

Description:

This function queries the Panel element's position and size on the Panel page.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel page with the name 'Calculation' contains a multiline editbox 'Status', which is shifted 3.3 mm to the top and its height is increased by 3.3 mm. The X-coordinate and the width remain unchanged.

```
X = PnGetPosition("Calculation.Status", 0))
Y = PnGetPosition("Calculation.Status", 1)
Height = PnGetPosition("Calculation.Status", 3)
PnSetPosition("Calculation.Status", X, Y - 3.3, 0, Height + 3.3)
```

See also:

PnSetPosition

PnGetSelectedItem

Scope: Panels

Determines the entry currently selected in a list with single-selection.

Declaration:

`PnGetSelectedItem ( TxElementName ) -> Index`

Parameter:

TxElementName	Name of the element in question
Index	
Index	Index of the selected entry (>=0). 0, if no entry is selected.

Applies to:

Listbox (Single selection), Droplist, Combobox, CCV Selector, Radiogroup

Examples:

The currently selected entry is read out of a list containing names of measurement files in FAMOS-format, and the corresponding file is opened.

```
i = PnGetSelectedItem("listFiles")
IF i > 0
   FileName$ = PnGetItemText("listFiles", i)
   fh = FileOpenDSF( FileName$,0)
   IF fh > 0
      ;...
      FileClose(fh)
   END
END
```

See also:

PnSelectItem, PnIsItemSelected

PnGetSelectedItemCount

Scope: Panels

Determines the number of selected entries in a list with multi-selection.

Declaration:

```
PnGetSelectedItemCount ( TxElementName ) -> Count
```

Parameter:

TxElementName	Name of the element in question
Count	
Count	Number of selected entries. 0, if no entry is selected.

Description:

Applies to:

Listbox (Multiple selection)

Examples:

The selected entries in a list box with multi-selection are to be evaluated at the push of a button. The button is only to be enabled if at least one entry is selected. For this purpose, the amount of entries selected in the event 'Selected' is checked:

--> The list box's event-sequence 'Selected'

```
Count = PnGetSelectedItemCount(PA1)
PnEnable("Button1", Count > 0)
```

See also:

PnIsItemSelected, PnSetItemSelection, PnGetSelectedItem

PnGetText

Scope: Panels

Queries the content or caption of the specified element.

Declaration:

```
PnGetText ( TxElementName ) -> TxText
```

Parameter:

TxElementName	Name of the Panel element to be queried
TxText	
TxText	Current content or labeling of the element

Description:

The function can only be applied to elements which have labeling or whose current state can be expressed by a text. Interpretation of the text depends on the element's type, e.g.:

Element	Meaning
Button	Button caption
Label	Content of the text box
Editbox (single line)	Content of the input box
Droplist	Currently selected entry
Combination box	Content of the input box

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

In a Panel, there is an input box for entering comments on a variable, and a button 'Apply'. When this button is pressed, the current content of the input box is to be entered as commentary in that variable which is currently selected in the Variables list (Measurements) as 1st measurement/1st channel.

Event-sequence 'Pressed' of the 'Apply'-button

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   TxComment = PnGetText("Comment")
   SetComm(<TxVarName>, TxComment)
END
```

See also:

PnGetValue, PnSetText, PnSetActivePage

PnGetValue

Scope: Panels

Queries the specified element's current numerical value.

Declaration:

```
PnGetValue ( TxElementName ) -> Value
```

Parameter:

TxElementName	Name of the Panel element to be queried
Value	
Value	Current value of the Panel-element

Description:

The function can only be applied to Panel elements whose current value can be expressed as a number. The interpretation of the value depends on the element type.

Element	Meaning
Input boxes	If the text in the input box can be converted to a number, this number is returned. Otherwise 0.
Switch	Returns 1 if the switch is pressed or 'checked', else 0.
Radiogroup	Returns the index of the selected option.
RangeEdit, TimeSpan	Returns the lower limit of the range set.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel has a page 'Filter', used for setting low-pass filter parameters. Two input boxes 'Order' and 'Input_Freq' serve to make the specifications for the filter order and the cutoff frequency. A switch 'Bessel' specifies whether to perform the calculation with Butterworth- or Bessel-characteristics (switch ON). When loading the Panel, the boxes are initialized with default values. Upon pressing the button, filtering is then performed for the current selection in the Variables list (Measurements), here: 1st channel/1st measurement.

Event-sequence 'Panel initialization'

```
PnSetValue("Filter.Order", 4)
PnSetValue("Filter.Input_Freq", 100)
PnSetValue("Filter.Bessel", 1)
```

Event-sequence 'Pressed' of the 'Filter!'-button

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   Order = PnGetValue( "Order")
   Freq  = PnGetValue("Input_Freq")
   Type  = PnGetValue("Bessel")
   IF Order > 1 AND Freq > 0
      IF Type = 1  ; Bessel
         filtrat = FiltLP( <TxVarName>, 1, 0, Order, Freq)
      ELSE           ; Butterworth
         filtrat = FiltLP( <TxVarName>, 0, 0, Order, Freq)
      END
   END
END
```

See also:

PnSetValue, PnGetText

PnGetValue2

Scope: Panels

Queries the current 2nd numerical value of the element specified.

Declaration:

`PnGetValue2 ( TxElementName ) -> Value`

Parameter:

TxElementName	Name of the Panel element to be queried
Value	
Value	Current 2nd value of Panel-element

Description:

This function can only apply to such Panel element whose current state can be expressed by two numbers.

Element	Meaning
TimeSpan	Returns the upper boundary of the range selected in the imc FAMOS time format.
RangeEdit	Returns the upper boundary of the range selected.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

TimeSpan, RangeEdit

Examples:

A Panel contains a Timespan-widget "dateRange" and a list box "list". The list box presents all files belonging to a given folder, whose file creation time lies within the range specified by "dateRange". A button "Update!" causes the currently set date range to be read out and the file list to be updated.

Event-sequence 'Pressed' of the 'Update!'-button

```
t_min = PnGetValue("dateRange")
t_max = PnGetValue2("dateRange")

PnDeleteItem("list", 0)
fileList = FsGetFileNames("c:\data", "*.*", 0, 0, 1)
FOREACH ELEMENT path in fileList
   t_file = FsGetFileTime(path)
   IF t_file >= t_min AND t_file <= t_max
      PnInsertItem("list", 0, FsSplitPath(path, 4), 0)
   END
END
```

See also:

PnGetValue, PnSetValue, PnSetValue2

PnInsertItem

Scope: Panels

Adds new entries to an element (list box, pop-down list, etc.).

Declaration:

```
PnInsertItem ( TxElementName, Index, TxContents, Option )
```

Parameter:

TxElementName	Name of the element to be changed
Index	Insert position. The first entry has the index 1. To append the entry to the end, enter a 0.
TxContents	New entry/entries. The data types allowed are Text and Text array.
Option	Option parameter
	0 : Default
	1 : The new entry is scrolled into view (if necessary).

Description:

If a text array is specified as the 3rd parameter, all elements of the text array are inserted in succession.

When the list is sorted, the given position will be ignored and the new entry is inserted.according to the selected sort order.

Applies to:

Listbox, Droplist, ComboBox

Examples:

A list box is filled with the names of all files located in the specified folder.

```
Dir$ = FsDlgSelectDirectory("Select folder", "", 0)
FileListID = FsFileListNew( Dir$, "*.*", 0, 0, 0)
FileCount = FsFileListGetCount(FileListID)
i = 1
WHILE i <= FileCount
   File$ = FsSplitPath(FsFileListGetName(FileListID, i), 4)
   PnInsertItem("listFiles", i, File$, 0)
   i = i + 1
END
FsFileListClose(FileListID)
```

Same task, but more effective due to the use of a text array variable:

```
Dir$ = FsDlgSelectDirectory("Select folder", "", 0)
Files$ = FsGetFileNames( Dir$,"*.*", 0, 0, 0)
Files$ = TxRegExMatch( Files$, "^(.+)\\([^/]+)$", "", 0, 2)
PnInsertItem("listFiles", 0, Files$, 0)
```

A list box is filled with the names of all variables which are in the FAMOS variables list when the program is started. Subsequently, the first entry is selected.

```
Count = VarGetInit(0)
i = 1
WHILE i <= Count
   PnInsertItem("ListVariables", i, VarGetName?(i), 0)
   i = i + 1
END
PnSelectItem("ListVariables", 1)
```

A list box is filled with all values belonging to a (short) data set.

```
Count = leng?(MyData)
i = 1
WHILE i <= Count
   TxVal = TForm( MyData[i], "")
   PnInsertItem("list1", 0, TxVal, 0)
   i = i + 1
END
```

See also:

PnGetItemCount, PnGetItemText, PnSetItemText, PnFindItem, PnDeleteItem

PnInsertPage

Scope: Panels

A new page is inserted into the active Panel.

Declaration:

```
PnInsertPage ( TxTemplateFile, PageNumber, PageName, InsertPosition ) -> Success
```

Parameter:

TxTemplateFile	Filename of the Panel file containing the template for the new page to insert. If the parameter is empty, the current Panel is used.
PageNumber	Determines which page from [TxTemplateFile] to use.
PageName	The name of the new page. If a page with the same name already exists, the function returns an error message. If an empty string is returned, the name of the new page is generated automatically.
InsertPosition	The new page is placed at the position specified here. A 0 means that it is appended at the end.
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

Using this function, a new page can be added to the current Panel. The new page can either be another page's duplicate or be imported from another Panel file.

If no complete path is specified with the filename of the template file, the system searches for the file in these folders according to this order:

- Project folder: When a project is active, the search is conducted initially in the current project's folder.
- Default folder for Panel-files: FAMOS-presettings for Panels/dialogs/sequences

Examples:

The data set "u0" is to be documented in the form of a 2-column table (x,y). The Panel file 'table_template.panel' serves as the page template, which contains, among other things, a table object with 50 lines. According to how long the data set is, the required amount of pages is generated and subsequently saved under a new name.

```
Length = leng?(u0)
Rows = 50
FirstSample = 1
WHILE FirstSample <= Length
   IF FirstSample = 1
      ; Open template
      PnLoad("table_template")
   ELSE
      ; append new page
      PnInsertPage("table_template", 1, "", 0)
   END
   ; cut y-files for table
   y = CutIndex( u0, FirstSample, FirstSample+Rows-1)
   PnTableSetColumn( "table1", 2, 1, y)
   ; build x-Data for table
   x = Ramp( (FirstSample-1)*xdel?(u0)+ xoff?(u0), xdel?(u0), leng?(y))
   PnTableSetColumn( "table1", 1, 1, x)
   FirstSample = FirstSample+Rows
END
PnPrint(0)
PnClose(0)
```

See also:

PnLoad, PnSave

PnIsItemSelected

Scope: Panels

Determines whether an entry in the list is selected

Declaration:

```
PnIsItemSelected ( TxElementName, Index ) -> IsSelected
```

Parameter:

TxElementName	Name of the element in question
Index	Index of the entry to be verified. the first entry has the index 1.
IsSelected	
IsSelected	The entry's selection status
	0 : Not selected
	1 : Selected

Applies to:

Listbox (Multiple selection)

Examples:

A listbox in a panel is filled with all of the values of a (short) data set. At the push of a button, all selected values are set to 0.

--> Event-sequence 'Panel initialization'

```
;Fills the list box with the data set's values:
Count = leng?(MyData)
i = 1
WHILE i <= Count
   TxVal = TForm( MyData[i], "")
   PnInsertItem("list1", 0, TxVal, 0)
   i = i + 1
END
```

--> Event sequence 'Button pressed'

```
;The selected samples are set to 0.
Count = PnGetItemCount("list1")
i = 1
WHILE i <= Count
   IF PnIsItemSelected("list1", i)
      MyData[i] = 0
      PnSetItemText("list1", 1, "0")
   END
   i = i + 1
END
```

See also:

PnGetSelectedItemCount, PnSetItemSelection, PnGetSelectedItem

PnLoad

A Panel-file is loaded and displayed.

Declaration:

```
PnLoad ( TxFilename ) -> Success
```

Parameter:

TxFilename	Name of the Panel file to be opened
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

Loads the Panel-file specified.

If the specified filename has no name extension, then the system assumes ".panel".

If no complete path is specified with the filename, the system searches for the file in this sequence of folders:

- Project folder: When a project is active, the search is conducted initially in the current project's folder.
- Default folder for Panel-files: FAMOS presettings for Panels/dialogs/sequences

To start a Panel in Dialog-mode, use the command Dialog().

This function was introduced with FAMOS V2022 and replaces the function DbLoadPanel() of older versions.

Multithreading: All functions for Panel remote control can be called anywhere and have a global effect. The Panel loaded here can therefore be used from all execution threads.

Examples:

A Panel file is opened. A variety of updates are performed, after which the Panel is printed and then closed again.

```
err = PnLoad("d:\templates\result.panel")
IF err <> 0
   BoxMessage("Error", GetLastError(), "!1")
ELSE
   ; various updates
   ; ...
   PnPrint(0)
   PnClose(0)
END
```

See also:

PnClose, Dialog

Supported since:

Version 2022

PnPrint

Scope: Panels

Prints the active Panel

Declaration:

```
PnPrint ( PageSelection )
```

Parameter:

PageSelection	Selection of page(s) to print
	-1 : The active page is printed
	0 : The entire Panel (all pages) is printed.
	>=1 : Page number; only the specified page is printed.

Description:

Prints the active Panel. The current printer settings ("File"/"Print") are taken into account.

Examples:

A Panel page contains a button 'Print'. When the button is pressed by the user, then in the associated event-sequence a text box on the same page is first filled with the current date and time, and next this page is exported in the PNG-format

-> Event-sequence 'Pressed' of the 'Print'-button

```
TxDate = TimeToText( TimeSystem?(), 3)
PnSetText( "Date", "Printed: " + TxDate)
PnPrint(-1)
```

See also:

PnExportPDF, PnExportGraphics

PnRemovePage

Scope: Panels

A page in the active Panel is deleted.

Declaration:

`PnRemovePage ( PageNumberOrTitle ) -> Success`

Parameter:

PageNumberOrTitle	Specifies the page number (1..) or the caption of the page to be deleted.
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

By means of this function, a page in the active Panel is deleted. The page to be deleted can be specified by either its name or its index.

Examples:

Deletes the page having the name "Intro":

```
PnRemovePage("Intro")
```

A Panel contains 10 pages for the display of a maximum of 10 channels to be loaded from a file.

Any superfluous pages are deleted, depending on the count of channels actually present:

```
fh = FileOpenDSF(MyFileName, 0)
channelCount = FileObjNum?(fh)
...
FOR i = 10 TO channelCount+1 STEP -1
   PnRemovePage(i)
END
```

See also:

PnShowPage

PnSave

Scope: Panels

Saves the active Panel

Declaration:

```
PnSave ( TxFileName ) -> Success
```

Parameter:

TxFileName	Filename, under which the Panel is to be saved
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

The active Panel is saved under the specified filename.

If the specifed filename doesn't have any name extension, '.panel' is appended.

If no full pathname is specified, then if there is an active project the current project folder is used. Otherwise, the default folder for Panel file specified in the FAMOS presettings is used.

Examples:

The data set "u0" is to be documented in the form of a 2-column table (x,y). The Panel file 'table_template.panel' serves as the page template, which contains, among other things, a table object with 50 lines. According to how long the data set is, the required amount of pages is generated and subsequently saved under a new name.

```
Length = leng?(u0)
Rows = 50
FirstSample = 1
WHILE FirstSample <= Length
   IF FirstSample = 1
      ; Open template
      PnLoad("table_template")
   ELSE
      ; append new page
      PnInsertPage("table_template", 1, "", 0)
   END
   ; cut y-files for table
   y = CutIndex( u0, FirstSample, FirstSample+Rows-1)
   PnTableSetColumn( "table1", 2, 1, y)
   ; build x-Data for table
   x = Ramp( (FirstSample-1)*xdel?(u0)+ xoff?(u0), xdel?(u0), leng?(y))
   PnTableSetColumn( "table1", 1, 1, x)
   FirstSample = FirstSample+Rows
END
TxFileName = BoxText?("Select filename", "", 0)
PnSave(TxFileName)
PnClose(0)
```

See also:

PnLoad, PnExportPDF

PnSelectItem

Scope: Panels

Selects an entry in a list with single-selection

Declaration:

```
PnSelectItem ( TxElementName, Index )
```

Parameter:

TxElementName	Name of the element.
Index	Index of the entry to be selected. The first entry has the index 1.

Applies to:

Listbox (Single selection), Droplist, Combobox, CCV Selector, Radiogroup

Examples:

Looks for an entry in a list box (with single-selection). If the entry exists, it is selected and scrolled into view, if necessary.

```
i = PnFindItem("list", "100.0")
IF i > 0
   PnSelectItem("list", i)
END
```

See also:

PnGetSelectedItem, PnSetItemSelection

PnSetActivePage

Scope: Panels

Determines the active (visible) page.

Declaration:

```
PnSetActivePage ( PageNumber )
```

Parameter:

PageNumber	Determines the page number (1..) of the page to be activated.

Description:

Multithreading: The functions for Panel remote control can be called anywhere and have a global effect. The Page selected here is therefore valid for all execution threads.

Examples:

The current Panel contains multiple pages which all contain a text box named 'Date' in the text box. All these boxes are updated with the current date.

```
Page = PnGetPageCount()
TxDate = TimeToText( TimeSystem?(), 3)
WHILE Page > 0
   PnSetActivePage(Page)
   PnSetText( "Date", TxDate)
   Page = Page-1
END
```

See also:

PnGetPageCount

PnSetFileSelection

Scope: Panels

Sets the selection in a file explorer's file list.

Declaration:

`PnSetFileSelection ( TxElementName, TxNewSelection, Format ) -> Success`

Parameter:

TxElementName	Name of the File Explorer element to be changed
TxNewSelection	Filenames to select. Data type Text or Text Array (if the element multi-selection is supported). Empty text, in order to delete the selection.
Format	Specifies in what format the new selection is stated.
	0 : Pathname. Either the complete path or only filename + extension can be specified.
	2 : Display name in the file list
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

The function sets the current selection in the file list. Both files and subfolders can be selected. These must be present in the current folder and currently visible, i.e. no changing of the current folder occurs. If the file list is configured for multi-selection, it is also possible to pass a text array with multiple files to be selected.

Special case: If the file list is not visible (mode: "Directory tree only") and the tree diagram is configured for display of files, the selection is performed in the tree. In this case, only the option 0 (complete filename) is permitted. If the filename is valid, the tree is expanded accordingly and the file is selected; if applicable, the current folder is changed.

Path-/filename vs. displayed name: The name of afile or folder may deviate from the name displayed, for example for special folders in which a localized name is displayed (e.g. Windows with input language "German", displayed name "c:\Programme", actual path: "c:\Program files") or with files in which the file extension mayhave been omitted (depending on the Windows-Explorer setting "Hide file extensions for known [data] types").

Examples:

The current folder for a FileExplorer widget is set. All files with the extension "raw" are selected.

```
DatFolder = "c:\Experiment001\data"
AllDatFiles = FsGetFileNames( DatFolder, "*.raw", 0, 0, 0)
PnSetFolder( "FileExplorer1", DatFolder)
PnSetFileSelection( "FileExplorer1", AllDatFiles, 0)
```

See also:

PnGetFolder, PnSetFolder, PnSetFileSelection

PnSetFolder

Scope: Panels

Sets the file Explorer's current folder to the path specified.

Declaration:

```
PnSetFolder ( TxElementName, TxNewFolder ) -> Success
```

Parameter:

TxElementName	Name of the File Explorer element to be changed
TxNewFolder	Full pathname of the desired folder
Success	
Success	Successful function execution. 0 if the function performs sucessfully, -1 in case of error. In case of error, the cause can be determined using the function GetLastError().

Description:

The function sets the File Explorer's current folder to the path specified.

It is necessary to always specify the actual path in the file system. This may deviate from the name displayed, for example for special folders in which a localized name is displayed (e.g. Windows with input language "German", displayed name "c:\Programme", actual path: "c:\Program files").

Note:To set the root folder (the initial folder for the folder hierarchy displayed), use the function PnSetProperty(.., "Rootfolder", ..).

Examples:

The current folder of a FileExplorer widget is set to the standard path for data files in FAMOS:

```
TxFolder = GetOption("Dir.DataFiles")
PnSetFolder( "FileExplorer1", TxFolder)
```

See also:

PnGetFolder, PnGetFileSelection, PnSetFileSelection, PnSetProperty

PnSetItemSelection

Scope: Panels

Sets of cancels the selection for an entry in a list with multi-selection.

Declaration:

```
PnSetItemSelection ( TxElementName, Index, OnOff )
```

Parameter:

TxElementName	Name of the element in question
Index	Index of the entry to be changed. The first entry has the index 1. In order to (de-)select all entries, enter 0.
OnOff	Selection On/ Off
	0 : Cancel selection
	1 : Selects an entry

Applies to:

Listbox (Multiple selection)

Examples:

Inverts the current selection in a list with multi-selection:

```
Count = PnGetItemCount("list1")
i = 1
   WHILE i <= Count
   Sel = PnIsItemSelected("list1", i)
   PnSetItemSelection("list1", i, NOT(Sel))
   i = i + 1
END
```

See also:

PnGetSelectedItemCount, PnIsItemSelected, PnSelectItem

PnSetItemText

Scope: Panels

Replaces an entry (listbox, droplist etc.) with the specified text.

Declaration:

`PnSetItemText ( TxElementName, Index, TxNewContents )`

Parameter:

TxElementName	Name of the element to be changed
Index	Index of the entry to be replaced. The first entry has the index 1.
TxNewContents	New text for the entry to be replaced

Applies to:

Listbox, Droplist, Combobox, Radiogroup

Examples:

A listbox in a panel is filled with all of the values of a (short) data set. At the push of a button, all selected values are set to 0.

--> Event-sequence 'Panel initialization'

```
;Fills the list box with the data set's values:
Count = leng?(MyData)
i = 1
WHILE i <= Count
   TxVal = TForm( MyData[i], "")
   PnInsertItem("list1", 0, TxVal, 0)
   i = i + 1
END
```

--> Event sequence 'Button pressed'

```
;The selected samples are set to 0.
Count = PnGetItemCount("list1")
i = 1
WHILE i <= Count
   IF PnIsItemSelected("list1", i)
      MyData[i] = 0
      PnSetItemText("list1", i, "0")
   END
   i = i + 1
END
```

See also:

PnGetItemCount, PnGetItemText, PnInsertItem, PnFindItem, PnDeleteItem

PnSetPosition

Scope: Panels

Sets the position und the size for the specified element.

Declaration:

```
PnSetPosition ( TxElementName, left, top, wide, high )
```

Parameter:

TxElementName	Name of the Panel element to be displayed.
left	X-coodinate of left upper corner
top	Y-coodinate of left upper corner
wide	Width
high	Height

Description:

With this function the position and size of a Panel-Elemente can be set. These values are given in millimeters.

If [wide] oder [high] are 0, the current width and height remain unchanged and only the position is changed.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel page with the name 'Calculation' contains a multiline editbox 'Status', which is shifted 3.3 mm to the top and its height is increased by 3.3 mm. The X-coordinate and the width remain unchanged.

```
X = PnGetPosition("Calculation.Status", 0))
Y = PnGetPosition("Calculation.Status", 1)
Height = PnGetPosition("Calculation.Status", 3)
PnSetPosition("Calculation.Status", X, Y - 3.3, 0, Height + 3.3)
```

See also:

PnGetPosition, PnShow

PnSetProperty

Scope: Panels

A property of the selected element is reset

Declaration:

```
PnSetProperty ( TxElementName, TxPropID, Value )
```

Parameter:

TxElementName	Name of the Panel element
TxPropID	Identifier of the property to set
	"FillColor" : Background color
	"FrameColor" : Frame color
	"TextColor" : Text color
	"Image" : Image file
	"RootFolder" : Root folder
	"FileFilter" : File filter
	"Minimum" : Minimum
	"Maximum" : Maximum
	"Increment" : Stepwidth
	"Resolution" : Resolution
	"SortOrder" : Sort order
	"SortColumn" : Sort column
Value	Numerical value or string to which the property is to be set

Description:

The property to set is delected by pre-defined identifiers. The adjustable properties depend on the type of the PanelPanel element.

The following properties have been defined thus far:

Identifier	Meaning	Applicable to:
"FillColor"	Background color (RGB-value)	All elements with background color
"FrameColor"	Frame color (RGB-value)	All elements with colored frame
"TextColor"	Text color (RGB-value)	Elements with text display
"Image"	Image file	'Image' element
"RootFolder"	Root folder	'FileExplorer' element
"FileFilter"	File filter	'FileExplorer' element (e.g. "*.raw" or "*.raw;*.dat")
"Minimum"	Minimum of the range to be set	'Slider' element, 'Spin', 'RangeEdit', 'TimeSpan'
"Maximum"	Maximum of the range to be set	'Slider' element, 'Spin', 'RangeEdit', 'TimeSpan'
"Increment"	Stepwidth in conjunction with value changes	'Spin' element
"Resolution"	Resolution of the value- or time-range	'RangeEdit' element, 'TimeSpan' (0: seconds; 1: minutes; 2: hours; 3: days; 4: week; 5: month; 6: year)
"SortOrder"	Sort order	'FileExplorer' element (1: ascending 2: descending)
"SortColumn"	Index of the column to be sorted	'FileExplorer' element

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event sequence and the event can be assigne to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel page contains a text box 'Max' for displaying the maximum value of the currently selected variable (1st measurement, 1st channel). Upon every change of the selection in the Measurement or Channel list, the box is updated accordingly. If the maximum value exceeds a certain limit, the box's background color changes to red.

Event-sequence 'Data selection changed''

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   maxval = Max(<TxVarName>)
   PnSetValue("Page1.Max", maxval )
   IF maxval > 10
      ; Limit exceeded (red background)
      PnSetProperty("Page1.Max", "FillColor", RGB( 255, 0, 0)))
   ELSE
      ; Default (white background)
      PnSetProperty("Page1.Max", "FillColor", RGB( 255, 255, 255)))
   END
END
```

In an experiment setup, multiple trials were run, each of whose results were saved in separate folders on the hard drive. Additionally, each folder contains a photo of the setup at the time of measurement, under the name "test.jpg". The data are now subsequently visualized in a one-page Panel which in addition to a curve window also contains an image object 'pic'. The associated photo is to be displayed with the channel (1.Kanal/1.Messung) display.

Event sequence 'Data selection changed''

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   ; name of the associated file
   FileName = FileName?(<TxVarName>)
   IF FileName <> ""
      ; construct filename for photo file
      PicFileName = FsGetParentDirectoryName(FileName)+ "\test.jpg"
      PnSetProperty( "pic", "Image", PicFileName)
   END
END
```

See also:

PnGetValue, PnGetText

PnSetText

Scope: Panels

Sets a new content or caption of the specified element.

Declaration:

```
PnSetText ( TxElementName, TxText )
```

Parameter:

TxElementName	Name of the Panel element to be changed.
TxText	Text to which the element is to be set

Description:

The function can only be applied to elements which have labeling or whose current state can be expressed by a text. Interpretation of the text depends on the element type, e.g.:

Element	Meaning
Button	Button caption
Label	Content of the text box
Editbox (single line)	Content of the input box
Droplist	Sets the selection to the specified text (which must be available in the selection list).
Combination box	Content of the input box

The function can not be applied to elements which are linked to a variable.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel page contains various text boxes which display some characteristic values of a currently selected variable (1st measurement, 1st channel). Upon every change of the selection in the Measurement or Channel list, the boxes are updated accordingly.

Event-sequence 'Data selection changed''

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   PnSetText("Page1.Name", TxVarName)
   PnSetText("Page1.Comment", Comm?(<TxVarName>))
   PnSetText("Page1.YUnit", Unit?(<TxVarName>, 1))
   PnSetText("Page1.XUnit", Unit?(<TxVarName>, 0))
END
```

See also:

PnGetText, PnSetValue

PnSetTimer

Scope: Panels

The timer associated with a Panel page is started/ reconfigured/ stopped.

Declaration:

```
PnSetTimer ( PageNumber, Interval )
```

Parameter:

PageNumber	Determines the page number (1..) of the timer to be controlled.
Interval	States the interval (in seconds) at which the timer is to 'tick'. Set to 0 in order to stop the timer.

Description:

For each Panel page, it is possible to activate a timer which works at a user-specified time interval. After each elapse of the interval, the "Timer"-event is generated, which triggers the running of an event-sequence defined to correspond to it.

To use the timer for a page, proceed as folllows:

- Activate the time during the page's Design stage (set property 'Timer' to 'active')
- In the FAMOS Sequence Editor you will find the event-sequence 'Timer' for the corresponding page. Here, enter the sequence of commnads to be run cyclically.
- Starts the timer with the desired time interval by means of the function PnSetTimer. For instance, you can start the timer automatically already when opening the Panel (call the function in the event-sequence 'Panel Initialization'), or the user can start it manually by clicking on a button (event-sequence 'Button Pressed').

FAMOS attempts to conform to the interval set as well as possible, but can not guarantee that the event will be triggered at exactly the clock rate specified. If either the operating system or FAMOS is currently busy with other tasks, then generation of the timer event may be subject to unpredictable delays. In particular, the event-sequence 'Timer' can only be performed if FAMOS is not currently (or no longer) performing any other sequences. If Timer events occur while a different sequence is running, the event-sequence asigned to the Timer are performed exactly one time, as soon as FAMOS has completed running the current sequence.

In consequence, it is clear that no time-consuming routines should be programmed to run within one Timer event-sequence. For example, a timer with a clock rate of 1s is practically impossible to realize if it takes more than 1s to run the associated evet-sequence.

Examples:

On a Panel page, measurement of the preceding data set 'speed' is to be simulated and the growing data set is to be displayed in a curve window. Along with the curve window, the page is also supplied with a button to use for starting/stopping the simulation; the button caption alternates between 'Start' and 'Stop'.

The curve window displays a temporary file 'Speed_Sim', to which the respective values of the original data set are appended cyclically (governed by the timer).

Event-sequence 'Panel Initialization'

```
SimIsRunning = 0
Speed_Sim = EMPTY
```

Event-sequence 'Pressed' belonging to the 'Start/Stop'-button

```
IF NOT(SimIsRunning)
   ; Start
   Speed_Sim = EMPTY
   PnSetTimer(1, 0.5) ; Update-Intervall 500ms
   SimIsRunning = 1
   PnSetText( "Button1", "Stop")
ELSE
   ; Stop
   PnSetTimer(1, 0)
   SimIsRunning = 0
   PnSetText( "Button1", "Start")
END
```

Event-sequence 'Timer'

```
ori_len= leng?(Speed)
next_index= leng?(Speed_Sim)+1
IF next_index <= ori_len
   ; appending 5 new values every 0.5s...
   Speed_Sim = Join( Speed_Sim, CutIndex( Speed, next_index, LowerValue( next_index+5, ori_len)))
ELSE
   ; Done, stopping timer...
   PnSetTimer( 1, 0)
   SimIsRunning = 0
   PnSetText( "Button1", "Start")
END
```

PnSetValue

Scope: Panels

Sets a new value for the specified element.

Declaration:

```
PnSetValue ( TxElementName, Value )
```

Parameter:

TxElementName	Name of the Panel-element to set
Value	Numerical value to which the Panel-element is to be set

Description:

The function can only be used on Panel elements whose current state can be expressed as a number. The interpretation of the value depends on the element type, e.g.:

Element	Meaning
Label	The numerical value is entered as the content of the text box.
Switch	With a 0, the status of the switch is set to "Off", otherwise to "On".
Radiogroup	The option with the specified index is selected.
RangeEdit, TimeSpan	Specifies the lower boundary of the range set.

The function can not be applied to elements linked to a variable.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event sequence and the event can be assigne to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel has a page 'Filter', used for setting low-pass filter parameters. Two input boxes 'Order' and 'Input_Freq' serve to make the specifications for the filter order and the cutoff frequency. A switch 'Bessel' specifies whether to perform the calculation with Butterworth- or Bessel-characteristics (switch ON). When loading the Panel, the boxes are initialized with default values. Upon pressing the button, filtering is then performed for the current selection in the Variables list (Measurements), here: 1st channel/1st measurement.

Event-sequence 'Panel initialization'

```
PnSetValue("Filter.Order", 4)
PnSetValue("Filter.Input_Freq", 100)
PnSetValue("Filter.Bessel", 1)
```

Event-sequence 'Pressed' of the 'Filter!'-button

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   Order = PnGetValue( "Order")
   Freq  = PnGetValue("Input_Freq")
   Type  = PnGetValue("Bessel")
   IF Order > 1 AND Freq > 0
      IF Type = 1  ; Bessel
         filtrat = FiltLP( <TxVarName>, 1, 0, Order, Freq)
      ELSE           ; Butterworth
         filtrat = FiltLP( <TxVarName>, 0, 0, Order, Freq)
      END
   END
END
```

See also:

PnGetValue, PnGetText, PnSetText

PnSetValue2

Scope: Panels

Sets the 2nd numerical value of the element specified.

Declaration:

```
PnSetValue2 ( TxElementName, Value )
```

Parameter:

TxElementName	Name of the Panel-element to set
Value	Numerical value to which the Panel-element is to be set

Description:

The function can only be applied to such Panel-elements whose current state can be expressed by two numbers.

Element	Meaning
TimeSpan	The value denotes the upper limit of the range selected. The time value provided must be expressed in the imc FAMOS time format which is generated by such functions as Time?(), TimeSystem?() and TimeJoin().
RangeEdit	This value specifies the upper boundary of the range selected.

The function can not be applied to elements linked to a variable.

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event sequence and the event can be assigne to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

TimeSpan, RangeEdit

Examples:

A range is to be excised out of a data set 'channel1'. The indices for the beginning and end end of the excerpt are specified by a range selection widget 'Range'. After loading the data set, the widget is initialized to correspond to the maximum allowable value for the data set length (= index of last value). The initial selection set is the middle fifth of the data set.

```
len = Leng?(channel1)
PnSetProperty("Range", "Minimum", 1)
PnSetProperty("Range", "Maximum", len)
PnSetValue("Range", floor(len*0.4))
PnSetValue2("Range", floor(len*0.6))
```

See also:

PnSetValue, PnGetValue2

PnShow

Scope: Panels

Controls the visibility of the specified element

Declaration:

```
PnShow ( TxElementName, Task )
```

Parameter:

TxElementName	Name of the Panel element to be displayed.
Task	Task
	0 : Hide element
	1 : Show element

Description:

Selection of the element to be addressed can either take the form [page name].[element name], or by using only the element name. If the page is not stated explicitly, the system seraches for the element as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Examples:

A Panel page with the name 'Filter' contains a button 'Execute', with which a currently selected chanel (1st measurement/1st channell) is to be filtered. If no such channel is selected in either the Measurement or Channel lists, the button is hidden.

Event sequence 'Data selection changed''

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   PnShow("Filter.Execute", 1)
ELSE
   PnShow("Filter.Execute", 0)
END
```

See also:

PnEnable

PnShowPage

Scope: Panels

Controls the visibility of a panel page.

Declaration:

`PnShowPage ( PageNumberOrTitle, Task )`

Parameter:

PageNumberOrTitle	Determines the page number (1..) of the title of the page to be controlled.
Task	Task
	0 : Hide page
	1 : Show page disabled
	2 : Show page normally

Description:

Examples:

After a panel having many pages is loaded, only the first 3 pages are displayed initially. Any additional page is blocked.

Event-sequence 'Panel Initialization'

```
PnShowPage( "Report", 1)
FOR iPage = 4 TO PnGetPageCount()
   PnShowPage( iPage, 0)
END
```

See also:

PnGetPageCount, PnSetActivePage

PnTableColumns?

Scope: Panels

Finds the number of columns in a table

Declaration:

```
PnTableColumns? ( TxTableName ) -> Result
```

Parameter:

TxTableName	Name of the Panel table to be queried
Result	
Result	Number of columns

Description:

Finds the number of columns in the specified table of the active Panel.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table, Datagrid

Examples:

The columns of a table are filled with the individual events of an event-based data set.

```
Count    = PnTableColumns?( "Tab1")
CountEvn = EventNum?( Data)
IF CountEvn < Count
   Count = CountEvn
END
i = 1
WHILE i <= Count
   PnTableSetColumn("Tab1",i,2, Daten[i])
   i = i + 1
END
```

See also:

PnTableRows?, PnTableSetCell, PnTableSetColumn, PnTableSetRow

PnTableGetCellText

Scope: Panels

Queries the content of a table cell.

Declaration:

```
PnTableGetCellText ( TxTableName, Column, Row ) -> TxContents
```

Parameter:

TxTableName	Name of the Panel table to be queried
Column	Column (1..)
Row	Row (1..)
TxContents	
TxContents	Content of the specified table cell

Description:

Gets the content of a cell in the specified table of the active Panel.

The inputs for the row and column determine the position of the cell to be read; the top left is [1,1]

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table

Examples:

Gets the content of a table cell (top left). This contains a placeholder "xxx" fot the name. The placeholder is replaced and updated in the table cell.

```
Tx$= PnTableGetCellText( "Tab1", 1, 1, 0)
Tx$= TErsetze( Tx$,"xxx","Heinz Muster")
err= PnTableSetCell( "Tab1", 1, 1, Tx$, 0 )
```

See also:

PnTableSetColumn, PnTableSetRow, PnTableSetCell

PnTableGetCellValue

Scope: Panels

Queries the numerical value of a table cell

Declaration:

```
PnTableGetCellValue ( TxTableName, Column, Row ) -> Value
```

Parameter:

TxTableName	Name of the Panel table to be queried
Column	Column (1..)
Row	Row (1..)
Value	
Value	Value in the specified table cell

Description:

Gets the content of a cell in the specified table of the active Panel.

The inputs for the row and column determine the position of the cell to be read; the top left is [1,1]

If the content can not be converted to a number, a 0 is returned.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table

Examples:

Gets the content of a table cell (top left). This contains a placeholder "xxx" fot the name. The placeholder is replaced and updated in the table cell.

The inputs for the row and column determine the position of the first cell to be queried; the top left is [1,1]

```
Tx$= PnTableGetCellText( "Tab1", 1, 1, 0)
Tx$= TErsetze( Tx$,"xxx","Heinz Muster")
PnTableSetCell( "Tab1", 1, 1, Tx$, 0 )
```

See also:

PnTableSetColumn, PnTableSetRow, PnTableSetCell

PnTableGetSelectedRows

Scope: Panels

Finds the currently selected rows in a Datagrid.

Declaration:

`PnTableGetSelectedRows ( TxElementName ) -> Result`

Parameter:

TxElementName	Name of the Panel element
Result	
Result	Data set containing the indices of the selected rows

Description:

The data set retuned can have as its length 0 (no row selected), 1 (one row selected) or > 1 (for multi-selection).

The indices found refer to the absolute position in the data set displayed. The visible index of the selected row may deviate in some circumstances, for instance if sorting had been applied to the table.

The order of the indices found corresponds to the visible order of the entries selected in the table, from top to bottom.

The index of the first row is 1.

The selection of the table to be accessed may either be made in the form [PageName].[ElementName], or be made via only the element name. If the page is not specified explicitly, the system searches for the element as follows:

- If the function is called within an event sequence and the event can be assigned to a Page (e.g. a botton's event 'Pressed'), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Datagrid

Examples:

A Panel-page contains a 2-column data grid 'tab'. Here, the data sets 'channel_x' and 'channel_y' are displayed. When confirmed by clicking on the button 'Execute!', the currently selected columns are imported and used to construct a new XY-data set.

Event-sequence 'Panel Initialization'

```
PnTableSetColumn("tab1", 1, 1, channel_x)
PnTableSetColumn("tab1", 2, 1, channel_y)
```

Event-sequence 'Pressed' of the 'Execute!'-button

```
sel = PnTableGetSelectedRows("tab1")
IF leng?(sel) = 0
   BoxMessage("Error", "Please first select the desired rows in the table!", "!1")
ELSE
   sel = Sort(sel, 1) ; if table is sortable
   x = EMPTY
   y = EMPTY
   FOR I = 1 TO leng?(sel)
      index = sel[I]
      x = Join(x, Channel_x[index])
      y = Join(y, Channel_y[index])
   END
   Result = xyOf(x,y)
END
```

See also:

PnTableSetColumn

PnTableRows?

Scope: Panels

Finds the number of rows in a table

Declaration:

`PnTableRows? ( TxTableName ) -> Result`

Parameter:

TxTableName	Name of the Panel table to be queried
Result	
Result	Number of rows

Description:

Finds the number of rows in the specified table of the active Panel.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table, Datagrid

Examples:

The rows of a table are filled with the individual segments of a segment-based data set.

```
Count     = PnTableRows?( "Tab1")
CountSeg = leng?( Data) / SegLen?( Data)
IF CountSeg < Count
   Count = CountSeg
END
i = 1
WHILE i <= Count
   PnTableSetRow ("Tab1",1,i, Data[i])
   i = i + 1
END
```

See also:

PnTableColumns?, PnTableSetCell, PnTableSetColumn, PnTableSetRow

PnTableSetCell

Scope: Panels

Sets the content of a table cell

Declaration:

```
PnTableSetCell ( TxTableName, Column, Row, Contents )
```

Parameter:

TxTableName	Name of the Panel table to be controlled.
Column	Column (1..)
Row	Row (1..) or 0 for column header
Contents	Number or string to transfer

Description:

A cell of the specified table in the active panel is occupied with a string or a single value.

The inputs for the row and column determine the position of the cell to which to write data; the top left is [1,1]

If a data set is specified for [Content] which is over 1 in length, the data set's last value is used.

The function can not be applied to table cells which are linked to a variable.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

This function can also be used to set the column header text for elements of the type [Datagrid]. For [Row], 0 must be entered, for [Content] only text is allowed.

Applies to:

Table, Datagrid (Column header only)

Examples:

The second column of a table is filled with a data set which has just been calculated. The first row contains the variable name, the second row the data set's maximum value in a fixed format. The numerical values start as of the 3rd row.

```
Kanal1= ...
PnTableSetCell( "Tab1", 2, 1, "Kanal1")
TxMax$ = TForm( Max( Kanal1), "f32")
PnTableSetCell(   "Tab1", 2, 2, TxMax$)
PnTableSetColumn( "Tab1", 2, 3, Kanal1)
```

See also:

PnTableSetColumn, PnTableSetRow, PnTableGetCellText

PnTableSetColumn

Scope: Panels

Sets the content of a table column.

Declaration:

```
PnTableSetColumn ( TxTableName, Column, Row, Contents )
```

Parameter:

TxTableName	Name of the Panel table to be controlled.
Column	Column (1..)
Row	Row (1..)
Contents	Permitted data types: Normal Waveform, Text Box

Description:

One column of the specified table in the active Panel is filled with a data set. The individual numbers/texts are entered according to the formatting specified for the table. The first number is entered at the specified position, all others below it.

The inputs for the row and column determine the position of the first cell to be filled; the top left is [1,1]

As many values are transferred as needed until either the last value of the data set or the last table row has been reached.

To transfer text to a table cell, use the function PnTableSetCell().

The function can not be applied to table cells which are linked to a variable.

If it is a data grid, the parameter [Row] must be set to 1.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table, Datagrid

Examples:

The columns of a table are filled with the individual events of an event-based data set.

```
Count    = PnTableColumns?( "Tab1")
CountEvn = EventNum?( Data)
IF CountEvn < Count
   Count = CountEvn
END
i = 1
WHILE i <= Count
   PnTableSetColumn( "Tab1",i,2, Daten[i])
   i = i + 1
END
```

See also:

PnTableSetRow, PnTableSetCell

PnTableSetDim

Scope: Panels

Sets the number of columns and rows of a table.

Declaration:

```
PnTableSetDim ( TxElementName, Columns, Rows, Option )
```

Parameter:

TxElementName	Name of the Panel element
Columns	New number of columns
Rows	New number of rows
Option	Option for the control of the table size.
	0 : The current table size remains unchanged.
	1 : The table size is changed.

Description:

With this function it is possible to set the number of columns and rows of a table.

If [Columns] or [Rows] are 0, the current number of columns or rows remain unchanged.

With [Option] the table size can be controlled as follows:

0: The current table position and size remain unchanged. The current column widths and row heights are changed proportionally.

1: The table size is adapted (the position left/top remains unchanged):

- Increasing number of columns: The width of the table increases. The new columns get the width and futher properties (not the content) from the previous last column.
- Increasing number of rows: The height of the table increases. The new rows get the height and futher properties (not the conten) from the previous last row.
- At decreasing the number of rows and columns, the surplus cells were removed. The height/width of the table is decreasing, while the remaining cells keep their size.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table

Examples:

The columns of a table are filled with the individual events of an event-based data set. The number of columns is set to the number of events and number of rows is set to the maximum length of the events + 1. Therefore the complete data set can be shown in the table, starting from the 2. row.

```
eventCount = EventNum?(data)
maxEventLength = 0
FOREACH EVENT event IN data
   maxEventLength = UpperValue(maxEventLength, Leng?(event))
END
PnTableSetDim("Tab1", eventCount, maxEventLength + 1,  1)
i = 1
WHILE i <= eventCount
   PnTableSetColumn("Tab1", i, 2, data[i])
   i = i + 1
END
```

See also:

PnTableColumns?, PnTableRows?, PnTableSetCell, PnSetProperty

PnTableSetProperty

Scope: Panels

A property of the selected table value is re-set.

Declaration:

```
PnTableSetProperty ( TxElementName, Column, Row, TxPropName, Value )
```

Parameter:

TxElementName	Name of the Panel element
Column	Column (1..)
Row	Row (1..)
TxPropName	Identifier of the property to set
	"FillColor" : Background color
	"FrameColor" : Frame color
	"TextColor" : Text color
Value	Numerical value or string to which the property is to be set

Description:

The property to be set is selected by pre-defined identifiers.

The following properties have been defined thus far:

Identifier	Meaning
"FillColor"	Background color (RGB-value)
"FrameColor"	Frame color (RGB-value)
"TextColor"	Text color (RGB-value)

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table

Examples:

A Panel page contains a table 'tab' for displaying various characteristic values of the currently selected variables (1st measurement, 1st channel). With each change of the selection in the Measurement or Channels list, the table is updated accordingly. Among other things, the maximum of the data set is found and entered in the 3rd line of the 2nd column. If the value exceeds a certain limit, the cell's background color is changed to red.

Event-sequence 'Data selection changed''

```
TxVarName = SelBuildVarName(1, 1, 0)
IF TxVarName <> ""
   maxval = Max(<TxVarName>)
   PnTableSetCell("Page1.tab", 2, 3, maxval )
   IF maxval > 10
      ; Limit exceeded (red background)
      PnTableSetProperty("Page1.tab", 2, 3, "FillColor", RGB( 255, 0, 0)))
   ELSE
      ; Default (white background)
      PnTableSetProperty("Page1.tab", 2, 3, "FillColor", RGB( 255, 255, 255)))
   END
   ; ... fill further lines in the table
END
```

See also:

PnSetValue, PnSetProperty

PnTableSetRow

Scope: Panels

Sets the content of a table row

Declaration:

```
PnTableSetRow ( TxTableName, Column, Row, Data )
```

Parameter:

TxTableName	Name of the Panel table to be controlled.
Column	Column (1..)
Row	Row (1..)
Data	Permitted data types: Normal Waveform, Text Box

Description:

One row of the specified table in the active Panel is filled with a data set. The individual numbers/texts are entered according to the formatting specified for the table. The first number is entered at the specified position, all others to the right of it.

The inputs for the row and column determine the position of the first cell to be filled; the top left is [1,1]

As many values are transferred as needed until either the last value of the data set or the last table column has been reached.

To transfer text to a table cell, use the function PnTableSetCell().

The function can not be applied to table cells which are linked to a variable.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event-sequence and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Table

Examples:

The rows of a table are filled with the individual segments of a segment-based data set.

```
Count     = PnTableRows?( "Tab1")
CountSeg = leng?( Daten) / SegLen?( Daten)
IF CountSeg < Count
   Count = CountSeg
END
i = 1
WHILE i <= Count
   PnTableSetRow ( "Tab1",1,i, Data[i])
   i = i + 1
END
```

See also:

PnTableSetColumn, PnTableSetCell

PnTableShowColumn

Scope: Panels

Controls the visibility of a datagrid column

Declaration:

`PnTableShowColumn ( TxTableName, Column, Task )`

Parameter:

TxTableName	Name of the Panel table to be controlled.
Column	Column (1..)
Task	Task
	0 : Hide column
	1 : Show column

Description:

Controls the visibility of a column of the specified table in the active Panel.

The selection of the table to be addressed can either take the form [page name].[element name], or be acomplished using on the element name. If the page is not stated explicitly, the element is found as follows:

- If the function is called within an event sequence and the event can be assigned to a Page (e.g. a botton's event 'Pressed'), then the system searches for an element with the specified name on this page.
- Otherwise, the system searches on the active (visible) page.

Applies to:

Datagrid

Examples:

At design time, a data grid with 10 columns was defined. At runtime, the columns were filled with the individual events of an event-based data set. If the data set contains fewer than 10 events, the unnecessary columns are hidden.

```
NumberOfCols  = PnTableColumns?( "Grid1")
CountEvn = EventNum?( Data)
IF NumberEvn > NumberOfCols
   NumberEvn = NumberColumns
END
i = 1
WHILE i <= NumberEvn
   PnTableSetColumn( "Grid1",i,1, Data[i])
   i = i + 1
END
WHILE i <= NumberColumns
   PnTableShowColumn( "Grid1",i, 0)
   i = i + 1
END
```

See also:

PnTableSetColumn, PnTableColumns?

PnTreeDeleteNode

Scope: Panels

Deletes nodes in a Treeview

Declaration:

`PnTreeDeleteNode ( TxElementName, Node, ChildPosition )`

Parameter:

TxElementName	Name of the tree view element
Node	Absolute index or unique key of the node to be deleted, or of the associated parent node. Enter a -1 in order to delete all nodes. With a 0, you address a node on the root level; the next parameter then specifies the position within the root level.
ChildPosition	Child-position
	0 : [Node] directly identifes the node(s) to be deleted.
	-1 : All child nodes of [Node] are deleted. If [Node] = 0, all nodes except the root node are deleted.
	>=1 : Position of the desired node among the immediate child nodes of [Node], or respectively within the root level, if a 0 was specified for [Node].

Description:

Examples:

```
; Deletes the entire content of the Treeview
PnTreeDeleteNode("TreeView1", -1, 0)

; Deletes all nodes except the root level
PnTreeDeleteNode("TreeView1", 0, -1)

; Deletes the 2nd node of the root level completely
PnTreeDeleteNode("TreeView1", 0, 2)

; Deletes the 3rd child node in the 2nd node of the root node
index = PnTreeGetNodeState("TreeView1", 0, 2, "AbsoluteIndex")
PnTreeDeleteNode("TreeView1", index, 3)

; Deletes all child nodes in the 2nd node of the root level
index = PnTreeGetNodeState("TreeView1", 0, 2, "AbsoluteIndex")
PnTreeDeleteNode("TreeView1", index, -1)

; Deletes the node having the key "#node11"
PnTreeDeleteNode("TreeView1", "#node_11", 0)

; Deletes all children of the node having the key "#node11"
PnTreeDeleteNode("TreeView1", "#node_11", -1)

; Searches for a node having the caption "Mr. Smith" and deletes the entire branch
; in which the node is located
index = PnTreeFindNodes("TreeView1", 0, 0, "Mr. Smith", "Caption")
IF leng?(index) = 1
   rootIndex = PnTreeGetNodeState("TreeView1", index, 0, "RootNode")
   PnTreeDeleteNode("TreeView1", rootIndex, 0)
END
```

See also:

PnTreeInsertNode, PnTreeSetNodeProp

Supported since:

Version 2023

PnTreeFindNodes

Scope: Panels

Search for nodes in a Treeview which meet a specified condition.

Declaration:

```
PnTreeFindNodes ( TxElementName, ParentNode, Option, TxPattern, TxCondition ) -> NodeIndizes
```

Parameter:

TxElementName	Name of the tree view element
ParentNode	Absolute index or unique key of the node among whose children the search is to be performed. Enter a 0 to search throughout all nodes or respectively within the root node.
Option	Search area
	0 : Search among all child nodes. For [ParentNode] = 0, the search is performed throughout all of the tree's nodes.
	1 : Only search among direct child nodes. For [ParentNode] = 0, the search is performed throughout the tree's root nodes.
TxPattern	Pattern for which to search
TxCondition	What condition must be met?
	"Caption" : The node caption exactly matches the search pattern. There is no case sensitivity.
	"NodeKey" : The node key exactly matches the search pattern. There is no case sensitivity.
	"Caption*" : The node's caption matches the search pattern. The search pattern can contain the wildcard characters "*" (any amount of any characters) and "?" (any one character). If the first character in the search pattern is a "!", the condition is negated; so all nodes are found which do NOT match the search pattern. There is no case-sensitivity.
	"NodeKey*" : The node-key matches the search pattern. The search pattern can contain the wildcard characters "*" (any amount of any characters) and "?" (any one character). If the first character in the search pattern is a "!", the condition is negated; so all nodes are found which do NOT match the search pattern. There is no case-sensitivity.
NodeIndizes	
NodeIndizes	Data set containing the absolute indices of the nodes found. The first node in the tree has the index 1. Empty data set (length: 0), when no node meets the condition.

Description:

Examples:

Searches for a node having the caption "Mr. Smith" and deletes the entire branch in which the nodes are located:

```
index = PnTreeFindNodes("TreeView1", 0, 0, "Mr. Smith", "Caption")
rootIndex = PnTreeGetNodeState("TreeView1", index, 0, "RootNode")
PnTreeDeleteNode("TreeView1", rootIndex, 0)
```

In a Tree-widget, the system finds all entries which are direct children of the node having the key "#datafiles" and whose captions end with ".dat". The captions found are saved in a text array.

```
TxDatFiles = TxArrayCreate(0)
fileNodeIndex = PnTreeFindNodes("TreeView1", 0, 0, "#datafiles", "NodeKey")
nodeIndizes = PnTreeFindNodes("TreeView1", fileNodesIndex, 1, "*.dat", "Caption*")
count = leng?(nodeIndizes)
FOR i = 1 TO count
   TxDatFiles[i] = PnTreeGetNodeProp("TreeView1", nodeIndizes[i], 0, "Caption")
END
```

See also:

PnTreeGetNodeProp, PnTreeGetNodeState, PnTreeGetNodes

Supported since:

Version 2023

PnTreeGetNodeCount

Scope: Panels

Gets the count of nodes in a Treeview.

Declaration:

`PnTreeGetNodeCount ( TxElementName, ParentNode, Option ) -> Count`

Parameter:

TxElementName	Name of the tree view element
ParentNode	Absolute index or unique key of the node whose child nodes are to be counted. Enter a 0 to find the count of all nodes, or respectively the count of all root nodes.
Option	Option
	0 : Include all child nodes. For [ParentNode] = 0, the system gets the count of all nodes in the tree diagram.
	1 : Include only the direct child nodes. For [ParentNode] = 0, the system gets the count of all root nodes.
Count	
Count	Node count

Description:

Examples:

```
; Getting the total count of all nodes in the tree diagram:
count = PnTreeGetNodeCount("TreeView1", 0, 0)

; Gets count of root nodes
root_count = PnTreeGetNodeCount("TreeView1", 0, 1)

; Gets the count of direct children of the last root node:
lastroot_index = PnTreeGetNodeState("TreeView1", 0, root_count, "AbsoluteIndex")
lastroot_children_count = PnTreeGetNodeCount("TreeView1", lastroot_index, 1)

; Getting the count of direct children of the node labeled "Source Files":
file_group_index = PnTreeFindNodes("TreeView1", 0, 0, "Source Files", "Caption")
file_count = PnTreeGetNodeCount("TreeView1", file_group_index, 1)
```

See also:

PnTreeGetNodeProp, PnTreeGetNodeState, PnTreeGetNodes

Supported since:

Version 2023

PnTreeGetNodeProp

Scope: Panels

Gets a property of the specified node.

Declaration:

```
PnTreeGetNodeProp ( TxElementName, Node, ChildPosition, TxPropertyID ) -> TxText
```

Parameter:

TxElementName	Name of the tree view element
Node	Absolute index or unique key of the node to be used, or respectively of the associated parent node. With a 0, you address a node on the root level; the next parameter then specifies the position within the root node.
ChildPosition	Child-position
	0 : [Node] directly identifies the node to be used.
	>=1 : Position of the desired node among the immediate child nodes of [Node], or respectively within the root level, if a 0 was specified for [Node].
TxPropertyID	Selection of the property to get
	"Caption" : Caption of the node
	"NodeKey" : Key/supplemental text. The text specified in this parameter can be used as a key so that later it is easy to specify this node in other functions for controlling the tree view. In this usage, the key naturally needs to be unique. There is no case sensitivity with keys. Alternatively, the text can also be used to save additional information with the node. For instance, it would be possible when filling the tree diagram with filenames to display only the "short" filename as the caption, while saving the complete file path with the key.
	"ImageKey" : Image-key. Specifies the miniature image to be displayed for this node. The available images must be defined, together with a unique identifying key, at Design-time in the tree view's property "Impage list". Enter an empty text "" in order to display no image.
TxText	
TxText	Current value of the property queried

Description:

For the purpose of addressing the desired node, a variety of possibilites are available:

- **Absolute index:** The absolute index of the node among all existing nodes. Corresponds to the row number in which the node is located if all nodes are expanded. The first node of the root level has the index 1.
- **Key:** Upon creating a node, the user can define an optional textual key, by means of which the node can be addressed later. The assigned key should therefore be absolutely unique. There is no case-sensitivity.
- **Child-position:** Specifies the position of the node among all immediate child nodes of a specific parent node. For nodes on the root level, the position within the root level (corresponds to the row number, if all nodes are collapsed). The first child node has the position 1.

Examples:

The captions of all of a Treeview's checked nodes are written to a text array.

```
indizes = PnTreeGetNodes("TreeView1", 0, 0, "Selected")
count = leng?(indizes)
TxaCheckedNodes = TxArrayCreate(count)
FOR i = 1 TO count
   caption = PnTreeGetNodeProp("TreeView1", indizes[i], 0, "Caption")
   TxaCheckedNodes[i] = caption
END
```

To all nodes of the root level which previously had no miniature image, a default images is assigned. This had been defined along with the key "GenericFileIcon" in the property "Image list" upon designing the Treeview.

```
count = PnTreeGetNodeCount("TreeView1", 0, 1)
FOR i = 1 TO count
   imagekey = PnTreeGetNodeProp("TreeView1", 0, i, "ImageKey")
   IF imageKey = ""
      PnTreeSetNodeProp("TreeView1", 0, i, "ImageKey", "GenericFileIcon")
   END
END
```

The captioning of all nodes ending with ".dat" is converted to upper case letters.

```
indizes = PnTreeFindNodes("TreeView1", 0, 0, "*.dat", "Caption*")
FOREACH SAMPLE index in indizes
   caption = PnTreeGetNodeProp("TreeView1", index, 0, "Caption")
   PnTreeSetNodeProp("TreeView1", index, 0, "Caption", TConv(caption, 2))
END
```

See also:

PnTreeInsertNode, PnTreeSetNodeProp

Supported since:

Version 2023

PnTreeGetNodes

Scope: Panels

Gets the currently selected/checked nodes, the root nodes, or the respective child nodes for a given parent node in the Treeview.

Declaration:

`PnTreeGetNodes ( TxElementName, ParentNode, Option, TxCondition ) -> NodeIndizes`

Parameter:

TxElementName	Name of the tree view element
ParentNode	Absolute index or unique key of the node among whose children the search is to be performed. Enter a 0 to search throughout all nodes or respectively within the root node.
Option	Sets the search depth
	0 : Include all child nodes.
	1 : Include only direct child nodes
TxCondition	What condition must be met?
	"Selected" : All selected nodes are returned.
	"Checked" : All checked nodes are returned.
	"All" : For a given parent node, all associated child nodes are returned. For [ParentNode]=0 and [Option]=1, all of the root level's nodes are returned.
NodeIndizes	
NodeIndizes	Data set containing the absolute indices of the nodes found. The first node in the tree has the index 1. Empty data set (length: 0), when no node meets the condition.

Examples:

```
; Gets the absolute indices of all selected nodes on the root level
sel_nodes_root = PnTreeGetNodes("TreeView1", 0, 1, "Selected")

; Gets the absolute indices of all checked nodes in the entire tree diagram:
checked_nodes_global = PnTreeGetNodes("TreeView1", 0, 0, "Checked")

; Gets the absolute indices of all root nodes
root_nodes = PnTreeGetNodes("TreeView1", 0, 1, "all")

; Gets all checked direct children of the node labeled "Source Files"
files_group_index = PnTreeFindNodes("TreeView1", 0, 0, "Source Files", "Caption")
files_checked = PnTreeGetNodes("TreeView1", file_group_index, 1, "Checked")

; Allowed, but pointless: Finds the absolute indices of all nodes
The result contains the integers from 1 to the total node count
all_nodes = PnTreeGetNodes("TreeView1", 0, 0, "all")
```

See also:

PnTreeGetNodeProp, PnTreeGetNodeState, PnTreeFindNodes

Supported since:

Version 2023

PnTreeGetNodeState

Scope: Panels

Gets the current state of a tree node.

Declaration:

```
PnTreeGetNodeState ( TxElementName, Node, ChildPosition, TxStateID ) -> State
```

Parameter:

TxElementName	Name of the tree view element
Node	Absolute index or unique key of the node to be used, or respectively of the associated parent node. With a 0, you address a node on the root level; the next parameter then specifies the position within the root node.
ChildPosition	Child-position
	0 : [Node] directly identifies the node to be used.
	>=1 : Position of the desired node among the immediate child nodes of [Node], or respectively within the root level, if a 0 was specified for [Node].
TxStateID	Which state property is to be returned?
	"Selected" : 1, if the node is selected. Else 0.
	"Checked" : 1, if the node is checked. 0 if not. -1 for uncertain check state (possible for parent nodes when "Check-Mode"= "recursive").
	"Expanded" : 1, if the node is expanded. Else 0
	"Level" : Level (indentation) of the node. Root nodes have the level 0.
	"ParentNode" : Absolute index of the parent node, or 0 if the node is not located in the root level.
	"RootNode" : Absolute index of the associated root node.
	"Position" : Position of the node among the direct children of the parent node. With nodes of the root level, the position within the root level.
	"AbsoluteIndex" : Absolute index of the node in the tree
	"IsLeaf" : 1, if the node is a leaf, so it is an end point in the hierarchy, without its own children. Else 0. Inverse of "HasChildren".
	"HasChildren" : 1, if the node has child nodes; else 0. Inverse of "IsLeaf".
State	
State	State returned; integer and >-1. Interpretation depends on [TxStateID].

Description:

For the purpose of addressing the desired node, a variety of possibilites are available:

- **Absolute index:** The absolute index of the node among all existing nodes. Corresponds to the row number in which the node is located if all nodes are expanded. The first node of the root level has the index 1.
- **Key:** Upon creating a node, the user can define an optional textual key, by means of which the node can be addressed later. The assigned key should therefore be absolutely unique. There is no case-sensitivity.
- **Child-position:** Specifies the position of the node among all immediate child nodes of a specific parent node. For nodes on the root level, the position within the root level (corresponds to the row number, if all nodes are collapsed). The first child node has the position 1.

Examples:

Gets the count of direct children of the 2nd root node.

```
index = PnTreeGetNodeState("TreeView1", 0, 2, "AbsoluteIndex")
children_count = PnTreeGetNodeCount("TreeView1", index, 1)
```

The captions of all selected nodes which are direct children of a root node are written in a text array.

```
selected_entries = TxArrayCreate(0)
all_selected_nodes = PnTreeGetNodes("TreeView1", 0, 0, "Selected")
FOREACH SAMPLE node IN all_selected_nodes
IF PnTreeGetNodeState("Treeview1", node , 0, "Level") = 1
   selected_entries = TxArrayInsert( selected_entries, PnTreeGetNodeProp("TreeView1", node, 0, "Caption"), -1)
END
```

A tree widget contains, among other things, a "Data files" branch in which measured value files are listed in various sub-branches. Double-clicking on a leaf node loads the associated file.

Event-sequence 'Double-clicked' for the tree-widget:

```
selected_index = PA2
root_index = PnTreeGetNodeState(   PA1, selected_index, 0, "RootNode")
IF PnTreeGetNodeProp(PA1, root_index, 0, "Caption") = "Data files"
   ; OK, the double click happened within the desired branch
   filename = PnTreeGetNodeProp(PA1, selected_index, 0, "Caption")
   FileLoad(filename, "", 0)
END
```

See also:

PnTreeGetNodeProp, PnTreeGetNodeState

Supported since:

Version 2023

PnTreeInsertNode

Scope: Panels

Adds a new node to a treeview.

Declaration:

```
PnTreeInsertNode ( TxElementName, ParentNode, ChildPosition, TxCaption [, TxKeyOrData] [, TxImageKey] ) -> Index
```

Parameter:

TxElementName	Name of the treeview-widget
ParentNode	Absolute index or key of the parent node to which the new node is to be appended. Enter a 0 in order to create a node in the root level.
ChildPosition	Position of the new node among the direct children of the specified parent node. For nodes on the root level, the position within the root level (corresponds to the row number, if all nodes are collapsed). The first child's index is 1. Enter a 0 in order to append the new node as the last child.
TxCaption	Caption of the new node
TxKeyOrData	Additional text saved with the node, e.g. as a key for later identification of the node or additonally noted data in textual form. (optional , Default value: "")
TxImageKey	Specifies the miniature image to be displayed for this node. The image is displayed at the left of the caption. The available images must be defined, together with a unique identifying key, at Design-time in the tree view's property "Impage list". Enter an empty text "" in order to display no image. (optional , Default value: "")
Index	
Index	Absolute index of the newly created node.

Description:

In order to address the desired parent node, there are two different possibilities available for the parameter [ParentNode]:

- **Absolute index:** The absolute index of the node among all existing nodes. Corresponds ot the row number in which the node is located, if all nodes are expanded. The first node of the root level has the index 1.
- **Key:** Upon creating a node, the user can define an optional textual key, by means of which the node can be addressed later. The assigned key should therefore be absolutely unique. There is no case-sensitivity.

Parameter [TxKeyOrData]: The text specified in this parameter can be used as a key so that later it is easy to specify this node in other functions for controlling the tree view. In this usage, the key naturally needs to be unique. There is no case sensitivity with keys. Alternatively, the parameter can also be used to save additional information with the node. For instance, it would be possible when filling the treeview with filenames to display only the "short" filename as the caption, while saving the complete file path with the key.

Examples:

Inserting a new root node at the last position in the root level with two child nodes. The root node is expanded and the second child node is selected.

```
index = PnTreeInsertNode("TreeView1", 0, 0, "Result data")
PnTreeInsertNode("TreeView1", index, 0, "Highpass10Hz")
PnTreeInsertNode("TreeView1", index, 0, "Highpass20Hz")
PnTreeSetNodeState("TreeView1", index, 0, "Expanded", 1)
PnTreeSetNodeState("TreeView1", index, 2, "Selected", 1)
```

The names of the sample files included standard with FAMOS are displayed in a treeview. All files having the extension "dat" are sorted under the root node "imc files"; all other files under the root node "Other files". What is displayed is the filename, the complete path name is saved as supplemental information. The two root nodes display miniature images (which were defined in the tree view's property "image list" upon designing the panels). The "imc files"-node is expanded. The node "Other files" was additionally provided with a key as an easy way to address when inserting its child nodes (addressing it by its absolute index would be more trouble because that index is not constant and would need to be found again upon each call).

```
PnTreeDeleteNode("Treeview1", -1, 0); clear content
PnTreeInsertNode("TreeView1", 0, 0, "imc files", "", "ImcFileIcon")
PnTreeInsertNode("TreeView1", 0, 0, "Other files", "#other", "GenericFileIcon")
files = FsGetFileNames("C:\Users\Public\Documents\imc\imc FAMOS\_Demo Projects\_Demo Files\dat", "*.*", 0, 0, 1)
FOREACH ELEMENT path IN files
   ext = FsSplitPath(path, 3)
   file = FsSplitPath(path, 4)
   IF ext = ".dat"
      PnTreeInsertNode("TreeView1", 1, 0, file, path)
   ELSE
```

```
      PnTreeInsertNode("TreeView1", "#other", 0, file, path)
   END
END
PnTreeSetNodeState("TreeView1", 1, 0, "Expanded", 1)
```

See also:

PnTreeDeleteNode, PnTreeSetNodeProp, PnTreeSetNodeState

Supported since:

Version 2023

PnTreeSetNodeProp

Scope: Panels

Sets a property of the specified node.

Declaration:

```
PnTreeSetNodeProp ( TxElementName, Node, ChildPosition, TxPropertyID, TxText )
```

Parameter:

TxElementName	Name of the tree view element
Node	Absolute index or unique key of the node to be used, or respectively of the associated parent node. With a 0, you address a node on the root level; the next parameter then specifies the position within the root node.
ChildPosition	Child-position
	0 : [Node] directly identifies the node to be used.
	>=1 : Position of the desired node among the immediate child nodes of [Node], or respectively within the root level, if a 0 was specified for [Node].
TxPropertyID	Selection of the property to be set
	"Caption" : Caption
	"NodeKey" : Node-key
	"ImageKey" : Image key
	"ChildCheckStyle" : Check-style of the child nodes
TxText	New value. The meaning depends on [TxPropertyID]:
	"ChildCheckStyle" : Check style of the child nodes. For [Node]=0 and [ChildPosition]=0, the check style is set for all root nodes.
	"Standard" The corresponding global setting for the widget is used. **"None"** No check element **"Check"** Checkbox **"Radio"** Options group (radio group).

Description:

Examples:

To all nodes of the root level which previously had no miniature image, a default images is assigned. This had been defined along with the key "GenericFileIcon" in the property "Image list" upon designing the Treeview.

```
count = PnTreeGetNodeCount("TreeView1", 0, 1)
FOR i = 1 TO count
   imagekey = PnTreeGetNodeProp("TreeView1", 0, i, "ImageKey")
   IF imageKey = ""
      PnTreeSetNodeProp("TreeView1", 0, i, "ImageKey", "GenericFileIcon")
   END
END
```

The captioning of all nodes ending with ".dat" is converted to upper case letters.

```
indizes = PnTreeFindNodes("TreeView1", 0, 0, "*.dat", "Caption*")
FOREACH SAMPLE index IN indizes
   caption = PnTreeGetNodeProp("TreeView1", index, 0, "Caption")
   PnTreeSetNodeProp("TreeView1", index, 0, "Caption", TConv(caption, 2))
END
```

See also:

PnTreeInsertNode, PnTreeGetNodeProp, PnTreeSetNodeState

Supported since:

Version 2023

PnTreeSetNodeState

Scope: Panels

The current state of a tree node is changed.

Declaration:

```
PnTreeSetNodeState ( TxElementName, Node, ChildPosition, TxStateID, StateValue )
```

Parameter:

TxElementName	Name of the tree view element
Node	Absolute index or key of the desired node or of the associated parent node. Specifying 0 means that a node of the root level is to be addresed; the next parameter then specifies the desired position within the root level. Specifying -1 means that the operation is to be applied to all tree nodes. The next parameter should then be set to 0.
ChildPosition	Child-position
	0 : Not used. [Node] directly identifiies the node to be used. If a 0 was specified for [Node], the operation is applied to all root nodes.
	-1 : All child nodes. The operation is applied to all child nodes of [Node]. If a 0 was specified for [Node], the operation is applied to all direct children of the root node.
	>=1 : Position of the desired node among the immediate child nodes of [Node], or respectively within the root level, if a 0 was specified for [Node].
TxStateID	Which state property is to be changed?
	"Selected" : The selection for the specified node is swichted on/off. Explicit deactivation is only allowed for trees having multiselection.
	"Checked" : The checked state of the node is switched on/off.
	"Expanded" : The node is expanded/collapsed.
	"Selected>" : The selection for the selected node and all its child nodes is switched on/off. Only allowed for trees having multiselection.
	"Checked>" : The check state of the node and of all its child nodes is switched on/off.
	"Expanded>" : The node and all its child nodes are expanded or collapsed.
	"Visible" : The system ensures that the node is visible to the user. For this purpose, if necessary all parent nodes are expanded and the node is scrolled into the visible region. [StateValue] must take the value 1.
StateValue	New state
	0 : Deactivates the selected property.
	1 : Activates the selected property.

Description:

Examples:

Collapsing all nodes in a Treeview:

```
PnTreeSetNodeState("TreeView1", 0, 0, "Expanded>", 0)
```

Checking of all direct child nodes having the key "#files". The node is then made visible to the user (if applicable, by expanding all parent nodes and scrolling to the visible region).

```
PnTreeSetNodeState("TreeView1", "#files", -1, "Checked", 1)
PnTreeSetNodeState("TreeView1", "#files", 0, "Visible", 1)
```

Inserting a new root node at the last position in the root level with 2 child nodes. The root node is expanded and the 2nd child node is selected.

```
index = PnTreeInsertNode("TreeView1", 0, 0, "Result data")
PnTreeInsertNode("TreeView1", index, 0, "Highpass10Hz")
PnTreeInsertNode("TreeView1", index, 0, "Highpass20Hz")
PnTreeSetNodeState("TreeView1", index, 0, "Expanded", 1)
PnTreeSetNodeState("TreeView1", index, 2, "Selected", 1)
```

See also:

PnTreeSetNodeProp, PnTreeGetNodeState

Supported since:

Version 2023

Pol

Transformation of a complex data set to polar coordinates (Magnitude/Phase).

Declaration:

```
Pol ( ComplexData ) -> ComplexAsMP
```

Parameter:

ComplexData	Complex data set to be transformed [BP], [DP] or [RI]
ComplexAsMP	
ComplexAsMP	Resulting complex data set in polar coordinates [BP]

Description:

A complex data set is transformed to polar coordinates, i.e. a display with magnitude and phase. If the data set is already expressed in polar coordinates, it remains unchanged; if the data set is expressed with magnitude in dB and phase, the magnitude is recalculated as linear.

- When the data set is expressed in rectangular coordinates, the phase is calculated in degrees.
- If the data set to be transformed is the type Dp, the function Pol has the same effect as the function idB.
- The parameter may be structured (events/segments).

Examples:

A spectrum is transformed to the usually more graphically clear display style BP:

```
MPspectrum = Pol(RIspectrum)
```

Corrects the values in a spectrum, which due to multiple calculations, may contain phases outside the range of -180° to +180° or magnitudes less than zero. The actual content of the spectrum remains unchanged.

```
MPcorr = Pol(Rect(BPspectrum))
```

See also:

Rect, dB, idB, Compl

Poly

Polynomial approximation of a data set's values

Declaration:

```
Poly ( Data, SvOrder, SvOption ) -> ResultPolynom
```

Parameter:

Data	Data set to be approximated by a polynomial; allowed data types: [ND], [XY]
SvOrder	Order of the approximating polynomial
SvOption	Defines the calculation type
	1 : Output of the function values
	2 : Output of the coefficients
ResultPolynom	
ResultPolynom	Function values/coefficients of the approximating polynomial

Description:

Polynomial approximation of the data set passed is performed; the data set is approximated by a polynomial of the specified order, using the method of least squares. The approximating polynomial can assume any order m from 1 to 9 (for XY-data, from 1 to 7) . The order m is specified as the parameter SvOrder.

The Poly function uses the following procedure: a measurement channel is given with the values g(x[i]) (i = 0, ..., data set length - 1). The measurement channel should be approximated by a polynomial of the mth order. Using the Poly function, the coefficients c[0], ..., c[m] are determined according to the method of least squares, so that the condition

$$g(x[i]) \approx k[0] + k[1] \cdot x[i] + k[2] \cdot x[i]^2 + \ldots + k[m] \cdot x[i]^m$$

is met for all i.

According to the entry for the parameter [SvOption], either the data set of the approximating polynomial or the data set of the coefficients of the approximating polynomial is returned.

Entering 1 for the parameter [SvOption] means that the function values of the approximating polynomial are returned. For the data type [ND], the curve returned has the same resolution as the source data set. With the data type [XY], the curve returned has 1000 points and is defined over the same X-domain as the source data set.

Entering 2 for the [SvOption] parameter means that the coefficients of the approximating polynomial are returned. The coefficients are returned in the order corresponding to the ascending exponents of the polynomial: the constant is returned first, and the coefficient belonging to the highest exponent of the polynomial is returned last.

- The parameter [SvOrder] must be entered as an integer between 1 and 9 (data type: ND) or 7 (data type: XY).
- The length of the data set to be approximated must be at least 2.
- The function Poly is a special case of the Appro() function.

Influence of x-values on numerical analysis:

For purposes of numerical analysis, it is best if all x-values are distributed as closely around 0 as possible. The quality of the results is negatively affected the more x-values are distant from 0.

To obtain a rough estimate, the following expression can be evaluated:

```
A1 = X0 ; or the lowest x-value for XY-data
A2 = X0 + DeltaX * (data set length-1); or the highest x-value for XY-data
A  = |A1 + A2| / (2*(A2-A1))
```

The greater the value of A, the less precise the result becomes.

The following approximate boundaries apply for A as a rule of thumb (depending on the order; with XY-data, approximately uniform distribution of x-coordinates is assumed):

Order	A
1	A <= 100000
2	A <= 100
3	A <= 10
4	A <= 5
5	A <= 3

6	A <= 2

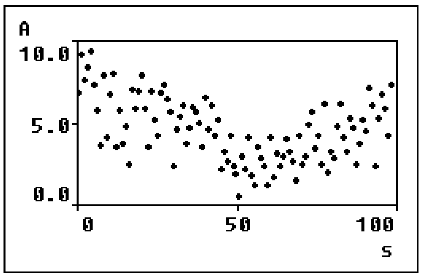
Accuracy

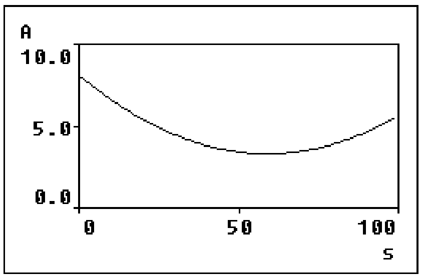
In application of the method of least squares, the products (c* x) are weighted against each other and the c is determined so that the squared error of the equation is minimized. It is possible that some of these products supply no meaningful information for the sum of all products. In this case, a very small c is calculated, which cannot be determined with any accuracy. Internal calculations are made with 15 places after the decimal, but it is a fact of the numeric method that irrelevant quantities are often the ones with the least accuracy. Even if the parameter c has the same magnitude, it is not the value of c, rather the product (c*x) which is important for this determination. For example, c1 = c2 = 1, but (c1*x) is approximately 10, while (c2* x *x) equals 1000. It is for this reason that it is safe to assume that the first product is a factor 100 less important for the summation of the products, and that the error of c1 thus can be 100 times greater than that of c2.

Examples:

```
NwFunctionValues = Poly(NwData, 2, 1)
NwCoeff = Poly(NwData, 2, 2)
```

The data set NDData, belonging to the left graph, is to be approximated by a second order polynomial. The data set NDData consists of 100 values. Entering 1 as the parameter [SvOption] (output of the function value of the approximating polynomial) yields the polynomial approximation result displayed at the right, the data set NDFctVal (function value):





Selecting 2 as the [SvOption] parameter (output of the coefficients of the approximating polynomial) yields the data set NDCoeff (coefficients); this data set contains 3 values:

```
koeff0 = 8.0577 , koeff1 = -162.87E-3 and koeff2 = 1.3804E-3
```

This is how to obtain the individual coefficients::

```
SvCoeff0 = NwCoeff[1]
SvCoeff1 = NwCoeff[2]
SvCoeff2 = NwDoeff[3]
```

See also:

Appro, ApproNonLin, eFit, LFit, Value

PolynomRoots

Available in: Professional Edition and above

Zeroes/roots of a polynomial

Declaration:

```
PolynomRoots ( Polynomial ) -> Zeroes
```

Parameter:

Polynomial	Polynomial
Zeroes	
Zeroes	Zeroes

Description:

The result is a list of the roots. The polynomial has as many roots as its degree indicates.

The roots are determined as complex numbers and returned in the form Real part / Imaginary part. If the root is a real number, the imaginary part is 0.

Interpretation of a polynomial having the coefficients p[i], where the coefficients in the data set are ordered according to increasing exponent:

```
Polynomial = p[1] + p[2] * x + p[3] * x^2 + p[4] * x^3 +...
```

The maximum degree supported is 20.

The polynomial must be real (not complex).

The function locates the roots by means of the eigenvalues of the companion matrix.

Examples:

Roots of a quadratic equation/2nd-degree polynomial

```
P = [ 24, -14, 2]  ; test data, 24-14x+2x^2=0
z = PolynomRoots ( P )
z_r = z.r
; z = [3, 4]
```

complex roots of a quadratic equation/2nd-degree polynomial

```
P = [ 58, -8, 2]  ; test data, 58-8x+2x^2=0
z = PolynomRoots ( P )
; z.r = [2, 2]
; z.i = [5, -5]
; 2+5i, 2-5i
```

See also:

All0

Pos

Returns the first position (X-coordinate) of a specified Y-value.

Declaration:

```
Pos ( Data, SvLevel ) -> EwPosition
```

Parameter:

Data	Data set to be examined. Types allowed: [ND],[XY].
SvLevel	Y-value whose X-coordinate is to be determined
EwPosition	
EwPosition	The X-coordinate determined

Description:

The x-coordinate (x-position) corresponding to a specified y-value in a data set is determined. The data set is assumed to be linear-interpolated between its sample values, so the specified y-value need not be a sample value. As a result, the x-coordinate generally will not fall exactly on a sample point.

The function thus determines the position of a value more precisely than the sampling interval.

If a y-value occurs more than once in a data set, only the first occurrence is determined. Use the function PosiEx2() if you wish to locate every occurrence of a certain y-value.

The unit of the returned value is the x-unit of the data set.

The unit of the y-value passed should naturally be the y-unit of the data set.

If the specified y-value does not lie within the range of the data set, i.e. no x-position can be assigned, a warning message is generated and the return value is set to -10^20 . The functions Min and Max can be used to determine the valid range of values in the data set.

Examples:

The first zero intercept of a data set is determined.:

```
pos1stZero = Pos(NDdata, 0)
```

A voltage increase from 0 to SvMax was measured. The rise time is defined as the length of time between the occurrence of the values 0.9 * SvMax and 0.1 * SvMax. The rise time can be calculated using the Pos function:

```
slopeTime = Pos(NDvoltage, 0.9 * SVmax) - Pos(NDvoltage, 0.1 * SVmax)
```

See also:

PosiEx2, PosiEx, Value2, SearchLevel

PosiEx

Returns the positions (x-coordinates) for a given y-value.

Declaration:

```
PosiEx ( Data, SvLevel ) -> Positions
```

Parameter:

Data	Data set to be examined. Types allowed: [ND],[XY].
SvLevel	Y-value whose X-coordinates are to be determined
Positions	
Positions	The X-coordinates determined

Description:

The x-coordinates (x-positions) in a data set which correspond to a specified y-value are located. The data set is imagined to be linearly interpolated between its sample values. Therefore, the y-value specified doesn't need to exactly coincide with a sample value. Thus, the x-coordinates found generally are not exact matches of sample values.

In other words, the function PosiEx locates x-positions at a higher resolution than the actual sampling interval provides.

The y-unit of the returned data set is the x-unit of the parameter data set.

The unit of the y-value passed should naturally be the y-unit of the data set.

If the target y-value is not in the data set's value range, i.e., no x-position can be assigned to it, a warning is issued and an empty data set (length = 0) returned.

This function is only included for reasons of backward compatibility and the more powerful function PosiEx2() is generally preferable for its purpose.

Examples:

All of a data set's zero intercepts are located and the position of the last zero intercept determined:

```
zeroes = PosiEx(NDdata, 0)
lastZero = zeroes[Leng?(zeroes)]
```

See also:

PosiEx2, Pos, Value2, SearchLevel

PosiEx2

Returns the positions of a specified Y-value

Declaration:

`PosiEx2 ( Data, SvLevel, SvSlope, SvReturn ) -> Positions`

Parameter:

Data	Data set to be examined. Types allowed: [ND],[XY].
SvLevel	Y-value whose x-coordinates are to be determined
SvSlope	Extra condition regarding the direction in which the relevant level is crossed
	0 : Condition not applicable
	1 : Positive slope (away from point). A value SVLevel is located on the slope if the following applies: y[k] <= SVLevel < y[k+1]
	2 : Positive slope (towards the point). y[k] < SVLevel <= y[k+1]
	3 : Negative slope (towards the point). y[k] >= SVLevel > y[k+1]
	4 : Negative slopee (towards the point). y[k] > SVLevel >= y[k+1]
SvReturn	Governs the form in which the position found is returned. The input signal is interpolated linearly and don this basis the x-positions are determined. The return value is then:
	0 : x, interpolated: the linearly interpolated x-value is used.
	1 : x, current: the x-value of the signal sample point directly BEFORE the interpolated x-value is used.
	2 : x, closest: the x-value of the signal sample point CLOSEST to the interpolated x-value is used.
	3 : Index, interpolated: The (interpolated) index of the value found.
	4 : Index, current: the index of the signal sample point directly BEFORE the interpolated x-value is used.
	5 : Index, closest: the index of the signal sample point directly CLOSEST to the interpolated x-value is used.
Positions	
Positions	The X-coordinates or indices determined.

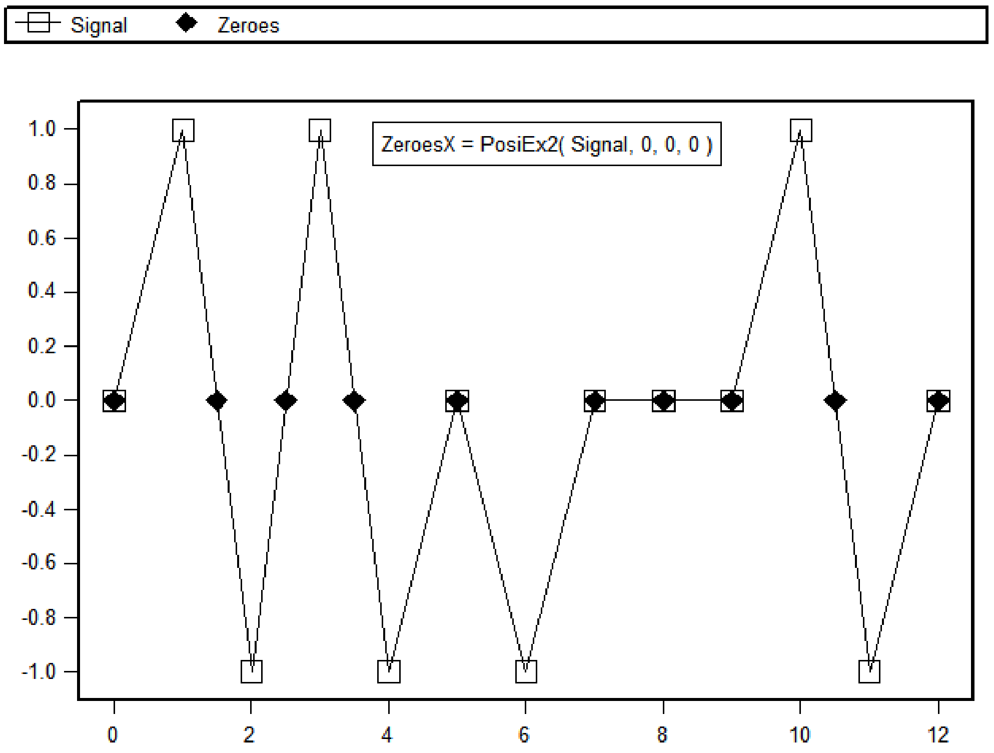
Description:

In a data set, the x-coordinates or indices associated with a specified y-value are determined. The data set is considered to be interpolated linearly between its sampled values. This means that the specified y-value does not need to coincide with any sampled value.

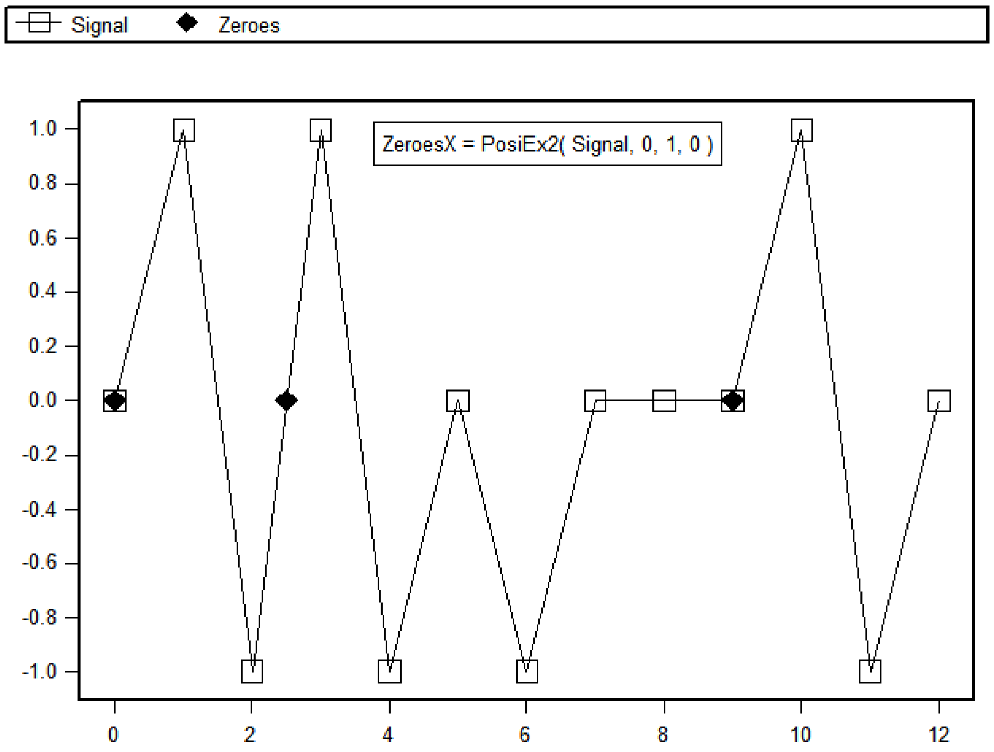
If the y-value provided y the user is not within the data set's range, so that it is not possible to assign a position to it, a warning is posted and an empty data set (length 0) is returned.

Edge condition:
The difference between options 1 and 2, the same as between 3 and 4, is in how the signal edge is defined if a point in the signal exactly coincides with the y-value specified and there is no unique way to assign it to an edge. This pertains to the signal's first and last points as well as all points at which the signal's slope is changing.
Options 1/3 mean that the slope between this point and the next one must be positive/negative. The signal's last point is thus never part of the result.
Options 2/4 mean that the slope between this point and the previous one must be positive/negative. The signal's first point is thus never part of the result.

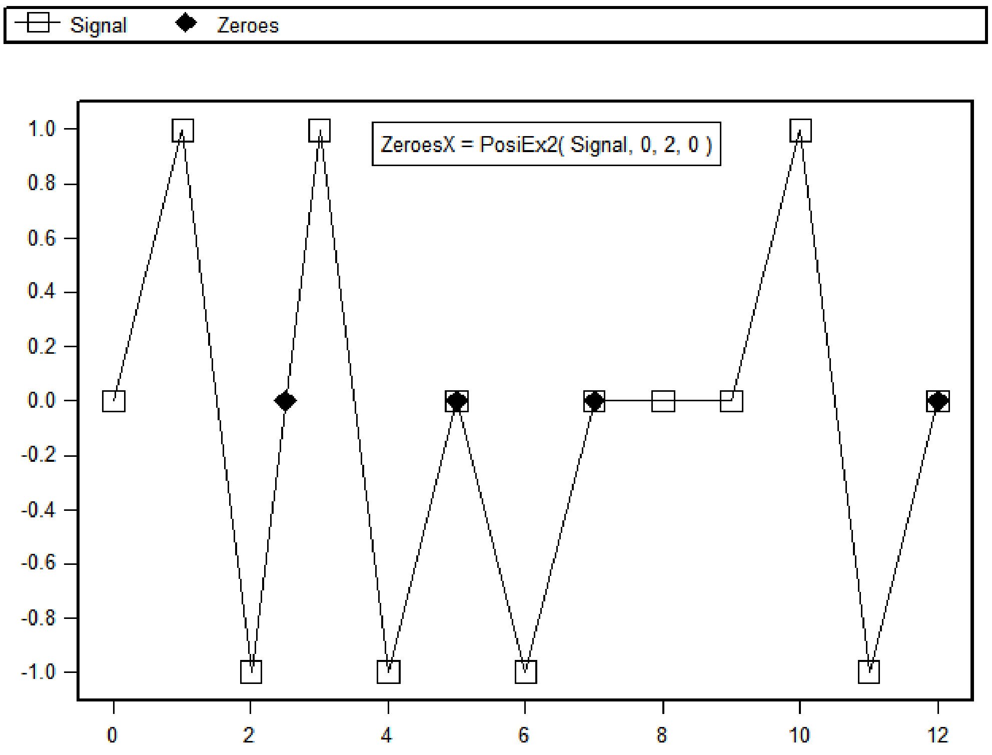
In the following example, the signal's zero crossing points are first found (SvEdge = 0):



Now, all zero-crossing points with positive edges are to be found. With SvEdge = 1, the slope is seen from the vantage point of the zero-crossing point. The zero-crossing at the end of the signal is thus ignored.



With SvEdge = 2, the slope is seen looking toward the zero-crossing. The zero-crossing at the beginning of the signal is thus ignored.



Examples:

Determines all positions where the signal crosses zero. Return of the interpolated x-values:

```
xZeroes = PosiEx2(Signal, 0, 0, 0)
```

To display the zero-crossings along with the signal, an XY data set is first generated and next displayed with a circle-symbol in the same curve window with the original signal.

```
Zeroes = XYof(xZeroes, Value2(Signal, xZeroes, 0))
CwNewWindow(Signal, "show")
CwNewChannel("append new axis", Signal)
CwNewChannel("append last axis", Zeroes)
CwSelectByIndex("line", 2)
CwLineSet("type", 0)
CwLineSet("symbol", 2)
```

The region in a signal between the first two zero-crossing points is to be extracted. Inclusion of the two zero-crossings must be certain.

```
zeroes = PosiEx2(signal, 0, 0, 4)
IF Leng?(zeroes) > 1
   front = zeroes[1]
   back = zeroes[2]
   IF signal[back] <> 0
      back = back + 1 ; move index after the zero
   END
   signalPart = CutIndex(signal, front, back)
END
```

Determines all positions in a signal at which the level 10 is reached with the signal coming from below in a positive slope. The index of the closeset point is returned.

```
xZeroes = PosiEx2(Signal, 10, 2, 5)
```

Determines all positions in a signal at which the level 10 is exited with the signal passing beyond it in a positive slope. The index of the last previous point is returned.

```
xZeroes = PosiEx2(Signal, 10, 1, 4)
```

See also:

Pos, PosiEx, Value2, SearchLevel

Power1

Available in: Professional Edition and above

Power calculation in one phase

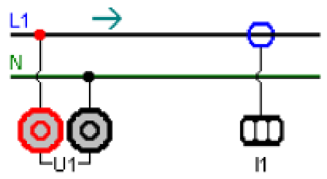
Declaration:

```
Power1 ( U1, I1 [, Interval data] ) -> Results group
```

Parameter:

U1	U1 = UL1 - N, phase voltage differential to Neutral
I1	I1, also IL1, current through the conductor, current direction (arrow) from the source to the load
Interval data	Interval data; the intervals represent the signals periods. Preferably a series of seamlessly successive intervals. The calculation is based on these periods. Especially for unknown or fluctuating signal frequencies. (optional)
Results group	
Results group	Group containing all calculation results: P: active power, PF: power factor, U*: voltage RMS values, I*: current RMS values, p_t instantaneous power, f: frequency

Description:



The functions PowerSelect() and PowerParameter() must first be called in order to complete the process of parameterizing the function.

If the parameter IntervalData is not specified, the calculation is performed based on the frequency which was specified in PowerParameter(). The number of signal periods over which the calculation is to be performed must be a multiple of the sampling interval when multiplied by the period duration. Thus, at 50Hz, either 1 or 10 periods are often specified; at 60Hz then 12 periods, with the sampling frequency typically 10kHz.

If the parameter IntervalData is specified, the interval lengths determine the period lengths. If there are gaps before/between/after the intervals, then for the purpose of calculations, continuously adjacent periods are assumed; the interval boundaries may no longer be used in such cases, but only their widths. The calculation starts at the first interval or at a whole number of interval widths before. If there are no intervals available, the frequency from PowerParameter() is used.

The interval data set can be determined using IntervalFromLevel(), for instance, and re-processed with other interval functions. As the basis for determining the zero crossing points, it is recommended to use an especially regular sinusoidal signal such as a voltage, for example.

Interval format: Equidistant "equi" or XY "xy", i.e. interval-code with position

The frequency is only determined when an interval data set is specified. Then it always shows the frequency on which the calculation is based, even at positions where intervals are missing.

When working with interval data, it is necessary to determine the zero crossings as precisely as possible in order to obtain exact results.

Determination of the frequency and the other results become (substantially) more exact when the analysis is performed over multiple periods.

For a precise analysis, at a mains frequency of 50Hz a sampling frequency of 10kHz makes sense.

The current direction in accordance with the load system. If an ammeter was connected the other way around, then for example instead of I1, the value -I1would be passed as the parameter.

Instead of the pair U1, I1, it would also be possible to analyze U2, I2 or U3, I3 of a 3-phase grid individually.

The current and voltage must have the same time base. They may have events; segments are also possible but not in conjunction with interval data.

The calculation only returns precise results if the interval width of the analysis is an integer number of samples.

Examples:

1-phase power measurement. default calculation at 50Hz over 10 periods; sampling frequence: 10kHz

```
PowerSelect()
PowerParameter(50, 10)
g = Power1 ( U1, I1 )
P = g:P
Q = g:Q
```

1-phase power measurement. Frequency fluctuates

```
PowerSelect()
PowerParameter ( 50, 1)
```

```
ivl = IntervalFromLevel ( U1, 0, 0, 0, 1, 0.0, 1e35, 0, "")
g = Power1 ( U1, I1, ivl )
P = g:P
Q = g:Q
```

All 3 phases of the three-phase network are analyzed individually and together; the RMS-values of the currents and voltages are caulculated only once.

```
PowerSelect()
PowerParameter(50, 10)
g = Power3 ( U1, I1, U2, I2, U3, I3, ivl )
PowerSelect(1, 0, 1, 0, 0, 1)
g1 = Power1 ( U1, I1, ivl )
g2 = Power1 ( U2, I2, ivl )
g3 = Power1 ( U3, I3, ivl )
P = g:P
Q = g:Q
P1 = g1:P1
```

See also:

PowerSelect, PowerParameter, Power2, Power3, IntervalFromLevel

Power2

Available in: Professional Edition and above

Calculation of ther power at the three-phase network with three-wire system

Declaration:

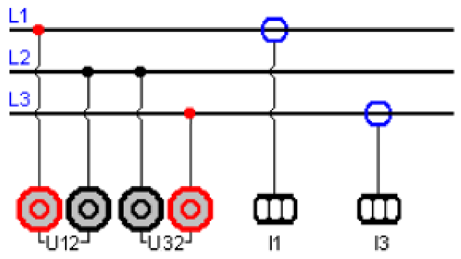
```
Power2 ( U12, I1, U32, I3 [, Interval data] ) -> Results group
```

Parameter:

U12	U12 = UL1 - UL2, Delta voltage; voltage between the outer conductors
I1	I1, also IL1, current through one phase line, current direction (arrow) from the source to the load
U32	U32 = UL3 - UL2, Delta voltage; voltage between the outer conductors
I3	I3, also IL3, current through one phase line, current direction (arrow) from the source to the load
Interval data	Interval data; the intervals represent the signals periods. Preferably a series of seamlessly successive intervals. The calculation is based on these periods. Especially for unknown or fluctuating signal frequencies. (optional)
Results group	
Results group	Group containing all calculation results: P: active power, PF: power factor, U*: voltage RMS values, I*: current RMS values, p_t instantaneous power, f: frequency

Description:

Instead of [U12, I1, U32, I3], it is also possible to use the pairs [U13, I1, U23, I2] or [U21, I2, U31, I3].



The Neutral conductor N is either not present or not conducting current. Typical in Delta circuit. The equation I1 + I2 + I3 = 0 applies.

The equations U12 = -U21, U32 = -U32, U31 = -U13 and U12 + U23 + U31 = 0 also apply.

The functions PowerSelect() and PowerParameter() must first be called in order to complete the process of parameterizing the function.

If the parameter IntervalData is not specified, the calculation is performed based on the frequency which was specified in PowerParameter(). The number of signal periods over which the calculation is to be performed must be a multiple of the sampling interval when multiplied by the period duration. Thus, at 50Hz, either 1 or 10 periods are often specified; at 60Hz then 12 periods, with the sampling frequency typically 10kHz.

If the parameter IntervalData is specified, the interval lengths determine the period lengths. If there are gaps before/between/after the intervals, then for the purpose of calculations, continuously adjacent periods are assumed; the interval boundaries may no longer be used in such cases, but only their widths. The calculation starts at the first interval or at a whole number of interval widths before. If there are no intervals available, the frequency from PowerParameter() is used.

The interval data set can be determined using IntervalFromLevel(), for instance, and re-processed with other interval functions. As the basis for determining the zero crossing points, it is recommended to use an especially regular sinusoidal signal such as a voltage, for example.

Interval format: Equidistant "equi" or XY "xy", i.e. interval-code with position

The frequency is only determined when an interval data set is specified. Then it always shows the frequency on which the calculation is based, even at positions where intervals are missing.

When working with interval data, it is necessary to determine the zero crossings as precisely as possible in order to obtain exact results.

Determination of the frequency and the other results become (substantially) more exact when the analysis is performed over multiple periods.

For a precise analysis, at a mains frequency of 50Hz a sampling frequency of 10kHz makes sense.

The current direction in accordance with the load system. If an ammeter was connected the other way around, then for example instead of I1, the value -I1would be passed as the parameter.

The current and voltage must have the same time base. They may have events; segments are also possible but not in conjunction with interval data.

The calculation only returns precise results if the interval width of the analysis is an integer number of samples.

Examples:

Power measurement at the three-phase network with Aron circuit

```
PowerSelect()
PowerParameter(50, 10)
g = Power2 ( U12, I1, U32, I3 )
P = g:P
Q = g:Q
```

Power measurement at the three-phase network with Aron circuit. Frequency fluctuates

```
PowerSelect ()
PowerParameter ( 50, 1)
ivl = IntervalFromLevel ( U12, 0, 0, 0, 1, 0.0, 1e35, 0, "")
g = Power2 ( U12, I1, U32, I3, ivl )
P = g:P
Q = g:Q
```

See also:

PowerSelect, PowerParameter, Power1, Power3, IntervalFromLevel

Power3

Available in: Professional Edition and above

Calculation of the power in a three-phase network with a 4-wire system

Declaration:

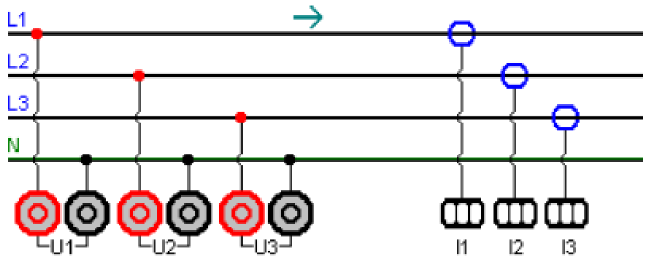
```
Power3 ( U1, I1, U2, I2, U3, I3 [, Interval data] ) -> Results group
```

Parameter:

U1	U1 = UL1 - N, phase-to-neutral voltage, phase-element voltage
I1	I1, also IL1, current through the conductor, current direction (arrow) from the source to the load
U2	U2 = UL2 - N, phase-to-neutral voltage, phase-element voltage
I2	I2, also IL2, current through one phase line, current direction (arrow) from the source to the load
U3	U3 = UL3 - N, phase-to-neutral voltage, phase-element voltage
I3	I3, also IL3, current through one phase line, current direction (arrow) from the source to the load
Interval data	Interval data; the intervals represent the signals periods. Preferably a series of seamlessly successive intervals. The calculation is based on these periods. Especially for unknown or fluctuating signal frequencies. (optional)
Results group	
Results group	Group containing all calculation results: P: active power, PF: power factor, U*: voltage RMS values, I*: current RMS values, p_t instantaneous power, f: frequency

Description:

Neutral line N is present and can conduct current. Typical in the star (Y) connection.



In order to perform additional analyses on the individual phases, it is possible to use Power1() for each of the phases. Power3() mainly returns the summarized values.

The functions PowerSelect() and PowerParameter() must first be called in order to complete the process of parameterizing the function.

If the parameter IntervalData is not specified, the calculation is performed based on the frequency which was specified in PowerParameter(). The number of signal periods over which the calculation is to be performed must be a multiple of the sampling interval when multiplied by the period duration. Thus, at 50Hz, either 1 or 10 periods are often specified; at 60Hz then 12 periods, with the sampling frequency typically 10kHz.

If the parameter IntervalData is specified, the interval lengths determine the period lengths. If there are gaps before/between/after the intervals, then for the purpose of calculations, continuously adjacent periods are assumed; the interval boundaries may no longer be used in such cases, but only their widths. The calculation starts at the first interval or at a whole number of interval widths before. If there are no intervals available, the frequency from PowerParameter() is used.

The interval data set can be determined using IntervalFromLevel(), for instance, and re-processed with other interval functions. As the basis for determining the zero crossing points, it is recommended to use an especially regular sinusoidal signal such as a voltage, for example.

Interval format: Equidistant "equi" or XY "xy", i.e. interval-code with position

The frequency is only determined when an interval data set is specified. Then it always shows the frequency on which the calculation is based, even at positions where intervals are missing.

When working with interval data, it is necessary to determine the zero crossings as precisely as possible in order to obtain exact results.

Determination of the frequency and the other results become (substantially) more exact when the analysis is performed over multiple periods.

For a precise analysis, at a mains frequency of 50Hz a sampling frequency of 10kHz makes sense.

The current direction in accordance with the load system. If an ammeter was connected the other way around, then for example instead of I1, the value -I1would be passed as the parameter.

The current and voltage must have the same time base. They may have events; segments are also possible but not in conjunction with interval data.

The calculation only returns precise results if the interval width of the analysis is an integer number of samples.

Examples:

Power measurement at three-phase network with 4 wires

```
PowerSelect()
PowerParameter(50, 10)
g = Power3 ( U1, I1, U2, I2, U3, I3 )
P = g:P
Q = g:Q
```

Power measurement at the three-phase network with four wires. Frequency fluctuates

```
PowerSelect ()
PowerParameter ( 50, 1)
ivl = IntervalFromLevel ( U1, 0, 0, 0, 1, 0.0, 1e35, 0, "")
g = Power3 ( U1, I1, U2, I2, U3, I3 , ivl )
P = g:P
Q = g:Q
```

See also:

PowerSelect, PowerParameter, Power1, Power2, IntervalFromLevel

PowerCepstrum

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power Cepstrum. The cepstrum is calculated using a moving window and linear averaging.

Declaration:

```
PowerCepstrum ( InputData, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th cepstrum is returned.
AveragingType	method of summarizing all cepstra
	0 : no averaging
	1 : averaging (arithmetic mean or linear averaging of the magnitude cepstra). The number of cepstra over which the average is taken is determined by the parameter 'Reduction'.
	2 : Peak Hold Max, from beginning. Maximum values, based on the cepstra calculated thus far in the algorithm
	3 : Peak Hold Max, interval. Maximum values, based on the number of cepstra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the cepstra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of cepstra which the parameter 'Reduction' dictates.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is a segmented waveform, where each segment represents a cepstrum.

Description:

RMS-spectrum of the double-sided, logarithmic (ln) power spectrum.

The dynamic range is limited to 8 powers of 10.

Algorithm:

- a section of the time-based waveform is taken, e.g. 1000 data points
- interpolation to obtain a power of 2 data points, e.g. 1024, if applicable
- multiplication with a window function, if applicable
- calculation of the power spectrum. Only the magnitude is kept. This leaves 513 data points.
- limiting of this spectrum to 8 powers of 10
- the natural logarithm of the magnitude is taken

- the resulting spectrum is extended to cover the corresponding negative frequencies. This returns 1024 data points.
- the RMS-magnitude spectrum of this double-sided spectrum is computed. This leaves 513 data points.
- the resulting cepstrum is subjected to subsequent averaging.

Examples:

```
PCepstra = PowerCepstrum ( Channel, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-cepstra, which each overlap their neighbors by 50%.

```
PCepstra = PowerCepstrum ( Channel, 2048, 1, 0, 10, 1, 0 )
```

This calculates a sequence of 2048 point-cepstra with a Hamming window. The average is taken of each group of ten consecutive cepstra and recorded with the results.

See also:

PowerCepstrum_exp, PowerCepstrum_1

PowerCepstrum_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power Cepstrum. Calculates a mean cepstrum. The averaging is taken of as many cepstra as there are windows within the waveform.

Declaration:

```
PowerCepstrum_1 ( InputData, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all cepstra
	1 : averaging (arithmetic mean or linear averaging of the magnitude cepstra). The mean is taken over all cepstra computed.
	2 : Peak Hold Max, maximum values, based on the cepstra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the cepstra calculated thus far in the algorithm
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged cepstrum

Description:

RMS-spectrum of the double-sided, logarithmic (ln) power spectrum.

The dynamic range is limited to 8 powers of 10.

Algorithm:

- a section of the time-based waveform is taken, e.g. 1000 data points
- interpolation to obtain a power of 2 data points, e.g. 1024, if applicable
- multiplication with a window function, if applicable
- calculation of the power spectrum. Only the magnitude is kept. This leaves 513 data points.
- limiting of this spectrum to 8 powers of 10
- the natural logarithm of the magnitude is taken
- the resulting spectrum is extended to cover the corresponding negative frequencies. This returns 1024 data points.
- the RMS-magnitude spectrum of this double-sided spectrum is computed. This leaves 513 data points.
- the resulting cepstrum is subjected to subsequent averaging.

Examples:

```
Cepstrum = PowerCepstrum_1 ( Channel, 1000, 0, 50, 1, 0 )
```

This calculates an averaged cepstrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values.

See also:

PowerCepstrum, PowerCepstrum_exp

PowerCepstrum_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power Cepstrum, exponential averaging

Declaration:

```
PowerCepstrum_exp ( InputData, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [, Base2] ) ->
Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th cepstrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	The result is a segmented waveform, where each segment represents a cepstrum.

Description:

RMS-spectrum of the double-sided, logarithmic (ln) power spectrum.

The dynamic range is limited to 8 powers of 10.

Algorithm:

- a section of the time-based waveform is taken, e.g. 1000 data points
- interpolation to obtain a power of 2 data points, e.g. 1024, if applicable
- multiplication with a window function, if applicable
- calculation of the power spectrum. Only the magnitude is kept. This leaves 513 data points.
- limiting of this spectrum to 8 powers of 10
- the natural logarithm of the magnitude is taken
- the resulting spectrum is extended to cover the corresponding negative frequencies. This returns 1024 data points.
- the RMS-magnitude spectrum of this double-sided spectrum is computed. This leaves 513 data points.
- the resulting cepstrum is subjected to subsequent averaging.

Examples:

```
Cepstra = PowerCepstrum_exp ( Channel, 1000, 0, 50, 2, 40.0, 0 )
```

This calculates a sequence of 1000 point-cepstra, which each overlap their neighbors by 50%. The channel has a sampling time of 10ms. Therefore, a spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

PowerCepstrum, PowerCepstrum_1

PowerDS

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power Density Spectrum with a moving window and linear averaging. Square of the RMS-spectrum, divided by the frequency line distance. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
PowerDS ( InputData, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
AveragingType	method of summarizing all spectra
	0 : no averaging
	1 : Averaging (arithmetic mean or linear averaging of the squares of the magnitude spectra). The mean value is formed over as many spectra as specified by the parameter Reduction.
	2 : Peak Hold Max, from beginning. Maximum values, based on the spectra calculated thus far in the algorithm.
	3 : Peak Hold Max, interval. Maximum values, based on the number of spectra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the spectra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of spectra which the parameter 'Reduction' dictates.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	segmented waveform, where each segment represents a power density spectrum.

Description:

The result is divided by the ENBW (Equivalent noise bandwidth) according to the window type used. E.g. division by 1.5 in the case of a Hanning window.

Examples:

```
PowerDensitySpectra = PowerDS ( Channel, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%.

```
PowerDensitSpectra = PowerDS ( Channel, 2048, 1, 0, 10, 1, 0 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

PowerDS_exp, PowerDS_1, PowerSpectrum

PowerDS_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

An averaged power density spectrum is determined. Square of the RMS-spectrum, divided by the frequency line distance. The averaging is taken of as many spectra as there are windows within the waveform.

Declaration:

```
PowerDS_1 ( InputData, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all spectra
	1 : Averaging (arithmetic mean or linear averaging of the squares of the magnitude spectra). The mean value is formed over all spectra calculated.
	2 : Peak Hold Max, maximum values, based on the spectra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the spectra calculated thus far in the algorithm
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged magnitude spectrum

Description:

The result is divided by the ENBW (Equivalent noise bandwidth) according to the window type used. E.g. division by 1.5 in the case of a Hanning window.

Internally applied multiplication factors for weighting with ENBW: Rectangle: 1.0; Hamming: 1.0 / 1.3628257; Hanning: 1.0 / 1.5000000; Blackman: 1.0 / 1.7267573; Blackman-Harris: 1.0 / 2.0043528; Flat top: 1.0 / 3.7702901

Examples:

```
PowerDensitySpectrum = PowerDS_1 ( Channel, 1000, 0, 50, 1, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values.

See also:

PowerDS, PowerDS_exp, PowerSpectrum_1

PowerDS_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power Density Spectrum with a moving window and exponential averaging. Square of the RMS-spectrum, divided by the frequency line distance. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
PowerDS_exp ( InputData, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

The result is divided by the ENBW (Equivalent noise bandwidth) according to the window type used. E.g. division by 1.5 in the case of a Hanning window.

Examples:

```
PowerDensitySpectra = PowerDS_exp ( Channel, 1000, 0, 50, 2, 40.0, 0 )
```

The channel has a sampling time of 10ms. Therefore, a 1000 point-spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

PowerDS, PowerDS_1, PowerSpectrum _exp

PowerParameter

Available in: Professional Edition and above

Determines which parameters and procedures to use to calculate the power.

Declaration:

```
PowerParameter ( f, Periods [, S_calc] [, Q_calc] [, Q_sign] [, PF_calc] )
```

Parameter:

f	Nominal frequency in Hz, e.g. 50 or 60
Periods	Number of periods over which the calculation is perfomed and after which a result is to be returned; e.g. 1 or 10 or 12
S_calc	Calculation of apparent power (optional , Default value: 0)
	0 : according to DIN 40110, S = Urms*Irms, Irms=sqrt(I1^2+I2^2+I3^2); 4-wire system: Urms=sqrt(U1^2+U2^2+U3^2); 3-wire system: Urms= sqrt((U12^2+U32^2+U13^2)/3)
	1 : 4-wire system: S = S1 + S2 + S3; 3-wire system S = (sqrt(3)/2)*(U12*I1+U32*I3)
Q_calc	Calculation of reactive power (optional , Default value: 0)
	0 : Reactive power = sqrt (total apparent power^2 - total active power^2)
	1 : Reactive power = Q1+Q2+Q3 (not possible with Aron circuit)
Q_sign	Reactive power sign (optional , Default value: 0)
	0 : Reactive power always positive
	1 : Reactive power with sign; positive for inductive, negative for capacitive (not possible with Aron circuit).
PF_calc	Calculation of power factor (optional , Default value: 0)
	0 : The active power's sign; positive when drawing power, negative for power output
	1 : inductive: positive when drawing power, negative when outputting; capacitive: negative when drawing power, positive for power output (not possible with Aron circuit)
	2 : always positive

Description:

The system determines the inductive/capacitive operation by analyzing the fundamental oscillation's phase reactive power U1*I1*sin(Phi1). To do this for 3 phases, the fundamental oscillation reactive power values of all phases are added.

With Phi as the angle by which the current lags, the inductive/capacitive operation is as follows: Quadrant 1: Phase Phi = 0 to 90 deg, motor-driven inductive; Quadrant 2: Phase Phi = 90 to 180 deg, generative inductive; Quadrant 3: Phase Phi = 180 to 270 deg generative capacitive; Quadrant 4: Phase Phi = 270 to 360 deg, motor-driven capacitive.

The active power is positive in Quadrants 1and 4; else negative.

The reactive power is positive in Quadrants 1 and 2, else negative if determined with the sign.

Examples:

1-phase power measurement. default calculation at 50Hz over 10 periods; sampling frequence: 10kHz

```
PowerSelect()
PowerParameter(50, 10)
g = Power1 ( U1, I1 )
```

1-phase power measurement. Default calculation at 60Hz over 12 periods; sampling frequency: 10kHz

```
PowerSelect()
PowerParameter(60, 12)
g = Power1 ( U1, I1 )
```

1-phase power measurement. Reactive power and power factor with sign

```
PowerSelect()
PowerParameter(50, 1, 0, 0, 1, 1)
g = Power1 ( U1, I1 )
```

See also:

PowerSelect, Power1, Power2, Power3

PowerSelect

Available in: Professional Edition and above

Determines which results a power calculation is to return.

Declaration:

```
PowerSelect ( [QS] [, RMS] [, PF] [, f] [, p_t] [, P] )
```

Parameter:

QS	Reactive- and apparent-power requested (optional , Default value: 1)
	0 : no
	1 : yes
RMS	RMS-values of the currents and voltages are requested. (optional , Default value: 1)
	0 : no
	1 : yes
PF	Power factor requested (optional , Default value: 1)
	0 : no
	1 : yes
f	Signal frequency requested (optional , Default value: 1)
	0 : no
	1 : yes
p_t	Instantaneous power requested (optional , Default value: 0)
	0 : no
	1 : yes
P	Active power requested (optional , Default value: 1)
	0 : no
	1 : yes

Description:

This function must be called once before calling the functions Power1(), Power2(), Power3().

If the frequency is desired, it is still only calculated if Power1(), Power2() or Power3() is called with an interval data set, because the frequency is not fixed.

Examples:

1-phase power measurement. Default calculation

```
PowerSelect()
PowerParameter(50, 10)
g = Power1 ( U1, I1 )
```

1-phase power measurement. Explicit selection of calculation of the active power and the power factor

```
PowerSelect(0, 0, 1, 0, 0, 1)
PowerParameter(50, 10)
g = Power1 ( U1, I1 )
```

See also:

PowerParameter, Power1, Power2, Power3

PowerSpectrum

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power spectrum with a moving window and linear averaging. The square of the RMS-spectrum. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
PowerSpectrum ( InputData, WindowWidth, WindowType, Overlapping, Reduction, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
AveragingType	method of summarizing all spectra
	0 : no averaging
	1 : Averaging (arithmetic mean or linear averaging of the squares of the magnitude spectra). The mean value is formed over as many spectra as specified by the parameter Reduction.
	2 : Peak Hold Max, from beginning. Maximum values, based on the spectra calculated thus far in the algorithm.
	3 : Peak Hold Max, interval. Maximum values, based on the number of spectra which the parameter 'Reduction' dictates.
	4 : Peak Hold Min, from beginning. Minimum values, based on the spectra calculated thus far in the algorithm
	5 : Peak Hold Min, interval. Minimum values, based on the number of spectra which the parameter 'Reduction' dictates.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

Examples:

```
PowerSpectra = PowerSpectrum ( Channel, 1000, 0, 50, 1, 0, 0 )
```

This calculates a sequence of 1000 point-spectra, which each overlap their neighbors by 50%.

```
PowerSpectra = PowerSpectrum ( Channel, 2048, 1, 0, 10, 1, 0 )
```

This calculates a sequence of 2048 point-spectra with a Hamming window. The average is taken of each group of ten consecutive spectra and recorded with the results.

See also:

PowerSpectrum_exp, PowerSpectrum_1, PowerDS

PowerSpectrum_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

An averaged power spectrum is computed. the square of the RMS-spectrum. The averaging is taken of as many spectra as there are windows within the waveform.

Declaration:

```
PowerSpectrum_1 ( InputData, WindowWidth, WindowType, Overlapping, AveragingType [, Base2] ) -> Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
AveragingType	method of summarizing all spectra
	1 : Averaging (arithmetic mean or linear averaging of the squares of the magnitude spectra). The mean value is formed over all spectra calculated.
	2 : Peak Hold Max, maximum values, based on the spectra calculated thus far in the algorithm
	4 : Peak Hold Min, minimum values, based on the spectra calculated thus far in the algorithm
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	One averaged magnitude spectrum

Description:

Examples:

```
PowerSpectrum = PowerSpectrum_1 ( Channel, 1000, 0, 50, 1, 0 )
```

This calculates an averaged spectrum. The averaging is performed on a sequence of 1000 point-spectra which each overlap their neighbors by 50%. The input channel contains approx. 20000 measured values.

See also:

PowerSpectrum, PowerSpectrum_exp, PowerDS_1

PowerSpectrum_exp

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Power spectrum with a moving window and exponential averaging. The square of the RMS-spectrum. The result is a segmented waveform, where each segment represents a spectrum.

Declaration:

```
PowerSpectrum_exp ( InputData, WindowWidth, WindowType, Overlapping, Reduction, TimeConstant [, Base2] ) ->
Result
```

Parameter:

InputData	Time waveform, the time scaled in seconds
WindowWidth	Width of time window in points, >= 4. If not a power of 2, then the system interpolates to a smaller sampling interval in accordance with the parameter 'Base2'.
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Overlapping	The time windows overlap by this percentage.
	0 : no overlapping
	> 0 : > 0 (but < 100) : Overlapping. Computation time increases with percentage.
	< 0 : < 0 This percentage of the window width is left as a margin before the next window. This means that these values will be disregarded by the calculations.
Reduction	>= 1: Only every n-th spectrum is returned.
TimeConstant	The time constant used in taking the exponential mean. Specified in seconds.
Base2	Perform internal calculation of FFT only with powers of 2 (Base 2), or also with other window widths? The value 3 is recommended. If omitted, 2 will be used. (optional)
	2 : If the window width is not a power of 2, the data are interpolated to a power of 2 for the purpose of an FFT-calculation.
	3 : FFT with all window widths which are products of powers of 2, 3, 5; no interpolation of time-domain data
Result	
Result	Segmented waveform, where each segment represents a spectrum.

Description:

Examples:

```
PowerSpectra = PowerSpectrum_exp ( Channel, 1000, 0, 50, 2, 40.0, 0 )
```

The channel has a sampling time of 10ms. Therefore, a 1000 point-spectrum is computed every 5s. These are smoothed with a time constant of 40.0s. Every second spectrum is returned.

See also:

PowerSpectrum_1, PowerSpectrum, PowerDS_exp

PptAddSlides

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Slides from a PowerPoint file are inserted into the presentation.

Declaration:

```
PptAddSlides ( SlideIndex, TxFilename [, SlideStartIndex] [, SlideEndIndex] ) -> Result
```

Parameter:

SlideIndex	Index of the slide following which the slides from the file are inserted. Enter a 0 if you wish to insert the slides before the first slide.
TxFilename	Complete pathname of a PowerPoint file whose slides are inserted.
SlideStartIndex	Index of the first slide which is to be inserted from the file. (1...) (optional) (optional)
SlideEndIndex	Index of the last slide from the file which is to be inserted (>= SlideStartIndex or -1) (optional) (optional)
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

Slides from a PowerPoint file are inserted into the presentation.

If the parameters 'SlideStartIndex' and 'SlideEndIndex' are not specified, then all slides from the PowerPoint file are inserted.

If only the parameter 'SlideStartIndex' is specified, then only the slide having the specified index is inserted from the PowerPoint file.

If both parameters are specified, then the slides are inserted from the PowerPoint file between the specified indices.

If the parameter 'SlideEndIndex'= -1 is specified, then the slides from 'SlideStartIndex' up to the last slide are inserted.

Examples:

The PowerPoint file 'Presentation1' is opened. Subsequently, all slides from the file 'Presentation2' are inserted at the beginning.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
count = PptGetSlidesCount( )
result= PptAddSlides( 0,"c:\imc\ppt\Presentation2.pptx")
count = PptGetSlidesCount( )
result= PptClosePresentation( )
```

The PowerPoint file 'Presentation1' is opened. Subsequently, the 2nd slide from the file 'Presentation2' is inserted after the last slide.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
count = PptGetSlidesCount( )
result= PptAddSlides( count,"c:\imc\ppt\Presentation2.pptx",2)
result= PptClosePresentation( )
```

See also:

PptGetSlidesCount, PptDuplicateSlide, PptDeleteSlide, PptMoveSlide

PptClosePresentation

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

The presentation will be closed.

Declaration:

```
PptClosePresentation ( ) -> Result
```

Parameter:

Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

With this function, the PowerPoint instance is simultaneously ended.

Any files still open are closed. Any changes made will be lost.

This function should always be called at the end of a sequence.

See also:

PptOpenPresentation, PptSavePresentation

PptDeleteSlide

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

A slide is deleted from the presentation.

Declaration:

```
PptDeleteSlide ( SlideIndex ) -> Result
```

Parameter:

SlideIndex	Index of the slide to be deleted. The first slide has the index 1.
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

A slide is deleted from the presentation.

Examples:

The first and last slide are deleted from the presentation.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
result= PptDeleteSlide( 1)
count = PptGetSlidesCount( )
result= PptDeleteSlide(count)
result= PptSavePresentation("c:\imc\ppt\Presentation3.pptx")
result= PptClosePresentation( )
```

See also:

PptAddSlides, PptGetSlidesCount, PptDuplicateSlide

PptDuplicateSlide

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Duplicates one of the presentation's slides.

Declaration:

`PptDuplicateSlide ( SlideIndex ) -> Result`

Parameter:

SlideIndex	Index of the slide to be duplicated (1...)
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

One of the presentation's slides is duplicated.

The duplicated slide is inserted into the presentation directly after the original slide.

Examples:

The presentation's last slide is duplicated.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
count = PptGetSlidesCount( ) ; Index der letzten Folie bestimmen
result= PptDuplicateSlide(count) ; slide duplication
result= PptSavePresentation("c:\imc\ppt\Presentation3.pptx")
result= PptClosePresentation( )
```

See also:

PptAddSlides, PptDeleteSlide, PptMoveSlide, PptGetSlidesCount

PptFindSlideByAlternativeText

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Finds a slide with a shape object having the specified alternative text.

Declaration:

`PptFindSlideByAlternativeText ( StartIndex, TxAlternativeText ) -> Result`

Parameter:

StartIndex	Index of the slide at which the search is to begin (1...)
TxAlternativeText	The alternative text of the shape object
Result	
Result	Result:
	>0 : Index of the slide containing the shape object
	=0 : There is no slide with a shape object having the specified alternative text.
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

From the 'startindex', the presentation is searched for slides which contain a shape object having the specified alternative text.

At the first occurrence of the shape object, the slide's index is returned.

See also:

PptAddSlides, PptDeleteSlide, PptGetSlidesCount, PptDuplicateSlide, PptMoveSlide, PptGetSlidesCount

PptGetSlidesCount

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

The number of slides in the presentation is determined.

Declaration:

```
PptGetSlidesCount ( ) -> Result
```

Parameter:

Result	
Result	Result:
	>=0 : Slide count
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

The count of slides is determined.

At fault condition, -1 is returned. The cause can be determined using the function GetLastError().

See also:

PptAddSlides, PptDeleteSlide, PptDuplicateSlide, PptMoveSlide

PptMoveSlide

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Moves the specified slide to a specific position in the presentation.

Declaration:

`PptMoveSlide ( SlideIndex, ToPosition ) -> Result`

Parameter:

SlideIndex	Index of the slide to be moved (1...)
ToPosition	Index of the new position (1...)
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

One of the presentation's slides is moved.

All other slides in the presentation are renumbered accordingly.

Examples:

The presentation's 1st slide is duplicated and moved to the presentation's last position.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
slideindex=1
result= PptDuplicateSlide(slideindex) ; slide duplication
slideindex= slideindex+1; index of the duplicated slide
count = PptGetSlidesCount( ) ; Index der letzten Folie bestimmen
result= PptMoveSlide(slideindex,count); duplicated slide moved to the last position
result= PptSavePresentation("c:\imc\ppt\Presentation3.pptx")
result= PptClosePresentation( )
```

See also:

PptAddSlides, PptDeleteSlide, PptGetSlidesCount, PptDuplicateSlide, PptGetSlidesCount

PptOpenPresentation

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

The specified file in the PowerPoint format is opened.

Declaration:

```
PptOpenPresentation ( TxFilename, Visible ) -> Result
```

Parameter:

TxFilename	Complete pathname of the file to be opened
Visible	Visibility of the work window
	0 : The PowerPoint work window is not visible.
	1 : The PowerPoint work window is visible.
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

The specified PowerPoint- Datei is opened. To do this, a PowerPoint instance is started unless it already has been.

If another presentation was opened by the same execution thread before this call, it will be closed.

Multithreading: Each execution thread uses its own Powerpoint instance. If, for example, Powerpoint is started in a parallel sequence function (BEGIN_PARALLEL) using PptOpenPresentation () and a document is loaded, further access to this document is only permitted within the same sequence function. If the Powerpoint instance was not explicitly closed with PptClosePresentation (), it is closed automatically at the end of the sequence function.

Examples:

A PowerPoint file is opened. Slides from another PowerPoint file are added. The presentation is saved under a different name.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1);
count = PptGetSlidesCount( )
result= PptAddSlides(count,"c:\imc\ppt\Presentation2.pptx")
result= PptSavePresentation("c:\imc\ppt\Result.pptx")
result= PptClosePresentation( )
```

See also:

PptSavePresentation, PptClosePresentation

PptPrintPresentation

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

The presentation is printed

Declaration:

```
PptPrintPresentation ( [FromSlideIndex] [, ToSlideIndex] ) -> Result
```

Parameter:

FromSlideIndex	Index of the first slide to be printed (1...) (optional) (optional)
ToSlideIndex	Index of the last slide to be printed (optional) (optional)
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

The entire presentation/slides from the presentation are printed.

If the parameters 'FromSlideIndex' and 'ToSlideIndex' are not specified, the entire presentation is printed.

If only the parameter 'FromSlideIndex' is specified, then only the slide having the specified index is printed.

If both parameters are specified, then the slides between the specified indices are printed.

Examples:

The presentation Presentation1.pptx is opened. All slides are printed.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
result= PptPrintPresentation( )
result= PptClosePresentation( )
```

The presentation Presentation1.pptx is opened. The 2nd slide is printed.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
result= PptPrintPresentation(2)
result= PptClosePresentation( )
```

See also:

PptOpenPresentation, PptSavePresentation

PptSavePresentation

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

The presentation is saved

Declaration:

```
PptSavePresentation ( TxFilename [, FileFormat] ) -> Result
```

Parameter:

TxFilename	Complete pathname under which the presentation is to be saved
FileFormat	Sets the file format for data saving. If the parameter is not specified, then the data are saved in the default format (optional). (optional)
	0 : Save presentation in the default format
	1 : save as PDF file.
	2 : save as JPG file.
	3 : save as PNG file.
	4 : save as BMP file.
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

The presentation is saved under the specified filename.

If the parameter 'FileFormat' is missing, the presentation is saved in PowerPoint's default format.

Data are also saved in the default format if the parameter =0.

With 'FileFormat' = 1, the presentation is saved as a PDF file.

With the formats 2, 3 and 4, a folder name is constucted from the filename. In this folder, each slide is saved as a JPG, PNG or BMP file.

Examples:

A presentation is opened and saved in various formats.

```
result=PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1);
result=PptSavePresentation("c:\imc\ppt\test.pptx"); Standardformat
result=PptSavePresentation("c:\imc\ppt\test1.pptx",0); Standardformat
result=PptSavePresentation("c:\imc\ppt\test.pdf",1)  ; PDF-format
result=PptSavePresentation("c:\imc\ppt\testa.jpg",2) ; JPG-format
result=PptSavePresentation("c:\imc\ppt\testb.png",3) ; PNG-format
result=PptSavePresentation("c:\imc\ppt\testc.bmp",4) ; BMP-format
result=PptClosePresentation( )
```

See also:

PptOpenPresentation, PptClosePresentation

PptSetCellText

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Sets the text in a table cell in the specified slide.

Declaration:

```
PptSetCellText ( SlideIndex, TxAlternativeText, Row, Column, TxContent ) -> Result
```

Parameter:

SlideIndex	Index of the slide. The index of the first slide is 1.
TxAlternativeText	The table's alternative text
Row	Row number of cell (1...)
Column	Column number of cell (1...)
TxContent	This text is transferred into the cell.
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

By means of the 'SlideIndex', the slide in the presentation is determined. The index corresponds to the slide's position in the presentation.

The system searches in the specified slide for the table having this alternative text.

Once the table has been found, the text supplied is transferred into the cell.

The cell is addressed by specifying its row and column numbers.

Like a text box, a cell can have multiple paragraphs. In this case, the text transferred is split off into partial strings.

The text is split at the characters CR ("~013") , LF ("~010") or NL ("~013"+ "~010").

Each partial string is assigned to the corresponding paragraph of the cell.

The alternative text for a table is defined in PowerPoint as follows:

1. On the Slide, select the Table object.
2. Right-click the mouse in the element and select "Format Shape".
3. Click on "Size and Properties" and then on "Alt Text".
4. In the box "Description", enter at designating text such as FAMOS_Table1. This designating text is used in the Kit-functions to locate the shape object.
5. **Important!** Don't enter the designating text in the box "Title"!
6. If you wish to use the alternative text for it's original function (barrier-free PowerPoint), then first enter the designating text in the box "Description", followed by a semicolon (;). After that enter the text intended for the barrier free PowerPoint. The semicolon is not part of the designating text.

Examples:

The presentation contains a table with 3 columns and 6 rows. The cells of the 2nd through 4th row are set.

```
result=pptOpenPresentation( "c:\imc\ppt\PresentationTable1.pptx",1)
result=PptSetCellText( 1,"Table1",2,1,"Channel 1"); Channel name -> column 1
result=PptSetCellText( 1,"Table1",2,2,"34.06"); Maximum -> column 2
result=PptSetCellText( 1,"Table1",2,3,"-3.02"); Minimum -> Column 3
result=PptSetCellText( 1,"Table1",3,1,"Channel 2")
result=PptSetCellText( 1,"Table1",3,2,"10.4555")
result=PptSetCellText( 1,"Table1",3,3,"1.02")
result=PptSetCellText( 1,"Table1",4,1,"Channel 3")
result=PptSetCellText( 1,"Table1",4,2,"210.4555")
result=PptSetCellText( 1,"Table1",4,3,"-23.02")
result=pptsavePresentation("c:\imc\ppt\test.pptx")
result=pptClosePresentation( )
```

See also:

PptSetText, PptSetPicture, PptSetCurve

PptSetCurve

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Transfers the selected curve window as a picture into the specified slide.

Declaration:

```
PptSetCurve ( SlideIndex, TxAlternativeText, SizeOption, Position, Screen ) -> Result
```

Parameter:

SlideIndex	Index of the slide. The first slide's index is 1.
TxAlternativeText	The picture's alternative text
SizeOption	Modification of picture size
	0 : The picture's orginal size is used.
	1 : The placeholder's size is used.
	2 : The size of the placeholder is used. In the process, the aspect ratio of the orginal picture is retained.
Position	The portion of the shape (picture placeholder) which retains its position when the shape is scaled
	0 : Top left
	1 : Bottom right
	2 : Centered
Screen	Monitor screen or printer
	"screen" : To generate the picture, the curve window's monitor screen settings are used.
	"printer" : To generate the picture, the curve window's printer settings are used.
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

A curve window is transferred into the presentation as a picture. For this purpose, the slide must have a picture or a placeholder.

The curve window needs to have already been generated and selected before this function is called. To do this, the Curve Window Kit's functions (Cw...- functions) can be used.

By means of the 'SlideIndex', the slide in the presentation is determined. The index corresponds to the slide's position in the presentation.

The system searches in the specified slide for the picture placeholder having the specified alternative text

Once the picture has been found, the curve window is transferred into the picture in the presentation.

With 'SizeOption' = 0, the curve window is transferred to the placeholder in its orginal size.
The parameter 'Position' determines at which position in the placeholder the picture is anchored.
If the curve window is larger than the placeholder, it may occur that other shape objects are obscured.

With 'SizeOption' = 1, the curve window is fitted into the area of the picture placeholder.
It completely fills the placeholder area. The parameter 'Position' is not relevant.
With this option, there can be distortions of the curve window.

With 'SizeOption' = 2, the curve window is fitted into the area of the picture placeholder while retaining its aspect ratio.
By means of the parameter 'Position', you can set at which position in the placeholder the curve window is anchored.

Proceed as follows to define the alternative text for a picture in PowerPoint :

1. On the Slide, select the Shape object.
2. Right-click the mouse in the element and select "Format Graphic".
3. Click on "Size and Properties" and then on "Alt Text".
4. In the box "Description", enter at designating text such as FAMOS_Cw1. This designating text is used in the Kit-functions to locate the shape object.
5. **Important!** Don't enter the designating text in the box "Title"!
6. If you wish to use the alternative text for it's original function (barrier-free PowerPoint), then first enter the designating text in the box "Description", followed by a semicolon (;). After that enter the text intended for the barrier free PowerPoint. The semicolon is not part of the designating text.

Examples:

In this sequence, two curve windows are created. Next, the presentation 'PresentationCW3' is opened.
The presentation has two placeholders on Slide 1, with the text alternatives 'CW1' and 'CW2'.
First the curve window 'Arch' is selected and transferred to the picture with the text alternative 'CW1'.
Then the curve window 'Weight' is selected and transferred into the picture with the text alternative 'CW2'.

```
FileLoad("C:\imc\Dat\bogen.dat","",0)
CwNewWindow( bogen,"show")
CwNewChannel( "append new axis", bogen)
FileLoad("C:\imc\Dat\gewicht.dat","",0)
CwNewWindow( gewicht,"show")
CwNewChannel( "append new axis", gewicht)
screen= "screen"
result= PptOpenPresentation( "c:\imc\ppt\PräsentationCW3.pptx",1);
CwSelectWindow(bogen)
result= PptSetcurve(1,"CW1",2,0,screen); Kurvenfenster "Bogen" -> CW1
CwSelectWindow(gewicht)
result= PptSetcurve(1,"CW2",2,0,screen); Kurvenfenster "Gewicht" -> CW2
result= PptSavePresentation("c:\imc\ppt\test.pptx")
result= PptClosePresentation( )
```

See also:

PptSetCellText, PptSetText, PptSetPicture

PptSetPicture

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Transfers the picture to a shape in the specified slide.

Declaration:

```
PptSetPicture ( SlideIndex, TxAlternativeText, TxPictureFilename, SizeOption, Position ) -> Result
```

Parameter:

SlideIndex	Index of the slide. The first slide's index is 1.
TxAlternativeText	The picture's alternative text
TxPictureFilename	Complete pathname of the picture file
SizeOption	Modification of picture size
	0 : The picture's orginal size is used.
	1 : The placeholder's size is used.
	2 : The size of the placeholder is used. In the process, the aspect ratio of the orginal picture is retained.
Position	The portion of the shape (picture placeholder) which retains its position when the shape is scaled
	0 : Top left
	1 : Bottom right
	2 : Centered
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

By means of the 'SlideIndex', the slide in the presentation is determined. The index corresponds to the slide's position in the presentation.

The system searches in the specified slide for the picture having this alternative text. It must be either a picture object or a picture placeholder.

Once the picture has been found, the picture from the file is transferred into the picture in the presentation.

With 'SizeOption' = 0, the picture from the file is transferred into the placeholder in its orginal size.
The parameter 'Position' specifies at which position in the placeholder the picture is anchored.
If the orginal picture is larger than the placeholder, it may occur that other shape objects are obscured.

With 'SizeOption' = 1, the original picture is fitted into the area of the picture placeholder.
It completely fills the placeholder area. The parameter 'Position' is not relevant.
With this option, there can be distortions of the picture.

With 'SizeOption' = 2, the original picture is fitted into the area of the picture placeholder while retaining its aspect ratio.
By means of the parameter 'Position', you can set at which position in the placeholder the picture is anchored.

The alternative text for a picture is defined in PowerPoint as follows:

1. On the Slide, select the picture object.
2. Right-click the mouse in the element and select "Format Graphic".
3. Click on "Size and Properties" and then on "Alt Text".
4. In the box "Description", enter at designating text such as FAMOS_Pic1. This designating text is used in the Kit-functions to locate the shape object.
5. **Important!** Don't enter the designating text in the box "Title"!
6. If you wish to use the alternative text for it's original function (barrier-free PowerPoint), then first enter the designating text in the box "Description", followed by a semicolon (;). After that enter the text intended for the barrier free PowerPoint. The semicolon is not part of the designating text.

Examples:

The picture 'Tulips.jpg' is transferred into the presentation at the placeholder having the alternative text 'Picture1'.
The original picture's aspect ratio is retained. It is centered over the placeholder.

```
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
result= PptSetPicture( 1,"Picture1","c:\imc\ppt\Tulips.jpg",2,2)
result= PptSavePresentation("c:\imc\ppt\Presentation4.pptx")
```

```
result= PptClosePresentation( )
```

See also:

PptSetCellText, PptSetText, PptSetCurve

PptSetText

Scope: PowerPoint remote control

Available in: Professional Edition and above (PowerPoint-Kit)

Sets the text in a text box in the specified slide.

Declaration:

```
PptSetText ( SlideIndex, TxAlternativeText, TxContent ) -> Result
```

Parameter:

SlideIndex	Index of the slide. The first slide's index is 1.
TxAlternativeText	The text box's alternative text
TxContent	This text is transferred into the text box.
Result	
Result	Result:
	0 : successful
	-1 : An error occurred. The cause can be determined by means of the function GetLastError().

Description:

By means of the 'SlideIndex', the slide in the presentation is determined. The index corresponds to the slide's position in the presentation.

In the specified slide, the system searches for the text box having the alternative text.
All text boxes which are to be set by the kit require an alternative text.

Once the text box is found, the text supplied is transferred into the text box.

The formatting of the text box remains intact.

If the text box has multiple paragraphs, then the text passed is split into multiple partial strings.

The text is split at the characters CR ("~013") , LF ("~010") or NL ("~013"+ "~010").

Each partial string is assigned to the corresponding paragraph.

The alternative text for a text box is defined in PowerPoint as follows:

1. On the Slide, select the shape object.
2. Right-click the mouse in the element and select "Format Shape".
3. Click on "Size and Properties" and then on "Alt Text".
4. In the box "Description", enter at designating text such as FAMOS_Text1. This designating text is used in the Kit-functions to locate the shape object.
5. **Important!** Don't enter the designating text in the box "Title"!
6. If you wish to use the alternative text for it's original function (barrier-free PowerPoint), then first enter the designating text in the box "Description", followed by a semicolon (;). After that enter the text intended for the barrier free PowerPoint. The semicolon is not part of the designating text.

Examples:

Slide 1 contains the text boxes with the alternative texts 'Marker1' and 'Marker2'. The text box 'Marker2' has 3 paragraphs.

```
cr="~013"
result= PptOpenPresentation( "c:\imc\ppt\Presentation1.pptx",1)
result= PptSetText(1,"Marker1","FAMOS Presentation")
measurementname ="Measurement 01"
measurementbegin="2017-09-25 14:30:00"
measurementend  ="2017-09-25 14:36:00"
textcontent=measurementname+cr+measurementbegin+cr+measurementend
result= PptSetText(1,"Marker2",textcontent)
result= PptSavePresentation("c:\imc\ppt\Presentation3.pptx")
result= PptClosePresentation( )
```

See also:

PptSetCellText, PptSetPicture, PptSetCurve

PrConfig

Scope: Report Generator

A configuration file is loaded to the Report Generator.

Declaration:

```
PrConfig ( TxFileName )
```

Parameter:

TxFileName	Name of file to load.

Description:

This function opens a configuration file into the Report Generator REPORT.EXE. This configuration file was previously created using the Report Generator. A configuration file is usually a mask consisting of dummy objects which serve as "place-holders". The place-holders can be filled with the desired contents, e.g. current graphs, values and text.

If the filename specified doesn't have an extension, the extension ".drb" is assumed.

Unless a complete pathname is supplied, the default directory valid for the calling application (the report directory) is used.

- This function is provided for reasons of compatibility. For newly created sequences or programs, RgDocOpen should be used.
- Both masks and completed reports can be loaded.
- The application REPORT.EXE is started (minimized) if it was not started before. Calling the application indirectly is preferable to opening it manually.
- A maximum of one (1) Report Generator application can be started.
- To specify a file name with no extension, end the name with a period (".")..

Examples:

```
TxError$ = PrConfig("report")
PrSet ("Herbert Mustermann", "Name")
```

This code example illustrates how to replace placeholders in a mask with current values using the function PrSet.

See also:

RgDocOpen

PrMove

Scope: Report Generator

Arranges an object relative to another object.

Declaration:

```
PrMove ( TxTitle, TxReference, Xmm, Ymm, Zero ) -> Zero
```

Parameter:

TxTitle	Object title
TxReference	Title of reference object
Xmm	Distance [mm] in x-direction to left edge of reference object
Ymm	Distance [mm] in y-direction to top of reference object
Zero	Reserved, always set to 0
Zero	
Zero	Result; always 0

Description:

This function moves an object TxTitle relative to a reference object TxTitleRef. The parameter SvXmm specifies the distance between the left edges of the objects is specified in millimeters. The parameter SvYmm specifies the distance between the top edges of the objects is specified in millimeters by the parameter SvYmm. If the reference object does not exist or is specified as an empty string (""), the upper left corner of the page is used as the reference point.

- Decimal values can be specified for SvXmm and SvYmm.
- Grid snap is ignored by this funct.
- The reference point for curve objects is the upper left corner of the coordinate system.

Examples:

```
Sv0 = PrMove("Text1", "Text2", 10, -15,0)
```

In this code example, an object ("Text1") is moved 10 mm to the right and 15 mm above the reference object ("Text2").

```
Sv0 = DrMove("Curve1", "", 5, 5,0)
```

In this code example, an object ("Curve1") is moved 5 mm to the right and 5 mm below the upper left corner of the page.

See also:

PrPrint

Scope: Report Generator

A finished document is printed out.

Declaration:

```
PrPrint ( )
```

Parameter:

Description:

The graphic in the current layout is printed out on the printer set for the system.

- This function is provided for reasons of compatibility. For newly created sequences or programs, RgDocPrint should be used.
- The Report Generator file REPORT.EXE must be loaded as an application.
- The printer is set up using the menu option "File"/"Printer Setup.." in the main window of the Print Layout.
- Reports can be saved and printed together during the night, instead of printing a single report immediately.

Examples:

```
TxError$ = PrConfig("report")
PrSet ("Herbert Mustermann", "Name")
PrPrint()
```

A report is usually created by loading a mask and then setting the elements to the current values. The ready report can then be printed.

See also:

RgDocPrint

PrRdClip

Scope: Report Generator

Copies the contents of the Clipboard to the Report Generator.

Declaration:

```
PrRdClip ( TxTitleRef, Xmm, Ymm, Zero, Zero, Format, Attribute, Zero ) -> Error
```

Parameter:

TxTitleRef	Title of reference object
Xmm	Distance [mm] in x-direction to left edge of reference object
Ymm	Distance [mm] in y-direction to top of reference object
Zero	Reserved, always set to 0
Zero	Reserved, always set to 0
Format	File format
	0 : Automatic selection of file format
	1 : Text
	2 : Bitmap (pixel graphics)
	3 : Metafile (vector graphics)
Attribute	Attribute, see below for description
Zero	Reserved, always set to 0
Error	
Error	Error status
	0 : No errors occurred
	1 : Incorrect parameter
	2 : Specified file format and file format of Clipboard different or Clipboard is empty
	3 : Report Generator not available or not operable
	4 : Insufficient memory or object is too large

Description:

Definition of Attribute parameter

SvAttrib	**Characteristics of the object**
0:	New object appears in foreground
Add +1:	New object appears in background
+2:	Contents of Clipboard copied to the reference object, whereas its characteristics are maintained

This function copies the contents of the Clipboard into a Print Layout document. The new object is arranged relative to the reference object TxTitleRef. The parameter SvXmm specifies the distance between the left edges of the objects is specified in millimeters. The parameter SvYmm specifies the distance between the top edges of the objects is specified in millimeters by the parameter SvYmm. If the reference object does not exist or is specified as an empty string (""), the upper left corner of the page is used as the reference point.

- Decimal values can be specified for SvXmm and SvYmm.
- Grid snap is ignored by this function.
- The reference point for curve objects is the upper left corner of the coordinate system.
- If a new text object is created and an existing text object is specified as the reference object, the font and color of the reference text object are applied to the new one.
- If the contents of the Clipboard are copied to the reference object (SvAttrib = 2 or 3), the type of the reference object is changed according to the file format of the Clipboard. Set parameters SvXmm and SvYmm to zero in this case.

Examples:

```
Error= DrRdClip("", 5, 10, 0, 0, 1, 1, 0)
```

In this code example, a text object (SvFormat=1) from the Clipboard is placed 5 mm to the right and 10 mm below the upper left corner of the page. Text is specified as the file format. The new object placed in the background (SvAttrib=1).

```
Error= DrRdClip("Text1", 0, 0, 0, 0, 1, 3, 0)
```

In this code example, a text object (SvFormat=1) from the Clipboard is copied onto a reference text object. The text in the reference text object is overwritten and the object appears in the background.

See also:

PrMove

PrSave

Scope: Report Generator

Saves a Report Generator configuration to a file.

Declaration:

```
PrSave ( TxFileName ) -> TxError
```

Parameter:

TxFileName	Name of file to be created. Can be specified complete with folder and filename extension.
TxError	
TxError	If the function successfully wrote the file, an empty text is returned; otherwise an error text.

Description:

The configuration currently stored in the Report Generator REPORT.EXE is saved to a file.

If the filename specified doesn't have an extension, the extension ".drb" is assumed.

Unless a complete pathname is supplied, the default directory valid for the calling application (the report directory) is used.

- This function is provided for reasons of compatibility. For newly created sequences or programs, RgDocSave should be used.
- Be careful not to overwrite a mask with a document. A document file cannot be used as a mask because all of the place-holders have been replaced.
- To specify a file name with no extension, end the name with a period (".").

Examples:

```
;... The Report is created.
PrSet ("Herbert Mustermann", "Name")
;...
Error$ = PrSave("report")
IF TComp(Error$,"")
   ok = BoxMessage("Caution!", Error$, "!1")
END
```

In this code example, a report is created and saved as "report.drb". Error handling is performed to check for saving errors. If an error occurs, the corresponding message is generated.

See also:

RgDocSave

PrSet

Scope: Report Generator

Assigns contents to place-holders in a Report Generator mask.

Declaration:

```
PrSet ( Contents, TxTitle )
```

Parameter:

Contents	New content to replace placeholder. Data type (waveform, single value, text) depends on the placeholder.
TxTitle	Title of Report Generator object to be changed.

Description:

This function enables you to fill a Report Generator mask with actual data, text, etc. All objects are identified by their titles.

Text objects:
Text objects may contain placeholders within the text itself, indicated by a "#"-prefix . When data is transferred to a text object, the first place-holder with the appropriate format is replaced. Place-holders for real numbers are "#e" and "#f" , for text "#s".

The following place-holders are defined for text objects:

#d	Current date in WINDOWS format
#z	Current time in WINDOWS format
#u	Units of a single value or y-units of a data set
#e?	Floating-point number, number of decimal places
#f?.?	Floating-point number, number of decimal places before and after decimal
#s	Text

"?" stands for digits.

Curve Objects:
Placeholders are not actually used for curve objects. When a curve window is transferred to a curve object, the new image replaces the old contents. If the curve object was empty (a mask), then the graph is simply pasted in.

A curve window is identified by means of a reference waveform. This curve window reference waveform is set when a curve window is created using the Cw*(..) functions of the Curve Kit or the FAMOS command "Show". It often corresponds to the first waveform displayed in the window and the window title.

- This function is provided for reasons of compatibility. For newly created sequences or programs, use the object-specific functions (menus: "Table", "Curve", "Text").
- This function can be used only when a Report Generator configuration file (mask) has been loaded.
- For a detailed description of "place-holders", please refer to the "Report Generator" chapter in the text object section.
- The place holders "#d" (current date) and "#z" (current time) are replaced the first time the text object is accessed via the function PrSet.
- If a complete waveform is to be transferred to a text object with real numbered place-holders, then a place-holder is replaced for each sample (y-value). In this way, a complete table (or column/ row) can be replaced using a single command.

A placeholder can be replaced only once. To create another document using the same mask, the mask must be reloaded using the PrConfig function. Be careful not to overwrite a configuration (mask) file!

Examples:

First, a mask is loaded.

```
TxError$ = PrConfig("report")
```

Two text objects are defined in the layout (this mask):

Title	Contents
Name	#s
Number	x: #e2 and #f2.1

The following lines are executed:

```
PrSet("Herbert", "Name")
PrSet(27.5, "Number")
PrSet(28.9, "Number")
```

Then, the text objects contain the following texts:

Title	Contents
Name	Herbert
Number	x: 2.75E+01 and 28.9

See also:

RgTextSet, RgTextGet, RgTextSetData, RgCurveSet, RgTableSetCell, RgTableSetRow, RgTableSetColumn

PrTitleI

Scope: Report Generator

The number of objects with titles is retrieved.

Declaration:

```
PrTitleI ( Zero ) -> SvAmount
```

Parameter:

Zero	Reserved, always set to 0
SvAmount	
SvAmount	Amount of objects with a title.

Description:

This function returns the number of titled objects in a Print Layout document.

This function is provided for reasons of compatibility. For newly created sequences or programs, RgObjGetCount should be used.

Examples:

```
Amounr = PrTitleI(0)
```

See also:

RgObjGetCount, RgObjGetTitle, RgObjGetType, DrTitleN

PrWin

Scope: Report Generator

Controls main Report-Generator window.

Declaration:

`PrWin ( Task )`

Parameter:

Task	Task
	1 : Display layout in normal size.
	2 : Display layout as icon.
	3 : Close layout.

Description:

This function enables you to control the Report Generator window according to the control parameter SvTask.

- This function is provided for reasons of compatibility. For newly created sequences or programs, RgWindow should be used.
- If the Report Generator was not already started, it is automatically started by this function.
- Opening the Print Layout via a function call is preferable to directly starting the executable REPORT.EXE, because this name may be changed in a future version.

Examples:

```
PrWin(1)
```

The layout is displayed as a normal window by the program. The user can now work directly in the Report Generator.

See also:

RgWindow

PulseDuration

Available in: Professional Edition and above

Duration/width, or even the frequency of pulses, determined in relation to time.

Declaration:

`PulseDuration ( Dataset, Window [, Slope] [, Calculation] [, Boundary factor] ) -> Result`

Parameter:

Dataset	Dataset
Window	Width of the averaging interval expressed in x-units; rounded to whole multiples of the sampling interval
Slope	Recognition of the signal edge which initiates a pulse (optional , Default value: "0-1")
	"0-1" : Transition from 0 to 1 (nonzero). The 1 is the exact position of the pulse beginning. Positive signal edge.
	"1-0" : Transition from 1 (nonzero) to 0 . The 0 is the exact position of the pulse beginning. Negative signal edge.
	"+" : Linearly interpolated zero crossing with positive signal edge. In a transition from <= 0 to > 0, the exact position lies between these values.
	"-" : Linearly interpolated zero crossing with negative signal edge. In a transition from >0 to <=0, the exact position lies between these values.
	"ifa" : Compatible with imc Inline FAMOS: All transitions from 0 to 1 (nonzero) and vice versa are taken into account. The average pulse duration is formed from all whole impulses in the window. If no impulse is completed within the window, the last pulse duration is returned or, if the window width times the count of windows is greater than the last pulse duration, the product of the window width and window count. Pulses from later times always remain disregarded. The boundary factor is ignored.
Calculation	What is calculated (optional , Default value: "t")
	"t" : Pulse duration, stated in the x-unit of the input data
	"n" : Pulse width, expressed as number of measurement points; may include decimal places
	"f" : Pulse frequency, calculated as reciprocal of the average pulse duration
	"rpm" : Rotation speed, calculated as the reciprocal of the average pulse duration, expressed in RPM units for an encoder with one pulse per revolution. The x-unit of the input data must be s.
Boundary factor	At the start and end of the signal, the signal is often located in the middle of a pulse. If the last adjacent pulse duration, multiplied by this factor, is greater than or equal to the boundary area witout any signal edge, then the last pulse duration is retained. Else, zero is returned. (optional , Default value: 1.5)
Result	
Result	Result

Description:

This function operates inside of a window of fixed width and generates one result value at the end of the window. The average pulse duration is formed from all impulses within the window. If any pulses are not contained completely within the window, they are counted in proportion to the fraction of time they are inside it. The duration of all pulses whose front or hind signal edge lies within the interval is subjected to weighted averaging, where the pulse's duration within the window is used as the weighting factor. Both pulses occurring in times before and after are thus taken into account.

The result values are the respective window start points. If the result is displayed as stair steps, then each step denotes the averaging interval.

If the parameter "slope" is set to "imc Inline FAMOS", then a different algorithm is used; see explanations above.

In typical applications, a boundary factor lies significantly > 1.0. With a continual sequence of pulses, it prevents the result from containing any jumps at the boundary.

With boundary factor = 0.0, extensions at the boundary are suppressed. If there are no pulses at the boundary within a window width, the result at these locations is zero.

The input data are equidistant and may contain events and segments.

Examples:

Period duration from unformed pulse signal

```
pulses = ( stri ( signal, 3, 2 ) > 0 )
duration = pulseDuration ( pulses, 0.1)
```

Rotation speed from pulse sequence

```
rpm = pulseDuration ( pulses, 0.1, "0-1", "rpm")
```

Pulse frequency from sinusoidal signal

```
f = pulseDuration ( sine, 0.1, "+", "f")
```

See also:

OtrTachoToSpeed, Peaks

PyCallFunction

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

A Python-function is executed.

Declaration:

```
PyCallFunction ( TxModuleName, TxFunctionName [, Arg1] [, Arg2] [, Arg3] [, Arg4] [, Arg5] [, Arg6] [, Arg7] [,
Arg8] [, Arg9] [, Arg10] [, Arg11] [, Arg12] [, Arg13] ) -> Result
```

Parameter:

TxModuleName	Name of the Python-module in which the function to be executed is defined.
TxFunctionName	Name of the function to be executed
Arg1	Argument #1 (optional) (optional)
Arg2	Argument #2 (optional) (optional)
Arg3	Argument #3 (optional) (optional)
Arg4	Argument #4 (optional) (optional)
Arg5	Argument #5 (optional) (optional)
Arg6	Argument #6 (optional) (optional)
Arg7	Argument #7 (optional) (optional)
Arg8	Argument #8 (optional) (optional)
Arg9	Argument #9 (optional) (optional)
Arg10	Argument #10 (optional) (optional)
Arg11	Argument #11 (optional) (optional)
Arg12	Argument #12 (optional) (optional)
Arg13	Argument #13 (optional) (optional)
Result	
Result	Result returned by the function

Description:

Initially the system checks whether a module of the specified name has already been imported into the Interpreter. If not, import is performed.

- With sub-modules of a package, the usual "period-notation" must be used, so for instance "PackageName.subModuleName".
- For permanently "built-in functions" in Python, an empty text or "builtins" must be specified.
- If the module specified is not located in Python's default search path, the function PyConfig("Sys.Path.Append", ...) must first be called.

Subsequently, the system searches for the function having the name specified and if it is found, the parameters passed are converted to Python-variables and the function is applied to these.

- For the parameter conversion, the same rules apply as described for the function PySetVar().

If the function provides a return value, the result is converted to a FAMOS variable.

- For the re-conversion, the same rules apply as described for the function PyGetVar().

Remarks:

- When a Python function can't be called directly, for example because the parameter types are not convertible from/to FAMOS, a small Python script can provide a remedy by encapsulating the desired function. In that case, the parameters are first transferred using PySetVar(), then the script is called either with PyRunFile() or repeated PyRun()-commands, and subsequently the results are retrieved using PyGetVar(). Then the script takes on the task of conditioning the parameters, calls the function, and converts the results to a form which can be retrieved by PyGetVar().
- If the Python runtime environment is not yet loaded, it is first initialized by an implicit call of the function PyInit().

- At fault condition (e.g. module or function unknown, incorrect parameter types), a runtime error is raised.

Examples:

The built-in Python-function 'divmod' returns the quotient and remainder of the integer division of the two arguments as a tuple of 2 elements. The resulting tuple is converted to a FAMOS data set of length 2.

```
result = PyCallFunction("", "divmod", 13, 3)
; identical to PyCallFunction("builtins", "divmod", 13, 3)
quotient_= result[1]  ; => 4
remainder= result[2]  ; => 1
```

The preceding code is functionally equivalent to:

```
PySetVar("", "a", 13)
PySetVar("", "b", 3)
PyRun("result = divmod(a,b)")
result = PyGetVar("", "result")
quotient_= result[1]
remainder= result[2]
```

Calls functions from external libraries, here illustrated by an example with 'NumPy'. The following example uses the FFT-function 'NumPy' and compares the result with the FFT()-function built into FAMOS.

```
; Create test signal
n = 1024
signal = sin(Ramp(0,0.01,n))

; Apply FFT from 'NumPy'
PyConfig("Converter.PyArrayType", "ndarray")
res = PyCallFunction("numpy.fft", "rfft", signal )
; Convert Cartesian to Polar coordinates
res = Pol(res)
; Undo normalization
res.b = res.b/n
; Calculate frequency line distance
freq = PyCallFunction("numpy.fft", "rfftfreq", ToInt(n))
res = xdel(res, (freq[2]-freq[1])/xdel?(signal))

; Compare with FAMOS FFFT
FFTOption 0 0
resFAMOS = FFT(signal)
Verify(Equal(resFAMOS, res,  1e-10, "absolute", 1e-10))
```

Calculation of the lower and upper quartiles as well as of the median using 'NumPy':

```
signal = ...
PyConfig("Converter.PyArrayType", "ndarray")
percs = PyCallFunction("numpy", "percentile", signal, [25, 50, 75])
lower_q = percs[1]
median = percs[2]
upper_q = percs[3]
```

If the interpolation type is additionally to be set for the "percentile"-function (by means of the specified optional parameter 'interpolation'), direct calling with PyCallFunction() is no longer possible, but must instead be emulated using PySetVar()/PyRun()/PyGetVar():

```
...
PySetVar("", "signal", signal)
PyRun("import numpy")
PyRun("percs = numpy.percentile(signal, [25, 50, 75], interpolation='nearest')")
percs = PyGetVar("", "percs")
...
```

See also:

PyGetVar, PySetVar, PyRun, PyRunFile, PyConfig

PyConfig

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

Control of the behavior of the embedded Python-Interpreter

Declaration:

```
PyConfig ( TxOption, TxValue ) -> Success
```

Parameter:

TxOption	Name of the option to set
	"Converter.PyArrayType" : Specifies into which Python data type a FAMOS data set is converted when transferred to Python.
	"Sys.Path.Append" : Adds an additional folder to the module search path of the Python-Interpreter.
TxValue	Value of the option to set
	"Converter.PyArrayType" : Array data type in Python
	"list": [Default] Python default data type 'list'. **"ndarray"**: Data type 'numpy.ndarray' (defined in the 'NumPy' expansion library). If the 'NumPy'-library is not present, the function returns a 0.
Success	
Success	1, if the function has been executed successfully; 0 at fault condition. In case of fault, the cause can be queried using the function GetLastError().

Description:

Python-data type for FAMOS data sets ("Converter.PyArrayType")

This option specifies which Python data type is to be used when transferring FAMOS data sets to Python. This pertains to the functions PySetVar() and PyCallFunction(). By default, Python's default data type 'list' is used. However, this is not effective for storage and processing of large data volumes.

Alternatively, the data type 'numpy.ndarray' can be used, which is defined in the widely prevalent 'NumPy' expansion library. This type is optimized for efficient data storage and high-speed processing of multi-dimensional arrays. This data type can only be set if the 'NumPy'-package is also installed, otherwise the function returns an error.

An exact listing of how the various FAMOS data types are implemented in Python is presented in the description of the function PySetVar().

Module search path ("Sys.Path.Append")

The module search path contains a list of folders in which the Python-Interpreter searches for modules to import. The module search path is recorded in the Python-variable "sys.path". After starting the Interpreter, this variable contains the default paths specified by the Python-installation as well as the current working directory.

The call

```
   PyConfig("Sys.Path.Append", "c:\my\folder")
```

is equivalent to the following Python-code:

```
   >>> import sys
   >>> sys.path.append('c:\\my\\folder')
```

Remarks:

- This function is required when Python-modules are to be used subsequently which are not located in Python's default folders.
- If the Python runtime environment is not yet loaded, it is first initialized by an implicit call of the function PyInit().

Multithreading: All functions of the Python-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are initially moved internally to the FAMOS main thread and executed from there, since the Python Runtime Environment does not support parallel calls from different threads.

Examples:

An analysis uses the supplemental library 'NumPy'. The default array type is accordingly set to 'ndarray' at the beginning.

```
IF 0 = PyConfig("Converter.PyArrayType", "ndarray")
   BoxMessage("Error", "NumPy-Package not found. Please install first (see https://numpy.org)."
   EXITSEQUENCE
```

```
END
signal = ...
; Apply FFT from 'NumPy'
res = PyCallFunction("numpy.fft", "rfft", signal )
```

A FAMOS-project contains a user-written Python-module named "myfunctions.py", from which the function "sum3" is to be called, which finds the sum of 3 parameters. Before the module is used, the FAMOS project folder must be added to the search path.

```
projectPath = GetOption("Dir.CurrentProject")
PyConfig("Sys.Path.Append", projectPath)
sum = PyCallFunction("myfunctions", "sum3", 1, 2.2, 3)
; sum = 6.2
```

See also:

PyInit, PySetVar, PyRun, PyCallFunction

PyGetVar

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

Queries the content of a Python variable.

Declaration:

```
PyGetVar ( TxModuleName, TxVariableName ) -> Content
```

Parameter:

TxModuleName	Name of the Python-module in whose context the variable exists.
TxVariableName	Name of the variable to query
Content	
Content	Content of the queried variable

Description:

Retrieves the content of the Python variable.

The system initially tests whether the specified module has already been imported into the Interpreter. If not, import is performed. In case of an empty name, the main module ("__main__") is used. For sub-modules of a package, the usual "period-notation" must be used, thus for example "PackageName.SubModuleName".

Subsequently, the system finds the variable with the name specified and if found, retrieves the variable's content.

Example:

Querying a global variable named 'test':

```
PyRun("test = 2.3")
test = PyGetVar("", "test")
; identical to: PyGetVar("__main__", "test")
```

If it is not a global variable on the level of the module, but an attribute of a class instance, then it must be specified in Python using the usual "period-notation", thus for example "ClassInstanceName.AttributeName"

Example:

Supposing the Python module 'persons' contains a definition of the class 'person', which contains the attributes 'name' and 'age' among others. The following code generates an instance of the class 'person' and later retrieves the attributes:

```
PyRun("import persons")
PyRun("harry = persons.person()")
...
TxName = PyGetVar("", "harry.name")
Age = PyGetVar("", "harry.age")
```

Remarks:

- If the Python runtime environment is not yet loaded, it is first initialized by an implicit call of the function PyInit().
- At fault condition (e.g. unknown module, unknown variable name or data type not supported), a runtime error is raised.

Data types

The data type of the FAMOS-variable generated is determined automatically from the Python-variable's data type.

Here, the following rules apply:

Python	FAMOS
Standard-Typen	
'NoneType'	Data set of length 0; data format: 8-Byte Real (like constant 'EMPTY')
'int'	Single value; data format: 8-Byte signed Integer
'float'	Single value; data format: 8-Byte Real
'complex'	Complex data set (RI) of length 1; data format: 8-Byte Real
'bool'	Single value; data format: 8-Byte Real

'str'	Text
Container types 'tuple', 'list', 'range', 'set', 'frozenset'	
with element number 0	Data set of length 0; data format: 8-Byte Real
all elements of type 'float'	Data set; data format: 8-Byte Real
all elements of type 'int'	Data set; data format: 8-Byte signed Integer
all elements of the type 'boole'	Data set; data format: 8-Byte Real
all elements of the type 'complex'	Complex data set (RI) of length 1; data format: 8-Byte Real
all elements of the type 'str'	Text array
all elements of a 1-dimensional container, where all containers have the same length N and all elements the same numerical type (float, int, complex)	Segmented data set; segment length: N
other element-types or various types	Data group. The channels of the group contain the name "Item1", "Item2"...; the content conforms to the conversion rules defined here. Caution when using 'set' - the elements have no defined order; the order of the channels in the group is then random.
Miscellaneous container-types	
'dictionary'	Data group. From each key/value-pair, a channel is constructed. The channel's name is constructed from the key name, the content of the channel is generated from the respective value according to the conversion rules defined here.
'bytes', 'bytearray'	Data set; 1-Byte unsigned Integer

NumPy-Arrays ('numpy.ndarray')

This data type is defined in the widely prevalent 'NumPy'-package. It is optimized for efficient data storage and high-speed processing of multi-dimensional arrays.

Homogeneous arrays of Dimension 1 or 2 are supported. For Dimension 2, a segmented data set is generated, where the rows of the NumPy-array form the segments.

The storage layout must be either "C-" or "F-contiguous" (C_CONTIGUOUS, F_CONTIGUOUS) and the data must be saved in "Little-Endian-Byte order".

NumPy-Array-element type	FAMOS
'float64', 'float32'	Data set (8 Byte Real)
'float128', 'float16'	not supported
'int8', 'int16', 'int32'	Data set (8 Byte Real)
'int64'	Data set (8-Byte signed Integer)
'uint8'	Data set (1-Byte unsigned Integer)
'uint16', 'uint32'	Data set (8 Byte Real)
'uint64'	not supported
'bool'	Data set (8 Byte Real)
'complex64', 'complex128'	Complex data set (RI, 8 Byte Real)
'str'	Text array
'datetime64', 'timedelta64'	not supported
'bytes' and all other types	not supported

Examples:

Gets the Python-version by means of the attribute "hexversion" of the "sys"-module.

```
version = PyGetVar("sys", "hexversion") ; datatype of the result is "signed 8 byte integer"
major = BitShift(version, -24, 64) ; major is always 3
minor = BitAnd(BitShift(version, -16, 32), 0xFF, 64); minor is e.g. 8 or 9
```

Finds the Python-module search path. The result is of the type Text Array; each element is a folder.

```
pythonPath = PyGetVar("sys", "path")
```

Calculation of the lower and upper quartiles as well as of the median using 'NumPy':

```
PyConfig("Converter.PyArrayType", "ndarray")
signal = ...
PySetVar("", "signal", signal)
PyRun("import numpy")
PyRun("percs = numpy.percentile(signal, [25, 50, 75], interpolation='nearest')")
percs = PyGetVar("", "percs")
lower_quantil = percs[1]
median = percs[2]
upper_quantil = percs[3]
```

A Python-dictionary is converted to a FAMOS data group:

```
PyRun("person = {'name' : 'John', 'age' : 36}")
person = PyGetVar("", "person")
```

The result is a data group with 2 elements:

```
person
   |__  name   ("John")
   |__  age   (36)
```

From a random sample of data, a 'NormalDist'-class is constructed from the 'statistics'-module, which represents a standard distribution of the sample with a mean value and standard deviation. The system finds the mean value and the standard deviation; additionally a long set of random data having the same distribution as the random sample is generated:

```
PyRun("import statistics")
PySetVar("", "probe_samples" , [1, 1.2, 7.2, 15, 4.2, 6.1])
PyRun("nd = statistics.NormalDist.from_samples(probe_samples)")
mean_ = PyGetVar("","nd.mean")
stdev_ = PyGetVar("","nd.stdev")
PyRun("samples = nd.samples(10000, seed = 1)")
samples = PyGetVar("","samples")
```

See also:

PySetVar, PyRun, PyRunFile, PyCallFunction

PyInit

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

The Python runtime environment is loaded in FAMOS and the Interpreter is initialized.

Declaration:

`PyInit ( [TxPythonFolder] ) -> Success`

Parameter:

TxPythonFolder	Home directory of the Python-installation to be used. Optionally, if it is empty or not specified, the Python folder is determined automatically. (optional , Default value: "")
Success	
Success	1, if the function has been executed successfully; 0 at fault condition. In case of fault, the cause can be queried using the function GetLastError().

Description:

It is not absolutely necessary to explicitly call this function. All FAMOS-functions which use the Python-Interpreter will perform it automatically if initialization has not occurred yet.

If the Python-Interpreter is already initialized at the time when it is called, the function does not do anything.

Explicitly calling this function can be useful in the following application situations:

- Specification of the home directory of the Python-installation to be used: Normally this parameter can be omitted, FAMOS then automatically uses the highest supported version which can be found on the computer. Explicit specification is necessary when multiple Python installations are on the computer and a specific version is to be used. The home directory of a Python-installation is the target folder selected upon setup, and contains, for example, the executable file 'python.exe'.
- The Python home directory is not entered in the PATH environment variable.FAMOS cannot then find Python automatically and the path must be specified here explicitly.
- Test for a compatible Python-installation before beginning the actual analysis, in order to be able to perform proper error handling at fault condition (e.g. a Python-version which is not supported). For instance this can be a message to the user requesting a check of the Python-installation, followed by controlled closing of the analysis. Otherwise, such a fundamental problem would only become noticeable during execution of the actual analysis and cause a runtime error, meaning an uncontrolled cancelling of the sequence.

Multithreading: All functions of the Python-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are initially moved internally to the FAMOS main thread and executed from there, since the Python Runtime Environment does not support parallel calls from different threads.

Examples:

At the beginning of an analysis, for which Python 3.9 is compulsory, the availability of an appropriate Python-installation is verified:

```
IF 0 = PyInit()
   BoxMessage("Can't start Python!", GetLastError(), "!1")
   EXITSEQUENCE
END
version = PyGetVar("sys", "hexversion")
; "sys.hexversion" returns the Python-version encoded in hexadecimal format
major = BitShift(version, -24, 64)
minor = BitAnd(BitShift(version, -16, 32), 0xFF, 64)
IF major <> 3 OR minor <> 9
   BoxMessage("Wrong Python version!", "Sorry, Python 3.9 requested!", "!1")
   PyTerminate()
   EXITSEQUENCE
END
...
```

On a PC having multiple installations of Python, the desired version is selected by specifying the home directory:

```
IF 0 = PyInit("c:\python\version_3_8")
   BoxMessage("No Python installation found!", GetLastError(), "!1")
   EXITSEQUENCE
END
...
```

See also:

PyTerminate, PyConfig, PySetVar, PyRun

PyRun

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

The Python-code specified is executed.

Declaration:

```
PyRun ( TxCode ) -> Success
```

Parameter:

TxCode	The Python-code to run. Allowed types: Text, Text Array
Success	
Success	1, if the function has been executed successfully; 0 at fault condition. In case of fault, the cause can be queried using the function GetLastError().

Description:

The specified Python-script-code is passed to the Python-Interpreter and executed.

It is also possible to pass multiple code lines in a text variable, these must be separated by a LineFeed-character (ASCII-Code 10, "~010").

When a text array is is transferred, each element is interpreted as a code line.

Remarks:

- For the purpose of executing complex scripts, the function PyRunFile() is usually more appropriate.
- If the Python runtime environment is not yet loaded, it is first initialized by an implicit call of the function PyInit().

Examples:

The Python-function 'sqrt', which belongs to the 'math'-module and serves to calculate the square root, is called. The following 3 sample calls are equivalent:

```
PyRun("import math")
PyRun("x = math.sqrt(2)")
```

```
PyRun("import math~010x = math.sqrt(2)")
```

```
codeLines = ["import math", "x = math.sqrt(2)"]
PyRun(codeLines)
```

The result of the calculation is subsequently retrieved:

```
x = PyGetVar("", "x")
```

Alternative calculation:

```
x = PyCallFunction("math", "sqrt", 2)
```

The default Python output is redirected to the FAMOS output window. The print()-function can thus be used to display texts there:

```
PyRun("person = {'name': 'Harry', 'age': 42}")
PyRun("print('My name is',person['name'],', age ', person['age'])")
; => FAMOS-output window: "My name is  Harry , age  42"
```

At the beginning of an analysis in Python, which uses functions from a personally-made library 'myfunctions.py', the system checks whether the module is even present on the current PC:

```
IF PyRun("import myfunctions") = 0
   BoxMessage("Error", "Please install the Python module 'myfunctions' first!", "!1")
   EXITSEQUENCE
END
```

From a random sample of data, a 'NormalDist'-class is constructed from the 'statistics'-module, which represents a standard distribution of the sample with a mean value and standard deviation. The system finds the mean value and the standard deviation; additionally a long set of random data having the same distribution as the random sample is generated:

```
PyRun("import statistics")
PySetVar("", "probe_samples" , [1, 1.2, 7.2, 15, 4.2, 6.1])
PyRun("nd = statistics.NormalDist.from_samples(probe_samples)")
mean_ = PyGetVar("","nd.mean")
stdev_ = PyGetVar("","nd.stdev")
```

```
PyRun("samples = nd.samples(10000, seed = 1)")
samples = PyGetVar("","samples")
```

See also:

PyRunFile, PyCallFunction, PyGetVar, PySetVar

PyRunFile

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

The specified Python-script-file is executed.

Declaration:

```
PyRunFile ( TxFileName [, TxParameter] ) -> Success
```

Parameter:

TxFileName	Complete pathname of the Python-script to be executed
TxParameter	Optional list of the parameters to be passed (optional)
Success	
Success	1, if the function has been executed successfully; 0 at fault condition. In case of fault, the cause can be queried using the function GetLastError().

Description:

The specified Python-script-file is passed to the Python-Interpreter and executed.

The (optional) list of parameters must be specified in the same syntax as if the script were directly passed to the Python-Interpreter ('python.exe'). The individual parameters are thus separated by spaces, parameters names containing spaces must be bracketed in extra quotation marks.

The call in the Windows command prompt

```
   python c:\py_scripts\myscript.py 2 "Hello world"
```

would thus be identical to

```
   PyRunFile("c:\py_scripts\myscript.py", "2 ""Hello world""")
```

Important difference between the two calls: The first call uses a new, independent Python-instance. In the second call, the script is run in the context of the Python-instance administered by FAMOS; any already imported modules or generated global variables are thus available to the script. Conversely, any global variables generated by the script remain intact after execution.

Remarks:

- If the Python runtime environment is not yet loaded, it is first initialized by an implicit call of the function PyInit().
- The script-file must be encoded in the character set UTF-8. If only ASCII-characters within the range up to 127 are used, this matches the standard-ANSI character set.

Examples:

The Python-script file 'c:\py_scripts\Add2Numbers.py" adds 2 numbers passed as the parameters and has the following content:

```
import sys
sum = float(sys.argv[1]) + float(sys.argv[2])
```

Call in FAMOS:

```
PyRunFile("c:\py_scripts\Add2Numbers.py","1.2 3.2")
sum = PyGetVar("","sum") ; => sum = 4.4
```

The script file 'untabify.py' which is included in the standard Python installation processes the file(s) passed as parameters and replaces tab characters with a specifiable number of space characters.

```
; replace each tab in "file1.txt" with 3 spaces:
PyRunFile("C:\Python\Tools\scripts\untabify.py", "-t 3 ""c:\my files\file1.txt""")
```

See also:

PyRun, PyCallFunction, PyGetVar, PySetVar

PySetVar

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

Sets the content of a Python-variable.

Declaration:

```
PySetVar ( TxModuleName, TxVariableName, Content )
```

Parameter:

TxModuleName	Name of the Python-module in whose context the variable is to be set.
TxVariableName	Name of the variable to be set
Content	New content

Description:

The specified content is assigned to the Python-variable addressed.

The system initially tests whether the specified module has already been imported into the Interpreter. If not, import is performed. In case of an empty name, the main module ("__main__") is used. For sub-modules of a package, the usual "period-notation" must be used, thus for example "PackageName.SubModuleName".

Subsequently, the system searches for the variable having the name specified. If it already exists, it is overwritten; otherwise it is newly created.

Example: Generates a global variable having the name 'test' and the data type 'float'.

```
PySetVar("", "test", 2.3)
; identical to: PySetVar("__main__", "test", 2.3)
```

If it is not a global variable on the level of the module, but an attribute of a class instance, then it must be specified in Python using the usual "period-notation", thus for example "ClassInstanceName.AttributeName"

Example: Supposing the Python-module 'persons' contains a definition of the class 'person', which contains the attributes 'name' and 'age' among others. The following code generates an instance of the class 'person' and sets the attributes:

```
PyRun("import persons")
PyRun("harry = persons.person()")
PySetVar("", "harry.name", "harry")
PySetVar("", "harry.age", 18)
```

Remarks:

- If the Python runtime environment is not yet loaded, it is first initialized by an implicit call of the function PyInit().
- At fault condition (e.g. unknown module, invalid variable name or data type), a runtime error is raised.
- Multithreading: All functions of the Python-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are initially moved internally to the FAMOS main thread and executed from there, since the Python Runtime Environment does not support parallel calls from different threads.

Data types

The Python data type of the variable generated is automatically determined from the type and format of the 3rd parameter.

The default converter type configured with PyConfig("Converter.PyArrayType", ...) is applied (either 'list' (default) [1] or 'numpy.nddarray' [2]).

Here, the following rules apply:

FAMOS-data type	**Python-data type**
Single values	
8-Byte Integer (full-scale)	Integer ('int')
Miscellaneous numerical data formats	Real number, 8-Byte ('float')
Normal data sets	
8-Byte Real with length = 0 (constant 'EMPTY')	None ('NoneType')
8-Byte Integer (full-scale)	[1]: Liste ('list' [..'int'..]) [2]: NumPy-Array ('numpy.ndarray' [..'int64'..])

1-Byte Integer unsigned (full-scale)	Bytearray
Miscellaneous numerical data formats	[1]: Liste ('list' [..'float'..]) [2]: NumPy-Array ('numpy.ndarray' [..'float64'..])
TimeStampASCII and other special formats	not supported
2-component data sets	
Complex, Cartesian, Length = 1	Complex number ('complex')
Complex, Cartesian, Length <> 1	[1]: Liste kompl. Zahlen ('list' [..'complex'..]) [2]: NumPy-Array ('numpy.ndarray' [..'complex128'..])
Complex, Magnitude/Phase	not supported
XY	not supported
Structured data (Events/Segments)	
Segmente ... Integer 1 Byte unsigned (full-scale)	[1]: Liste von Bytefeldern ('list' [..'bytearray'..]). Jedes Bytefeld entspricht einem Segment. [2]: 2-dimensionales NumPy-Array von Bytes ('numpy.ndarray' [..'uint8'..]). Jedes Segment bildet eine Zeile der Matrix.
Segmente ... sonstige Formate	[1]: Liste von Listen ('list' [..'list'..]). Jedes Listenelement entspricht einem Segment. [2]: 2-dimensionales NumPy-Array ('numpy.ndarray'). Jedes Segment bildet eine Zeile der Matrix. Der Datentyp der Listen-/Feldelemente ergibt sich wie vorstehend beschrieben.
Events	not supported
Miscellaneous data types	
Text	String ('str')
Text array	List of strings ('list' [..'str'..])
Group	Dictionary ('dict'). Each channel of the group is translated to a key/value-pair. The key corresponds to the respective channel name.

Notes on the type 'numpy.ndarray'

This data type is defined in the very prevalent 'NumPy'-package. It is optimized for efficient data storage and high-speed processing of multi-dimensional arrays. The NumPy-arrays generated are saved in "C-contiguous" (C_CONTIGUOUS) style.

Examples:

Generating global variables of the data types 'float' and 'int':

```
PySetVar("", "test_float", 2.3)
PySetVar("", "test_int", ToInt(2))
```

Generating global variables of the data types 'list' and 'numpy.ndarray' (data type of the elements: 'float'):

```
PySetVar("", "test_list", [1,2,6,7])
PyConfig("Converter.PyArrayType", "ndarray")
PySetVar("", "test_ndarray", [1,2,6,7])
```

Generating a Python-list with texts:

```
PySetVar("", "test_str_list", ["The", "answer", "is", "42"])
```

Generating a Python-dictionary ('dict') from a FAMOS-data group:

```
person:name = "Harry"
person:age = ToInt(42)
PySetVar("", "test_dict", person)
PyRun("print(test_dict)")
; => FAMOS-output window "test_dict = {'name': 'Harry', 'age': 42}"
```

See also:

PyGetVar, PyRun, PyRunFile, PyConfig, PyCallFunction

PyTerminate

Scope: Python remote control

Available in: Professional Edition and above (Python-Kit)

Shutdown the Python-Interpreter

Declaration:

`PyTerminate ( )`

Parameter:

Description:

The Python-Interpreter previously initialized by PyInit() or another Python-accessing function is closed.

An explicit call of this function is not compulsory. Termination of the Python-Interpreter occurs automatically upon closing FAMOS.

One possible application is provisional termination of the Interpreter, for instance in order to subsequently use PyInit() to start a new, "clean" instance of the Interpreter. Although this is permitted in principle, it can be risky and is therefore not recommended. Python-expansion modules from 3rd-party manufacturers can have problems after such an "Interpreter-reboot", since some architectures may not enable them to be unloaded properly. One example is the library 'NumPy', which may no longer work properly following PyTerminate()/PyInit().

If the Python-Interpreter is not initialzed at the moment of the call, the function will not do anything.

Examples:

At the beginning of an analysis for which Python 3.9 is compulsory, the availability of an appropriate Python-installation is verified. If a different version is found, the Python runtime environment is immediately closed:

```
IF 0 = PyInit()
   BoxMessage("Can't start Python!", GetLastError(), "!1")
   EXITSEQUENCE
END
version = PyGetVar("sys", "hexversion")
; "sys.hexversion" returns the Python-version encoded in hexadecimal format
major = BitShift(version, -24, 64)
minor = BitAnd(BitShift(version, -16, 32), 0xFF, 64)
IF major <> 3 OR minor <> 9
   BoxMessage("Wrong Python version!", "Sorry, Python 3.9 requested!", "!1")
   PyTerminate()
   EXITSEQUENCE
END
...
```

See also:

PyInit

R_ChisqTest

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

Function for performing the Chi-square-test

Declaration:

```
R_ChisqTest ( x [, correct] [, p] ) -> Result
```

Parameter:

x	Data with the properties of the random sample
correct	Continuity correction (optional , Default value: 1)
	0 : Continuity correction is not applied.
	1 : Continuity correction is applied. (default)
p	Normal data set with probabilities (only applicable to goodness-of-fit test) (optional)
Result	
Result	The result of the Chi-square test isa data group with the elements:
	statistic : Chi-square value
	parameter : Count of degrees of freedom
	p.value : p-value of the test
	method : The character sequence indicates which type of the Chi-square test was performed.
	observed : Observed frequencies
	expected : The expected frequencies according to the null hypothesis
	residuals : Pearson residues
	stdres : Standard residues

Description:

The Chi-square test belongs to the group of hypothesis tests based on a Chi-square distribution. The core of the test concept consists of a comparison of an empirical frequency with a theoretical frequency. It gives a statement on whether the observed frequencies differ significantly from those which would be expected.

Test prerequisites:

The random sample should comprise at least 50 examination units.
If the sample size is between 20 and 50, then the continuity correction should be applied.
The expected frequencies in all cells of the contingency table should be greater than 5 .
The degrees of freedom = (r-1) * (c-1) should be greater than 1 (r=row count and c=column count of the contingency table). If this condition is not met, then the continuity correction should be applied.

There is a distinction between the following tests:

Goodness-of-fit test

The goodness-of-fit test examines whether an empirically observed frequency distribution can be described by a specific theoretical distribution. It tests whether the data conform to a specific probability distribution. Thus for example, it is possible to examine whether the face values of a dice roll are uniformly distributed or whether a concrete empirical characteristic is normally distributed in the population.
If the parameter **'x' is a normal data set**, the goodness-of-fit test is performed. In this case 'x' is treated as a single-dimensional contingency table. The entries belonging to 'x' are the observed frequencies. They must be positive numbers. These relative class frequencies are then compared with the specified probabilities.
The goodness-of-fit test always tests against the null hypothesis.
The null hypothesis states that the attributes of 'x' conform to the specified probability distribution 'p'.
The probabilities can be specified as the parameter 'p'. The value count in the data set 'p' must match that of 'x'. If 'p' is not specified, then uniform distribution is assumed.
The calculated p-value 'p.value' is the probability of observing the calculated test statistic according to the null hypothesis. The smaller the p-value, the less probable is the validity of the null hypothesis. It is customary to reject the null hypothesis at a p-value <= 0.05 (or 5%).

Test of independence

In the test of independence, 2 discrete characteristics a and b belonging to a sample are examined for stochastic independence. The issue of interest is whether the two characteristics are interrelated to or independent of each other.
The hypotheses of the test are:
The null hypothesis states that the characteristics a and b are independent.
The alternative hypothesis states that the characteristics a and b are not independent.

In order to perform the test of independence, the parameter **'x' must be a normal segmented data set**. This is converted to a matrix and transferred into a contingency table. On the basis of this contingency table, the test of independence is performed. In the contingency table, the degrees of the first characteristic form the columns and the degrees of the second characteristic form the rows. The cells contain the frequencies of both characteristics, meaning the count of occurrences, which apply to both the criterion for the row and also that of the column.
Example: In the contigency table, the purchasing behavior of customers is reflected.
Characteristic a= gender; characteristic b = purchasing behavior; sample size = 1000 customers
The frequencies for the purchasing behavior is subdivided by gender and "buy"/"don't buy" are transferred in the form of a matrix (segmented data sets). The function produces a contingency table from the matrix by adding the boundary sums.

	Matrix				Contingency table		
Characteristic a / b	buy	don't buy		Characteristic a / b	buy	don't buy	Sum
Men	180	170	=>	Men	180	170	350
Women	240	410		Women	240	410	650
				Sum	420	580	1000

The continuity correction should be set when the calculation is applied to a contingency table with 2 columns and 2 rows.

When using the calculated p-value, a test is decided as follows:
p.value < alpha -> The null hypothesis is rejected. The alternative hypothesis is accepted.
p.value >= alpha -> The null hypothesis accepted.
Where alpha the previously selected significance level; 0.05 (5%) is customary.

Continuity correction

The continuity correction is applied when there is only one degree of freedom or the sample size is between 20 and 50. The so-called Yates' correction improves the comparability of test variable with the theoretical distribution for smaller samples sizes with only one degree of freedom.

Results of the Chi-square-test

statistic	Chi-square - Value = sum((obsserved frequencies - expected frequencies)^2 /expected frequencies)
parameter	Number of degrees of freedom:= (row count -1) * (column count -1) of the contingency table
p.value	The p-Wert is a measure of probability for evidence against accepting the null hypothesis. Lower probabilities provide stronger eveidence that the null hypothesis is not accepted. If the p-value <= alpha (significance level), then the null hypothesis is rejected. If the p-value > alpha, the null hypothesis is accepted.
method	The type Chi-square test conducted
observed	Compilation of observed data counts (frequencies)
expected	The expected frequencies according to the null hypothesis Expected frequency[i,j]= (column sum[i] * row sum[j])/ total count
residuals	Pearson residues Residue[i,j]= (observed frequency[i,j] - expected frequency[i,j]) / sqrt(expected frequency[i,j])
stdres	Standard residues

Examples:

In the **goodness-of-fit test**, for a significance level of alpha = 1%, we test whether the sample provides evidence for or against uniform distribution of dice roll outcomes. The die investigated was rolled 500 times with the following frequency distribution
for face values of 1, 2, 3, 4, 5 and 6:
absolute frequency 75, 78, 96, 103, 77, 71
Null hypothese for i=1,...,6 is pi = P({face value =i}) = 1/6
Alternative hypothesis for i=1,...,6 ist pi = P({face value =i}) <> 1/6

```
ni=[75,78,96,103,77,71]
result=R_ChisqTest(ni)
The Chi-square test returns the result as the data group 'result':
```

	statistic	=	10.048
	parameter	=	5
	p.value	=	0.0738866
	method	=	"Chi-squared test for given probabilities"
	observed	=	75,78,96,103,77,71
	expected	=	83.33,83.33,83.33,83.33,83.33,83.33
	residuals	=	-0.9129, -0.5842, 1.3876, 2.1544, -0.6938, -1.3510
	stdres	=	-1.0000, -0.6400, 1.5200, 2.3600, -0.7600, -1.4800

We obtain a Chi-square value = 10.048 with 5 degrees of freedom.
The p-value of the test is 0.0738866. It is substantially above the significance level of 0.01, so that one can assume that the distribution is uniform and the dice is legit.

Can one conclude based on the numbers obtained that the merchant's sales distribution deviates significantly from the overal market? The null hypothesis states that the merchant's sales distribution matches the distribution given for the overall population. In this **goodness of fit test**, the distribution of the population is not uniformly distributed, so that the distribution must be supplied as the parameter 'p'.

```
x=[33400,35410,2610,12520,16340,9840,6070,7620,1190]; sales figures of the merchant, by brand
p=[27,28,2,10,13,8,5,6,1]; market share in percent
result=R_ChisqTest( x,0,p)
The Chi-square test returns the result as the data group 'result':
```

	statistic	=	26.3469
	parameter	=	8
	p.value	=	0.000915901
	method	=	"Chi-squared test for given probabilities"
	observed	=	33400, 35410, 2610, 12520, 16340, 9840, 6070, 7620, 1190
	expected	=	33750, 35000, 2500, 12500, 16250, 10000, 6250, 7500, 1250
	residuals	=	-1.9052, 2.1915, 2.2000, 0.1789, 0.7060, -1.6000, -2.2768, 1.3856, -1.6971
	stdres	=	-2.2298, 2.5828, 2.2223, 0.1886, 0.7569, -1.6681, -2.3360, 1.4292, -1.7056

We obtain a Chi-square value = 26.3469 with 8 degrees of freedom.
At 0.0009, the p-value is below the significance level of 0.01. The merchant can thus assume that his sales profile is different from that of the overall market.

With the **test of independence**, it is to be examined whether there is any correlation between eye color and hair color. The null hypothesis states: Hair and eye color are mutually independent. The eye and hair color of 592 people was observed.

Hair/Eyecolor	blue	brown	green	nut
blonde	94	7	16	10
brown	84	119	29	54
red	17	26	14	14
black	20	68	5	15

From the observed frequencies, a segmented data set is compiled. The segments form the columns.

```
blue=[94,84,17,20]
brown=[7,119,26,68]
green=[16,29,14,5]
hazel=[10,54,14,15]
x=join(blue,brown)
x=join(x,green)
x=join(x,hazel)
SetSegLen(x,leng?(blue))
result=R_ChisqTest(x)
```

	statistic	=	138.29
	parameter	=	9
	p.value	=	2.32529e-25
	method	=	"Pearson's Chi-squared test"

We obtain a Chi-square value = 138.29 with 9 degrees of freedom.
Since the p-value is < 0.05, the null hypothesis can be rejected. There is a correlation between eye color and hair color.

Using the **test of independence**, we examine whether the proportions of men and women in the labor force is different. The null hypothesis states that the proportion of labor force participation is independent of gender. The survey encompasses 3468 persons.

.	male	female
full-time	1026	545
part-time	41	309
working on the side	73	135

not working	619	720

From the observed frequencies, a segmented data set is generated and the Chi-square test is conducted.

```
man=[1026,41,73,619]
women=[545,309,135,720]
x=join(man,women)
SetSegLen(x,leng?(man))
result = R_ChisqTest(x)
```

	statistic	=	377.938
	parameter	=	3
	p.value	=	1.32914e-81
	method	=	"Pearson's Chi-squared test"

Both at a significance level of 5% and of 1%, the null hypothesis rejected. From this it is possible to conclude that there is a correlation between labor force participation and gender.

R_Execute

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

An R script is run.

Declaration:

```
R_Execute ( TxRScript [, TxRVarNames] [, Variable 1] [, Variable 2] [, Variable 3] [, Variable 4] [, Variable 5]
[, Variable 6] [, Variable 7] [, Variable 8] ) -> Result
```

Parameter:

TxRScript	The R script is passed and run.
TxRVarNames	Specification of the R-variable name for the R-Script. The variable names are separated by a semicolon. (optional)
Variable 1	1st variable according to the TxRVarNames definition (optional)
Variable 2	2nd variable according to the TxRVarNames definition (optional)
Variable 3	3rd variable according to the TxRVarNames definition (optional)
Variable 4	4th variable according to the TxRVarNames definition (optional)
Variable 5	5th variable according to the TxRVarNames definition (optional)
Variable 6	6th variable according to the TxRVarNames definition (optional)
Variable 7	7th variable according to the TxRVarNames definition (optional)
Variable 8	8th variable according to the TxRVarNames definition (optional)
Result	
Result	Result of the R script

Description:

The function performs 3 steps:

1. The FAMOS variable passed: 'Variable 1' ... 'Variable 8' are converted and assigned to the specified R variables.

2. The R script is passed to the R-System and run.

3. The return variable or the result of the R script's last expression are determined, converted to a FAMOS variable and returned as 'Result'.

The R script can contain multiple expressions. The individual expressions are separated by semicolons. When the R script is passed, all the expressions are executed.

If an R-Script is to be loaded from a file and executed, then the TxRScript must begin with the keyword **RScript:**. Next, the filename follows. Example: R_Execute("**RScript:**"+"C:\Program Files\R\R-3.3.1\tests\MaxMinAvg.R","sd_value;input",input)

The parameter TxRVarNames contains the R-names of the return variables and/or the R-names of the input variable. All R-names must be separated from each other with a semicolon.

TxRVarNames	Transfer to the R-system	Return from the R-system
Parameter missing or empty	No transfer of input variable	The result of the R script's last expression is returned.
No return variable is specified. Example: ";a;b"	The input variables a and b are transferred to the R-system.	The result of the R script's last expression is returned.
No input or return variables have been specified. Example: ";"	No transfer of input variable	The result of the R script's last expression is returned.
Only the return variable is specified. Example: "c;"	No transfer of input variable	The value of the return variable c is returned.
The return variable and input variabels are specified. Example: "c;a;b"	The input variables a and b are transferred to the R-system.	The value of the return variable c is returned.

The number of names of input variables must match the number of parameters 'Variable 1'...'Variable 8'. If there are more input names in TxRVarNames than parameters 'Variable 1'...'Variable 8', then the sequence will abort. If there are fewer input names in TxRVarNames than parameters 'Variable 1'...'Variable 8', then the excess parameters are ignored.

The conversion rules for the input variables are described in the function R_SetVar(). It does not matter whether the input variables are passed for an R script with the function R_Execute() or previously by calls of the function R_SetVar().

More specifications can be made for the return variable. In R, variablen can have a complex structure. For example, the R-function t.test(var1, var2) generates a Data Frame with the components 'statistic', 'parameter', 'p.value', 'conf.int', 'estimate', 'null.value', 'alternative', 'method' and 'data.name'. However, if only the p.value is of interest for subsequent analysis, this component can be returned by specifying a component name.

The component name is separated from the variable name by a colon. Example: The R script result=t.test(group1, group2) is executed and only the p.value is to be returned.

```
p_value =R_Execute("result=t.test(group1, group2)","result:p.value;")
```

If the R script does not contain any assignment to a result variable, then a component of the last expression can also be returned.

```
p_value =R_Execute("t.test(group1, group2)",":p.value;")
```

In this case the variable name is omitted and only the desired component is sepcified.

For conversion of the returns to FAMOS, the same rules apply as in the function R_GetVar().

Multithreading: All functions of the R-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are first moved internally to the FAMOS main thread and executed from there, since the R runtime environment does not support parallel calls from different threads.

Examples:

Simple calculation by means of R

```
ramp=Ramp(1,1,100)
rscript="mean_value=mean(input)"
result =R_Execute(rscript,"mean_value;input",ramp)
```

The variable 'ramp' is created in FAMOS. The values of these variables are assigned to the R variable 'input'. The R script calculates the mean value of these variables and assigns the results of the R variable to mean_value. The return variable is converted to the FAMOS variable 'result'.

Simultaneous calculation of multiple values with one script

```
ramp = Ramp(1,1,100)
rscript=        "min_value=min(input);"
rscript=rscript+"max_value=max(input);"
rscript=rscript+"mean_value=mean(input);"
rscript=rscript+"sd_value=sd(input);"
sd_Value1  =R_Execute(rscript,"sd_value;input",ramp); = 29.0115
min_Value1 =R_GetVar( "min_value"); =1
max_Value1 =R_GetVar( "max_value"); =100
mean_Value1=R_GetVar( "mean_value");= 50.5
```

The variable 'ramp' is created in FAMOS. Then the R script is assembled from the individual instructions for calculating Minimum, Maximum, Mean Value and Standard Deviation. It is important for a semicolon to indicate the end of an instruction. The script is executed and the return variable 'sd_value' is returned. The other values are then later read from the R-system using the function R_GetVar().

Calculation with the help of a script-file

The file MaxMinAvgSd.R has the following content:

```
## Example
min_value <-min(input);
max_value <-max(input);
mean_value <-mean(input);
sd_value <-sd(input)
```

```
ramp = Ramp(1,1,100)
rFile="C:\Program Files\R\R-3.3.1\tests\MaxMinAvgSd.R"
rscript="RScript:"+rFile
sd_Value2  =R_Execute(rscript,"sd_value;input",ramp)
min_Value2 =R_GetVar("min_value")
max_Value2 =R_GetVar( "max_value");
mean_Value2=R_GetVar( "mean_value");
```

The function R_Execute assigns the FAMOS variable 'ramp' to the R variable 'input'. Subsequently, the R-Script file MaxMinAvgSd.R is loaded and executed. After execution, the function returns the value of the R variable 'sd_value'. The other values are subsequently read from the R-system by means of the function R_GetVar().

Transfer and execution of a user-defined function

```
x=Ramp(1,1,10)
y=Ramp(1,1,10)
R_Function="matrix_mult <- function(a,b){ c = a * b; return (c);}"
R_Execute(R_Function)
z=R_Execute("matrix_mult(x,y)",";x;y",x,y)
```

It is possible to write and execute user-defined functions in R. In the 3rd line, the script for user-defined functions is set. With the help of this function, the values of two vectors are multiplied. In the following line, this function is passed to the R-System. In the last line, the two FAMOS variables 'x' and 'y' are passed and the function is executed. The result of the expression is returned as a the FAMOS variable 'z'.

Generating a data set with normally distributed random numbers.

The arguments of the function rnorm are the mean value of the distibution, the standard deviation and the number of samples.

```
Rscript="rnorm(mean=2,sd=3,n=100)"
x=R_Execute(Rscript)
```

See also:

R_GetVar, R_SetVar

R_GetOption

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

Gets the R-System's settings

Declaration:

```
R_GetOption ( TxParameterName ) -> TxParameterValue
```

Parameter:

TxParameterName	Name of the declared optional parameter
	"R_HOME" : R-Home folder
	"R_KITVERSION" : Version of the imc R- Kit
	"R_PROCESS" : 64- or 32-Bit version of the R-System
	"R_VERSION" : Version of the R-System
TxParameterValue	
TxParameterValue	Parameter value

Description:

With these functions, the R- System's settings can be obtained. The names of the optional parameters are not case-sensitive.

Examples:

The R-System's parameters to be queried are obtained by means of the following calls.

```
R_Path=R_GetOption("R_HOME") ; -> C:\Program Files\R\R-3.3.2\bin\x64
R_Version=R_GetOption("R_version"); -> 3.3.2
R_KitVersion=R_GetOption("R_KITVERSION"); -> 7.2.1.0
R_Process=R_GetOption("R_PROCESS"); -> x64 or i386
R_Unknown=R_GetOption("R_Unknown"); -> Abort: The optional parameter 'R_Unknown' is not declared.
```

See also:

R_GetVar

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

Gets the value of an R variable.

Declaration:

```
R_GetVar ( TxRVarName ) -> Result
```

Parameter:

TxRVarName	Symbolic name of the R variable
Result	
Result	Data group, normal data set, normal segmented data set, data set with Real- and Imaginary part, segmented data setwith Real and Imaginary part, text or textarray with the value of the R variable

Description:

The R variable of the name specified is found in the R-system and its content is converted to a FAMOS variable. If the R variable does not exist, the sequence aborts.

The name of the R variable is case-sensitive.

For conversion of an R variable to a FAMOS variable, the following rules apply:

R variable type	The FAMOS variable's type
Character vector with 1 element	Text
Character vector with more than one element	Text array
Numeric vector	Normal data set, 8-Byte Real
Numeric matrix	Normal, segmented data set, 8-Byte Real
Complex vector	Complex data set (Real and Imaginary parts), 8-Byte Real
Complex matrix	Complex, segmented data set (Real and Imaginary parts), 8-Byte Real
Integer vector	Normal data set, 4-Byte Integer
Integer matrix	Normal, segmented data set, 4-Byte Integer
Logical vector	Normal data set, unsigned 1-Byte Integer
Logical matrix	Normal, segmented data set, unsigned 1-Byte Integer
Raw Vector	Normal data set, unsigned 1-Byte Integer
Raw matrix	Normal, segmented data set, unsigned 1-Byte Integer
List	Data group
Data frame	Data group
Pair list	Data group
Language object	Data group
S4 object	Data group
Symbol	Text
Factor	Normal data set, 4-Byte Integer

If the R variable's structure is too complex, it can not be converted to a FAMOS variable. The structure is too complex if, for example, the R variable is a list and this list contains an additional list. In this case, the sequence would abort with an appropriate error message.

In order to obtain the results in spite of this, it is possible to read the components of the R variable individually. Toward this end, the component name must be specified. The syntax is **VariableName:ComponentName**. The component name is separated from the variable name by a colon.

The component name can be either a name or a number. If it is a number, then it is interpreted as an index. The component is determined according to this index.

If the component name is a name, then the system attempts to find this name in the Names list of variables. For this purpose, the components of the R-variable must be named.

If a component of a matrix is to be returned, then the component name is structured as follows: **VariableName:RowName,ColumnName**. The row and column names may neither be a number nor a name. They are separated by a comma.

mat	The entire matrix is returned.
mat:	The entire matrix is returned.
mat:,	The entire matrix is returned.
mat:row,	The matrix's row 'row' is returned.
mat:,col	The matrix's column 'col' is returned.
mat:row,col	The value of the matrix's row 'row' and column 'col' is returned.

If the R variable contains values which lie outside of FAMOS' valid numerical range, then they are replaced as follows: For the R variable types numerical vector, numerical matrix, complex vector and complex matrix, the following applies:

Value	Return
Division by zero	0
Value > 1e35	1e35
Value < -1e35	-1e35
NaN	1e35

For the R variable types Integer vector and Integer matrix, the following applies:

Value	Return
Division by zero	-2147483648
NA (missing value)	-2147483648
NaN (no number)	-2147483648

For the R variable types logical vector, logical matrix, Raw vector and Raw matrix:

Value	Return
Division by zero	0
NA (missing value)	0
NaN (no number)	0
Values outside of 0...255	0

Examples:

With the R script, 2 numerical vectors are generated and the function t.test() is run. The result is assigned to the variable 'result'.

The function t.test() returns a list with the components 'statistic', 'parameter', 'p.value', 'conf.int', 'estimate', 'null.value', 'alternative', 'method' und 'data.name'

```
RScript=        "group1 <- c(30.02, 29.99, 30.11, 29.97, 30.01, 29.99);"
RScript=RScript+"group2 <- c(29.89, 29.93, 29.72, 29.98, 30.02, 29.98);"
RScript=RScript+"result= t.test(group1, group2)"
result =R_Execute(RScript)
```

```
The function returns the result of the last expression in the form of the data group 'result':                   statistic   (=1.95901)                  parameter   (=7.03056)
There are two ways to determine only the value of p.value:
p_value1=R_GetVar("result:p.value"); returns the named component p.value of the variable 'result'.
p_value2=R_GetVar("result:3"); returns the 3rd component of the variable 'result'.
```

By means of R script, a Data Frame is generated in R. The first component is a numerical vector, the second a character vector and the third a logical vector.

```
RScript="d <- c(1,2,3,4);"
RScript=RScript+"e <- c('red','white','blue',NA);"
RScript=RScript+"f <- c(TRUE,TRUE,TRUE,FALSE);"
RScript=RScript+"mydata <- data.frame(d,e,f,stringsAsFactors=FALSE);"
RScript=RScript+"names(mydata) <- c('ID' ,'Color','Passed')";  The components of the Data Frame are named.
R_Execute(RScript);  The variable 'mydata' is generated.
myDataGroup=R_GetVar("mydata");  The variable 'mydata' is read and converted to a data group.
The component 'ID' is converted to a normal data set, the component  'Color' to a text array, and the component 'Passed' to a normal data set in the format unsigned 1-Byte Intege

There are two ways to read components of the R variable:
The 1st component of the variable 'mydata' is read.
ID   = R_GetVar("mydata:1")
The named component 'Color' of the variable 'mydata' is read.
Color= R_GetVar("mydata:Color")
```

See also:

R_SetVar

R_Norm

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

Functions for calculating densities, probability distributions and quantiles of the standard normal distribution

Declaration:

```
R_Norm ( TxFnt, x [, mean] [, sd] ) -> Result
```

Parameter:

TxFnt	Name of the function determining what is to be calculated
	"d" : Probability density at the position x
	"p" : Value of the distribution function at the position x
	"q" : Quantile of the specified probability x
x	Single value or normal data set x
mean	Expected/mean value (default=0) (optional , Default value: 0)
sd	Standard deviation (default =1.0) (optional , Default value: 1)
Result	
Result	Result of the calculation

Description:

The calculations are performed by means of R. I.e., an R-script is generated from the parameters and run. Subsequently, the result of the expression is read and converted to a FAMOS variable.

x can be a single value or a normal data set.

The result value count is determined by the count of x values.

Specification of the mean value is optional. If it is not specified, then calculations are based on the default value =0.

Specification of the standard deviation is optional. If it is not specified, then calculations are based on the default value =1.

A negative standard deviation causes an error in the calculation. In such a case, the result value is 1e35.

Examples:

Density: Calculation of the probability desnity at the position x=3

```
y=R_Norm("d",3); y= 0.00443185
```

Distribution function: The value of the distribution function at the position x states the probability of a value <= x being observed.

```
y=R_Norm("p",1.644854); y= 0.95
```

The probability of an observation <= 1.644854 is 0.95

Quantile: Calculation of a value, of which the probability of not being exceeded is 0.95:

```
y=R_Norm("q",0.95,0,1); y= 1.64485
```

R_SetVar

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

Assigns a value to an R variable.

Declaration:

```
R_SetVar ( TxRVarName, FAMOSVariable )
```

Parameter:

TxRVarName	Symbolic name of the R variable
FAMOSVariable	The content of the FAMOS variable is transferred to the R variable.

Description:

Using this function R variables can be defined and values assigned to them.

This value can be a single value, a FAMOS data set, a text, a text array or a data group.

A FAMOS variable is converted according to thefollowing rules:

The FAMOS variable's type	R variable type
Normal data set	Numeric vector
Normal segmented data set	Numeric matrix
Data set with Real and Imaginary parts	Complex vector
Segmented data set with Real and Imaginary parts	Complex matrix
Data set with magnitude and phase	Complex vector
Segmented data set with magnitude and phase	Complex matrix
Text	Character vector
Text array	Character vector
Data group	R list

Data sets with events, data sets with magnitude and phase in db, and data sets in the format Time Stamp ASCII are not supported.

When converting a segmented data set, each segment becomes a column in the R matrix.

Data sets having a magnitude and phase are converted to data sets with Real and Imaginary parts, and subsequently to a complex vector or a matrix.

A data group is converted to an R list. The components of the R list are derived from the elements of the data group.

Multithreading: All functions of the R-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are first moved internally to the FAMOS main thread and executed from there, since the R runtime environment does not support parallel calls from different threads.

Examples:

The FAMOS variable 'fa' is assigned to the R variable 'a'.

```
fa=1000
R_SetVar("a",fa); single value -> Numeric vector
```

The Text variable 't' is assigned to the R variable 'ch'.

```
t="123.4"
R_SetVar("ch",t); Text -> Character vector
```

The segmented data set 'seg' is assigned to the R variable 'mat'.

```
seg=Ramp(1,1,100)
SetSegLen(seg,20)
R_SetVar("mat",seg); Normal segmented data set -> Numeric matrix
```

A data group 'roller' is created. This group is assigned to the R variable 'roller'. The R variable 'roller' is of the type R List.

```
weight=Ramp(0,1,3) ; Normal data set
Weight[1]=1.9
Weight[2]=3.1
Weight[3]=3.3
depression=Ramp(0,1,4) ; 1 value more than Weight
```

```
depression[1]=2
depression[2]=1
depression[3]=5
depression[4]=5
doc= TxArrayCreate(2) ; Text array with 2 elements
doc[1]="abc"
doc[2]="def"
dsseg=Ramp(1,1,100); Normal segmented data set
SetSegLen( dsseg,20)
roller=GrNew(); creating the data group 'roller'
GrChannAppend(roller,weight)
GrChannAppend(roller,depression)
GrChannAppend(roller,doc)
GrChannAppend(roller,dsseg)
R_SetVar("roller",roller) ; FAMOS data group --> R list
```

See also:

R_GetVar

R_tTest

Scope: R remote control

Available in: Professional Edition and above (R-Kit)

Function for performing the t-test

Declaration:

```
R_tTest ( x [, y] [, alternative] [, mu0] [, paired] [, var_equal] [, conf_level] ) -> Result
```

Parameter:

x	Data of the first sample
y	Data of the second sample (in the Two-sample t-test) (optional , Default value: 0)
alternative	Formulation of the alternative hypothesis (optional , Default value: "t")
	"t" : H1: mu <> mu0 (default)
	"g" : H1: mu > mu0
	"l" : H1: mu < mu0
mu0	Hypothetiszed population mean (default = 0) (optional , Default value: 0)
paired	Paired samples (optional , Default value: 0)
	0 : Independent samples (default)
	1 : Paired samples
var_equal	Variances of the samples are equal (optional , Default value: 0)
	0 : Variance heterogeneity (default)
	1 : Variance homogeneity (Welch-test)
conf_level	Confidence level (default =0.95) The confidence level states with what probability the population parameter (e.g. the mean value) is within the confidence interval. (optional , Default value: 0.95)
Result	
Result	The result of the t-test is a data group having the elements:
	statistic : Value of the t-statistic
	parameter : Number of degrees of freedom. The degrees of freedom represent the number of variables in a system which vary independently of each other, while the arithmetic mean remains fixed.
	p.value : The p-value is a measure of probability for the evidence against assuming the null hypothesis. Its value lies in the range between 0 and 1. Lower probabilities indicate stronger evidence that the null hypothesis is not accepted. The p-value can be compared with an alpha-value in order to decide whether the null hypothesis (H0) is to be rejected. If the p-value <= alpha (significance level), the null hypothesis is rejected. If the p-value > alpha, the null hypothesis is not rejected.
	conf.int : The confidence interval indicates the value range in within which there is a certain probability that the population parameter lies. How high this probability is to be is determined by the confidence level.
	estimate : The estimated mean value or the mean value in dependence on whether the test was a One-sample t-test or Two-sample t-test.
	null.value : The specified hypothesis pertains to either the mean value or the difference of mean values, for a One-sample t-test or Two-sample t-test, respectively.
	alternative : The character sequence describes the alternative hypothesis.
	method : The character sequence indicates which type of t-test was performed.

Description:

The t-test describes a group of hypothesis tests with a t-distributed test test variable. It can be used to determine whether two samples are statistically significantly different. Only 2 samples are compared, which must be normally distributed.
In a hypothesis test, two hypotheses that are contrary to a population are examined: the null hypothesis and the alternative hypothesis.
The null hypothesis H0 says, that a parameter of the population equals a certain value.
The alternative hypothesis H1 states that the parameter of the population differs from the value of the parameter of the population in the null hypothesis.
With the t-test a distinction is made between:

One-sample t-test

The One-sample test uses the mean value of a random sample to check whether the mean value differs from a specified target value. It is

assumed that the sample data are normally distributed. If the sample is large, the t-test can also be used without the assumption of normal distribution can be used.
For the one-sample test, the second sample 'y' must be empty or not specified.
The setpoint is given as 'mu0'.The call is:
result = R_tTest (x, empty, "t", mu0)

The hypotheses for the single sample test are:

Null hypothesis	Meaning
H0: mu = mu0	The mean value of the population (mu) is equal to the hypothetical mean value (mu0).
Alternative hypothesis	.
H1: mu != mu0	"t" The mean value of the population (mu) differs from the hypothetical mean value (mu0).
H1: mu > mu0	"g" The mean value of the population (mu) is greater than the hypothetical mean value (mu0).
H1: mu < mu0	"l" The mean value of the population (mu) is less than the hypothetical mean value (mu0).

Two-sample t-test

The Two-sample test uses the mean values of two independent samples to check how the mean values of two populations compare behave towards each other. The mean value difference is tested against mu0 or against 0. It is assumed that the sample data are normally distributed or there are sufficiently large sample sizes. The data from both samples must first be examined (using an F-test) for homogeneity of variance. If there is homogeneity of variance, the call is:
result = R_tTest (x, y, "t", 0.0,0,1)
, where 'x' is the data from the first sample and 'y' is the data from the second sample. The parameter 'var_equal' = 1 means that homogeneity of variance is assumed.
If there is heterogeneity of variance, the call is:
result = R_tTest (x, y)
This call corresponds to the Welch test.

The hypotheses for the Two-sample t-test are:

Null hypothesis	Meaning
H0: mux-muy = delta0	The difference between the mean values of the populations (mux-muy) is equal to the hypothetical difference (delta0). In this case, the parameter mu0 equals delta0.
Alternative hypothesis	.
H1: mux-muy != delta0	"t" The difference between the mean values of the populations (mux-muy) is not equal to the hypothetical difference (delta0). In this case, the parameter mu0 matches delta0.
H1: mux-muy > delta0	"g" The difference between the mean values of the populations (mux-muy) is greater than the hypothetical difference (delta0). In this case, the parameter mu0 matches delta0.
H1: mux-muy < delta0	"l" The difference between the mean values of the populations (mux-muy) is less than the hypothetical difference (delta0). In this case, the parameter mu0 matches delta0.

Paired t-test

The Paired t-test examines for two interrelated samples whether the mean difference of the measured values is different. It is performed if two surveys were conducted in the same examination group, and these data are now to be examined. The value count in both samples must be identical. The differences of the paired measurement values must be normally distributed. It is not adequate to show that the two samples conform to a normal distribution.
The call is:
result=R_tTest(x,y,"t",0,1)
where 'x' is the sample data of the 1st survey and 'y' the sampel data of the 2nd survey.

The hypotheses for the Paired t-test are:

Null hypothesis	Meaning
H0: mud=mu0	The mean value of the differences of the population (mud) is equal to the hypothetical mean of the differences (mu0).
Alternative hypothesis	.
H1: mud != mu0	"t" The mean value of the population's differences (mud) is not equal to the hypothetical mean value of the differences (mu0).
H1: mud > mu0	"g" The mean value of the population's differences (mud) is greater than the hypothetical mean value of the differences (mu0).
H1: mud < mu0	"l" The mean value of the population's differences (mud) is less than the hypothetical mean of the differences (mu0).

Examples:

From a delivery of diodes with the desired pass-through resistance of 100 mOhm, a random sample of size 10 is taken and measured.

```
resistance =[114.62,110.10,106.31,99.30,107.28,108.35,113.64,117.92,130.15,102.74]
```

```
result=R_tTest(resistance,empty,"t",100.00)
The sample test delivers the result as the data group 'result':
```

	statistic	=	4.0066
	parameter	=	9
	p.value	=	0.00307962
	conf.int	=	104.8072, 117.2748
	estimate	=	111.041
	null.value	=	100
	alternative	=	"two.sided"
	method	=	"One Sample t-test"

We obtain the mean value = 111.04 and the t-value = 4.0066 with 9 degrees of freedom.
The test's p-value is 0.00308. This is substantially below the significance level of 0.05, so that the null hypothesis must be rejected.
With high probability, the mean pass-through resistance does not match the desired pass-through resistance of 100 mOhm.

Given two shipments of diodes, it is to be determined whether the pass-through resistance of the two shipments is identical (the difference between the expected values is = 0). From each shipment, a random sample of 10 dioden is taken and measured. It is assumed that the variance of both is the same.

```
resistanceA =[114.62,110.10,106.31,99.30,107.28,108.35,113.64,117.92,130.15,102.74]
resistanceB =[101.77,109.86,131.41,105.29,104.49,118.62,108.60,139.09,113.72,114.91]
mu0=0
var_equal=1
result=R_tTest(resistanceA,resistanceB,"t",mu0,0,var_equal)
The two-sample test delivers the result as the data group 'result':
```

	statistic	=	-0.79341
	parameter	=	18
	p.value	=	0.437873
	conf.int	=	-13.6251, 6.1551
	estimate	=	111.0410, 114.7760
	null.value	=	0
	alternative	=	"two.sided"
	method	=	"Two Sample t-test"

We obtain the mean value of resistanceA = 111.04 and of resistanceB = 114.77.
The resulting t-value is = -0.7934 with 18 degrees of freedom.
The null hypothesis is assumed: the p-value of the test is 0.4379.
With high probability, the two shipments have the same mean resistance.

The tensile strengths of wires are examined using a machine A and a machine B. To test the equality of the two machines, a sample of 12 wires is divided and each half is tested at one of the machines. The resulting tensile strength measurements are:

```
machineA=[ 35, 46, 34, 27, 37, 59, 52, 61, 21, 31, 37, 27]
machineB=[ 39, 51, 32, 23, 41, 53, 51, 55, 19, 36, 37, 26]
paired=1
result=R_tTest( machineA,machineB,"t",0,paired)
The paired Two-sample test delivers the result as the data group 'result':
```

	statistic	=	0.286513
	parameter	=	11
	p.value	=	0.77981
	conf.int	=	-2.2273, 2.8940
	estimate	=	0.333333
	null.value	=	0
	alternative	=	"two.sided"
	method	=	"Paired t-test"

One obtains a t-value of = 0.2865 with 11 degrees of freedom. The mean value of the differences is 0.333. The null hypothesis can be accepted, since the p-value is = 0.7798. The machines have equal mean tensile strength measurements.

Ramp

Generates a ramp (straight line with slope 1) with specifiable initial value, point interval and length

Declaration:

```
Ramp ( SvStart, SvDelta, SvLength ) -> Ramp
```

Parameter:

SvStart	Start value
SvDelta	Sampling interval (Delta-X)
SvLength	Length
Ramp	
Ramp	Resulting data set in the form of a ramp

Description:

A straight line with the slope 1 is generated, given by the equation

```
f(x) = x
```

The line generated takes the form of a ramp.

- The data set generated has no units. The length (3rd parameter) must be an integer greater than 0. The sampling interval (2nd parameter) must be greater than 0.
- The Ramp() function is generally used to create test channels for comparison.

Examples:

A data set with the values 2.0, 2.1, 2.2, 2.3, 2.4 is generated:

```
Ramp5 = Ramp(2, 0.1, 5)
```

A data set with 10 values is created, where all values are equal to zero. Such data sets with known sampling rates and number of values are often required as a basis for further operations requiring a data set data type:

```
TenZeroes = Ramp(0, 1, 10) * 0
```

A sine-shaped data set is created. It contains two periods of a sine function with an amplitude 3 A and a phase shift of PI/4. The predefined constant PI is used:

```
Sinus = 3 'A' * sin(Ramp(PI/4, PI/128, 512))
```

Data recorded before the trigger are joined with data after the trigger:

```
NDcomplete = Join(NDpreTrigger, NDpostTrigger)
```

See also:

Random, Leng, XOff, XDel

Random

Generates random numbers with user-specified distribution

Declaration:

```
Random ( SvCount, SvDistribution, SvPar1, EwPar2, SvInit ) -> RandomNumbers
```

Parameter:

SvCount	Length of the data set generated
SvDistribution	Selection for distribution
	0 : Uniform distribution
	1 : Exponential distribution
	2 : Normal distribution
	3 : Binomial distribution
SvPar1	For uniform distribution, the minimum; for binominal distribution, the maximum value. The numbers generated lie within the range (0,1,2...SvPar1). Else, set to 0.
EwPar2	For uniform distribution, the maximum. For binominal distribution, the desired distribution value p (0< p <1). Else, set to 0.
SvInit	Value for initializing the pseudo-random number generator. 0 means no re-initilaization; else >0 and integer
RandomNumbers	
RandomNumbers	Data set containing random numbers

Description:

Examples:

```
r1 = Random(10000, 0, -1, 1, 0)
```

Generates 10000 uniformly distributed random numbers between -1 and +1.

```
r21 = Random(10000, 2, 0, 0, 11)
```

Generates 10000 normally distributed random numbers.

```
r22 = Random(10000, 2, 0, 0, 11)
```

Identical with r21 (since same value for initialization),

```
r4 = Random(10000, 3, 1, 0.7, 24)
```

Generates a data set of length 10000, which consists of the values 0 (approx. 30%) and 1 (approx.70%).

See also:

Ramp

RangeSet

Available in: Professional Edition and above

Input data values which lie within a specific value range of the controlling channel are set to a different value.

Declaration:

```
RangeSet ( input data, Controlling channel, Bottom condition, Lower boundary, Top condition, Upper boundary,
Action [, Substitute value] ) -> Result
```

Parameter:

input data	input data
Controlling channel	Controlling channel
Bottom condition	Condition for lower boundary
	">=" : Controlling channel >= Lower boundary
	">" : Controlling channel >= Lower boundary
Lower boundary	Lower boundary
Top condition	Condition for upper boundary
	"<=" : Controlling channel <= Upper boundary
	"<" : Controlling channel < Upper boundary
Upper boundary	Upper boundary
Action	Action
	"=" : Set to substitute value
	"+" : Add a substitute value
	"first" : When the signal enters the range, use the first value of the input data as the substitute value
	"before" : When the signal enters the range, use the previous value of the input data as the substitute value. 0.0, if no such previous value exists.
Substitute value	Substitute value (optional , Default value: 0)
Result	
Result	Result

Description:

The two conditions are joined in an AND expression.

The real number input data may contain events and segments. Equidistant and XY-data are supported, with XY-data, the Y-component is subjected to calculations.

The controlling channel is equidistant. Assignment of a value of the input data and of a controlling channel value is performed value-by-value in order.

The input data and the controlling channel must have the same length and structure (segments and events).

On the topic of replacing Lost Value, Overflow, Not a Number, Overmodulation, Sensor Breakage, refer to LostValueReplace().

Examples:

When determining a rotation speed from pulses, at times of immobility, there is no RPM-value above zero due to the long pulse distance. Set all values between 0 and 10 to 0.0.

```
rpm = RangeSet ( rpm, rpm, ">=", 0, "<=", 10, "=", 0 )
```

With gear recognition, reverse appears as 7, but should be made -1.

```
gear = RangeSet ( gear, gear, ">=", 7, "<=", 7, "=", -1 )
```

If the temperature is too low, the force measured is invalid and is set to 0.

```
force = RangeSet ( force, temperature, ">=", -1e35, "<=", -40, "=", 0 )
```

See also:

CodeRange, LowerValue, UpperValue, Top, Set, LostValueReplace

Recip

Reciprocal

Declaration:

```
Recip ( Parameter ) -> Result
```

Parameter:

Parameter	Parameter. Allowed types: [ND],[XY].
Result	
Result	Reciprocal of the parameter

Description:

The reciprocal or inverse value is calculated.

Remarks

- The x-coordinate(s) of the parameter and the result are the same.
- The reciprocal of the unit is also formed; e.g. from "V" we get "1/V".
- Division by zero is not allowed; i.e. the parameter may not become zero. If it does, an appropriate warning is issued.
- The parameter may be structured (events/segments).

Examples:

From an electrical conductance value, the calculation of the reciprocal determines elecrical resistance value:

```
resistance = Recip(conductance)
```

This line is identical to the following:

```
resistance = 1 / conductance
```

See also:

/(Division)

Rect

Transformation of a complex data set to Cartesian (rectangular) coordinates (Real/Imaginary part).

Declaration:

```
Rect ( ComplexData ) -> ComplexAsRI
```

Parameter:

ComplexData	Complex data set to be transformed [BP], [DP] or [RI]
ComplexAsRI	
ComplexAsRI	Resulting complex data set in Cartesian (rectangular) coordinates [RI]

Description:

complex data set is transformed to rectangular coordinates with real and imaginary parts. The data type of the original data set is irrelevant; if the data set already exists in rectangular coordinates, this function has no effect.

- The y-units are adjusted according to the transformation performed.
- The parameter may be structured (events/segments).

Examples:

A spectrum is transformed to the usually more graphically clear display style BP:

```
MPspectrum = Pol(RIspectrum)
```

A spectrum is calculated and expressed in Cartesian (rectangular) coordinate:

```
RIdata = Rect(Spec(NDdata))
```

See also:

Pol, idB, Compl

Red

Subsequent resampling with user-specified reduction factor

Declaration:

```
Red ( Data, SvFactor ) -> Result
```

Parameter:

Data	Data set to be resampled: Allowed data types: [ND]
SvFactor	Reduction factor
Result	
Result	Result of re-sampling.

Description:

A data set is resampled to reduce the number of data. The factor by which data are to be reduced can be specified; during sampling, every nth value of the specified data set is entered in the resulting resampled data set.

- The unit of the data set is retained, but the sampling rate is increased by the specified factor.
- The reduction factor should not have an associated unit and should be an integer greater than 1.
- The function Red() reverses the functions Ipol() and Lip().
- Please note that aliasing effects can occur during resampling when high frequency components are present in the data set. Perform low-pass filtering with a smoothing function before resampling, for example Smo.

Examples:

```
NDshort = Red(NDlong, 5)
```

A data set with a length of 10000 points is reduced by a factor of 5 to 2000 points, taking each fifth point of the original data set.

```
NDShort = Red(Smo5(NDlong), 2)
```

Aliasing effects are suppressed by filtering the data set with a low-pass filter before sampling. For greater reduction factors, the Smo function is usually more convenient.

See also:

Red2, RedEx, RSamp, RSampEx, IPol, Lip

Red2

Subsequent resampling in order to produce a power of two.

Declaration:

```
Red2 ( Data ) -> Result
```

Parameter:

Data	Data set to be resampled: Allowed data types: [ND]
Result	
Result	Result of re-sampling.

Description:

This function is designed especially for use with function such as FFT, ACF, CCF and Spec. A data set is resampled so that its length becomes a power of two. The above functions can then be applied to the data set without any values being discarded.

The Red2 function works like the function RSamp. The data set is treated as if linear interpolation were performed on its sampling values and these values are sampled with an appropriate sampling rate. The new sampling rate can be somewhat greater or smaller than the original. The ratio between the new and old sampling rates in the Red2 function is usually not an integer.

If the length of the data set is slightly greater than a power of two (up to 10% greater), the length is reduced to the smaller power of two. If the length is greater than this limit, it is extended to the next larger power of two. This method is selected to prevent any significant aliasing effects and so that the data sets do not become unmanageable (calculation time!).

- The units remain unchanged, but the sampling rate is adjusted.
- The assumption that the values between the sampling rates can be approximated through linear interpolation is not always valid. It is recommended to perform spline interpolation before applying the Red2 function.
- Red2 takes the maximum data set length which can be processed into account for the functions FFT, ACF, CCF and Spec. Extremely long data sets are then reduced to a maximum of 134.217.728 (2^27) values.

Examples:

```
MPspectr = Spec(Red2(NDdata))
```

The spectrum of a data set with 2000 points is calculated.

```
NDacf = ACF(Red2(NDdata))
```

The autocorrelation function of a data set with 6000 points is calculated.

See also:

Red, RSamp, RSampEx, IPol, Lip, FFT, CCF

RedEx

Available in: Professional Edition and above

Sampling with user-specified reduction factor and start

Declaration:

```
RedEx ( Dataset, Reduction factor [, Start index] ) -> Result
```

Parameter:

Dataset	Dataset
Reduction factor	Reduction factor, every nth sample is applied; integer >= 1
Start index	Start index, >= 1; the first sample is taken at this index. (optional , Default value: 1)
Result	
Result	Result

Description:

A data set is subjected to a later sampling. In the process, the data volume is reduced. You can specify a factor by which the data volume is reduced. When sampling, every n-th value of the data set passed is entered in the data set to be generated.

The data set's unit is retained, the sampling interval dx increased by the factor passed.

When a data set is sampled, aliasing effects occur when the data set contains high frequency components. Before sampling, you should use a smoothing function or low-pass filtering, e.g. Smo().

If a Start index > 1 is specified, the result's x-offset is increased accordingly.

The data set may contain events and segments, it may be equidistant, XY or complex.

Examples:

The 2nd, 5th, 8th measurement values, etc. are cut out.

```
short = RedEx ( Data, 3, 2 )
```

See also:

Red, Lip, IPol, RSampEx, CutIndex, MatrixIpol

RemoveSamples

Values are deleted from a data set.

Declaration:

```
RemoveSamples ( Data, SvSampleIndex, SvCount, SvEventIndex, Zero )
```

Parameter:

Data	Data set from which values are to be deleted
SvSampleIndex	Index of the (first) value to be deleted. The index of the data set's first value is 1. With event-based data, the index references the beginning of the selected event.
SvCount	Number of values to be deleted. -1, in order to delete all the way to the end of the data set or of the selected event.
SvEventIndex	For event-based data, the index (1..) of the desired event is to be specified here. 0 for non-event-based data.
Zero	Reserved parameter. Always set to 0.

Description:

This function can be applied to normal (equidistantly sampled) data sets as well as to [XY]- and complex data.

The data type of the variables remains unchanged; so for normal waveforms, the time/x-axis is 'lumped together'. For instance, if the time assignment of all values is to remain intact, the data set must first be converted to an equivalent XY-data set (see example).

With [XY] and [Complex], the corresponding value in the 2nd component (X, Phase, Imaginary part) is also deleted.

Segmented data sets and data of the type TimeStamp-ASCII are not allowed.

The function deletes a maximum of [Count] values beginning with [SampleIndex], but at most until the end of the data set or the current event.

Examples:

Removes the first two samples of a data set.

```
RemoveSamples(Signal, 1, 2, 0, 0)
```

The third event of a data set is truncated after the 10th sample.

```
RemoveSamples(SignalWithEvents, 11, -1, 3, 0)
```

From an equidistantly sampled waveform, a portion in the middle is removed, but the time reference of subsequent samples is to remain unchanged. To this end, the waveform is first converted to an XY-data set.

```
SignalXY = XYof(Ramp(xoff?(Signal), xdel?(Signal), Leng?(Signal)), Signal)
RemoveSamples(SignalXY, 200, 100, 0, 0)
```

See also:

Cut, CutIndex, SamplesGate, Repl, ReplIndex

RENAME

A variable is assigned a new name.

Declaration:

```
RENAME OldName NewName
```

Parameter:

OldName	Old variable name
NewName	New variable name

Description

A variable is renamed. The old name is transferred as the first parameter, the new name as the second.

If a variable with the desired new name already exists, this variable is deleted and the old variable is overwritten by the new name.

Examples:

A waveform is loaded, renamed and displayed:

```
Data = ...
RENAME Data {BrakeTest - Test #1}
SHOW {BrakeTest - Test #1}
```

A channel is a data group is renamed and displayed:

```
MyGroup:Channel1 = data
RENAME MyGroup:Channel1 MyData
SHOW MyGroup:MyData
```

See also:

SHOW, DELETE

RenameMeasurement

A measurement is renamed.

Declaration:

```
RenameMeasurement ( TxOldName, TxNewName ) -> SvSuccess
```

Parameter:

TxOldName	Current name of measurement to be renamed
TxNewName	New name of the measurement
SvSuccess	
SvSuccess	Success of the function (optional)
	0 : Either there is no measurement having the name [TxOldName], or there is already a measuremnt having the name [TxNewName].
	1 : OK

Description:

The function renames a measurement.

The concept of a variable's association with a measurement is primarily applicable to the Data Source Browser. There, when a file of measurement values is loaded, a measurement name is automatically assigned to each variable generated, in order to be able to easily distinguish variables having the same name but different data sources. Changing the measurement name affects all variables which were previously assigned to this measuremnet and causes automatic updating of the Measurement- and Channel lists in the Variables list/Measurement view.

If [TxOldName] is currently recorded in the data selector's list of measurements (meaning, is assigned to a symbolic measurement number), the corresponding list entry is also renamed.

For all curve windows representing a channel belonging to this measurement, the reference to the measurement will be renamed. The display remains intact.

The new name may not match the name of any existing measurement.

The working of the function corresponds to that of the command "Rename Measurement" in the Measurement list's context menu, in the Variables list/Measurement view.

Folders or filenames in the file system are not changed by this function.

Examples:

Once a new measurement has been created using the Data Source Browser, the system checks whether it contains a channel named 'speed'. If yes, the measurement name assigned automatically is replaced by the channel's trigger time.

Event sequence 'Measurement available'

```
TxVarName = SelBuildVarName(PA1, "speed", 0)
IF TxVarName <> ""
   TxNewMeasName = TimeToText(Time?(<TxVarName>), 0)
   RenameMeasurement(PA1, TxNewMeasName)
END
```

All currently loaded measurements whose names begin with the prefix 'BLF' are assigned a more distinctive name:

```
measurements = MeasNames?("BLF*")
FOREACH ELEMENT name IN measurements
   RenameMeasurement(name, TReplace(name, "BLF", "BottomLeftSensor"))
END
```

See also:

SetMeasurementName, MeasNames?

Repl

Replaces a part of a data set with new data.

Declaration:

```
Repl ( Data, NewPart ) -> ResultData
```

Parameter:

Data	Data set to be changed; allowed types: [ND].
NewPart	Portion to be inserted
ResultData	
ResultData	Changed data set with corresponding new values

Description:

In the data set [Data], data points are replaced with data points from the the data set [NewPart], in accordance with the x-offset and the length of [NewPart]. Thus, [NewPart] is copied into [Data] at a specified location. The function Repl() transfers the data points point-by-point, even when the sampling intervals are different.

The function Repl() is especially suitable for copying parts of data set which were cut out with the function Cut() and editedback into the whole data set.

Data points are only transferred from the specified part on the condition that they do not lie outside of the original total data set.

If the sampling intervals of both data sets don't match, use the function RSamp, in order to assimilate the sampling intervals.

Alternatively, you can work with the function ReplIndex() if you wish to specify the insertion position by the index of the point in the data set. This function is also suitable for XY-data sets.

Examples:

The Cut function is used to cut out a section from x = 1 to x = 2 from the data set NwData. This section is smoothed and then copied back into the original data set with the Repl function. This result appears as if the range from x = 1 to x = 2 had been smoothed but the rest of the data set was left unchanged.

```
NDpart = Cut(NDdata, 1, 2)
NDpart = Smo5(NDpart)
NDdata = Repl(NDdata, NDpart)
```

See also:

ReplIndex, Cut, CutIndex, ValueIndex

ReplIndex

Replaces a part of a data set with new data.

Declaration:

```
ReplIndex ( Data, NewPart, SvStartIndex ) -> ResultData
```

Parameter:

Data	Data set to be changed; allowed types: [ND],[XY].
NewPart	Portion to be inserted
SvStartIndex	Index in [Data] at which the replacement begins
ResultData	
ResultData	Changed data set with corresponding new values

Description:

In the data set [Data] the data points are replaced with data points from [NewPart] startingg at a specified position [StartIndex].

The length of [NewPart] determines the number of data points to be replaced.

[SvStartIndex] must be an integer between 1 and the length of the data set [Data].

The lengths of the result and of the first parameter are the same, or, if applicable, surplus values from [NewPart] are ignored (a warning is issued).

The data types of [Data] and [NewPart] must be the same. If they are XY-data sets, the xcoordinates are also replaced. With real number data sets, only the y-values are replaced, the x-scaling (Offset and sample interval or x-increment) remain intact.

ReplIndex is particularly suitable for recopying data sections cut out using the CutIndex function and then processed back into the data set from which they originated.

Alternatively, you can use the function Repl which requires you to specify the starting point of the replacement in terms of its x-coordinate.

Examples:

The y-values of "Wave" are doubled, excepting those of the first and the last points:

```
part = CutIndex(Data, 2, leng?(Data)-1)
part = part * 2
Data = ReplIndex(Data, part, 2)
```

See also:

Repl, Cut, CutIndex, ValueIndex, MatrixMerge, MatrixFromLine

REQUEST

Available in: Professional Edition and above

Receive data via DDE
This command is obsolete; instead of it, the more powerful function DDEINQ() should be used.

Declaration:

```
REQUEST Application Topic Item VariableName
```

Parameter:

Application	Name of the DDE-application to be addressed
Topic	Designation of the DDE topic
Item	Name of the item to be queried
VariableName	Name of the FAMOS variable to be received

Description

FAMOS operates as a DDE client, the application addressed as a DDE server. The entries "Application" and "topic" cause FAMOS to select another DDE-capable application and request a conversation. If the request is complied with, FAMOS then supplies the name of the item desired. FAMOS waits until the server either transfers the data, or denies the request. Then imc FAMOS ends the conversation.

The variable specified in "Variable" now contains the data received. If the transfer wasn't possible, it contains the return value from the DDE application addressed.

The data can be single values, as well as data sets in ASCII or FAMOS DDE format.

- The individual parameters for the REQUEST command may not contain space characters.
- FAMOS sends an error message, if DDE application (server) addressed doesn't answer or the data request was denied.
- If the application (server) addressed is busy, FAMOS waits until it is free.
- FAMOS commands are case-insensitive (upper or lower-case letters are equally valid). However, some DDE applications do distinguish between upper and lower case. Have regard, therefore, for correct spelling.

Examples:

```
REQUEST Trans Timebase TB x
```

"Trans" is the name of an imaginary, DDE-capable application, "Timebase" the name of the DDE-topic. The return value is written to variable "x".

```
REQUEST Trans1 Status Trigger Svtrig
REQUEST Trans2 Status Armed SVarm
```

Data can be requested from varying DDE-applications. After execution of this command the two single value variables contain information about the status of the "trans"-devices. For example, the variable SVTrig could be =1.0, if the "Trans1" trigger was tripped

See also:

DDEInq, DDESend, DDESet

RGB

Forming a color value from the color component

Declaration:

```
RGB ( SvRed, SvGreen, SvBlue ) -> SvColorValue
```

Parameter:

SvRed	Red component of the color; 0..255
SvGreen	Green component of the color; 0..255
SvBlue	Blue component of the color; 0..255
SvColorValue	
SvColorValue	Color value

Description:

Using the intensity values (0..255) of the 3 basic colors, a color value is computed, which can be passed to the function SetColor().

Data format of the result: 4 Byte unsigned Integer

As of Version 7.5: As well as single values, the parameters supplied can also be data sets, which are then jointly subjected to calculations value-by-value. Events and segments are allowed, but all 3 parameters msut have exactly the same structure (same total length and segment length, same event lengths). The data set of results then has the same structuring. When there is segmenting, the system assumes that the result can be interpreted as image data in the RGB-format and the corresponding internal flag is set; see also the function SetFlag().

Examples:

```
fb = RGB(255, 0, 0) ;Red
fb = RGB(0, 0, 0) ;Black
fb = RGB(0, 0, 255) ;Blue
fb = RGB(255, 255, 255) ;White
```

In the following example, the video file 'sample.avi' is loaded in the current Panel video player and the 100th picture is extracted. Subsequently, the picture is converted first to grey tones and finally to a black/white image.

```
VpVideoLoad("c:\tmp\sample.avi", 1)
VpSetPosFrames(100, 1)
image = VpGetImages(1)
RedValue =   RGBConvert(image, "R")
GreenValue = RGBConvert(image, "G")
BlueValue =  RGBConvert(image, "B")
; weighted grey tone conversion
; coefficients according to "Luma coding" as per "ITU-R Recommendation BT.601":
GreyValue = 0.3* RedValue + 0.59* GreenValue + 0.11 *BlueValue
imageGrey = RGB(GreyValue, GreyValue, GreyValue)
; conversion black/white. All grey tones > 127 become white; all others black:
BWValue = (GreyValue > 127)*255
imageBlackWhite = RGB(BWValue, BWValue, BWValue)
```

Generates an image data set with 640x480 pixels; all pixels red:

```
r = Leng(0, 640*480) + 255
g = r*0
b = r*0
image = RGB(r, g, b)
SetSegLen(image, 640)
SetFlag(image, 1, 1)
```

See also:

RGBConvert, SetColor, Color?, SetFlag, VpGetImages

RGBConvert

Available in: Professional Edition and above

Converts/processes the RGB-data passed.

Declaration:

```
RGBConvert ( RGB_Data, Calculation [, Parameter1] [, Parameter2] ) -> Result
```

Parameter:

RGB_Data	Input data in RGB-format
Calculation	Calculation
	"R" : Red-component (R in RGB) in the range: 0 .. 255
	"G" : Green-component (G in RGB) in the range: 0 .. 255
	"B" : Blue-component (B in RGB) in the range: 0 .. 255
	"RGB" : Input data to be interpreted as RGB colors are converted to the 4-Byte RGB format with a color flag. Rounding is performed.
	"GREY" : Input data within the range 0 .. 255 are converted to 1-Byte grayscale format with color flag. Rounding and limiting to the value range 0 .. 255 are performed.
	"bright1" : For RGB-input data, the brightness sqrt(0.299*R^2 + 0.587*G^2 + 0.114*B^2) in the range: 0 .. 255
	"bright2" : For RGB-input data, the brightness (0.2126*R + 0.7152*G + 0.0722*B) in the range: 0 .. 255
	"bright3" : For RGB-input data, the brightness (0.299*R + 0.587*G + 0.114*B) in the range: 0 .. 255
	"HSV-H" : For RGB-input data, the hue. H-component in HSV-representation, in the range: 0 .. 359. E.g. 0 for red, 120 for green, 240 for blue.
	"HSV-S" : For RGB-input data, the saturation. S-component of HSV-representation in the range: 0 .. 255, corresponding to 0 .. 100%.
	"HSV-V" : For RGB-input data, the lightness. V-component of the HSV-representation in the range: 0 .. 255, corresponding to 0 .. 100%.
	"HSL-H" : For RGB-input data, the hue. H-component of HSL-representation in the range: 0 .. 359. E.g. 0 for red, 120 for green, 240 for blue.
	"HSL-S" : For RGB-input data, the saturation. S-component of HSL-representation in the range: 0 .. 255, corresponding to 0 .. 100%.
	"HSL-L" : For RGB-input data, the luminance. L-component of HSL-representation in the range: 0 .. 255, corresponding to 0 .. 100%.
	"add" : For RGB-input data, adds the RGB-value provided as Parameter1.
	"sub" : For RGB-input data, the RGB-value specified as Parameter1 is subtracted.
	"inv" : Input data located in the range 0 .. 255 are inverted; result = 255 - input data. The results are located in the range: 0 .. 255.
	"1.GREY" : Input data located in the range 0.0 .. 1.0 are converted to 1-Byte grayscale format with color flag. Rounding and limiting to the value range 0 .. 255 are performed.
	"similar12" : For RGB-input data, determines an RGB-value's respective greatest similarity to a color transition between the RGB-colors stated in Parameter1 and Parameter2. The result is 255 for equivalence and 0 for absolutely no match.
Parameter1	1st parameter; interpretation depends on the calculation type (optional , Default value: 0)
Parameter2	2nd parameter; interpretation depends on the calculation type (optional , Default value: 0)
Result	
Result	Result

Description:

The equidistant input data can have events and segments.

For results in the range 0..255 (interpreted as grayscale shades) and results with RGB-values, the Color flag (flag for color information) is set.

Examples:

Determine R, G, B from RGB

```
colors = RGB(200, 100, 0)
R = RGBConvert( colors, "R" )
G = RGBConvert( colors, "G" )
B = RGBConvert( colors, "B" )
```

Determines the brightness, resulting in gray scale shades: 0..255

```
colors = RGB(200, 100, 0)
bright1 = RGBConvert( colors, "bright1" )
bright2 = RGBConvert( colors, "bright2" )
bright3 = RGBConvert( colors, "bright3" )
bright4 = RGBConvert( colors, "HSV-V" )
bright5 = RGBConvert( colors, "HSL-L" )
```

Highlights all locations in a picture which are somewhat green to dark green.

```
picture = Random(36,0, 0, 256*256*255, 17 )
setseglen(picture,6)
picture = RGBConvert( picture, "RGB" )
picture = MatrixIpol(picture,10,10,1)
sim = RGBConvert( picture, "similar12", RGB(0,150,0), RGB(0,250,0) )
decide = ( sim > 170 )
decide = RGBConvert( decide, "1.GREY" )
```

See also:

RGB, SetFlag

RgCurveSet

Scope: Report Generator

Sets contents of a curve object in the active document

Declaration:

```
RgCurveSet ( Title, Curve, Zero ) -> Error code
```

Parameter:

Title	Title of curve object in document
Curve	Selects the the curve window to be transferred. See remarks.
Zero	Reserved parameter, to be set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function transfers the contents of the specified curve window to the curve object in the active document. If the curve object already contains a graphic, the graphic is replaced.

Selection of the curve window (2nd Parameter):

With **free curve windows**, the window is identified by its title variable. A title variable can be specified using the Cw*(...) functions of the curve kit or using the FAMOS command "Show". Usually, the title variable is the first variable to be displayed in the window and to appear in the title bar.

For curve windows **embedded in an user-defined dialog**, the name of the dialog element must be specified.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

The size and appearance of the graph in the curve object depends on the curve object settings in the report. These settings affect not only the placement of the graph but also the line width, axis style etc. Furthermore, depending on the curve object specifications, the transfer settings can either be defined in the curve object or taken from the "Clipboard Settings" in the curve window itself.

- This function can only be used when a Report Generator configuration has been loaded.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A waveform is loaded and displayed in a curve window. A report is loaded and a curve object in it is filled with the curve window. Then the report is printed.

```
Channel1 = FileObjRead(file, 1)
TxErr$ = CwLoadCCV(Channel1, "channel1.ccv")
err = RgDocOpen("Table.drb", 0)
IF err = 0
   err = RgCurveSet("Curve1", Channel1, 0)
   IF err = 0
      RgDocPrint(0)
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen , PrSet

RgDocClose

Scope: Report Generator

The Report Generator closes the active document.

Declaration:

```
RgDocClose ( Option ) -> Error code
```

Parameter:

Option	Option parameter
	0 : The active document is closed.
	1 : All documents are closed.
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The active document or all documents are closed. The Report Generator itself remains open.

The documents aren't saved; any changes since last save are discarded.

- Any documents opened using RgDocOpen function should be closed using the RgDocClose function after being processed. This prevents having too many opened documents at a given time.
- If more than one document is open, the active document must be specified using the function RgDocSetActive. This ensures that the correct document is used.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, a document ("Table.drb") is opened, a table object is created and filled with data. The report is printed and then closed.

```
Data = ...
err = RgDocOpen("Table.drb", 0)
IF err = 0
   err = RgTableSetColumn("tab1", 2, 2, Data, 0)
   IF err = 0
      RgDocPrint(0)
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen, RgDocSetActive, RgWindow

RgDocCopy

Scope: Report Generator

A whole page of the active document is copied to the Clipboard.

Declaration:

```
RgDocCopy ( PageNumber ) -> Error code
```

Parameter:

PageNumber	Number of the page to be copied (1..).
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The active document is completely copied to the Clipboard. The Windows-Metafile-Format is used towards this end.

- For purposes of compatibilty with older versions, a 0 (zero) is also permitted as the parameter [PageNumber]. It denotes that the first page is copied.
- If at a given time only one document is open, it is clear which document is the active one. Otherwise, the function RgDocSetActive can be used to ascertain which document would be affected.
- In case of error (return value < 0), the corresponding error text can be inquired using the function RgGetErrorText.

Examples:

A report is opened and completely copied into the Clipboard. Subsequently, "WinWord" is instructed via DDE to paste the contents of the Clipboard to the current Word-Document.

```
err = RgDocOpen("Table.drb", 0)
IF err = 0
   err = RgDocCopy(0)
   IF err = 0
      res = DDESend("WinWord", "System", "[EditPaste]")
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen, RgDocSetActive, RgWindow

RgDocExport

Scope: Report Generator

The Report Generator exports the active document in a graphics format to be selected.

Declaration:

```
RgDocExport ( TxFileName, Format, Resolution, ColorDepth/PDF method, Option ) -> Error code
```

Parameter:

TxFileName	Filename, under which the active document is to be saved.
Format	Graphics format
	0 : Windows-Metafile (*.emf)
	1 : Aldus Placeable Metafile (*.wmf)
	2 : Windows Bitmap (*.bmp)
	3 : JPEG-FileFormat (*.jpg)
	4 : Portable Networks Graphic-FileFormat (*.png)
	5 : PDF-Format (*.pdf)
Resolution	Specifies the resolution to select for Bitmap formats (BMP, PNG, JPG). The unit is 'dpi' (dots per inch). Typical values, for example, are 150dpi or 300dpi. A 0 means that the Report Generator uses the default. Set to 0 for formats other than BMP, JPG, or PNG.
ColorDepth/PDF method	Specifies the color type to generate for the Bitmap formats BMP and PNG respectively the export method for PDF. Set to 0 for any other format. The meaning depends from the value of the [Format] parameter:
	5 : PDF: Export method
	0 The presetting in FAMOS ("Options"/"File-Export"/"PDF") is observed. **1** auto (minimum file size) **2** Bitmap **3** Vektor graphic preferred
	2,4 : Bitmap: Color depth
	0 Report Generator default **1** 16 colors **2** 256 colors **3** 16 million colors
Option	JPEG: Quality (in percent 10%..100%), a 0 means that the Report Generator uses the default. Set to 0 for formats other than JPEG.
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The Report Generator exports the active document under the specified filename and in the specified format.

Multi-page documents: If the document has multiple pages, all pages are exported in PDF-format. With all other formats, the first page is used. If you wish to export another page, use the function [RgDocExportEx].

If an empty text is entered for the filename, the report is filed under its current title.

If the specified filename doesn't have an extension, the default extension for the selected file format is used.

Unless a full path name is provided, the current project folder is used if a project is active. Otherwise, the folder defined by the last call of RgSetDir is used. If this function has not yet been called, the default folder set in FAMOS (folder for report files) is used.

- The region exported is the smallest possible rectangle to contain all Report objects and is thus generally smaller than the size of a page.
- If more than one document is open, the active document must be specified using the function RgDocSetActive. This ensures that the correct

document is saved.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.
- To specify a file name without an extension, the last character in the name must be a period (".").

Examples:

A report is opened. A curve object it contains is updated. The report is saved under a new name in JPEG-format (100% quality, resolution 150dpi) as well as in PDF-format and is then closed again.

```
CwNewWindow(Data, "show")
err = RgDocOpen("cu_mask", 0)
IF err = 0
   err = RgCurveSet("curve1", Data, 0)
   IF err = 0
      err = RgDocExport("cu_0001.jpg", 3, 150, 0, 100)
      err = RgDocExport("cu_0001.pdf", 5, 0, 0, 0)
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen, RgDocSave, RgDocSetActive, RgWindow

RgDocExportEx

Scope: Report Generator

The Report Generator exports the active document's specified page to a selectable graphics format.

Declaration:

```
RgDocExportEx ( TxFileName, PageNumber, Format, Resolution, ColorDepth/PDF method, Option ) -> Error code
```

Parameter:

TxFileName	Filename, under which the document exported is to be saved.
PageNumber	Specifies the number (1..) of the page to be exported. For PDF, 0 (a zero) is permitted, which means the entire document is exported.
Format	Graphics-format
	0 : Windows-metafile (*.emf)
	1 : Aldus Placeable metafile (*.wmf)
	2 : Windows Bitmap (*.bmp)
	3 : JPEG-FileFormat (*.jpg)
	4 : Portable Networks Graphic-FileFormat (*.png)
	5 : PDF-Format (*.pdf)
Resolution	Specifies the selected resolution for the bitmap formats (BMP, PNG, JPG). The unit is 'dpi' (Dots per Inch). Typical values include 150dpi or 300dpi. A 0 means that the Report Generator's default is used. Must be set to 0 (zero) for formats other than BMP, JPG, PNG.
ColorDepth/PDF method	Specifies the color type to generate for the Bitmap formats BMP and PNG respectively the export method for PDF. Set to 0 for any other format. The meaning depends from the value of the [Format] parameter:
	5 : PDF: Export method
	0 The presetting in FAMOS ("Options"/"File-Export"/"PDF") is observed. **1** auto (minimum file size) **2** Bitmap **3** Vektor graphic preferred
	2,4 : Bitmap: Color depth
	0 Report Generator default **1** 16 colors **2** 256 colors **3** 16 million colors
Option	PDF: Specify 0 to create a new file, 1 to append to an existing file. JPEG: Quality (in percent 10%..100%); a 0 means that the Report Generator default is used. Must be set to 0 (zero) for formats other than PDF/JPEG.
Error code	
Error code	Success of the function
	0 : Function executed successfully
	< 0 : Error code

Description:

The Report Generator exports the selected page of the active document under the specified filename and in the specified format.

If an empty text is specified as the filename, the report is saved under its current title.

If the specified filename has no extension, the default extension is used as the file format selected.

Unless a complete pathname is specified, the current project folder will be used if there is an active project. Otherwise, the folder determined by the last call of RgSetDir is used. If this function has not been called yet, the default folder set in FAMOS (folder for report files) is used.

- The region exported is the smallest possible rectangle containing all report objects and is thus usually smaller than the page size.
- As long as only one document is open at any one time, the active document is unambiguous. Otherwise, the function RgDocSetActive must

be used to ascertain that only the desired document is affected.
- At fault condition (return value < 0), the function RgGetErrorText can be used to get the associated error text.
- If a filname has no filename extension, the filename is ended with ".".

Examples:

The report template "template.drb" contains two pages having the same structure, and filled with different curve graphics. The report is saved under a new name both **page-by-page** in JPEG-format (quality: 100%, resolution: 150dpi) and as a whole in PDF-format, and then closed.

```
RgDocOpen("template.drb",0)

; filling 1st page
RgDocSetActivePage(1)
CwLoadCCV(data1, "data1.ccv")
RgCurveSet("curve1", data1, 0)

; filling 2nd page
RgDocSetActivePage(2)
CwLoadCCV(data2, "data2.ccv")
RgCurveSet("curve1", data2, 0)

RgDocExportEx("cu_0001_page1.jpg",1, 3, 150, 0, 100)
RgDocExportEx("cu_0001_page2.jpg",2, 3, 150, 0, 100)
RgDocExport("cu_0001.pdf", 5, 0, 0, 0)
```

See also:

RgDocOpen, RgDocSave, RgDocSetActive, RgWindow

RgDocGetPageCount

Scope: Report Generator

Determines the amount of pages in the current report.

Declaration:

`RgDocGetPageCount ( ) -> PageCount`

Parameter:

PageCount	
PageCount	
	>= 0 : Amount of pages
	< 0 : Error Code.

Description:

- In case of error (return value: < 0), the associated error text can be inquired using the function RgGetErrorText.

Examples:

In the current document, all pages except the first two are deleted.

```
LastPage = RgDocGetPageCount()
WHILE LastPage > 2
   RgDocRemovePage(LastPage)
   LastPage = LastPage-1
END
```

See also:

RgDocInsertPage

RgDocInsertPage

Scope: Report Generator

A new page is added to the current report.

Declaration:

```
RgDocInsertPage ( TxTemplateFile, PageNumber, InsertPosition, Zero ) -> ErrorCode
```

Parameter:

TxTemplateFile	Filename of the report file containing the template for the new page to be inserted. If the parameter is empty, the current document is used.
PageNumber	Determines which page from [TxTemplateFile] to use.
InsertPosition	The new page is inserted at the position stated here. A 0 (zero) means that it is appended to the end.
Zero	Reserved, always set to 0 (zero)
ErrorCode	
ErrorCode	Function outcome
	0 : Function performed successfully
	< 0 : ErrorCode

Description:

This function can be used to add a new page to the current report. The new page can either be a duplicate of a page from the active report, or be imported from a different report file.

The new page is activated, which means that subsequent calls to object-specific functions only affect this page (see also RgDocSetActivePage(..)).

- In case of error (return value: < 0), the associated error text can be inquired using the function RgGetErrorText.

Examples:

The data set "u0" is to be recorded in the form of a 2-column table (x,y). The report "table.drb" serves as the page template; it contains among other things a table object with 50 lines.

```
Length = leng?(u0)
Rows = 50

FirstSample = 1
WHILE FirstSample <= Length
   IF FirstSample = 1
      ; open template
      RgDocOpen("table.drb", 0)
   ELSE
      ; append new page
      RgDocInsertPage("table.drb", 1, 0, 0)
   END

   ; clip y-data for table
   y = CutIndex(u0, FirstSample, FirstSample+Rows-1)
   RgTableSetColumn("table", 2, 1, y, 0)

   ; construct x-data for table
   x = Ramp((FirstSample-1)*xdel?(u0)+ xoff?(u0), xdel?(u0), leng?(y))
   RgTableSetColumn("table", 1, 1, x, 0)

   FirstSample = FirstSample+Rows
END
RgDocSave("u0_table.drb", 0)
RgDocClose(0)
```

See also:

RgDocRemovePage

RgDocNew

Scope: Report Generator

The Report Generator creates a new document.

Declaration:

```
RgDocNew ( Title, Zero ) -> Error code
```

Parameter:

Title	Title for new element
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The Report Generator is instructed to create a new document under the specified title.

All other opened documents remain open and the new document becomes the active document.

If the Report Generator has not yet been started, it is started as an icon. Otherwise, it is considered to be the active (foreground) application.

- In most applications, this function is not particularly appropriate. Creating a complete report document via kit functions is usually not necessary. Instead, we recommend that you create your report "masks" manually and then open them using RgDocOpen.
- Specifying an empty text ("")for the title causes a default title ("Report" + consecutive number) to be used.
- The title serves only to identify the document (function RgDocSetActive). When necessary a complete name and path can be specified when saving the document (function RgDocSave).
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A new report ("temp") is created and a bitmap graphic from the Windows Clipboard is pasted into the document. The report is printed and then closed without being saved.

```
err = RgDocNew("temp", 0)
IF err = 0
   err = DrRdClip("", 5, 10, 0, 0, 2, 0, 0)
   IF err = 0
      RgDocPrint(0)
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen, RgDocClose, RgDocSetActive, RgWindow, DrRdClip

RgDocOpen

Scope: Report Generator

Opens the specified document in the Report Generator.

Declaration:

```
RgDocOpen ( TxFileName, Zero ) -> Error code
```

Parameter:

TxFileName	Name of the file to be loaded.
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function opens an already existing document in the Report Generator. All other opened documents remain open and the new document becomes the active document.

If the Report Generator has not yet been started, it is started as an icon. Otherwise, it is considered to be the active (foreground) application.

If a file extension is not specified, the extension "*.drb" is used.

Unless a complete path is supplied along with the filename, the system searches for the Report file in this sequence of folders:

- Current working folder: This is the folder from which the calling sequence was opened.
- Project folder: When a project is loaded, the system searches for it in the current project folder.
- Default folder for Report files: This folder is defined by the last call of RgSetDir. If this function has not yet been called, The FAMOS default setting for Report files is used.

Any documents opened using RgDocOpen function should be closed using the RgDocClose function after being processed. This prevents having too many opened documents at a given time.

If more than one document is open, the active document must be specified using the function RgDocSetActive. This ensures that the correct document is used.

To specify a file name without an extension, the last character in the name must be a period (".").

In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Multithreading: The functions of the Report kit can be called anywhere and have a global effect. The report loaded here is therefore valid for all execution threads.

Examples:

A report is loaded and the curve object ("curve1") is updated from the specified curve window. If no errors occur, the document is printed and closed.

```
CwNewWindow(Data, "show")
err = RgDocOpen("Curve.drb", 0)
IF   err = 0
   err = RgCurveSet("curve1", Data, 0)
   IF err = 0
      RgDocPrint(0)
   END
   RgDocClose(0)
END
```

See also:

RgDocClose, RgDocSetActive, RgWindow

RgDocPrint

Scope: Report Generator

The Report Generator prints the active document.

Declaration:

```
RgDocPrint ( Selection ) -> Error code
```

Parameter:

Selection	Selection of the region to be printed. Enter either a zero to print the entire document, or the desired page number.
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The Report Generator is instructed to print the active document. The printer set under "File/ Printer Setup" is used.

- Before this function can be called, the Report Generator must be started and at least 1 document must be open. If several documents have been opened, then use the function RgDocSetActive to specify which document is to be printed.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A report is loaded, printed and closed again.

```
err = RgDocOpen("Table.drb", 0)
IF err = 0
   err = RgDocPrint(0)
   IF err < 0
      TxError$=RgGetErrorText(err)
      ok=BoxMessage("Print",TxError$,"!1")
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen, RgDocSetActive

RgDocPrintSetup

Scope: Report Generator

Setting the printer to use for subsequent printout of document.

Declaration:

```
RgDocPrintSetup ( TxPrinter, TxOutput, Zero ) -> Error code
```

Parameter:

TxPrinter	Printer name
TxOutput	Name of output medium. Either empty string for default or a filename.
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function is for setting the printer and output medium to use for a subsequent printout of the document by means of RgDocPrint().

One possible application of this function is to print the document to a file.

The printer name (first parameter) must be entered as it appears in the Report Generator dialog "Printer setup".

Appropriate entires for the output name (second parameter) would be an empty string or a filename. For an empty string, the printer's default connection as defined in the Windows Control Panel, (e.g. local printer port LPT1), is used. If a filename is entered, the printout is diverted to that file.

- In case of error (return value < 0), the corresponding error text can be retrieved using the function RgGetErrorText.

Examples:

A report is loaded and printed toa file with the "Acrobat PDF Writer". The resulting PDF-file can be read using the "Acrobat Reader"

```
err = RgDocOpen("Report1.drb", 0)
IF err = 0
   If = RgDocPrintSetup("Acrobat PDFWriter", "c:\tmp\Report1.pdf", 0)
   IF err = 0
      RgDocPrint(0)
   ELSE
     ; Printer driver not installed or filename not valid
      TxMessage = RgGetErrorText(err)
      ok = BoxMessage("Error!", TxMessage, "!1")
   END
RgDocClose(0)
END
```

See also:

RgDocPrint

RgDocRemovePage

Scope: Report Generator

A page is deleted from the current report.

Declaration:

```
RgDocRemovePage ( PageNumber ) -> ErrorCode
```

Parameter:

PageNumber	States the page number (1..) of the page to be deleted.
ErrorCode	
ErrorCode	Function outcome
	0 : Function performed successfully
	< 0 : Error code

Description:

A page is removed from the current report.

A report must at least have one page, so deleting from a single page report is not possible.

- In case of error (return value: < 0), the associated error text can be inquired using the function RgGetErrorText.

Examples:

In the current document, all pages except for the first two are deleted.

```
LastPage = RgDocGetPageCount()
WHILE LastPage > 2
   RgDocRemovePage(LastPage)
   LastPage = LastPage-1
END
```

The report template "template.drb" contains 10 pages. Depending on the data to be documented, maybe only the first 8 will be needed. In this case, the extra pages are deleted prior to printing.

```
RgDocOpen("template.drb",0)
ExtendedReport = 0

Page = 1
WHILE Page <= 8
   RgDocSetActivePage(Page)
   ; filling the respective page
   ; ...
   ; if applicable: ExtendedReport = 1
   Page = Page+1
END

IF ExtendedReport
   RgDocSetActivePage(9)
   ; ... fill page
   RgDocSetActivePage(10)
   ; ... fill page
ELSE
   ; delete extra pages
   RgDocRemovePage(10)
   RgDocRemovePage(9)
END
RgDocPrint(0)
```

See also:

RgDocInsertPage

RgDocSave

Scope: Report Generator

The Report Generator saves the currently active document.

Declaration:

```
RgDocSave ( TxFileName, Zero ) -> Error code
```

Parameter:

TxFileName	Filename under which the active document is to be saved.
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The Report Generator saves the active document under the specified filename.

If an empty text string ("") for the file name, the document is saved using its current title as the file name.

If a file extension is not specified, the extension "*.drb" is used.

Unless a full path name is provided, the current project folder is used if a project is active. Otherwise, the folder defined by the last call of RgSetDir is used. If this function has not yet been called, the default folder set in FAMOS (folder for report files) is used.

- If more than one document is open, the active document must be specified using the function RgDocSetActive. This ensures that the correct document is saved.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.
- To specify a file name without an extension, the last character in the name must be a period (".").

Examples:

A report is opened and the curve object ("curve1") is updated. The document is saved using a different name and then closed.

```
CwNewWindow(Data, "show")
err = RgDocOpen("cu_mask", 0)
IF err = 0
   err = RgCurveSet("curve1", Data, 0)
   IF err = 0
      err = RgDocSave("cu_0001", 0)
   END
   RgDocClose(0)
END
```

See also:

RgDocOpen, RgDocClose, RgDocSetActive, RgWindow

RgDocSetActive

Scope: Report Generator

The Report Generator activates an open document.

Declaration:

```
RgDocSetActive ( TxTitle, Zero ) -> Error code
```

Parameter:

TxTitle	Title of active report
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function specifies which open document in the Report Generator is to be active. This function is therefore only useful when more than one document is open. Most functions only affect the currently active document.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.
- Multithreading: The functions of the report kit can be called anywhere and have a global effect. The report activated here is therefore valid for all execution threads.

Examples:

If several documents are open, a prompt box asks the user if the most recently opened document is to be printed. As long as the message box is open, the user can manually activate another document. Therefore, make sure that the intended report is active before confirming the print command.

```
err = RgDocOpen("Whatever", 0)
...
err = RgDocOpen("d:\drb\drb1.drb", 0)
IF err = 0
   ok = BoxMessage("Drb1", "Print ?", "?2")
   IF ok = 1
      err = RgDocSetActive("drb1.drb", 0)
      IF err =0
         RgDocPrint(0)
      END
   END
END
```

See also:

RgDocOpen, RgDocClose, RgWindow

RgDocSetActivePage

Scope: Report Generator

Determines which page is the current one. Subsequent object-specific function calls refer to the page selected here.

Declaration:

`RgDocSetActivePage ( PageNumber ) -> ErrorCode`

Parameter:

PageNumber	Sets the page number (1..) of the page to be activated. A 0 (zero) means that object-specific functions are to be used on all pages.
ErrorCode	
ErrorCode	Function outcome
	0 : Function performed successfully
	< 0 : ErrorCode

Description:

The page selected here is used by some Report Generator Kit functions to set the target range of the executed operation. This pertains to all functions, for instance, which change or query object properties. The object addressed is identified by means of the object title. If a particular page is selected by means of RgDocSetActivePage(..), then the system only searches for an object with the specified title on this page.

Without such a call (or following RgDocSetActive(0)), on the other hand, the system searches on all of the document's pages. Functions which query the properties use the first object found. Functions which set properties may apply the operation repeatedly to all objects found. This is practical if all pages have the same structure and one particular element (e.g. date in footer) is to be set to the same content for all pages.

- The function RgDocInsertPage(..) also sets the current page.
- In case of error (return value: < 0), the associated error text can be inquired using the function RgGetErrorText.
- Multithreading:The functions of the report kit can be called anywhere and have a global effect. The page selected here is therefore valid for all execution threads.

Examples:

The report template "template.drb" contains 3 pages. The first is a fixed title page, the other two have the same structure and are to be filled with various curve graphs. Further, all pages have a text box in their legend, which is filled with the current date.

```
RgDocOpen("template.drb",0)

; fill 2nd page
CwLoadCCV(data1, "data1.ccv")
RgDocSetActivePage(2)
RgCurveSet("curve1", data1, 0)

; fill 3rd page
CwLoadCCV(data2, "data2.ccv")
RgDocSetActivePage(3)
RgCurveSet("curve1", data2, 0)

; enter date on every page
RgDocSetActivePage(0)
RgTextSet("Date", TimeToText(TimeSystem?(), 0),0)
```

See also:

RgDocInsertPage

RgGetErrorText

Scope: Report Generator

Determines error text from error code

Declaration:

```
RgGetErrorText ( Error Code ) -> TxErrorText
```

Parameter:

Error Code	Error code (< 0)
TxErrorText	
TxErrorText	Corresponding error message

Description:

This function returns the error message which corresponds to an error code.

Examples:

This code example tries to open a document in the Report Generator. If an error occurs, the message is displayed in a pop-up message box.

```
err = RgDocOpen("d:\drb\drb1.drb", 0)
IF err < 0
   Message$ = RgGetErrorText(err)
   ok = BoxMessage ("Error!", Message$, "!1")
ELSE
   ...
END
```

RgObjDelete

Scope: Report Generator

A report object is deleted from the active report.

Declaration:

```
RgObjDelete ( Title ) -> Error code
```

Parameter:

Title	Object title
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The specified object is deleted.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

The contents of a text variable are to be transferred to the text object "MyText". If this text is empty, the text object is deleted.

```
IF TLeng(TxContents) > 0
   RgTextSet("MyText", TxContents, 0)
ELSE
   RgObjDelete("MyText")
END
```

See also:

RgObjMove

RgObjFind

Scope: Report Generator

Searches a document for an object.

Declaration:

```
RgObjFind ( TxTitle, Zero ) -> Result
```

Parameter:

TxTitle	Title of object searched for
Zero	Reserved, always set to 0
Result	
Result	Function result status
	< 0 : Error code.
	= 0 : Object not found
	> 0 : Object with this title found.
	1 : Text
	2 : Table
	3 : Curve
	10 : Line
	11 : Line (vertical)
	12 : Line (horizontal)
	13 : Polyline
	20 : Rectangle
	21 : Ellipse
	22 : Polygon
	30 : Bitmap
	31 : Metafile
	40 : OLE-object

Description:

This function searches in the active document for an object with the specified title. If it is found, then the object type code is returned.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the system limits the object search (for the specified title) to this particular page. Otherwise, the search covers all pages of the document and the first object found is used.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A Report Generator document is opened and a search for the text object "Date" is performed. If it is found, then it receives the current date.

```
err   = RgDocOpen("report1", 0)
seek = RgObjFind ("Date", 0)
IF seek = 1
   TxDate$ = TimeInText(TimeSystem?(), 1)
   RgTextSet("Date", TxDate$, 0)
END
```

See also:

RgObjGetTitle, RgObjGetType, RgObjGetCount

RgObjGetCount

Scope: Report Generator

This function determines the number of objects in the active document.

Declaration:

```
RgObjGetCount ( Zero ) -> Reserved
```

Parameter:

Zero	Reserved, always set to 0
Reserved	
Reserved	
	>= 0 : Number of object
	< 0 : Error code

Description:

This function determines the number of objects, independent of type, in the active document.

This function is generally used together with RgObjGetTitle or RgObjGetType.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the number of objects of that particular page is determined. Otherwise the number of objects on all pages is returned.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, the font color of all text objects with a title is set to green.

```
err    = RgDocOpen("report1", 0)
Amount = RgObjGetCount(0)
index  = 1
WHILE index <= Amount
   IF RgObjGetType(index) = 1 ; this means text
      TxTitle$ = RgObjGetTitle(index)
      f = RGB(0,255,0)
      err = RgObjSetColor(TxTitle$, 2, f, 0, 0)
   END
   index = index +1
END
```

See also:

RgObjGetTitle, RgObjFind, RgObjGetType

RgObjGetPos

Scope: Report Generator

A report object's position is retrieved.

Declaration:

```
RgObjGetPos ( Title , Position, Option ) -> Position
```

Parameter:

Title	Object title
Position	Positon parameter
	0 : X-position of upper left corner
	1 : Y-position of upper left corner
	2 : Width of the object.
	3 : Height of the object.
Option	Option
	0 : Default.
	1 : Curve objects only: determine the position of the coordinate system.
Position	
Position	Position, distance [in millimeters] from top or left edge of page

Description:

The position of an object in the active document is retrieved.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the system limits the object search (for the specified title) to this particular page. Otherwise, the search covers all pages of the document and the first object found is used.

In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A text object with the title "MyText" is shifted 1cm to the right and its height is doubled.

```
x = RgObjGetPos("MyText", 0, 0)
y = RgObjGetPos("MyText", 1, 0)
RgObjMove("MyText", x+10, y, "", 0)
height = RgObjGetPos("MyText", 3, 0)
RgObjSetSize("MyText", -1, height*2, 0)
```

See also:

RgObjSetSize, RgObjMove

RgObjGetTitle

Scope: Report Generator

Retrieves the title of an object.

Declaration:

```
RgObjGetTitle ( ObjectIndex ) -> TxTitle
```

Parameter:

ObjectIndex	Object index, 1.. number of objects
TxTitle	
TxTitle	Title of specified object

Description:

This function determines the title of an object using its index number. This function is used together with RgObjGetCount(..).

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the index refers to the list of objects of this particular page. Otherwise, the index refers to the list of all objects (on all pages).

Examples:

In a report, the color of all horizontal lines having a title is set to red.

```
err    = RgDocOpen("report1", 0)
Amount = RgObjGetCount(0)
index  = 1
WHILE index <= Amount
   IF RgObjGetType(index) = 12
      TxTitle$ = RgObjGetTitle(index)
      f = RGB(255,0,0)
      err = RgObjSetColor(TxTitle$, 0, f, 0, 0)
   END
   index = index +1
END
```

See also:

RgObjGetCount, RgObjFind, RgObjGetType

RgObjGetType

Scope: Report Generator

Retrieves object type.

Declaration:

```
RgObjGetType ( ObjectIndex ) -> Type
```

Parameter:

ObjectIndex	Object index, 1.. number of objects
Type	
Type	Object type
	< 0 : Error code.
	1 : Text
	2 : Table
	3 : Curve
	10 : Line
	11 : Line (vertical)
	12 : Line (horizontal)
	13 : Polyline
	20 : Rectangle
	21 : Ellipse
	22 : Polygon
	30 : Bitmap
	31 : Metafile
	40 : OLE-object

Description:

Determines the type of a object in the active document using its index number. This function is generally used together with RgObjGetCount.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the index refers to the list of objects of this particular page. Otherwise, the index refers to the list of all objects (on all pages).

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, the date is inserted into the upper left cell of all tables present.

```
TxDate$ = TimeInText(TimeSystem?(), 1)
err      = RgDocOpen("report1", 0)
Amount   = RgObjGetCount (0)
index    = 1
WHILE index <= Amount
   IF RgObjGetType(index) = 2
      TxTitle$ = RgObjGetTitle(index)
      err=RgTableSetCell(TxTitle$,1,1, TxDate$, 0)
   END
   index = index +1
END
```

See also:

RgObjGetCount, RgObjFind, RgObjGetType

RgObjMove

Scope: Report Generator

The position of a report object is changed.

Declaration:

```
RgObjMove ( Title , X-position, Y-position, ReferenceObject, Zero ) -> Error code
```

Parameter:

Title	Object title
X-position	New distance of object from the left edge of page in millimeters.
Y-position	New distance of object from the top edge of page in millimeters.
ReferenceObject	If the object is given, the two coordinates are in reference to this object's position (upper left corner), if one is given.
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The position of an object in the active document is changed.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A text object with the title "MyText" is shifted 1cm to the right and its height is doubled.

```
x = RgObjGetPos("MyText", 0, 0)
y = RgObjGetPos("MyText", 1, 0)
RgObjMove("MyText", x+10, y, "", 0)
height = RgObjGetPos("MyText", 3, 0)
RgObjSetSize("MyText", -1, height*2, 0)
```

See also:

RgObjSetSize, RgObjGetPos

RgObjSetColor

Scope: Report Generator

The color of an object is changed.

Declaration:

```
RgObjSetColor ( Title , Option, RGBColor, Column, Row ) -> Error code
```

Parameter:

Title	Object title
Option	Option
	0 : Frame or line color
	1 : Background color
	2 : Font color
RGBColor	Color
	>=0 : RGB-value of desired color
	-1 : Transparent
Column	Column index or 0
Row	Row index or 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

Changes the color (frame/line color, background color or font color) of an object in the active document.

For table objects, changes apply to the cell specified by the coordinates [Column, Row]; the upper left corner is [1,1]. If both coordinates are 0, the color change applies to the whole table. Otherwise only the specified cell is affected.

For all other object types, Column and Row must both be set to 0.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- The third parameter of this function represents the RGB color value. This color is represented as a mixture of the 3 fundamental colors. To create these values use the FAMOS function RGB.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, the RGB value for red is computed and assigned to a text object (TxTitle$) whenever the maximum value is exceeded.

```
TxTitel$ = ...
Maxim = max(Data)
err = RgTextSet(TxTitle$, TForm(Maxim, "f32"), 0)
IF Maxim > 50
   f = RGB(255,0,0)
   err = RgObjSetColor(TxTitle$, 1, f, 0, 0)
END
```

See also:

RGB

RgObjSetSize

Scope: Report Generator

The size of a report object is changed.

Declaration:

```
RgObjSetSize ( Title , Width, Height, Zero ) -> Error code
```

Parameter:

Title	Object title
Width	New width of object in millimeters, or -1 for no change.
Height	New height of object in millimeters, or -1 for no change.
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

The size of an object in the active document is changed.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

A text object with the title "MyText" is moved 1cm to the right and its height is doubled.

```
x = RgObjGetPos("MyText", 0, 0)
y = RgObjGetPos("MyText", 1, 0)
RgObjMove("MyText", x+10, y, "", 0)
height = RgObjGetPos("MyText", 3, 0)
RgObjSetSize("MyText", -1, height*2, 0)
```

See also:

RgObjMove, RgObjGetPos

RgSetDir

Scope: Report Generator

Sets default folder.

Declaration:

```
RgSetDir ( TxFolder ) -> Error code
```

Parameter:

TxFolder	New default folder
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

Specifies the default directory for loading and saving Report Generator documents. When the file path is not supplied with document functions (e.g. RgDocOpen), this directory is used.

This command doesn't apply to the "compatibility" functions such as PrConfig.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, a document mask is loaded from the "masks" directory and data is transferred to te objects. Once successful, the document is saved in a "results" directory.

```
err = RgSetDir("d:\drb\masks")
IF err = 0
   err = RgDocOpen("report1", 0)
END
err = RgCurveSet(...)
...
err = RgSetDir("d:\drb\result")
err = RgDocSave("report1", 0)
```

See also:

RgDocOpen, RgDocSave

RgTableColumns?

Scope: Report Generator

Retrieves the number of cloumns in a table.

Declaration:

```
RgTableColumns? ( Title ) -> Result
```

Parameter:

Title	Table title
Result	
Result	
	> 0 : Amount of columns
	< 0 : Error code

Description:

This function determines the number of columns in the specified table of the active document.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the system limits the object search (for the specified title) to this particular page. Otherwise, the search covers all pages of the document and the first object found is used.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, events from a multi-shot waveform are assigned column-wise to a table.

```
Amount    = RgTableColumns?("Tab1")
AmountEvn = EventNum?(Data)
IF AmountEvn < Amount
   Amount = AmountEvn
END
i = 1
WHILE i <= Amount
   err = RgTableSetColumn("Tab1",i,2, Data[i],0)
   i = i + 1
END
```

See also:

RgTableRows?, RgTableSetCell, RgTableSetColumn, RgTableSetRow

RgTableGetCellText

Scope: Report Generator

Retrieves contents of a table's cell.

Declaration:

```
RgTableGetCellText ( Title, Column, Row, Zero ) -> TxContents
```

Parameter:

Title	Table title
Column	Column (1..)
Row	Row (1..)
Zero	Reserved, always set to 0
TxContents	
TxContents	Contents of specified table cell

Description:

This function retrieves the contents of a cell in the specified table of the active document.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the system limits the object search (for the specified title) to this particular page. Otherwise, the search covers all pages of the document and the first object found is used.

- The maximum length of the return value is 255 characters.

Examples:

In this code example, the contents of the upper left cell are retrieved. This contains a place-holder "xxx" for the user's name. The place holder is replaced with the user's name is inserted there and the table is updated.

The declarations for the row and column determine the position of the cell described; the upper left is [1,1].

```
Tx$= RgTableGetCellText("Tab1", 1, 1, 0)
Tx$= TReplace(Tx$,"xxx","Heinz Muster")
err = RgTableSetCell("Tab1", 1, 1, Tx$, 0)
```

See also:

RgTableSetColumn, RgTableSetRow, RgTableSetCell

RgTableRows?

Scope: Report Generator

Retrieves number of rows in a table.

Declaration:

```
RgTableRows? ( TxTitle ) -> Result
```

Parameter:

TxTitle	Table title
Result	
Result	
	> 0 : Number of rows
	< 0 : Error code

Description:

This function determines the number of rows in the specified table of the active document.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the system limits the object search (for the specified title) to this particular page. Otherwise, the search covers all pages of the document and the first object found is used.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, segments from a segmented waveform are inserted column-wise to a table.

```
Amount    = RgTableRows?("Tab1")
AmountSeg = leng?(Data) / Segleng?(Data)
IF AmountSeg < Amount
   Amount = AmountSeg
END
i = 1
WHILE i <= Amount
   err = RgTableSetRow ("Tab1",1,i, Data[i],0)
   i = i + 1
END
```

See also:

RgTableColumns?, RgTableSetCell, RgTableSetColumn, RgTableSetRow

RgTableSetCell

Scope: Report Generator

Sets contents of a table cell

Declaration:

```
RgTableSetCell ( TxTitle, Column, Row, Contents, Zero ) -> Error code
```

Parameter:

TxTitle	Table title
Column	Column (1..)
Row	Row (1..)
Contents	Number or text to transfer
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function assigns a text or number to the specified cell in a table of a the active document. With numbers, the numerical format of the cell is used.

The declarations for the row and column determine the position of the cell described; the upper left is [1,1].

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- If a data set with more than 1 value is specified for <XXContents>, then only the first value is used.
- Numerical values assume the cell format (in terms of decimal places, unit prefix etc.) that has been set for this table. These attributes can be changed in the "Properties" dialog for table objects, on the "Number" tab.
- If you do not wish to have the number converted to this preset format, you can convert the number to a text using TForm and then insert this into the cell.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

The second column of a table is filled with a waveform which was just computed. The first row contains variable names, the second row the waveform's maximum value in a fixed format. The actual data values begin in the third row, and these use the numerical format as defined in the table.

```
Channel1= ...
err = RgTableSetCell("Tab1", 2, 1, "Channel1",0)
TxMax$ = TForm(Max(Channel1), "f32")
err = RgTableSetCell("Tab1", 2, 2, TxMax$, 0)
err = RgTableSetColumn("Tab1", 2, 3, Channel1, 0)
```

See also:

RgTableSetColumn, RgTableSetRow, RgTableGetCellText

RgTableSetColumn

Scope: Report Generator

Sets the contents of a table column.

Declaration:

```
RgTableSetColumn ( TxTitle, Column, Row, Contents, Zero ) -> Error code
```

Parameter:

TxTitle	Table column
Column	Column (1..)
Row	Row (1..)
Contents	Data to be transferred
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function assigns a data set to a column using the numerical format of the table. The first number is inserted at the specified position, all remaining data below.

The declarations for the row and column determine the position of the cell described; the upper left is [1,1].

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- Data are transferred until either the end of the data set or the last table row has been reached.
- Numerical values use assume the cell format (decimal places, unit prefix etc.). These attributes cam be changed in the "Properties" dialog under "Number".
- To transfer text to a table cell, use the function RgTableSetCell.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, events from a multi-shot waveform are inserted column-wise to a table.

```
Amount    = RgTableColumns?("Tab1")
AmountEvn = EventNum?(Data)
IF AmountEvn < Amount
   Amount = AmountEvn
END
i = 1
WHILE i <= Amount
   err = RgTableSetColumn("Tab1",i,2, Data[i],0)
   i = i + 1
END
```

See also:

RgTableSetRow, RgTableSetCell

RgTableSetRow

Scope: Report Generator

Sets the contents of a row in a table.

Declaration:

```
RgTableSetRow ( TxTitle, Column, Row, Data, Zero ) -> Error code
```

Parameter:

TxTitle	Table title
Column	Column (1..)
Row	Row (1..)
Data	Data to transfer
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function assigns a data set to a column using the numerical format of the table. The first number is inserted at the specified position, all remaining data to the right thereof.

The declarations for the row and column determine the position of the cell described; the upper left is [1,1].

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- Data are transferred until either the end of the data set or the last table row has been reached.
- Numerical values use assume the cell format (decimal places, unit prefix etc.). These attributes cam be changed in the "Properties" dialog under "Number"
- To transfer text to a table cell, use the function RgTableSetCell.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, segments from a segmented waveform are inserted column-wise to a table.

```
Amount    = RgTableRows?("Tab1")
AmountSeg = leng?(Data) / Segleng?(Data)
IF AmountSeg < Amount
   Amount = AmountSeg
END
i = 1
WHILE i <= Amount
   err = RgTableSetRow ("Tab1",1,i, Data[i],0)
   i = i + 1
END
```

See also:

RgTableSetColumn, RgTableSetCell

RgTextGet

Scope: Report Generator

Retrieves the contents of a text object.

Declaration:

```
RgTextGet ( TxTitle, Zero ) -> TxContents
```

Parameter:

TxTitle	Object title
Zero	Reserved, always set to 0
TxContents	
TxContents	Object contents

Description:

This function retrieves the contents of a specified text object in the active document.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, the system limits the object search (for the specified title) to this particular page. Otherwise, the search covers all pages of the document and the first object found is used.

- The maximum length of the return value is 255 characters.

Examples:

In this code example, the contents of a text object are retrieved. This contains a place-holder "xxx" for the user's name. The place holder is replaced with the user's name is inserted there and the table is updated.

```
Tx$= RgTextGet("Signature", 0)
Tx$= TReplace(Tx$,"xxx","Heinz Muster")
err = RgTextSet("Signature", Tx$, 0)
```

See also:

RgTextSetData, RgTextSet

RgTextSet

Scope: Report Generator

Sets contents of a text object.

Declaration:

```
RgTextSet ( TxTitle, TxContent, Zero ) -> Error code
```

Parameter:

TxTitle	Object title
TxContent	New object contents
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function inserts text into a text object. Any text which was there before is completely overwritten.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, the contents of a text object are retrieved. This contains a place-holder "xxx" for the user's name. The place holder is replaced with the user's name is inserted there and the table is updated.

```
Tx$= RgTextGet("Signature", 0)
Tx$= TReplace(Tx$,"xxx","Heinz Muster")
err = RgTextSet("Signature", Tx$, 0)
```

See also:

RgTextSetData, RgTextGet

RgTextSetData

Scope: Report Generator

Fills place holder in text object.

Declaration:

```
RgTextSetData ( TxTitle, Data, Zero ) -> Error code
```

Parameter:

TxTitle	Object title
Data	Data to be transferred (waveform, single value or text)
Zero	Reserved, always set to 0
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function transfers data to place-holders in a text object.

The following place-holders are defined for text objects:

#d

Current date in WINDOWS format

#z

Current time in WINDOWS format

#u

Units of a single value or y-units of a data set

#e?

Floating-point number, number of decimal placesn

#f?.?

Floating-point number, number of decimal places before and after decimal

#s

Text

Note: "?" stands for digits.

Multi-page reports: A current page previously selected using RgDocSetActivePage(..) or RgDocInsertPage(..) is affected. If a page was previously selected accordingly, then the system searches only on this page for an object having the specified title. Otherwise, the object search is carried out on all of the document's pages and, if applicable, the operation is carried out multiple times, for every appropriate object.

Note:

The place-holder concept for text objects was much more important with earlier versions of the Report Generator (Report Generator). This was mostly due to the lack of powerful remote control functions and table objects.

If you are integrating the reports in new programs or sequences, we generally recommend the usage of "masks", i.e. formatting a document directly in the Report Generator using "dummy" text objects. The contents of these can be replaced with the appropriate text using the function RgTextSet. Alternatively, the contents of a text object can be retrieved using the function RgTextGet, edited and then replaced using RgTextSet.

Table objects should always be used to display data from a waveform in a tabular form.

- For a detailed description of "place-holders", please refer to the "Report Generator" chapterin the text object section.
- The place holders "#d" (current date) and "#z" (current time) are replaced the first time the text object is accessed via the function "RgTextSetData"..
- If a complete waveform is to be transferred to a text object with real numbered place-holders, then a place-holder is replaced for each sample (y-value). In this way, a complete table (or column/ row) can be replaced with a single command.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Once a place-holder has been replaced, it cannot be retrieved. If you want to create another report using the mask document, you have to open it again using RgDocOpen!

Examples:

First, a mask document is loaded.

```
TxError$ = RgDocOpen ("report", 0)
```

2 text objects are defined in this mask:

Title

Contents

Name

#s

Number

x: #e2 and #f2.1

The following lines are executed:

```
RgTextSetData("Name", "Herbert", 0)
RgTextSetData("Number", 27.5, 0)
RgTextSetData("Number", 28.9, 0)
```

As a result, the text objects now contain the following:

Title

Contents

Name

Herbert

Number

x: 2.75E+01 and 28.9

See also:

RgTextSet, RgTextGet, RgCurveSet, RgTableSetCell, RgTableSetRow, RgTableSetColumn

RgWindow

Scope: Report Generator

Starts or closes the Report Generator

Declaration:

```
RgWindow ( Task ) -> Error code
```

Parameter:

Task	Task
	1 : Start Report Generator as a normal window
	2 : Start Report Generator as an icon
	3 : Close Report Generator
Error code	
Error code	Function result status
	0 : Function executed successfully
	< 0 : Error code

Description:

This function enables you to launch, close or minimize/maximize the Report Generator.

- Any changes not saved are lost when the Report Generator is closed.
- In case of error (return value < 0), the function RgGetErrorText can be used to retrieve the corresponding error message.

Examples:

In this code example, a mask document is opened. If the Report Generator is not already open, it is started (minimized). Various objects are updated in the document and the finished report is displayed in normal size so that the user can decide whether or not to print the document as it is. Finally, the Report Generator is closed.

```
err = RgDocOpen("d:\usr\report1", 0)
IF err = 0
   ...
   ;; various updatesn
   ...
   err = RgWindow(1)
   ok =  BoxMessage ("Question", "Print?" "?2")
   IF ok
      RgDocPrint(0)
   END
   RgWindow(3)
END
```

See also:

RgDocOpen, RgDocClose

RMS

The RMS value (root-mean-square) of a data set's numerical values is determined.

Declaration:

```
RMS ( Data ) -> SvRMS
```

Parameter:

Data	Data whose RMS-value is to be determined [ND].
SvRMS	
SvRMS	RMS-value of the data set

Description:

The root of the mean value of the square of all values in a data set are calculated. In electrical engineering, this value is referred to as the true RMS. The RMS is defined as the root of the mean square; the mean square is the sum of the squares of all values, divided by the number of values.

The true RMS never has a value less than zero. Unlike arithmetic mean, positive and negative values do not cancel each other, because their squares are always positive.

Examples:

The RMS-value of a voltage plot is calculated:

```
RMS_voltage = RMS(voltage)
```

See also:

Mean, StDev, MvRMS, ExpoRMS, Stat

Rosette

Available in: Professional Edition and above

With rosettes, the principal strain and the principal stress are calculated from the strains measured.

Declaration:

```
Rosette ( GridA, GridB, GridC, Rosette shape, Calculation [, Poisson's ratio] [, Modulus of elasticity] [,
Minimal strain] ) -> Result
```

Parameter:

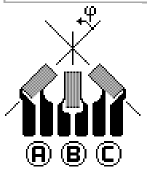
GridA	Grid A
GridB	Grid B
GridC	Grid C
Rosette shape	Which shape of rosette?
	"0-45-90" : 0-45-90 degree rectangular rosette
	"0-60-120" : 0-60-120 degree delta rosette
Calculation	What is calculated?
	"eps1" : Principal strain 1, eps1
	"eps2" : Principal strain 2, eps2
	"sig1" : Principal stress 1, sig1
	"sig2" : Principal stress 2, sig2
	"sigV" : Comparison stress as per Mises, sigV
	"phi" : Orientation angle phi, principal direction 1 measured against Grid A
Poisson's ratio	Transverse contraction number, transverse strain number, Poisson number, e.g. for meatals in the range 0.3 to 0.4 (optional , Default value: 0)
Modulus of elasticity	Modulus of elasticity in GPa, e.g. for steel: 210 GPa (optional , Default value: 0)
Minimal strain	Limit for the strains from which the angle is determined by calculation of arctan. If the values are <= this limit, the angle is set to zero. (optional , Default value: 0)
Result	
Result	Result

Description:

Numbering of the measurement grid: in order to obtain correct values in measuring with 3-element rosettes, the measurement grids must be numbered in a very specific way.

It doesn't matter whether the rosette is square, or a rectangular or a round rosette. Only the angle between the grids is important and defines the calculation.

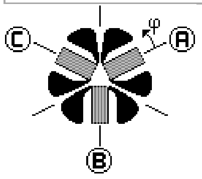
0-45-90 degree rectangular rosette



Principal strain

$$\varepsilon_{1,2} = \frac{\varepsilon_a + \varepsilon_c}{2} \pm \frac{1}{\sqrt{2}}\sqrt{(\varepsilon_a - \varepsilon_b)^2 + (\varepsilon_c - \varepsilon_b)^2}$$

0-60-120 degree delta rosette



Principal strain

$$\varepsilon_{1,2} = \frac{\varepsilon_a + \varepsilon_b + \varepsilon_c}{3} \pm \sqrt{\left(\frac{2\varepsilon_a - \varepsilon_b - \varepsilon_c}{3}\right)^2 + \frac{1}{3}(\varepsilon_b - \varepsilon_c)^2}$$

Principal stress

$$\sigma_1 = \frac{E}{1-\nu^2}(\varepsilon_1 + \nu\varepsilon_2) \qquad \sigma_2 = \frac{E}{1-\nu^2}(\varepsilon_2 + \nu\varepsilon_1)$$

Comparison stress as per Mises

$$\sigma_v^2 = \sigma_1^2 + \sigma_2^2 - \sigma_1\sigma_2$$

Units

The input channels for the grids A, B and C are generally stated in the unit micrometer/meter or microstrain or 1e-6. The minimal strain for the calculation of the angle must be stated in the same unit. The principle strains eps1 and eps2 are then also determined in te same unit.

The principal stress and the comparison stress according to Mises are determined in N/mm^2. The necessary precondition is that the input channels be expressed in micrometers/meters. Also, the modulus of elasticity must be stated in GPa. If they are expressed in any other unit, the user must subsequently correct he result's unit.

The angle is determined in degrees.

For the principal strains determined, the expression eps1 >= eps2 applies, i.e. the principal strain 1 is the greater of the two principal strains.

The Plus-sign in the formula calculates eps1, and the Minus-sign calculates eps2.

The Plus-sign in the formula for the principal strain causes absolute values to increase when the center point of the Mohr circles is positive, and the Minus-sign causes the same when the center point is negative.

Angle calculation with minimal strain

The angle phi found is counted from Grid A going counterclockwise. It is the principal direction 1. It is expressed in the range -90 to +90 degrees.

Counterclockwise angle measurements are thus expressed with a positive sign, and with negative sign when clockwise.

Principal direction 2 always points perpendicular to Principal direction 1, and can be determined subsequently by adding 90 degrees.

Angle calculation with minimum strain: To determine an angle, the arctan of a fraction is found. If this fraction's numerator and denominator are both very small, the angle will tend to be very inexact. If the sum of the absolute values of the numerator and denominator are less than or equal to the specified "Minimal strain", the angle is set to zero for that reason.

In illustration, one could say that for a strain of zero, no angle can be determined, naturally. Thus, when there is signal noise around zero, this prevents the angle from taking on random values, but instead being stated as zero.

When in doubt, the parameter "Minimal strain" is not stated and it receives its default value.

General notes

All 3 input channels (GridA, GridB, GridC) must have the same time base and the same structure with regard to length, segments and events.

The function expects equidistant input data. If the data are XY-data, the function can be applied to the .Y-component.

The Poisson ratio and modulus of elasticity are only needed for calculating the stresses and otherwise can be set to 0.

Examples:

```
eps1 = Rosette ( epsA, epsB, epsC, "0-45-90", "eps1")
eps2 = Rosette ( epsA, epsB, epsC, "0-45-90", "eps2")
phi = Rosette ( epsA, epsB, epsC, "0-45-90", "phi", 0, 0, 0.001)
poisson = 0.3
emod = 210 ; GPa
sig1 = Rosette ( epsA, epsB, epsC, "0-45-90", "sig1", poisson, emod)
sig2 = Rosette ( epsA, epsB, epsC, "0-45-90", "sig2", poisson, emod)
sigV = Rosette ( epsA, epsB, epsC, "0-45-90", "sigV", poisson, emod)
```

Round

A data set's Y-values are rounded to the specified precision.

Declaration:

```
Round ( Data, SvFactor ) -> Rounded
```

Parameter:

Data	Data set to be rounded. Allowed types: [ND],[XY].
SvFactor	The value are rounded to integer multiples of this value.
Rounded	
Rounded	The rounded data set

Description:

The Y-values are rounded in such a way that the results only contain integer multiples of the specified factor. With a factor of 0.01, for instance, you round all values off to two decimal places; with a factor of 0.25, you obtain values such as 0, 0.25, 0.5, 0.75...

If a value is located exactly between two rounding values, the system rounds to the higher absolute value.

```
Result = Round(0.5, 1)  ; Result:  1
Result = Round(-0.5, 1)  ; Result: -1
```

Since real numbers can usually not be represented exactly, you may obtain unexpected results if the value displayed does not exactly match the value saved in the system. For example, the number 0.3 may be saved internally as 0.299999..., but the result of Round(0.3, 0.2) would be 0.2 instead of the expected 0.4 (from rounding up) .

Examples:

Rounding to whole numbers:

```
Result = Round(2.156, 1)  ; Result: 2.0
```

Rounding to 2 decimal digits:

```
Result = Round(2.156, 0.01)  ; Result: 2.16
```

Rounding to multiples of 0.25:

```
Result = Round(2.156, 0.25)  ; Result: 2.25
```

Rounding to multiples of 100:

```
Result = Round(51, 100)  ; Result: 100
```

See also:

Floor

RSamp

Prototype-resampling; sampling of a data set at the sampling times of a reference, with linear interpolation

Declaration:

```
RSamp ( Data, Reference ) -> Result
```

Parameter:

Data	Data set to be re-sampled. Allowed data types: [NW],[XY]
Reference	Reference, prototype for re-sampling
Result	
Result	Result of re-sampling.

Description:

The first data set transferred is imagined to be extended in both x-directions to the length of the reference channel, with its begin and end values continued beyond the original range. For instance, for all x-coordinates less than the x-offset, the beginning value of the data set will be assigned as the y-coordinate. A constant extrapolation is performed. The specified data set is imagined to be linearly interpolated within its x-range.

This function is thus defined for all x-coordinates and can be sampled at the times (x-coordinates) dictated by the reference channel (second parameter), producing two data sets with completely synchronous sampling. The generated data set has the same sampling time and the same x-offset as the reference channel.

With XY-data, the x-track must be monotonically increasing.

RSamp is particularly convenient when two data sets with different sampling times are to be compared or calculated in some way. The addition of data sets only makes sense when the data sets have the same sampling rates (see the + (Addition) function).

- The x-units of both parameters should be the same.
- The RSamp function is useful in the x-range common to both parameters. Use the Cut function to limit the x-range to that of the reference channel.
- When the RSamp function reduces the amount of data, the aliasing effects common to all sampling functions may occur. Use a smoothing function before sampling (Smo).
- To combine two data sets with different sampling times, the RSamp function can in principle be used on each of the data sets. In view of accuracy and aliasing effects, it is generally better to resample the data set with the slower sampling rate. Select only the faster-sampled data set if the amount of data would become too large. Use a smoothing function first to prevent any aliasing effects.
- If the sampling times of the data sets are integer multiples of each other, the functions Red, Ipol or Lip can be used instead of the RSamp function.

If the linear interpolation upon which the RSamp function is based is insufficient (e.g. too angular), the data set to be resampled should be processed with spline interpolation (Ipol function) first, for example:

```
NDbetter = RSamp(IPol(NDdata, 10), NDref)
```

A factor of 10 is appropriate when the sampling of the reference channel is approximately 10 times as high.

Examples:

The instantaneous power of the 50Hz power source is to be plotted. The voltage was sampled for 1 second with a sampling rate of 0.2ms. The current has a much higher frequency because a current rectifier is connected; therefore the current was sampled at a rate of 0.005 ms over the same period of time. The instantaneous power is the product of the current and voltage.

```
NDpower = NDcurrent * RSamp(NDvoltage, NDcurrent)
```

A data set with an unusual sampling rate of 0.15 s is to be resampled 100 times for a duration of 2 s, at a sampling rate of 0.1 s.

```
NDnew = RSamp(NDdata, Ramp(2, 0.1, 100))
```

See also:

RSamp0, RSampEx, RedEx, Red, Red2, IPol, Lip

RSamp0

Prototype re-sampling, sampling of a waveform at a reference waveform's sampling times, with constant interpolation.

Declaration:

```
RSamp0 ( Data, Reference ) -> Result
```

Parameter:

Data	Data set to be re-sampled. Allowed data types: [NW],[XY]
Reference	Reference, prototype for re-sampling
Result	
Result	Result of re-sampling.

Description:

The waveform named as the first parameter is re-sampled at the points in time (x-coordinates) determined by the reference waveform (2nd parameter).

The result value associated with a particular reference channel's x-coordinate is considered to be the input data set's value at the x-coordinate closest to the reference channel's coordinate.

Thus, **in contrast to the function RSamp, interpolation is not linear**, the result data set only consists of values which also exist in the input data set.

With XY-data, the x-track must be monotonically increasing.

The function RSamp0() is best used when the input signal consists of pre-defined discrete values. This is the case, for instance, with digital data, or also with measurement data which can only have the integer format (e.g. the currently operable gear in a transmission system).

Re-sampling provides for synchronized records of the two waveforms. The data set generated has the same sample times and x-offset as the reference data set.

The result takes the same data format as the sampled waveform.

Outside of its x-range, the first waveform's values are considered to be its actual initial value, or respectively, end value (the boundary values are repeated outwards). For instance, for all x-coordinates less than the x-offset, the waveform's initial value is taken as the y-coordinate (constant extrapolation).

Examples:

The speed <velocity> and gear (<gear>, taking only the values 1 - 5) of a vehicle are measured. The task is to find all points in time at which the speed is less than 50 km/h while in 3rd gear. Since the two channels were sampled at different times, they must be assimilated to the same sampling time before they can be correlated with each other.

```
gear_resampled = RSamp0(gear, velocity)
result = (gear_resampled = 3) AND (velocity < 50)
```

See also:

RSamp, RSampEx, Red, RedEx, Red2

RSampEx

Prototype-resampling, a waveform is resampled at the sampling times of a reference data set, with adjustable interpolation..

Declaration:

```
RSampEx ( Data, Reference, SvInterpolation, SvOption ) -> Result
```

Parameter:

Data	Waveform to be resampled. Permitted waveforms are of the types NW (sampled equidistantly) and XY..
Reference	Reference, prototype for re-sampling
SvInterpolation	Interpolation type
	0 : Linear. The input waveform is interpolated linearly to find the values at the reference points. This is analogous to the behavior of the function RSamp()..
	1 : Constant, preceding value. The input waveform is interpolated at a constant level, i.e. each value remains valid until the next value is determined. Thus, the return value for any particular x-coordinate of the reference is such value of the input waveform whose x-coordinate is immediately PRIOR to the reference coordinate.
	2 : Constant, closest value. The return value for any particular x-coordinate of the reference is such value of the input waveform whose x-coordinate is CLOSEST to the reference coordinate. This is analogous to the behavior of the function RSamp0.
SvOption	Option parameter
	0 : The trigger time of the two waveforms is not taken into account. The result has the same trigger time as the input waveform.
	1 : The trigger times of the two waveforms are taken into account. The result has the same trigger time as the prototype.
Result	
Result	Result of re-sampling.

Description:

The waveform supplied as the first parameter is resampled at the times (x-coordinates) determined by the reference waveform (2nd parameter). Re-sampling causes both waveforms to have synchronized sampling values. The waveform generated has the same sampling time and the same x-offset as the reference waveform.

Linear interpolation is generally used with continuous input signals.

The **constant** interpolation types are most effectively used when the input signal consists of pre-defined, discrete values. This is the case, for instance, with digital data, or for measured data which by nature can only take integer values (for example, the current gear which is engaged in a transmission). The resulting data set consists only of values which also exist in the input data set, and has te same data format.

Outside of its x-range, the first waveform's values are considered to be its actual initial value, or respectively, end value (the boundary values are repeated outwards). For instance, for all x-coordinates less than the x-offset, the waveform's initial value is taken as the y-coordinate (constant extrapolation).

Linear interpolation with digital input data is not posible; the interpolation parameter is automatically corrected to the value 1 (constant interpolation).

Examples:

The speed <velocity> and gear (<gear>, taking only the values 1 - 5) of a vehicle are measured. The task is to find all points in time at which the speed is less than 50 km/h while in 3rd gear. Since the two channels were sampled at different times, they must be assimilated to the same sampling time before they can be correlated with each other.

```
gear_resampled = RSampEx(gear, velocity, 2, 0)
result = (gear_resampled = 3) AND (velocity < 50)
```

See also:

RSamp0, RSamp, Red, RedEx, Red2

SamplesGate

Available in: Professional Edition and above

Includes all values in the result which are selected by a controlling data set.

Declaration:

```
SamplesGate ( Dataset, Control ) -> Result
```

Parameter:

Dataset	Dataset
Control	Control
Result	
Result	Result

Description:

This function adopts a measured value from the input data set if the corresponding value in the control data set is nonzero. The measured value is discarded if the corresponding value of the reference data set is zero.

Both parameters have the same structure regarding length and events.

The data set may not have any segments.

The data set may be equidistant, XY, or complex.

The x-offset (x0) remains intact.

Examples:

Determine all measured values for which the gear is 4

```
Data4 = SamplesGate ( Data, Gear = 4 )
```

Delete all values for which a marker is set (Marked[i] is 1).

```
Remain = SamplesGate ( Data, not ( Marked ) )
```

See also:

RemoveSamples, ValueIndex, CutIndex

SAVE

A variable is saved to a file. The imc/FAMOS file format is used.

Declaration:

```
SAVE VariableName Filename
```

Parameter:

VariableName	Variable to be saved
Filename	Filename (or also complete path), under which the variable is to be saved.

Description

This command is obsolete. Instead of it you can use the more powerful and convenient function FileSave().

The variable specified as the parameter is saved. If no file name is specified, the variable is stored under its name in the directory currently selected for saving variables.

If a file name is specified, the variable is saved under this name. When a complete path is specified, this path is used; otherwise the path currently set for saving files is used.

The filename may also specified to contain quotation marks. This can be necessary, if, for instance, the path contains spaces.

To save files in the imc/FAMOS format, you can also use the more powerful functionsFileSave() or FileOpenDSF(). With these functions, you can also, for example, save multiple data sets together in one file.

Examples:

```
data = Int(data)
SAVE data
```

The variable data is integrated and saved to a file.

```
Hist = Histo(Data1, 10, 6)
SAVE Hist c:\data\histogr\hist001
Hist = Histo(Data2, 10, 6)
SAVE Hist c:\data\histogr\hist002
```

A histogram is calculated for each of the waveforms Data1 and Data2 and saved in the specified directory under the names 'hist001.dat' and 'hist002.dat'.

```
Hist:N001 = Histo(Data1, 10, 6)
Hist:N002 = Histo(Data2, 10, 6)
SAVE Hist c:\data\histogr\hist002
```

A data group consisting of 2 channels is created and stored under the specified file name.

```
SAVE Daten1 "c:\imc\My Data Files\result.dat"
```

The filename contains spaces and therefore it must be written inside of quotation marks.

See also:

FileSave, FileOpenDSF, SDIR, ASCSAVE, LOAD

SavitzkyGolay

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Savitzky-Golay filter for smoothing data sets

Declaration:

```
SavitzkyGolay ( InputData, Order, Left, Right, Edge ) -> Result
```

Parameter:

InputData	Input data to be smoothed
Order	Order/degree (0..9) of the polynomial used
Left	Number of points at left, >= 0
Right	Number of points at right, s>= 0
Edge	Influencing of the signal transients and decay at the edges
	0 : Edge regions are extended at a constant level from the last valid value.
	1 : Edge regions are completely filled with zeroes.
	2 : Edge regions result from filtering, and it is assumed that the data sets consists of zeroes outside of these regions.
	3 : Edge regions result from filtering, and it is assumed that the data sets is extended at constant level ouside of these regions.
	4 : Edge regions are truncated. The length is shortened, the x-offset is shifted.
Result	
Result	Filtered data

Description:

The Savitzky-Golay filter is a digital filter whose coefficients are determined by polynomial regression.

Although it was used mainly in spectroscopy in the past, the filter is useful not only for smoothing spectra, but also any arbitrary signals.

The width of the resulting digital filter is [left points + right points + 1].

The points to the left give weighting to past values. The points to the right give weighting to future values. If the count is > 0, the filter is not causal.

When the width is large, the computational demands increase.

The points on the left determine how wide the transients at the start are. The points on the right determine how wide the decay at the end is.

There is only a smoothing effect when the order < [left points + right points].

Available from imc FAMOS 7.1 onwards

Examples:

```
Smoothed = SavitzkyGolay( vibration, 3, 10, 10, 0 )
```

Scale

Scales a data set by means of linear transformation to a specified new value range.

Declaration:

```
Scale ( Data, SvTop, SvLow ) -> Result
```

Parameter:

Data	Data set to be scaled; allowed types: [ND],[XY]
SvTop	Upper boundary
SvLow	Lower boundary
Result	
Result	Rescaled data set. Y-value range lies between [SvBottom] and [SvTop]

Description:

The y-value range of a data set is rescaled to a predefined range. The new range is defined by an upper and lower limit (maximum and minimum). The second and third parameters define the new value range. All numerical values of the data set are transferred linearly from the old to the new value range. This function allows convenient normalizing to a suitable value range.

- The first parameter may be structured (events/segments).
- The unit of the data set remains unchanged. To normalize to a unitless number, divide by 1 times the unit.
- The function Scale searches for the minimum and maximum of the data set and calculates the transformation formula from these values and the desired range.

Examples:

```
NDnorm = Scale(NDdata, 0, 100) * 1 '%'
```

A data set with the numerical values 0.7, 0.8, 0.9, 0.8 should be set to a new range of 0 ... 100%: the generated data set then contains the values 0%, 50%, 100%, 50%.

See also:

Clip, Min, Max

SDIR

Sets folder for saving files

Declaration:

```
SDIR Folder
```

Parameter:

Folder	Complete pathname of the desired folder

Description

Instead of the command SDIR, the function SetOption() should be used in ewly created sequences.

The folder for saving files is given a new setting.

Once this command has been executed, this folder is used for saving files. This folder is used by the commands SAVE and ASCSAVE, if no full pathname is specified for the file.

The folder name may also be expressedd in quotation marks. This is obligatory when the name contains spaces.

This command can also be called without any parameters (so without any specified fodler). Then the folder set under "Options/Folders" is again used as the default.

The folder elected here remains valid until:

- the command SDIR is called again
- the function SetOption("Dir.DataFiles",...) is called
- a file is saved in a different folder by means of the dialog box "Save" in the menu "File"
- a new folder is designated for saving files by means of the dilog under "Options"/"Folders"

Multithreading: The command has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
FAMOS
LOAD data
data = 2 * data + 3
SAVE data
SDIR c:\test
SAVE data
```

The data set DATA is loaded and changed, then saved in the currently designated folder; then saved once again under C:\TEST\DATA.DAT.

```
SDIR "c:\My tests on 12/1/98"
```

The pathname contains spaces and must therefore be written in quotation marks.

See also:

SetOption, SAVE, ASCSAVE, LDIR, FileSave, FileOpenDSF

SDOF_Response

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

The SDOF (single degree of freedom system) is excited by the acceleration plot provided. The response is calulated.

Declaration:

`SDOF_Response ( Acceleration, Frequency, Attenuation, Model, AppendSamples ) -> Response`

Parameter:

Acceleration	The course of the acceleration in time, the time scaled in seconds.
Frequency	Natural frequency of the undamped system, in "Hz"
Attenuation	0 <= Attenuation< 0.9. (attenuation ratio) The system's relative attenuation. Typically 0.05 or 0.01. 0.0 is an undamped system.
Model	model of calculation
	0 : absolute acceleration model
	1 : relative displacement model
AppendSamples	The result will be longer than the input data by the number of samples specified, >= 0
Response	
Response	Response

Description:

The function determines the response of the system comprising a spring, a mass and a damper having a given natural frequency and damping.

The base (support) of the SDOF is excited by a measured acceleration. That acceleration is measured in absolute terms (with respect to a fixed support). In consequense of the acceleration the SDOF will move.

If relative displacement is calculated, the displacement is the (zero initialised) distance from the base. The acceleration of the SDOF, however, is calculated as an absolute value with respect to ground, not with repect to the base.

absolute acceleration model. The result is absolute acceleration values, where the acceleration is scaled exactly as in the acceleration specified as the parameter, typically in "g" (gravitational acceleration) or "m/s^2".

relative displacement model. The result is the relative deflection (path) and comes with the unit "m" (Meter). This requires that the acceleration-to-time curve be scaled in "m/s^2", not in "g".

The function assumes the system to be in steady state initially, which means displacement=0 and velocity=0.

The result is plotted over the natural frequency.

Then an appropriate length should be chosen.

Relationship with shock response spectrum (SRS): The SRS is calculated by calling SDOF_RResponse() for each desired frequency, then determining the mininum/maximum of the result.

The result has the same sampling frequency as the input. If minimum or maximum values are to be determined subsequently, or (rainflow) class-counting is to be performed, the following applies:

If the natural frequency is small compared with the sampling frequency, the result may lose accuracy, because sampling does not occur exactly at the extremum. E.g the error can be up to 1% if the ratio of frequencies is 23. Alternatively, the input can be interpolated beforehand or afterward.

If AppendSamples = 0, then the response is equal to the one calculated for the primary SRS (initial SRS).

Particularly with low natural frequencies, it sometimes occurs that within the (brief) duration of the plot of acceleration over time, the system response does begin with an oscillation, but neither reaches the maximum or the minimum before the acceleration ends.

If later on only maximum values are determined, then it is sufficient to append a length equal to only one period of the natural frequency.

If (rainflow) class-counting is to be performed subsequently, as many periods must be appended as it takes for the response to decay to a negligible value. For damping of 0.05, that can be about 30 periods of the natural frequency.

The input data can have events and segments.

(algebraic) signs, directions

The relative displacement z is the distance between the mass and the moving base. The moving base itself moves at the absolute acceleration specified as the parameter. All variables are measured in the same direction; positive meaning away from the moving base.

Both the mass and the moving base are considerd initially at rest. Next, constant positive acceleration is to be applied to the moving base:

At the first moment, the mass is not yet moving; instead the spring becomes compressed. Gradually the mass goes into motion. Initially it moves in the same direction as the moving base. Its absolute acceleration is also positive.

The mass next moves ever faster in the same direction as the moving base. But the moving base initially still accelerates faster than the mass. In consequence, the distance z between the mass and the moving base is initially decreasing. Since the intial value of z is taken as 0, this means z becomes negative.

z < 0 therefore indicates that the mass is closer to the moving base than at the start. z > 0 means that the mass is further away from the moving base than at the start.

Examples:

The base of the SDOF is continuously accelerated. The absolute acceleration of the SDOF is to be calculated; natural frequency=1000Hz, damping=0.05

```
SDOF_Acc = SDOF_Response ( Acceleration, 1000, 0.05, 0, 0)
```

The base of an SDOF is agitated by brief acceleration (shock) (in m/s^2). The relative displacement of the SDOF (in m) is to be calculated. Natural frequency=100Hz, damping=0.05. The result is extended by 30 periods of the natural frequency in order to also obtain the oscillation after the agitation subsides.

```
f0 = 100 ; Hz
append_samples = floor ( 30 / (xdel?(Acceleration) * f0 ))
RelativeDisplacement = SDOF_Response ( Acceleration, f0, 0.05, 1, append_samples)
```

A selected line of the shock response spectrum is to be examined.

```
SRS = ShockResponseSpectrum ( Acceleration, 5.0, 500.0, 100, 0.05, 14, 0 )
B = SDOF_Response(Acceleration, SRS[1].x, 0.05, 0, 1+1 / (SRS[1].x*xdel?(Acceleration)) )
B_max = abs ( max ( B )) ; == SRS[1].y
```

Examining the arithmentical signs during brief constant acceleration

It is seen how abs_a slowly increases. But a is already large for the whole time. By definition, z begins at 0. It then becomes ever smaller. In other words, less than 0, because the mass and the moving base approach each other. Only later does the spring push the mass forward faster, causing z to increase.

```
a = ramp(0,1e-4,100)*0+1
abs_a = SDOF_Response(a,100.0,0.0,0,0)
z = SDOF_Response(a,100.0,0.0,1,0)
```

See also:

ShockResponseSpectrum

SearchLevel

Finds selectable level and slope conditions in a data set.

Declaration:

```
SearchLevel ( Data, SvLevelCondition, SvLevel1, SvLevel2, SvSlopeCondition, SvSlope1, SvSlope2, SvOption ) ->
XYFound
```

Parameter:

Data	Data set to be searched. Allowed types: [ND],[XY]
SvLevelCondition	Condition for level (amplitude)
	0 : Any level
	1 : less than or equal to..
	2 : Greater than or equal to....
	3 : In the interval from.. to..
	4 : Outside of the interval from.. to..
SvLevel1	Level value, whose meaning depends on the level condition. For Options 1 and 2, the limit; for 3 and 4, the lower interval boundary. Otherwise, set to 0.
SvLevel2	Level value, whose meaning depends on the level condition. For Options 3 and 4, the upper interval boundary. Otherwise, set to 0.
SvSlopeCondition	Condition for slope
	0 : Any slope
	1 : less than or equal to..
	2 : Greater than or equal to....
	3 : In the interval from.. to..
	4 : Outside of the interval from.. to..
SvSlope1	Slope value, whose meaning depends on the slope condition. For Options 1 and 2, the limit; for 3 and 4, the lower interval boundary. Otherwise, set to 0.
SvSlope2	Slope value, whose meaning depends on the slope condition. For Options 3 and 4, the upper interval boundary. Otherwise, set to 0.
SvOption	Option for linking level and slope conditions
	0 : The function determines all xy-pairs, which fulfill both conditions at the same time. The function therefore searches for specified states.
	1 : The function searches for all transitions, in which the level condition changes from false to true. The slope condition must additionally be fulfilled. Linear interpolation is performed on the x-position at the transition. The function thus searches for transitions, not for states.
XYFound	
XYFound	XY-data set with the pairs x, y, which fulfill defined conditions

Description:

This function searches for points in a data set which fulfill specified level and slope conditions.

A condition for the y-value (level, amplitude) can be defined first.

SvLevelCondition	
0	Any level
1	Level <=SvLevel1
2	Level >= SvLevel1
3	SvLevel1 <= Level <= SvLevel2 (value in the interval from .. to..)
4	Level <=SvLevel1 OR Level >=SvLevel2 (value not in the interval from.. to..)

It is also possible to specify a condition for the slope at a point.

SvSlopeCondition

0	Any slope
1	Slope <=SvSlope1
2	Slope >=SvSlope1
3	SvSlope1 <= Slope <= SvSlope2 (slope in the interval from .. to..)
4	Slope <= SvSlope1 OR Slope >=SvSlope2 (slope not in the interval from .. to ..)

The last parameter [SvOption] specifies how the conditions are to be linked. The effect of these option parameters is illustrated in the following example:

The data set to be checked has the values 0, 2, 6, 7, 9 V at a sampling rate of 1s. The function searches for a level > 4V using the formula

```
result = SearchLevel(data, 2, 4, 0, 0, 0, 0, 0)
```

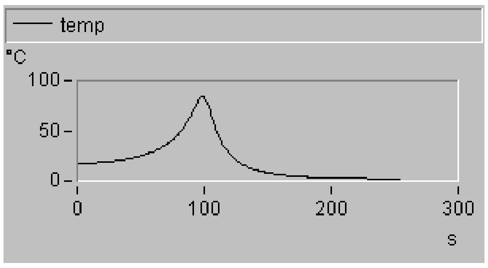
The function returns an XY-data set with the value pairs (6V, 2s), (7V, 3s), (9V, 4s) If the option is set to search for transitions:

```
result = SearchLevel(data, 2, 4, 0, 0, 0, 0, 1)
```

the function returns a value pair, namely (4V, 1.5s). This is the time determined (through linear interpolation) at which the condition was met.

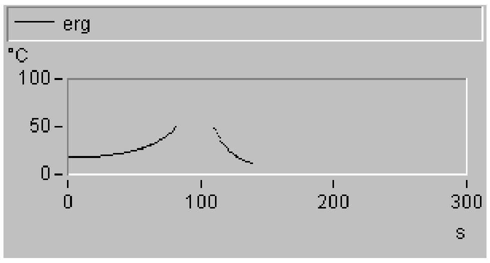
Examples:

Data set to be examined:



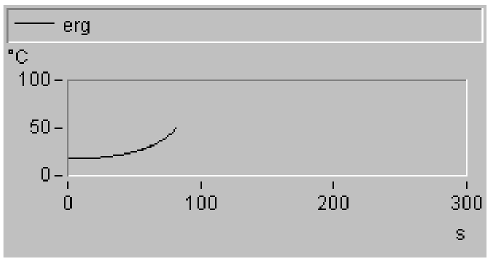
Search for all values in the range 10-50 °C, regardless of slope:

```
result = SearchLevel(temp, 3, 10, 50, 0, 0, 0, 0)
```



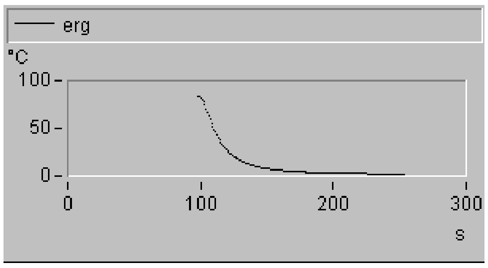
Search for all values in the range of 10-50 °C with positive slope:

```
result = SearchLevel(temp, 3, 10, 50, 2, 0, 0, 0)
```



Search for all values with a negative slope, regardless of level:

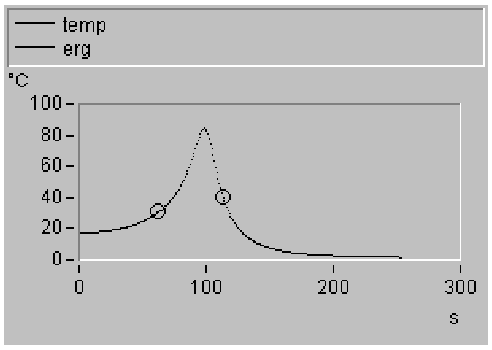
```
result = SearchLevel(temp, 0, 0, 0, 1, 0, 0, 0)
```



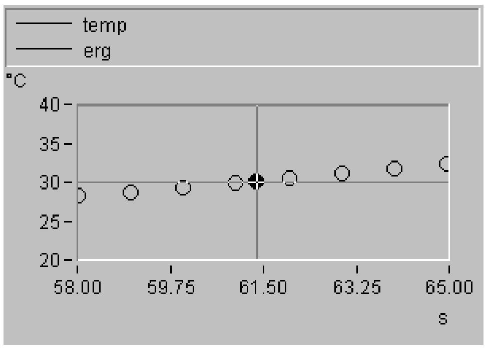
Searches for all transitions in which the temperature enters in the interval 30- 40°C from the top or bottom:

```
result = SearchLevel(temp, 3, 30, 40, 0, 0, 0, 1)
```

The result data set consists of 2 points (shown below as circles).



With extensive zooming, the interpolation at the 30°C -limit becomes clear:



See also:

Top, All0, xMax, PosiEx2

SegLen?

Determines a data set's segment length

Declaration:

```
SegLen? ( Data ) -> SvSegLeng
```

Parameter:

Data	Data set whose segment length is to be determined
SvSegLeng	
SvSegLeng	Segment length; 0 means no segmenting

Description:

This function returns a data set's segment length. A length of 0 means no segmenting.

Examples:

```
IF SegLen?(data) > 0
   SetSegLen(data, 0)
END
```

Any segmenting of the data set is voided.

See also:

SetSegLen, MatrixInfo

SelBuildVarName

From the given channel and measurement identifier a complete FAMOS variable name is constructed. Channel and measurement may be specified either by fixed name or by their index in the Data selector's measurement/channel list.

Declaration:

```
SelBuildVarName ( Measurement, Channel, SvOption ) -> TxVarName
```

Parameter:

Measurement	Name of the measurement or index (1,2,..) of the measurement list entry.
Channel	Name of the channel or index (1,2,..) of the channel list entry.
SvOption	Options parameter
	0 : The function checks whether a corresponding FAMOS variable exists. If not, an empty text is returned.
	1 : The variable name is returned unchecked.
TxVarName	
TxVarName	Complete variable name for FAMOS.

Description:

The returned variable name has the following structure:

```
ChannelName@MeasurementName
```

If channel- and/or measurement name do not comply with the rules for a valid variable name (e.g. contain spaces or special characters), then the corresponding identifier will be enclosed in {...}, whereby the returned variable name can still be used in sequences.

To subsequently use the returned variable name in sequences, you will need to use the Indirection Operator for text variables **<...>**. Before using the returned name, you should also check whether a variable with this name actually exists in imc FAMOS (see example #2).

The contents of the data selector's measurement/channel list is determined either by manual selection in the Variables list/ Measurement View and/or the functions SelMeasListSetName()/SelChanListSetName().

Examples:

ChannelName	MeasurementName	Result
"Channel1"	"Test1"	"Channel1@Test1"
"Channel1"	Empty	"Channel1"
"CH**1"	"Test1"	"{CH**1}@Test1"
"CH**1"	"Test[1]"	"{CH**1}@{Test[1]}"

All entries in the data selector are enumerated. If an entry refers to an existing FAMOS variable, the X-offset of this variable will be adjusted.

```
MCount = SelMeasListSize?()
FOR M = 1 TO MCount
   CCount = SelChanListSize?()
   FOR C = 1 TO CCount
      TxVarName = SelBuildVarName(M, C, 0)
      ; Option 0: If variable does not exist, return empty text!
      IF TxVarName <> ""
         XOFFSET <TxVarName> 10
      END
   END
END
```

In case the Measurement #1 in the Dataselector contains the channels "Voltage" and "Current", the power is calculated and displayed in a separate curve window.

```
TxUName = SelBuildVarName(1, "Voltage", 0)
TxIName = SelBuildVarName(1, "Current", 0)
; Option 0: If variable does not exist, return empty text!
IF (TxUName <> "") AND (TxIName <> "")
   Power = <TxUName> * <TxIName>
   SHOW Power
END
```

See also:

SelMeasListName?, SelMeasListSize?, SelChanListName?, SelChanListSize?, SelMeasListSetName, SelChanListSetName

SelChanListName?

An entry from the dataselector's channel list is returned.

Declaration:

```
SelChanListName? ( SvIndex ) -> TxName
```

Parameter:

SvIndex	Index of the requested entry.
TxName	
TxName	Name of the channel.

Description:

This function is seldom used. To build a complete FAMOS variable name from measurement and channel name, use the function "SelBuildVarName" instead.

The contents of the data selector's measurement/channel list is determined either by manual selection in the Variables list/ Measurement View and/or the functions SelMeasListSetName()/SelChanListSetName().

Examples:

All entries in the Data Selector's Channels list are enumerated and displayed.

```
count = SelChanListSize?()
c = 1
WHILE c <= count
   TxMeas = SelChanListName?(c)
   BoxOutput(TxMeas, EMPTY, "", 1)
   c = c + 1
END
```

See also:

SelMeasListName?, SelChanListSize?, SelBuildVarName, SelMeasListSetName, SelChanListSetName

SelChanListSetName

An entry in the Data Selector's channel list is set.

Declaration:

`SelChanListSetName ( SvIndex, TxName, GroupName [, SvOption] )`

Parameter:

SvIndex	Index of the entry in the Data Selector's channel list to be set. The index of the first entry is 1.
TxName	Name of the channel. An empty text deletes the existing entry.
GroupName	Name of the higher-level group, or empty text
SvOption	Option (optional , Default value: 0)
	0 : The change takes immediate effect (default).
	1 : The change takes effect only upon calling SelListControl(0).

Description:

This function sets an entry in the Data Selector's channel list, thus assigning a number to a channel name.

Thus, the function represents the automation of manually selecting a channel in the Variables list/Measurement view.

To specify a channel belonging to a group, either the channel name and group name can be specified separately in the 2nd and 3rd parameters, or they can be specified together in the 2nd parameter in the customary notation 'GroupName:ChannelName'. In the latter case, the 3rd parameter is left empty.

The parameter [option] controls whether the change takes effect immediately or only after an explicit call of SelListControl(0). This affects the updating of display elements which use the Data Selector (e.g. a curve window with a link to a measurement- or channel number), as well as the display of the corresponding numbering in the Variables list/Measurement view and the resolution of the Panel-event 'Data selection changed'. Enter a "1" if you wish to set multiple entries in the Data Selector in succession, and then in conclusion call SelListControl(0).

The system does not check whether a channel having the specified name actually exists at the time of the call.

Multithreading: The function has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

A Panel contains a curve window, which is configured for displaying the 1st channel of the 1st selected measurement ('Channel #1 @ Measurement #1') and a button 'Show'. The user initially loads all desired measurements manually. At the push of a button, then for each measurement, all channels starting with 'U_' are each displayed for 2 seconds.

Event-sequence 'Button pressed (Show)':

```
measurements = MeasNames?("*")
FOREACH ELEMENT name in measurements
   SelMeasListSetName(1, name, 1)
   channels = MeasChanNames?(name, "U_*")
   FOREACH ELEMENT chan in channels
      SelChanListSetName(1, chan, "", 1)
      SelListControl(0) ; refresh curve windows now
      Sleep(2)
   END
END
```

See also:

SelChanListName?, SelMeasListSetName, SetMeasurementName, SelBuildVarName, MeasChanNames?

SelChanListSize?

The amount of sequentially numbered entries in the data selector channel list is returned.

Declaration:

```
SelChanListSize? ( ) -> SvCount
```

Parameter:

SvCount	
SvCount	Count of entries in the channel list.

Description:

Only channels having a sequential numbering from 1 are taken into account; in the rare case of a "gap" in the channel list, the count is interrupted. If, for instance, the channels #1 through #3 are selected, a 3 is returned. But if channels #1, #2 and #4 are selected, because channel #3 is not occupied, the result is 2.

The contents of the data selector's measurement/channel list is determined either by manual selection in the Variables list/ Measurement View and/or the functions SelMeasListSetName()/SelChanListSetName().

Examples:

All entries in the data selector are enumerated. If an entry refers to an existing FAMOS variable, the X-offset of this variable will be adjusted.

```
MCount = SelMeasListSize?()
M = 1
WHILE M <= MCount
   CCount = SelChanListSize?()
   C = 1
   WHILE C <= CCount
      TxVarName = SelBuildVarName(M, C, 0)
      ; Option 0: If variable does not exist, return empty text!
      IF TxVarName <> ""
         XOFFSET <TxVarName> 10
      END
      C = C+1
   END
   M = M+1
END
```

See also:

SelMeasListName?, SelChanListName?, SelBuildVarName, SelMeasListSetName, SelChanListSetName

SelListControl

Control of the Data Selector

Declaration:

`SelListControl ( SvTask )`

Parameter:

SvTask	Task
	0 : Displays (curve window, Panel elements, Variables list/Measurement view) are updated. In consequence, the Panel-event 'Data selection changed' may be triggered.
	1 : The Measurement and Channel lists are completely deleted. The change takes effect immediately.
	2 : The Measurement and Channel lists are completely deleted. The change only takes effect upon the next call of SelListControl(0). Use this value if you wish to refill the list subsequently.

Description:

With [Task] = 0, any previous changes of the Data Selector are put into effect. This affects the updating of display elements which use the Data Selector (e.g. a curve window with a link to a measurement- or channel number), the display of the associated numbering in the Variables list/Measurement view, and the triggering of the Panel-event 'Data selection changed'.

Examples:

The Data Selector's entries are set to fixed names:

```
SelListControl(2) ; delete current content
SelMeasListSetName(1, "Measurement_12_04_2015", 1)
SelMeasListSetName(2, "Measurement_12_05_2015", 1)
SelChanListSetName(1, "channel1", "", 1)
SelChanListSetName(2, "channel2", "", 1)
SelListControl(0) ; Refresh display
```

See also:

SelMeasListSetName, SelChanListSetName, MeasNames?, SetMeasurementName

SelMeasListName?

An entry from the dataselector's measurement list is returned.

Declaration:

```
SelMeasListName? ( SvIndex ) -> TxName
```

Parameter:

SvIndex	Index of the requested entry.
TxName	
TxName	Name of the measurement.

Description:

This function is seldom used. To build a complete FAMOS variable name from measurement and channel name, use the function "SelBuildVarName" instead.

The contents of the data selector's measurement/channel list is determined either by manual selection in the Variables list/ Measurement View and/or the functions SelMeasListSetName()/SelChanListSetName().

Examples:

All entries in the Data Selector's Measurements list are enumerated and displayed.

```
MCount = SelMeasListSize?()
FOR M = 1 TO MCount
   TxMeas = SelMeasListName?(M)
   BoxOutput(TxMeas, EMPTY, "", 1)
END
```

All channels belonging to measurement #1 in the data selector are deleted.

```
measurement = SelMeasListName?(1)
DELETE *@{<measurement>}
```

See also:

SelMeasListSize?, SelChanListSize?, SelBuildVarName, SelMeasListSetName, SelChanListSetName

SelMeasListSetName

Sets an entry in the Data Selector's Measurements list.

Declaration:

```
SelMeasListSetName ( SvIndex, TxName [, SvOption] )
```

Parameter:

SvIndex	Index of the entry to set in the Measurements list. The first entry's index is 1.
TxName	The measurement's name. An empty text deletes the existing entry. "." stands for [no measurement].
SvOption	Option (optional , Default value: 0)
	0 : The change takes immediate effect (default).
	1 : The change takes effect only upon calling SelListControl(0).

Description:

The function sets an entry in the Data Selector's Measurements list, thus assigning a number to a measurement's name.

This function represents the automation of manual selection of a measurement in the Variables list/Measurement view.

The parameter [option] controls whether the change takes effect immediately or only after an explicit call of SelListControl(0). This affects the updating of display elements which use the Data Selector (e.g. a curve window with a link to a measurement- or channel number), as well as the display of the corresponding numbering in the Variables list/Measurement view and the resolution of the Panel-event 'Data selection changed'. Enter a "1" if you wish to set multiple entries in the Data Selector in succession, and then in conclusion call SelListControl(0).

The system does not check whether at the time of the call there actually is any measurement with the specified name.

Multithreading: The function has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

A panel contains a curve window which is configured for display of the 1st channel of the 1st selected measurement ('Channel #1 @ Measurement #1') and a button 'Show'. The user initially manually loads all measurements desired and selects a channel name (assigned the number #1) in the Variables list/Measurement View. At the push of a button, all currently available measurements are enumerated and each associated curve plot for the selected channel is displayed for 2 seconds.

Event-sequence 'Button pressed (Show)':

```
measurements = MeasNames?("*")
FOREACH ELEMENT name IN measurements
   SelMeasListSetName(1, name)
   Sleep(2)
END
```

The same situation as above, but here, for each measurement, all channels which start with 'U_' are displayed in succession.

```
measurements = MeasNames?("*")
FOREACH ELEMENT name in measurements
   SelMeasListSetName(1, name, 1)
   channels = MeasChanNames?(name, "U_*")
   FOREACH ELEMENT chan in channels
      SelChanListSetName(1, chan, "", 1)
      SelListControl(0) ; refresh curve windows now
      Sleep(2)
   END
END
```

A folder structure contains a number of subfolders, each representing a measurement and which contain the channels 'Voltage' and 'Current'. A Panel contains 3 curve windows, which are configured for the display of the channels 'Voltage', 'Current' and 'Power' of the first selected measurement. When a folder is loaded by means of the Data Source Browser, the new measurement is automatically selected as the 1st measurement, the variable 'Power' is calculated and saved in the same folder as the source data. The Panel display is updated accordingly.

Event-sequence 'Measurement available'

```
IF PA2 = 0 ; new measurement
   SelMeasListSetName(1, PA1)  ; new measurement gets the number 1
   SelUseMeasurement(1)
   Power = Voltage * Current
   SetMeasurementName(Power, PA1)
   fileName = FsSplitPath(FileName?(Voltage), 8) + "\power"
   FileSave(fileName, "", 0, Power)
END
```

See also:

SelMeasListName?, SelChanListSetName, SelBuildVarName, SetMeasurementName, MeasNames?

SelMeasListSize?

The amount of sequentially numbered entries in the data selector measurement list is returned.

Declaration:

```
SelMeasListSize? ( ) -> SvCount
```

Parameter:

SvCount	
SvCount	Count of entries in the measurement list.

Description:

Only measurements having a sequential numbering from 1 are taken into account; in the rare case of a "gap" in the measurement list, the count is interrupted. If, for instance, the measurements #1 through #3 are selected, a 3 is returned. But if measurements #1, #2 and #4 are selected, because measurement #3 is not occupied, the result is 2.

The contents of the data selector's measurement/channel list is determined either by manual selection in the Variables list/ Measurement View and/or the functions SelMeasListSetName()/SelChanListSetName().

Examples:

All entries in the data selector are enumerated. If an entry refers to an existing FAMOS variable, the X-offset of this variable will be adjusted.

```
MCount = SelMeasListSize?()
FOR M = 1 TO MCount
   CCount = SelChanListSize?()
   FOR C = 1 TO CCount
      TxVarName = SelBuildVarName(M, C, 0)
      ; Option 0: If variable does not exist, return empty text!
      IF TxVarName <> ""
         XOFFSET <TxVarName> 10
      END
   END
END
```

See also:

SelMeasListName?, SelChanListSize?, SelBuildVarName, SelMeasListSetName, SelChanListSetName

SelUseMeasurement

Formula Interpreter: Sets which measurement of which variables are used.

Declaration:

```
SelUseMeasurement ( MeasNameOrNumber ) -> SvStatus
```

Parameter:

MeasNameOrNumber	The measurement to be used. Either number or text. The number then refers to the corresponding entry in the Data Selector (Variables list/Measurement View). A "1" means, for example, that the first selected measurement is used. Alternatively, the measurement's name can also be stated directly, this always works regardless of the current assignment in the data selector.
SvStatus	
SvStatus	Result of function
	0 : No measurement is currently assigned to the measurement number entered.
	1 : OK

Description:

The measurement specified here is used by the Formula Interpreter to locate variables. The variable name is then automatically supplemented with this measurement. This is very helpful if sequences are to be used on channels having the same name byte belonging to different measurements.

The contents of the data selector's measurement/channel list is determined either by manual selection in the Variables list/ Measurement View and/or the functions SelMeasListSetName()/SelChanListSetName().

The function offers a simple alternative to the complete variable designation structure ChannelName@MeasurementName by means of the function SelBuildVarName().

A 0 or empty text for the parameter means that no standard measurement is used.

Once a standard measurement has been constructed using SelUseMeasurement(), when generating a new variable by means of making an assignment, the simplified notation

```
NewVariable@ = ...
```

can also be used. The new variable is then automatically asignedd to the current standard measurement.

Example: The measurements currently processed contain the channel 'current'. The channel belonging to Measurement #1 is to be provided with a fixed offset:

Variant 1:

```
SelUseMeasurement(1)
current = current + 10
```

Variant 2:

```
TxVarName = SelBuildVarName(1, "current", 0)
<TxVarName> = <TxVarName> + 10
```

The measurement set here remains valid until:

- next call of the function SelUseMeasurement()
- next restart of a top-level sequence
- Menu command 'Restart'.

Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Note: Unless a measurement is explicitly stated with a variable, the formula interpreter first searches among the variables with the selected measurement assignment. Only when this fails will the search be continued among the variables without measurement assignment.

Examples:

The measurements currently processed contain the channels 'Voltage' and 'Current'. For the currently selected Measurement #1, the power is calculated and displayed in a separate curve window.

```
IF SelUseMeasurement(1) = 0
   EXITSEQUENCE
END
Power = Voltage * Current
; or: Power@ = Voltage * Current ;=> the variable 'Power' is assigned to the current measurement.
```

```
SHOW Power
```

If the above sequence is to be structured in a more robust way, you can have an additional check useing the function VarExist?() of whether the expected channels are actually available at runtime:

```
IF SelUseMeasurement(1)
   IF VarExist?("Voltage") AND VarExist?("Current")
      Power = Voltage * Current
      SHOW Power
   END
END
```

A Panel serves the purpose of visualizing measured data. The measurements to be investigated each contain a channel "channel1", which is to be smoothed prior to display and subseuent evaluation. The smoothing is performed in the Panel-event sequence "Measurement available":

```
IF PA2 = 0  ; new measurement?
   SelUseMeasurement(PA1)  ; PA1: Name of the new measurement
   channel1 = Smo(channel1, 2)
END
```

See also:

SelMeasListName?, SelChanListName?, SelBuildVarName, SelMeasListSetName, SelChanListSetName

SEQUENCE

Run sequence

Declaration:

```
SEQUENCE Filename Par1 Par2 Par3 Par4 Par5 Par6 Par7 Par8 Par9 ...
```

Parameter:

Filename	Filename of the sequence file
Par1	1st parameter
Par2	2nd parameter
Par3	3rd parameter
Par4	4th parameter
Par5	5th parameter
Par6	6th parameter
Par7	7th parameter
Par8	8th parameter
Par9	9th parameter
...	More parameter (up to 20)

Description

A sequence is executed. The name of the sequence is a file name and is transferred as the first parameter. If the sequence to be executed expects parameters, these must be transferred as further parameters. The parameters passed are addressed within the sequence called via the symbolic designators PA1, PA2 etc.

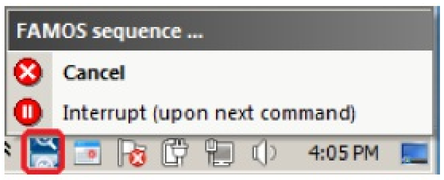
The filename may also specified to contain quotation marks. This can be necessary, if, for instance, the path contains spaces.

Unless a complete path is provided along with the filename, the system searches for the sequence file in this order of folders:

- Project folder: When a project is opened, the system looks for it in the current project folder.
- Current working directory: Upon starting FAMOS, initially set to the folder specified under "Options"/ "Folders". It can be modified using either the command MDIR or the function SetOption(). Otherwise it is the folder from which the sequence making the call was opened.

Please refer to the chapter 'Sequences' for more information about working with sequences, especially those with parameters.

- The maximum number of parameters is 20.
- The number of parameters actually passed can be determined within the sequence using the ParametersPassed?() function.
- When a sequence is active, the Windows taskbar's notification area contains an additional icon. Right-clicking the mouse calls a context menu which offers a command for interrupting the current run.
- To interrupt, it's also possible to use the combination of keys "CTRL" + "Break".



Examples:

```
SEQUENCE Series
```

The sequence named Series is executed; it does not expect any parameters.

```
SEQUENCE Sum Wave1 Wave2
```

A sequence named Sum is executed. It calculates the sum of two channels. The data sets to be added are transferred as parameters.

```
SEQUENCE c:\imc\seq\LoadAll.seq d*.dat
```

A sequence used to load a file is executed for all files which begin with a "D" and have the extension ".DAT" .

```
SEQUENCE "c:\imc\My Sequences\Evaluate.dat" Data1
```

The filename contains a space and therefore it must be written inside of quotation marks

See also:

Dialog, SetOption, MDIR, ParametersPassed?

Set

Sets points of a data set at a specified x-position to specified new y-values.

Declaration:

```
Set ( Data, XPositions, YValues ) -> ResultData
```

Parameter:

Data	Data set to be changed; allowed types: [ND].
XPositions	X-coordinates at which the associated y-values are to be changed
YValues	Y-values to which the data set's values are to be set at the specified x-coordinates
ResultData	
ResultData	Changed data set with corresponding new values

Description:

A data set's numerical value can be reset to a new value in a targeted way.

To be specified:

- the data set in which a value is to be re-set,
- the x-coordinate(s) at which the new value is to be entered,
- the new y-coordinate(s), i.e. the new numerical value(s).

A new data set is generated in which a series of values is changed from the original data set passed to the function. The 2nd and 3rd parameters must have the same length.

The data set's units are adopted. The 2nd parameter should represent the x-unit, the 3rd should represent the y-unit.

The x-position of the value to be set as the x-coordinate, not as the index.

If the x-position lies outside the x-range of the data set, no value is changed.

Alternatively, you can use the function SetIndex if you wish to specify the setting position in terms of the point's index in the data set. This function is also adapted to XY-data sets.

To change numerical values in a data set manually without automating the procedure, use the Data Editor in imc FAMOS.

To change individual values, you can also accomplish this directly by assignment:

```
NDData[ Index] = newSingleValue
```

Examples:

The value is set to the position 5s to 4.2A in a data set with the range 0s to 20s .:

```
NDdata = Set(NDdata, 5 's', 4.2 'A')
```

The seventh point in the data set is set to the value 5. The x-coordinate of the point is calculated from the point's index within the data set and the x-scaling.

```
NDnew = Set(NDold, xOff?(NDold) + (7 - 1) * xDel?(NDold), 5)
; or much more simply:
NDNew[7] = 5
```

See also:

SetIndex, Value2, ValueIndex, MatrixSet, MatrixFromLine, Repl, ReplIndex

SetBoxPos

Sets position of input/output boxes

Declaration:

```
SetBoxPos ( TxFuncName, SvLeft, SvTop, SvWidth, SvHeight, Zero )
```

Parameter:

TxFuncName	Name of the window-creating function
	"BoxValue?" : BoxValue?
	"BoxText?" : BoxText?
	"BoxOutput" : BoxOutput
	"BoxMessage" : BoxMessage
	"BoxVarSelector" : BoxVarSelector
	"DlgFileName" : DlgFileName
	"FsDlgSelectDirectory" : FsDlgSelectDirectory
SvLeft	X-position of the top left corner of box
SvTop	Y-position of top left corner of box
SvWidth	Width of the box
SvHeight	Height of the box
Zero	Reserved parameter. Always set to 0.

Description:

Sets the display position for boxes created by subsequent calls to BoxValue?(), BoxText?(), BoxOutput(), BoxMessage() or DlgFileName().

Position specifications refer to screen pixels. At a resolution of 800x600, the top left corner of the screen has the coordinates (0,0) and the bottom right the coordinates (799,599).

Boxes generated by means of BoxMessage() or DlgFileName() always have a fixed, automatically detetermined size. [Height] and [Width] are ignored here.

All boxes have a default height and width. Smaller values of [Width] and [Height] are ignored.

The coordinates may be corrected if necessary, so that the box fits completely within the visible region.

If [Left] is set to -1, the position upon the next call is again determined automatically.

Multithreading: The function has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

```
SetBoxPos("BoxMessage", 0, 0, 0, 0, 0)
BoxMessage("Result: ", res, "f20", 0)
```

Output top left in the screen corner with default width and height

```
SetBoxPos("BoxText?", 600, 450, 200, 150, 0)
TxName = BoxText?("Name of the variable?","var1",0)
```

Output bottom right (at screen resolution of 800x600 pixels).

See also:

BoxOutput, BoxMessage, BoxValue?, BoxText?, DlgFileName

SetColor

Assigns a color for display of a data set as a curve.

Declaration:

```
SetColor ( Data, SvColorValue )
```

Parameter:

Data	Data set whose color attribute is to be configured.
SvColorValue	Color value; -1 means automatic color assignment

Description:

Normally, data sets are displayed in the curve window configuration's color for graphs. But you ca assign to the data set a fixed color in which it will appear in any curve window, regardless of the curve window's current color setting.

The color value is a so-called RGB-value in which the portions of the 3 primary colors red, green and blue are stated.

To achieve a color value from primary color components, you can use the function RGB().

Examples:

```
green = RGB(0, 255, 0)
clr = Color?(data)
IF clr = -1
   SetColor(data, green)
END
```

Unless a fixed color has already been assignedd to the data set, it will be colored green in the future.

See also:

Color?, RGB

SetComm

Specifies the comment on a data set, text or data group.

Declaration:

```
SetComm ( Dataobject, TxComment )
```

Parameter:

Dataobject	Data set, text or data group whous comment is to be specified
TxComment	New comment

Description:

Examples:

The comment on a data set is queried. If none has been specified (length: 0), the user is prompted to enter a comment, which will then be assigned to the data set.

```
txComment = Comm?(data)
IF TLeng(txComment) = 0
   txComment = BoxText?("Please enter comment:", "",0)
   SetComm(data, txComment)
END
```

See also:

Comm?

SetDataFormat

A data set is converted to a different data format.

Declaration:

```
SetDataFormat ( Data, SvFormatCode [, SvScalMin] [, SvScalMax] )
```

Parameter:

Data	Data set whose data format is to be set
SvFormatCode	Format specification
	0 : 4 Byte real (float)
	1 : 8 Byte real (double)
	2 : 1 Byte integer
	3 : 2 Byte integer
	4 : 4 Byte integer
	5 : 1 Byte unsigned integer
	6 : 2 Byte unsigned integer
	7 : 4 Byte unsigned integer
	8 : Digital
	10 : 6 Byte unsigned integer
	12 : 8 Byte integer
	13 : 8 Byte unsigned integer
SvScalMin	With integer data formats, the lower value range boundary; else unimportant. (optional , Default value: 0)
SvScalMax	With integer data formats, the upper limit of the value range; else not applicable. When [SVScalMin] and [SVScalMax] are identical, scaling is performed automatically. If the optional parameters [SVScalMin] and [SVScalMax] are both not specified, the complete numerical range of the integer data format is assumed. (optional , Default value: 0)

Description:

This function converts a data set to a new data format.

The data format specifies how the individual values are saved in memory/the data carrier. The memory requirements for a data set, the value range and the achievable precision are determined by the data format.

When an XY-data set is transformed using this function, the data format of the y-component is set; when a complex data set is transformed, the data format of the magnitude or real part is set. To convert the other components, use the component identification codes:

```
SetDataFormat(MagnitudePhase.P, 0, 0, 0)
SetDataFormat(XYdata.X, 5, 0, 255)
```

The data format set by this function can change due to subseqent processing of the data set. To prevent this and to make the format permanent, you must activate the "Fixed data format" property using the function SetFlag.

<u>Special features of integer data formats:</u>

For integer data formats, a scaling factor and offset are additionally taken into account internally, in order to derive the physical values from the integer raw data:

```
PhysicalValue = IntegerValue * ScaleFactor + Offset
```

For signed formats having the bit width [BitCount], the characteristic values [ScaleFactor] and [Offset] are calculated as follows:

```
Offset = (SvScalMin+SvScalMax)/2
ScaleFactor = (SvScalMax-Offset) / (2^(BitCount-1)-1)
```

Thus, for a 1:1-mapping of the entire possible number range, specify [SvScalMin] = -2^(BitCount-1)-1 and [SvScalMax] = 2^(BitCount-1)-1.

For unsigned formats, we have the equation:

```
Offset = SvScalMin
ScaleFactor = (SvScalMax-Offset) / (2^BitCount-1)
```

Thus, for a 1:1-mapping of the entire possible number range, specify [SvScalMin] = 0 and [SvScalMax] = 2^BitCount-1.

Example:

Format = 2 (1Byte signed), SvScalMin= -127, SvScalMax = 127 ==> Offset = 0, ScaleFactor = 1

Format = 5 (1Byte unsigned), SvScalMin= -128, SvScalMax = 127 ==> Offset = -128, ScaleFactor = 1

If the optional parameters [SVScalMin] and [SVScalMax] are omitted, then for integer formats, a 1:1 mapping (ScaleFactor=1, Offset=0) is applied.

Examples:

```
format = DataFormat?(myData)
IF format <> 0
   SetDataFormat(myData, 1)
   SetFlag(myData, 0, 1)
END
```

If the data set is not already in the 4-bytes real format, it is converted. Next, this format is fixed, so that it isn't changed by subsequent calculations.

Examples for scaling in conjunction with integer formats:

```
SetDataFormat(myData, 2)            ;Value range -128..127 (Scaling factor  = 1, offset = 0)
SetDataFormat(myData, 2, -127, 127) ;Value range -128..127 (Scaling factor = 1, offset = 0)
SetDataFormat(myData, 2, 0, 10)     ;Value range 0..10     (Scaling factor = 5/127, offset = 5)
SetDataFormat(myData, 5)            ;Value range 0..255    (Scaling factor = 1, offset = 0)
SetDataFormat(myData, 5, 0, 10)     ;Value range 0..10     (Scaling factor = 10/255, offset = 0)
```

See also:

DataFormat?, GetScale, SetFlag

SetDisplayY

Assigns a fixed Y-scaling for the display in the curve window.

Declaration:

```
SetDisplayY ( Data, SvMin, SvMax )
```

Parameter:

Data	Data set whose Y-scaling is to be specified
SvMin	Bottom scale value
SvMax	Top scale value

Description:

A fixed Y-axis scaline can also be assiged to a data set as an additional property. It is then used for the display of the data set in a curve window if the curve window's setting "Automatic Y-axis scaling" is active.

The fixed scaling is canceled using SvMin = SvMax = 0. The data set is then scaled automatically according to its range of values.

Examples:

```
yMin = DisplayY?(data, 0)
IF yMin > 0
   SetDisplayY(data, 0, DisplayY?(data, 1))
ELSE
   yMax = DisplayY?(data, 1)
   IF yMax < 0
      SetDisplayY(data, DisplayY?(data, 0),0)
   END
END
```

If no fixed scaling has been assigned to the data set and the zero-line is not included, either the bottom or top scale value is corrected to 0.

See also:

DisplayY?, SetColor

SetFlag

This function toggles special attributes of a waveform on and off.

Declaration:

```
SetFlag ( Data, SvFlag, SvOnOrOff )
```

Parameter:

Data	Waveform whose attribute is to be changed.
SvFlag	Attribute selection
	0 : The current data format of the waveform is fixed, i.e. for subsequent processing, the data format (and for integer formats the scaling) is retained if possible.
	1 : The values of the data set can be interpreted as color information for 1 pixel of an image. Only allowed for data formats '4 byte unsigned' (color information is encoded as RGB-value) or '1 byte unsigned' (color information is encoded as grey scale value in the range 0..255). By default, the attribute is only set by import filters for image files or special functions such as VpGetImages(). It is used by the curve window to optimize the display of image data.
SvOnOrOff	New value
	0 : Off
	1 : On

Description:

This function sets certain data set attributes which can only take the boolean values [On] (or "True") or [Off] (or "False").

Activating the option [SvFlag] = 1 (interpret as image data) is only allowed for data sets having the data format "4 Byte unsigned Integer" or "1 Byte unsigned integer". For segmented data, each segment corresponds to one image row, the data set's first sample corresponds to the pixel at bottom left.

Examples:

The data format of the data set [Signal] is set and fixed as 1-Byte Integer (unsigned, value range 0..255). Any subsequent processing (see line [*]) doesn't change the data format. Without the call to SetFlag, the [Signal's] data format after line [*] would be converted to 4- or 8-Byte Real (depending on the global default for the result data format of math functions).

```
signal = ...
SetDataFormat(signal, 5, 0, 255)
SetFlag(signal, 0, 1)
;...
signal = signal/2  ;[*]
```

See also:

Flag?, RGB, VpGetImages, SetDataFormat

SetIndex

Sets points in a data set to new y-values at specified indices.

Declaration:

```
SetIndex ( Data, Indices, YValues ) -> ResultData
```

Parameter:

Data	Data set to be changed; allowed types: [ND],[XY].
Indices	Indices, at which the associated y-values are to be changed
YValues	Y-values to which the data set's values are to be changed at the specified positions
ResultData	
ResultData	Changed data set with corresponding new values

Description:

Sets one or more points of a data set to new y-values at specified indices.

The positions [indices] specified must lie between 1 and the data set lenth of [Data].

Indices lying outside of this range are ignored (a warning is posted).

A new data set is generated in which a series of values is changed from the original data set passed to the function. The 2nd and 3rd parameters must have the same length.

There first point in a data set has the index 1; the index of the last point matches the data set length.

Alternatively, you can use the function Set, which requires the positions of the replacement points to be specified in terms of their x-coordinates.

To change numerical values in a data set manually without automating the procedure, use the Data Editor in imc FAMOS.

If you wish to change a single value in FAMOS, you can also accomplish this directly by assignment:

```
NwData[ Index] = SvNewValue
```

Examples:

The second value of the data set is set to 10.

```
data = SetIndex(data, 2 , 10)
; equivalent to:
data[2] = 10
```

Every second Y-value of the data set is doubled:

```
Indizes = Ramp(1, 2, Leng?(Data)/2) ;Indizes = 1,3,5..
NewY= ValueIndex(Data, Indizes) * 2
DataNew = SetIndex(Data, Indizes, NewY)
```

See also:

Set, Value2, ValueIndex, MatrixSet, MatrixFromLine, Repl, ReplIndex

SetMeasurementName

Sets the name of the measurement to which the variable is assigned.

Declaration:

```
SetMeasurementName ( Variable, TxMeasName )
```

Parameter:

Variable	Variable to be changed
TxMeasName	Name of the measurement to which the variable is to be assigned.

Description:

An empty measurement name means that any association to a measurement is deleted.

The concept of a variable's measurement association has until now mainly been applicable to the Data Source Browser. There, when a file of measured values is loaded, a measurement name is automatically assigned to each variable generated, as a way of conveniently distinguishing between multiple variables having the same name buut belonging to different data sources. Changing a variable's measurement association automatically causes the system to update the measuremetn and Cahnnel lists in the Variables List/Measurement View.

Since FAMOS 7.1, when generateing a variable by means of making an assignment, the desired measurement name can also be specified directly (see examples).

Examples:

Once a new measurement has been set up using the Data Source Browser, a check is made of whether it contains a channel with the name "speed". If yes, then the channel's maximum value is calculated and the result is marked with the current measurement name.

Event-sequence 'Measurement available'

```
TxVarName = SelBuildVarName(PA1, "speed", 0)
IF TxVarName <> ""
   MaxSpeed = max(<TxVarName>)
   SetMeasurementName(MaxSpeed, PA1)
END
Or more simply:
```

```
...
IF TxVarName <> ""
   MaxSpeed@<PA1> = max(<TxVarName>)
END
```

The measurements currently being processed contain the channels 'Voltage' and 'Current'. The power associated with the currently selected measurement #1 is computed and assigned the corresponding measurement name. This means that the calculated variable is automatically displayed in the Variables List/Measurement View also, in the Channels list.

```
IF SelUseMeasurement(1) = 0
   EXITSEQUENCE
END
TxMeasName = MeasurementName?(Voltage)
Power = Voltage * Current
SetMeasurementName(Power, TxMeasName)
```

Or more simply:

```
IF SelUseMeasurement(1) = 0
   EXITSEQUENCE
END
Power@ = Voltage * Current
```

See also:

MeasurementName?

SetOption

Sets various defaults, such as default folders and parameters for math functions.

Declaration:

```
SetOption ( TxOptionName, TxNewSetting ) -> OldSetting
```

Parameter:

TxOptionName	Designation of the option
	"Dir.DataFiles" : Default folder for loading/saving measurement data files. Used by the commands LOAD, SAVE and the like, as well as the functions FileOpenDSF() and FileOpenFAS().
	"Dir.Sequences" : Default folder for loading sequences. Used by the command SEQUENCE.
	"Func.WarnLevel" : Determines which warning messages triggered during the execution of functions are displayed
	"Func.NoInfoMessages" : Some functions (Stat, LFit) print their results by default in the output window. Output can be suppressed with this option.
	"Func.FFT.Window" : Determines the FFT window function. Used by functions FFT(), Spec() and the like.
	"Func.FFT.Mode" : The FFT algorithm implemented requires the length of the input data set to be a power of 2. This option determines how to deal with other data set lengths.
	"Func.ResultFormat" : Determines the data format in which the functions return their results by default.
	"Func.ErrorBoxes" : Some functions have the choice of responding to an error by posting an error box or by indicating the error (silently) by their return values. To date only pertains to the file function group including FileOpenDSF() and the like.
	"DLLImport.DefinitionFile" : Filename containing definitions for external DLL-functions. The functions listed here are imported to FAMOS via the general-purpose DLL-interface and can be used in sequences. The file needs to have been generated by means of the dialog 'Options'/'Register DLL-functions'. If no complete path is specified, the system looks for the file in the following folders in succession: Project folder (when project active) - current sequence folder - default folder for definitions-files (see 'Options'/'Folders'). If no file extension is specified, ".def" is assumed.
	"Display.DecimalSeparator" : Sets the character displayed for separating real numbers' decimal places.
	"DDE.Text.NumFormat" : Determines the numerical format for sending data in text format by DDE.
	"DDE.Text.Delimeter" : Determines the separator(s) used between 2 numbers when sending data in text-format by DDE.
	"DDE.TimeOut" : Sets the maximum amount of time FAMOS waits for an answer in DDE communication with another application.
	"Units.Ctrl.Compatible" : Units: Compatibility
	"Units.Display.Greek" : Units: Font for Greek letters
	"Units.Display.Ohm" : Units: Ohm
	"Units.Create.Delim" : Units: Separator character between units, e.g. V*A
	"Units.Create.Nm" : Units: Separator character for Nm and the like
	"Units.Create.Pow.1/2" : Units: Use special character for 1/2
	"Units.Create.Pow.2" : Units: Use special character for ^2
	"Units.Create.Pow.3" : Units: Use special character for ^3
	"Units.Create.Pow.Neg" : Units: Negative exponents in denominator
	"Units.Create.u" : Units: generate u instead of µ (prefix 10^-6)
	"Units.Create.Num.Space" : Units: Spaces between automatically amended power of 10 and unit
	"Units.Create.1e3" : Units: 10, 100, 1000
	"Units.Create.1e-3" : Units: 0.1, 0.01, 0.001
	"Units.Create./s" : Units: Unit with numerator = 1
	"Units.Read.cal" : Units: Unit symbol cal
	"Units.Read.Exp" : Units: Number as exponent
	"Units.Read.g" : Units: g
	"Units.Read.Gs" : Units: Unit symbol for Gauss

	"Units.Read.hp" : Units: Unit symbol hp
	"Units.Read.kt" : Units: Unit symbol kt
	"Units.Read.L" : Units: Unit symbol L
	"Units.Read.lb" : Units: Unit symbol lb
	"Units.Read.oz" : Units: Unit symbol oz
	"Units.Read.pt" : Units: Unit symbol pt
	"Units.Read.quot" : Units: Units in quotation marks, e.g. "Clocks"
	"Units.Read.s" : Units: Plural s allowed
	"Units.Read.ton" : Units: Unit symbol ton
	"Units.Read.u" : Units: Unit symbol u
	"Units.Reduce.C" : Units: Generate C (Coulomb)
	"Units.Reduce.F" : Units: Generate F (Farad)
	"Units.Reduce.H" : Units: Generate H (Henry)
	"Units.Reduce.J" : Units: Generate J (Joule)
	"Units.Reduce.Ohm" : Units: Generate Ohm
	"Units.Reduce.Pa" : Units: Generate Pa (Pascal)
	"Units.Reduce.S" : Units: Generate S (Siemens)
	"Units.Reduce.T" : Units: Generate T (Tesla)
	"Units.Reduce.Wb" : Units: Generate Wb (Weber)
TxNewSetting	[TxOptionsName] determines the possible settings:
	"Func.FFT.Window" : FFT window function

"Rectangle"	Rectangular window.
"Hamming"	Hamming window
"Hanning"	Hanning window
"Blackman"	Blackman window
"Blackman_Harris"	Blackman/Harris window
"Flat_Top"	Flat-Top window

	"Func.FFT.Mode" : Set FFT mode (if waveform length is not a power of 2)

"Truncate"	The waveform length is truncated to the next lower power of 2.
"AppendZeroes"	The waveform length is extended to the next higher power of 2 with appended zeroes.

	"Func.ResultFormat" : Set default data format for results

"Auto"	Automatically
"Float"	4 Byte real
"Double"	8 Byte Real

	"Func.WarnLevel" : Which warning messages are displayed?

"None"	No warnings
"Important"	Important warnings
"All"	All warnings

	"Func.ErrorBoxes" : Error boxes (file functions)

"Yes"	The error is indicated and sequence processing interrupted.
"No"	An error code is returned, if possible. The caller is responsible for interpreting the return value.

	"Func.NoInfoMessages" : Information output

"No"	The results of functions like Stat() or LFit() are displayed in the output window.
"Yes"	No output of results.

"Display.DecimalSeparator" : Decimal separator

"."	Period
","	comma
"region"	Windows-setting (Control Panel / Region)

"Units.Ctrl.Compatible" : Compatibility

"no"	no
"7.0"	Like imc FAMOS 7.0 and predecessors

"Units.Display.Greek" : Font for Greek letters

"keep"	Greek characters in selected font
"symbol"	Use 'Symbol' font

"Units.Display.Ohm" : Ohm

"Omega"	Prefer Greek letter Omega
"Ohm"	Always display string Ohm

"Units.Create.Delim" : Separator character between units, e.g. V*A

"dot"	Period
"*"	*
"blank"	Space

"Units.Create.Nm" : Separator charater for Nm and the like

"no"	no (e.g. Nm)
"yes"	yes (e.g. N*m)

"Units.Create.Pow.1/2" : Use special character for 1/2

"yes"	yes
"no"	no, always write as ^1/2

"Units.Create.Pow.2" : Use special character for ^2

"yes"	yes
"no"	no, always write as ^2

"Units.Create.Pow.3" : Use special character for ^3

"yes"	yes
"no"	no, always write as ^3

"Units.Create.Pow.Neg" : Negative exponents in the denominator

"1/m"	yes (e.g. 1/m)
"m^-1"	no (e.g. m^-1)

"Units.Create.u" : generate u instead of µ (prefix 10^-6)

"no"	no
"yes"	yes

"Units.Create.Num.Space" : Spaces between automatically amended power of 10 and unit

"yes"	yes, e.g. 10^3 V
"no"	no, e.g. 10^3V

"Units.Create.1e3" : 10, 100, 1000

"e3"	give preference to 10^3 etc.
"1000"	give preference to 1000 etc.

"Units.Create.1e-3" : 0.1, 0.01, 0.001

"e-3"	give preference to 10^-3 etc.
".001"	give preference to 0.001 etc.

"Units.Create./s" : Unit with numerator = 1

"default"	regular (e.g. 1/m)
"no1"	omit 1(e.g. /m)

"Units.Read.cal" : Unit symbol cal

"internat"	International 1 cal = 4.1868 J
"thermo"	Thermochemical 1 cal = 4.184 J

"Units.Read.Exp" : Number as exponent

"no"	no, e.g. keep "g CO2"
"try"	Interpret in clear arrangements, e.g. "m2/s2" as "m^2/s^2"

"Units.Read.g" : g

"gram"	Gram
"gravity"	Standard gravity 9.81 m/s^2

"Units.Read.Gs" : Unit symbol for Gauss

"Gs"	Gs (Default)
"G"	G

"Units.Read.hp" : Unit symbol hp

"NIST"	mechanical horsepower, NIST, 745.6999 W
"metric"	metric horsepower, 735.49875 W = 1 PS
"electric"	electric horsepower, 746 W
"hydraulic"	hydraulic horsepower, 745.69988145 W

"Units.Read.kt" : Unit symbol kt

"kiloton"	Kiloton
"knot"	Knots

"Units.Read.L" : Unit symbol L

"Liter"	Liter
"Lambert"	Lambert

"Units.Read.lb" : Unit symbol lb

"Force"	Force, 1 lb = 1 lbs = 4.4 N = 1 lbf
"Mass"	Mass, 1 lb = 1 lbs = 0.45 kg = 1 lbm

"Units.Read.oz" : Unit symbol oz

"Force"	Force, 1 oz = 0.28 N
"Mass"	Mass, 1 oz = 0.028 kg

"Units.Read.pt" : Unit symbol pt

"pintUS"	United States liquid pint, 1 pt = 473.1765 ml
"pintUK"	Imperial pint, UK, 1 pt = 568.26125 ml
"pintDry"	United States dry pint, 1 pt = 550.6104713575 ml
"point"	printer point, 1 pt = 0.35146e mm

"Units.Read.quot" : Units in quotation marks, e.g. "Clocks"

"no"	Don't interpret contents
"try"	Attempt interpretation of contents, e.g. "N" as N

"Units.Read.s" : Plural s allowed

"no"	no (e.g. Volts = V * s)
"yes"	yes (e.g. Volts = V)

"Units.Read.ton" : Unit symbol ton

"short"	ton (short) (2000 lb), US
"long"	ton (long) (2240 lb), UK

"Units.Read.u" : Unit symbol u

"atom"	Unified atomic mass unit
"mju"	µ (Prefix 10^-6)

"Units.Reduce.C" : Generate C (Coulomb)

"no"	no
"yes"	yes

"Units.Reduce.F" : Generate F (Farad)

"no"	no
"yes"	yes

"Units.Reduce.H" : Generate H (Henry)

"no"	no
"yes"	yes

"Units.Reduce.J" : Generate J (Joule)

"no"	no
"yes"	yes

"Units.Reduce.Ohm" : Generate Ohm

"yes"	yes
"no"	no

"Units.Reduce.Pa" : Generate Pa (Pascal)

"no"	no
"yes"	yes

"Units.Reduce.S" : Generate S (Siemens)

	"no"	no
	"yes"	yes
	"Units.Reduce.T" : Generate T (Tesla)	
	"no"	no
	"yes"	yes
	"Units.Reduce.Wb" : Generate Wb (Weber)	
	"no"	no
	"yes"	yes
OldSetting		
OldSetting	Previous value of the option changed.	

Description:

This function enables changing of various default settings' values which are applied in the execution of commands and math functions.

All options which can be set here are initialized with the defaults valid upon activation of imc FAMOS, as they are set in the dialogs "Options/ Folders", "Options/DDE" or "Options/Functions".

These default settings can be changed within the course of a sequence by calling this function.

The option value specified here remains valid until either imc FAMOS is re-started or the respective Options dialog is called and exited with [OK], which resets all defaults to their current values in the dialogs.

This function is supported from Version 4.0 onward. Some of the settings which can be changed here can also be affected by 'old' commands such as FFTOPTION, SDIR, MDIR... But the function SetOption, aside from being easier to use (especially when using the Function Assistant), has the advantage that the old state is returned. This makes it easily possible to reverse changes made, when the sequence has been completed. This is seriously recommended in order to avoid side effects. For newly created sequences, SetOption is generally preferable.

In addition to the values for the 2nd parameter mentioned above, "Reset" can also be specified. This command (re)activates the FAMOS default setting (as set in the dialog "Options") for the respective option.

Multithreading: The function has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

The following sequence fragment first restores the default sequence folder set in the dialog. Then the sub-folder of the set default data folder, "Test1", is set as the new data folder. Next, the system tries to use a loop to load the files "experim1.dat" to "experim10.dat". If a file is not present, no error message is posted (error boxes were already de-activated) and the procedure continues with the next file. After successful file loading, a sequence is called from the default sequence folder for processing the files.

```
SetOption("Dir.Sequences", "Reset")
StdFolder = GetOption("Dir.DataFiles")
SetOption("Dir.DataFiles", StdFolder + "\test1")
SetOption("Func.ErrorBoxes", "No")
i = 1
WHILE i <= 10
   fh = FileOpenDSF("experim"+ TForm(i, ""), 0)
   IF fh > 0
      data = FileObjRead(fh, 1)
      SHOW data
      SEQUENCE Auswertung.seq data
      ; ... further analysis
      err = FileClose(fh)
   END
   i = i+1
END
```

The following subsequence carries out an FFT on the waveform it is given. The waveform is shortened to the next smaller power of 2, and Blackman windowing is used. The results are expressed in Double format (8 Byte Real). Warnings are suppressed. Upon finishing the sequence, the old settings are restored.

```
; Sequence DoFFT
; Call with "SEQUENCE DoFFT MyData"
oldWindow = SetOption("Func.FFT.Window", "Blackman")
oldMode   = SetOption("Func.FFT.Mode", "Truncate")
oldFormat = SetOption("Func.ResultFormat", "Double")
oldLevel  = SetOption("Func.WarnLevel", "None")
Result = FFT(PA1)
...; further analyses
```

```
SetOption("Func.FFT.Window", oldWindow)
SetOption("Func.FFT.Mode", oldMode)
SetOption("Func.ResultFormat", oldFormat)
SetOption("Func.WarnLevel", oldLevel)
```

Data are transfered to EXCEL by means of DDE. The numerical format is set as fixed point with decimal comma and 6 decimal places. First, a column of the table is filled (separator ASCII-13+ ASCII-10 = line break). Next, the separator is set to Tabulator (ASCII-9) and a row of the table is filled.

```
ColData = ...
RowData = ...
SetOption("DDE.Text.NumFormat", "f1.6")
SetOption("DDE.Text.Delimeter", "~013~010")
result = DDESet("EXCEL", "Mappe1", "Z2S1:Z100S1", ColData, 1)
SetOption("DDE.Text.Delimeter", "~009")
result = DDESet("EXCEL", "Mappe1", "Z1S1:Z1S100", RowData, 1)
```

See also:

GetOption, FFTOPTION, MDIR, LDIR, SDIR

SetSegLen

Specifies a data set's segment length.

Declaration:

`SetSegLen ( Data, SvSegLeng )`

Parameter:

Data	Data set whose length is to be specified
SvSegLeng	New segment length; 0 means no segmenting.

Description:

The segment length for a data set is specified. The data set is suplemented with zeroes up to a multiple of the segment length (for scalable integer data formats the unscaled raw values are set to zero).

Examples:

```
IF SegLen?(data) > 0
   SetSegLen(data, 0)
END
```

Any segmenting of the data set is voided.

See also:

SegLen?, MatrixInit, MatrixChangeDim

SetTime

Specifies a data set's trigger time.

Declaration:

```
SetTime ( Data, SvTime )
```

Parameter:

Data	Data set whose trigger time is to be specified
SvTime	New trigger time

Description:

A data set's trigger time is specified. The time value specified must be expressed in the imc FAMOS time format, such as functions like Time?(), TimeSystem?() and TimeJoin() generate.

Examples:

The trigger time of one data set is assigned to another data set:

```
time = Time?(dataA)
SetTime(dataB, time)
```

See also:

Time?, TimeJoin, TimeSystem?, TimeSplit

SetUnit

Specifies a data set's units

Declaration:

`SetUnit ( Data, TxUnit, SvCode )`

Parameter:

Data	Data set whose unit is to be changed
TxUnit	New unit
SvCode	Selection of unit to be set
	0 : X-unit for single-component data. For XY-data, the unit of the X-component. For complex data, unit of the phase/imaginary part.
	1 : Y-unit for single-component data. For XY-data, the unit of the Y-component. For complex data, unit of the magnitude/real part.
	2 : Z-unit
	3 : Unit of the parameter for 2-component data

Description:

Examples:

A data set's Y-unit is queried. If the unit is "W", it is converted to "VA":

```
unitY = Unit?(data, 1)
cmp = TComp(unitY, "W")
IF cmp = 0
   SetUnit(data, "VA", 1)
END
```

See also:

Unit?, ConvertUnit, XUNIT, YUNIT

SetZDel

The increment in the z-direction (Delta-Z) is set

Declaration:

```
SetZDel ( Data, SvZDelta )
```

Parameter:

Data	Data set whose z-increment is to be specified
SvZDelta	New Delta-Z (>0)

Description:

The increment in the z-direction is set. Examples of the use of this value include for the scaling of the z-axis in 3D-displays of segmented data.

See also:

ZDel?, ZOff?, SetZOff

SetZOff

Sets the offset in the z-direction

Declaration:

```
SetZOff ( Data, SvZOffset )
```

Parameter:

Data	Data set whose z-offset is to be specified
SvZOffset	New z-offset

Description:

A data set's initial value in the z-direction is set. Examples of the use of this value include for the scaling of the z-axis in 3D-displays of segmented data.

See also:

ZDel?, ZOff?, SetZDel

Sharpness

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

The sharpness [acum] is calculated from the specific loudness [sone/Bark].

Declaration:

```
Sharpness ( Specific_Loudness, Method ) -> Result
```

Parameter:

Specific_Loudness	The specific loudness over a Bark-scale, calculated by e.g. LoudnessSpectrum(). Its y-unit is sone/Bark, its x-unit is Bark.
Method	Method for calculating sharpness
	0 : DIN 45692:2009-08
	1 : Aures
	2 : von Bismarck
Result	
Result	The result takes the unit "acum".

Description:

The calculation is based on the specific loudness over Barks, which is a plot of the specific loudness N' over a Bark scale.

If the specific loudness is a normal dataset, the resulting sharpness will be a single value.

If the specific loudness is a dataset with segments, the resulting sharpness will be a normal dataset with one sample per input segment.

The x-axis of the specific loudness has a typical range of 0 to 24 Barks. Values beyond this will not be taken into account.

The calculation of sharpness is based on the specific loudness which itself is based on the thirds spectrum. The precision of the sharpness is strongly influenced by the precision of the thirds filter. Even the small tolerances of class 1 filters can cause significant deviations in sharpness: E.g. 0.4dB is similar to 5%, but the influence is non-linear.

Examples:

Sharpness from a microphone signal "mic", measured in Pa over time. Recorded outdoors

```
Thd = SpecThirds_1( mic, 25, 12500, 0)
Thd_dB = dB ( Thd / 2e-5)
N = LoudnessSpectrum( Thd_dB, 2, 0 )
S = Sharpness( N, 0 )
```

Sharpness over time from a microphone signal "mic", measured in Pa over time. The noise is supposed to be almost static; it changes only slowly.

```
Thd = SpecThirds( mic, 25, 12500, 0, -2, 1)
Thd_dB = dB ( Thd / 2e-5)
N = LoudnessSpectrum( Thd_dB, 2, 0 )
S = Sharpness( N, 0 )
```

Specific loudness from a microphone signal "mic", measured in Pa over time. Recorded inside a vehicle. Aures algorithm

```
Thd = SpecThirds_1( mic, 25, 12500, 0)
Thd_dB = dB ( Thd / 2e-5)
N = LoudnessSpectrum( Thd_dB,3, 0 )
S = Sharpness( N, 1 )
```

Third octave spectrum with narrow-band noise at 1kHz and sharpness of about 1.0 acum

```
Thd_dB = [-60,-60,-60,-60,-60,-60,-60,-60,-60,-60,-60,-60,-60,0,20,40,60,40,20,0,-60,-60,-60,-60,-60,-60,-60,-60]
xoffset Thd_dB 14
N = LoudnessSpectrum( Thd_dB,2, 0 )
S = Sharpness( N, 0 )
```

See also:

LoudnessSpectrum, SpecThirds_1

ShockResponseSpectrum

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Shock Response Spectrum, SRS

Declaration:

```
ShockResponseSpectrum ( Acceleration, f_Start, f_Stop, FrequencyLines, Attenuation, Model, MiniMaxi ) -> Result
```

Parameter:

Acceleration	The course of the acceleration in time, the time scaled in seconds.
f_Start	SRS starting frequency. Scaled in Hz. Evaluation begins at this frequency.
f_Stop	SRS stopping frequency. Scaled in Hz. Evaluation ends at this frequency. f_Start < f_Stop. f_Stop should be far less than half the sampling rate, preferably smaller by a factor of 5. For reasons intrinsic to the algorithm, significant imprecision results from very high f_Stop.
FrequencyLines	This number of frequency lines is calculated. >= 1. The frequency lines are distributed over the frequency range with equal logarithmic spacing from f_Start to f_Stop. The number specified should at least be large enough to result in 6 lines per octave.
Attenuation	0 <= Attenuation< 0.9. (attenuation ratio) The system's relative attenuation. Typically 0.05 or 0.01. 0.0 is an undamped system.
Model	model of calculation
	14 : absolute acceleration model. Calculation of all values both primary and secondary
	15 : PVSS: pseudo velocity. Calculation of all values, both primary and secondary
	16 : relative displacement model. Calculation of all values, both primary and secondary
	17 : absolute acceleration model. Only calculation of primary values during the shock
	18 : PVSS: pseudo velocity. Only calculation of primary values during the shock
	19 : relative displacement model. Only calculation of primary values during the shock
	20 : absolute acceleration model. Only calculation of secondary values, after the shock
	21 : PVSS: pseudo velocity. Only calculation of secondary values after the shock
	22 : relative displacement model. Only calculation of secondary values after the shock
MiniMaxi	Stipulates whether minima and maxima are calculated
	0 : The greater of the absolute values of the minimum and the maximum of the system response is found. This is the most common setting.
	4 : positive SRS or maximum positive SRS: The highest positive value is calculated. Zero, if there are no positive values.
	5 : negative SRS or maximum negative SRS: The most negative value is calulated. Its sign is omitted. Zero, if there are no negative values.
Result	
Result	SRS Spectrum

Description:

absolute acceleration model. The result is a spectrum with values of acceleration, where the acceleration is scaled in the same manner as for the accelration-to-time curve, typically in "g" (gravitational acceleration) or "m/s^2". This is the most common model.

relative displacement model. The result is the relative deflection (path) and comes with the unit "m" (Meter). This requires that the acceleration-to-time curve be scaled in "m/s^2", not in "g".

primary SRS or initial SRS. Calculation only during the shock, not after the shock. Minimum and maximum are calculated only for the duration of the given plot of acceleration over time.

secondary SRS or residual SRS: Calculation only after the end of the shock, not during the shock. Minimum and maximum are calcuated only after the end of the supplied acceleration over time.

Normally a MaxiMax calculation is performed. Positive and negative SRS are only required for the analysis of symmetry.

PVSS: pseudo velocity. It is calculated from the relative displacement shock spectrum, which is multplied by 2*PI*f. This requires that the acceleration-to-time plot be scaled in "m/s^2", not in "g".

The function assumes the system to be in steady state initially, which means displacement=0 and velocity=0.

The function determines the maximum displacement of a SDOF with given natural frequency and damping ratio.

The result is plotted over the natural frequency.

Particularly with low start frequencies, it sometimes occurs that the system response to the course of the acceleration over time begins with an oscillation, but neither reaches the maximum or the minimum before the acceleration ends.

Then a calculation of all values, both primary and secondary, shold be performed.

The function uses the input's sampling rate for determining the extreme values.

If the natural frequency is small compared with the sampling frequency, the result may lose accuracy, because sampling does not occur exactly at the extremum. E.g the error can be up to 1% if the ratio of frequencies is 23. Alternatively, the input can be interpolated beforehand or afterward.

No longer supported parameters:

Model=0: absolute acceleration model

Model=1: relative displacement model

Model=2: absolute acceleration model, without appended zeroes

Model=3: relative displacement model, without appended zeroes

Model=4: absolute acceleration model, different calculation

Model=5: PVSS: pseudo velocity

Model=6: relative displacement model, different calculation

Model=7: absolute acceleration model, without appended zeroes, different calculation

Model=8: PVSS: pseudo velocity, without appended zeroes

Model=9: relative displacement model, without appended zeroes, different calculation

Model=10: absolute acceleration model; only secondary

Model=11: PVSS: pseudo velocity; only secondary

Model=12: relative displacement model; only secondary

MiniMaxi=1: The absolute value of the system response maximum is found.

MiniMaxi=2: The lesser of the absolute values of the minimum and the maximum of the system response is found.

MiniMaxi=3: The absolute value of the system response minimum is found.

Examples:

Finds the SRS. The acceleration-to-time curve has a sampling time of 0.1ms and contains 5000 data points. Its y-unit is "m/s^2". The SRS is determined between 5.0Hz and 500Hz with 100 nodes. The attenuation is 0.05. The Maximax-spectrum is determined as the absolute acceleration. The result is subsequently displayed in the curve window in two-logarithmic-axes representation.

```
SRS = ShockResponseSpectrum ( Acceleration, 5.0, 500.0, 100, 0.05, 14, 0 )
```

In this case the frequency is rescaled to correspond to the third-octave scaling convention, in other words (1Hz: 0, 10Hz: 10, 100Hz: 20, ...). For the graphical representation in the curve window, the x-axis has to be set to third-octave axis labelling.

```
SRS = ShockResponseSpectrum ( Acceleration, 5.0, 500.0, 100, 0.05, 14, 0 )
SRS_Thirds = XYof ( log ( SRS.x ) * 10, SRS.y )
```

Numerical example, half sine pulse, 1ms, 1m/s^2. That is a theoretical signal. After the shock there is a constant relative velocity. That signal cannot be produced on a shaker.

```
Acceleration = sin ( ramp( 0, 1e-5, 101 ) * PI2 * 500 )
yunit Acceleration m/s^2
xunit Acceleration s
SRS = ShockResponseSpectrum ( Acceleration, 0.1, 10000.0, 1000, 0.0, 14, 0 )
```

Numerical example, half sine pulse, 1ms, 1m/s^2. However, it begins with a period of constant speed. At the end of the shock the velocity is zero again. The resulting pseudo velocity shock spectrum is determined.

```
in1 = sin ( ramp ( 0, 1e-5, 101 ) * PI2 * 500 )
insum = sum( in1 )
in2 = xdel ( leng ( 0, 50000 ), xdel?(in1))
Acceleration = join ( in2-insum/leng?(in2), in1)
yunit Acceleration m/s^2
xunit Acceleration s
PVSS = ShockResponseSpectrum ( Acceleration, 0.1, 20000.0, 1000, 0.0, 15, 0 )
```

See also:

SDOF_Response

SHOW

A variable is displayed in a free-floating window.

Declaration:

```
SHOW VariableName
```

Parameter:

VariableName	Variable to be displayed

Description

The variable specified as a parameter is displayed. If the variable is a waveform or data group, a free-floating curve window is opened.

If the variable is a text variable, it is displayed in a special text window.

If a window already exists for this variable, it is brought into the foreground.

Wildcards can also be specified in order to display a series of variables. The wildcard '?' stands for exactly one arbitrary character; the wildcard '*' stands for an undefined number of any characters.

```
SHOW *
```

All variables are displayed.

```
SHOW ??
```

All variables whose names are exactly 2 characters long are displayed.

```
SHOW *1a*
```

All variables containing the string '1a' (at the beginning or end) are displayed.

```
SHOW a*:channel?
```

All channels with the name 'channel' belonging to all data groups whose name begins with an 'a', are displayed along with one arbitrary character.

Examples:

A data set is enlarged with spline interpolation and displayed. This data set's maximum value is calculated and displayed.

```
Data = IPol(Data, 3)
SHOW Data
x = Max(Data)
SHOW x
```

See also:

CwLoadCCV, CwNewWindow

signum

Available in: Professional Edition and above

Function extracting a number's sign

Declaration:

```
signum ( x ) -> Result
```

Parameter:

x	x
Result	
Result	Result

Description:

The function returns 1 when x > 0; returns -1 when x < 0; returns 0 when x = 0.

The function only performs comparison with exactly 0.0. If only approximately zero is meant, then the signal can be processed beforehand, e.g with round().

The input data can have events and segments. Equidistant and XY-data are supported; with XY-data, the Y-component is subjected to the calculation.

This function is only implemented for real (not complex) number arguments.

Examples:

```
sign = signum ( x )
```

sin

Sine, trigonometric function

Declaration:

```
sin ( Parameter ) -> Result
```

Parameter:

Parameter	Input data (angle). Allowed types: [ND],[XY].
Result	
Result	Sine of the parameter

Description:

The trigonometric function sin is calculated using radians or degrees, corresponding to the unit of the specified parameter.

The x-coordinate(s) of the results and the parameter are the same.

Remarks

- The parameter of the cos function should not have a unit or the units 'rad', 'degr', or '° '. Any other units will cause an error message to be generated and the unit will be used for the result.
- The parameter may be structured (events/segments).
- The corresponding inverse function is asin().

Examples:

For the corresponding unit, the parameter is interpreted in degrees.

```
one = sin(90 '°')
```

Two options for creating a data set to display exactly the first half-wave of the sine function.

```
NDhalfWave = sin(Ramp(0, PI/100, 100))
NDhalfWave = sin(Ramp(0, 1, 180) * 1 'degree')
```

See also:

cos, tan, asin

SlClip

Slope limiting for a data set

Declaration:

```
SlClip ( Data, SvMaxSlope ) -> Filtrate
```

Parameter:

Data	Data set to be filtered [NW].
SvMaxSlope	Maximum permitted slope dy/dx
Filtrate	
Filtrate	Filtered data set

Description:

Slope limiter, the maximum slope dy/dx between two adjacent values in a data set is clipped. The function can thus be used to filter out measurement values from a data set, which are invalid due to the signal's physically limited rate of change.

The slope at one point is thus defined as the slope to the next measurement value and is calculated as follows:

```
Slope[i] = (y[i+1]-y[i]) / dx
```

If the calculated slope is greater than the maximum slope in one point, the next data point with the specified maximum slope is calculated and the algorithm is continued with the new value.

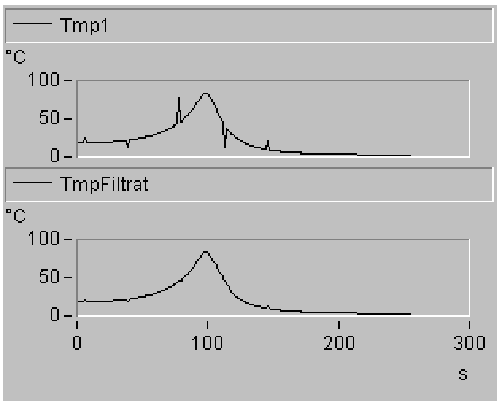
```
y[i+1] = y[i] + SvMaxSlope * dx
```

The data set's first value remains unchanged since it is used as the initial value for the algorithm.

Examples:

The measurement values of a temperature sensor were distorted due to noise in data transfer. Occasionally values occur, which should be filtered out. Temperature changes of more than 4° Celsius per second at one measuring point are not possible.

```
TempFilt = SlClip(Temp, 4)
```



See also:

Smo, Hyst, SearchLevel

Sleep

Pause, processing of the sequence is interrupted for a specified amount of time..

Declaration:

```
Sleep ( SvWaitingTime )
```

Parameter:

SvWaitingTime	Waiting time (in seconds)

Description:

The processing of the sequence is interrupted for the specified amount of time, the function returns only when the interval has elapsed.

While the function is waiting, FAMOS cannot be operated (exception: curve window). However, a running sequence can be stopped in the usual ways ("Sequence"-symbol with context menu in System Tray or [Ctrl+Break]).

During the waiting time, FAMOS is partially accessible via DDE: data can be received, but no instructions can be carried out.

Examples:

All channels in a file of readings are opened in a loop. Each channel is displayed for 5s, then closed and the next channel is opened.

```
fh = FileOpenDSF("c:\tmp\test.dat", 0)
IF fh > 0
   count = FileObjNum?(fh)
   n = 1
   WHILE n <= count
      data = FileObjRead(fh, n)
      SHOW data
      Sleep(5)
      n = n+1
   END
END
```

See also:

PAUSE

sMax

Stat()-function: A data set's maximum value

Description

sMax is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Alternatively, the function Max() can be used.

Examples:

Peak-to-peak value of a sinusoidal voltage signal:

```
Stat(U)
Uss =  sMax - sMin
```

This is equivalent to:

```
Uss =  Max(U) - Min(U)
```

See also:

Stat, Max, sMin

sMaxPos

Stat()-function: Position (x-value) of a data set's maximum value

Description

sMaxPos is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Examples:

Determining a voltage plot's maximum value:

```
Stat(U)
u_max = sMax
t_max = sMaxPos
```

This is equivalent to:

```
u_max = Max(U))
t_max = Pos(U, u_max)
```

See also:

Stat, sMax, Pos, PosiEx2

sMean

Stat()-function: Arithmetic mean of a data set

Description

sMean is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Alternatively, the function Mean() can be used.

Examples:

"Zero-meaning" a data set:

```
Stat(U)
U =  U - sMean
```

This is equivalent to:

```
U =  U - Mean(U)
```

See also:

Stat, Mean, sRMS

sMin

Stat()-function: A data set's minimum value

Description

sMin is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Alternatively, the function Min() can be used.

Examples:

Peak-to-peak value of a sinusoidal voltage signal:

```
Stat(U)
Uss =  sMax - sMin
```

This is equivalent to:

```
Uss =  Max(U) - Min(U)
```

See also:

Stat, Min, sMax

sMinPos

Stat()-function: Position (x-value) of a data set's minimum value

Description

sMinPos is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Examples:

Determining the minimum of a voltage plot:

```
Stat(U)
u_min = sMin
t_min = sMinPos
```

This is equivalent to:

```
u_min = Min(U))
t_min = Pos(U, u_min)
```

See also:

Stat, sMin, Pos, PosiEx2

Smo

Smoothing with a specified averaging time.

Declaration:

```
Smo ( Data, SvWidth ) -> Smoothed
```

Parameter:

Data	Waveform to be smoothed [NW]
SvWidth	Width of the smoothing interval
Smoothed	
Smoothed	Smoothed data set

Description:

The specified data set is smoothed by taking a weighted average over a define interval. The width of this interval is determined through the second parameter: the larger the interval width, the more pronounced the effect of smoothing.

Periodic noise occurring in the data set can be suppressed completely by setting the interval width equal to the period of the noise or an integer multiple of the period.

The Smo() function is a digital filter with a time constant in the same order of magnitude as the interval width.

The weighting function is triangular if the interval width is greater than five points of the data set. The weighting function can only be an odd number of points; otherwise the specified width will be rounded to the next possible value.

The coefficients for the digital filter (triangular filter, preserves the mean value) underlying the smoothing function , which represent the parameters for the smoothing width in points, can most easily be calculated in FAMOS from the filter's impulse response, see example below.

- If the chosen interval is so small that only five points are averaged, the Smo function has the same effect as the Smo5() function; for three points, as the Smo3() function.
- The implemented filter is a non-causal filter with a phase of zero. This means that the position of any slopes will not be distorted; the length of the data set does not change.
- Because this function uses a non-causal filter, the edges of the data set exhibit natural start-up phenomena.
- For filtering near the edges, it is assumed that the data sets are extended with the same values as the edges.
- The interval width may not be greater than the length of the specified data set.
- To suppress only high-frequency distortion, it is more effective to use the Smo function several times over a short interval rather than once over a longer interval.

Fof a filter width of n points, the digital filter can be described by the following differential equation:

$$y_{[k]} = \sum_{i=-\frac{n-1}{2}}^{\frac{n-1}{2}} \frac{2n-4\,|\,i\,|}{n^2+1} * u_{[k+i]}$$

Examples:

```
NDsmooth = Smo(NDdata, 1E-3 's')
```

A data set with a sampling rate of 10-4s is smoothed with a filter of the width 10-3s to reduce many high-frequency distortions.

Determine the filter coefficients as funtion of the smoothing interval (in samples) with the pulse response

```
imp = Ramp(0,1,100)*0
imp[50] = 1
resp = Smo(imp, SMO_INTERVAL_IN_POINTS)
```

Example: Signal with sampling time 0.15s, interval width 1s => SMO_INTERVAL_IN_POINTS = 7

[resp] contains the values which differ from 0: (0.04, 0.12, 0.20, 0.28, 0.20, 0.12, 0.04)

Thus:

```
y[k] = 0.04*u[k-3] +0.12*u[k-2] +0.2*u[k-1] +0.28*u[k] +0.20*u[k+1] +0.12*u[k+2] +0.04*u[k+3]
```

See also:

Smo3, Smo5, DFilt, Median, FiltLP, FiltLpZ, SavitzkyGolay

Smo3

Smoothing over 3 points

Declaration:

```
Smo3 ( Data ) -> Smoothed
```

Parameter:

Data	Data set to be smoothed [ND]
Smoothed	
Smoothed	Smoothed data set

Description:

The data set is smoothed by averaging each group of 3 adjacent points. The underlying digital filter applies the following equation:

```
y[k] = 0.25 * u[k-1] + 0.5 * u[k] + 0.25 * u[k+1]
```

Here k is a running index; u the value of the transferred data set, and y a value of the resulting data set.

This is a non-causal filter which does not change the position of any slopes since its phase is equal to zero.

The length of the data set is not changed by smoothing.

For filtering near the edges of the data set, the data set is assumed to be extended with the same constant values from its end values.

Examples:

This function is applied twice for targeted suppression of high-frequency noise.

```
NDsmooth = Smo3(Smo3(NDdata))
```

See also:

Smo5, Smo, DFilt, Median, FiltLP, FiltLpZ, SavitzkyGolay

Smo5

Smoothing over 5 points

Declaration:

```
Smo5 ( Data ) -> Smoothed
```

Parameter:

Data	Data set to be smoothed [ND]
Smoothed	
Smoothed	Smoothed data set

Description:

The data set is smoothed by taking a weighted average of five adjacent values. The underlying digital filter applies the following equation:

```
y[k]=(1/9)*u[k-2]+(2/9)*u[k-1]+(3/9)*u[k]+(2/9)*u[k+1]+(1/9)*u[k+2]
```

Here k is a running index; u the value of the transferred data set, and y a value of the resulting data set.

This is a non-causal filter which does not change the position of any slopes since its phase is equal to zero.

The length of the data set is not changed by smoothing.

For filtering near the edges of the data set, the data set is assumed to be extended with the same constant values from its end values.

The smoothing effect is more pronounced than from the function Smo3().

Examples:

This function is applied twice for targeted suppression of high-frequency noise.

```
NDsmooth = Smo5(Smo5(NDdata))
```

See also:

Smo3, Smo, DFilt, Median, FiltLP, FiltLpZ, SavitzkyGolay

SolveLinEq

Available in: Professional Edition and above

Solution of a system of linear equations

Declaration:

```
SolveLinEq ( Matrix A, Column vector b [, Error handling] [, MultipleSolution] ) -> Result
```

Parameter:

Matrix A	Matrix A with coefficients of the left side
Column vector b	Column vector b with coefficients of the right side
Error handling	Determines the system response to an error. (optional , Default value: 0)
	0 : Cancel and post error message
	1 : Return empty data set
MultipleSolution	Behavior in response to infinitely many solutions (optional , Default value: 0)
	0 : Return only unique solutions. Infinite amounts of solutions is an error.
	1 : Return both unique and multiple solutions. In the case of infinitely many solutions, one is selected.
Result	
Result	Solution vector

Description:

A matrix is a segmented data set. The segments are the columns.

In this context, a vector refers to a single-row or single-column matrix.

A column vector is a matrix with only one column. It is a data set without segments.

A column vector can also be represented as a segmented data set with exactly one segment.

A * x = b is solved, meaning the value of x is determined. A is a square matrix. x and b are column vectors.

The result of the function is a column vector, meaning a data set without segments.

When called with incorrect parameter values or insufficient memory, the system cancels the operation with the usual error message.

Units and sampling intervals are not taken into account.

Homogenous system of linear equations

The parameter MultipleSolution governs the system response

If only unique solutions are desired, the following applies: Return the trivial solution (all zeroes) only if it is the only one and thus unique. Infinitely many (non-trivial) solutions are an error.

If multiple solutions are also desired, the following applies: if it exists, the non-trivial solution is returned by the ssytem selecting one of the infinite many solutions. For example, the result can be multiplied by any arbitrary factor and is still a solution. If the non-trivial solution does not exist, the trivial solution (all zeroes) is returned.

If the matrix A can be inverted, then only the trivial solution exists, otherwise also the non-trivial.

Examples:

Solution of a sytem of linear equations A * x = b

```
A = leng(0,9) ; test data
setsegLen( A,3)
A[3,1] = 4 ; row 1, column 3
A[1,2] = 1
A[2,3] = 2
; A:
; 0 0 4
; 1 0 0
; 0 2 0
b = [4, 2, -2]
xi = SolveLinEq ( A, b )
; result: xi = [2, -1, 1]
```

See also:

MatrixInverse

Sort

Sorting of a data set's y-values

Declaration:

```
Sort ( Data, SvOption ) -> Sorted
```

Parameter:

Data	Data set to be sorted; allowed types: [ND],[XY].
SvOption	Defines the type of sorting
	1 : [only for ND] output of the ascendingly ordered y-values
	2 : [ND] output of the descendingly ordered y-values
	3 : [ND] Ascendingly ordered y-values; output of associated x-positions
	4 : [ND] Descendingly ordered y-values; output of associated x-positions
	5 : [only for XY] Ascendingly ordered according to Y
	6 : [XY] Descendingly ordered according to Y
	7 : [XY] Ascendingly ordered according to X
	8 : [XY] Descendingly ordered according to X
Sorted	
Sorted	Sorted data set

Description:

The values of a data set are sorted using the quick-sort algorithm; the values can be sorted in either ascending or descending order.

The result data type is the same as the data type of the data set to be sorted; the data format is also retained.

<u>The values 1 through 4 for the parameter SvChoice are only permitted for equidistantly sample data sets (NW)::</u>

By specifying 1 for the parameter [SvOption], the y-values of the data set are sorted and outputted in ascending order.

Selecting 2 for the parameter [SvOption] sorts and outputs the y-values of the data set in descending order.

When 3 is specified for the [SvOption] parameter, the y-values of the data set are sorted in ascending order and the (non-sorted) x-positions of these y-values are output. The x-position of the smallest y-value is output first, followed by the x-position of the second-smallest y-value, etc. The x-position of the largest y-value is outputted last.

When 4 is specified as the [SvOption] parameter, the y-values of the data set are sorted in descending order and the (non-sorted) x-positions of these y-values are output. The x-position of the largest y-value is output first, followed by the x-position of the second-largest y-value, etc. The last value outputted is the x-position of the smallest y-value.

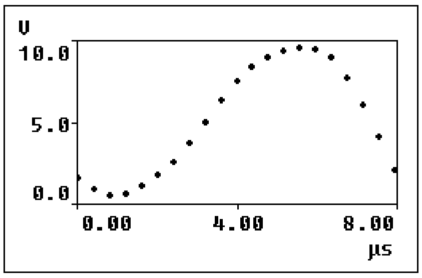
<u>The values 5 through 8 are only permitted for XY-data:</u>

For 5 or 6, the XY-data set is sorted in such a way that the Y-values in the result are arranged ascending/ descending, respectively.

For 7 or 8, the XY-data set is sorted in such a way that the X-values in the result are arranged ascending/ descending, respectively. Option 7 is particularly useful for transforming an XY-data set with a non-monotonic X-track into a data set with a monotonic X-track.

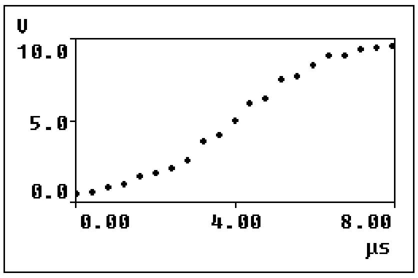
Examples:

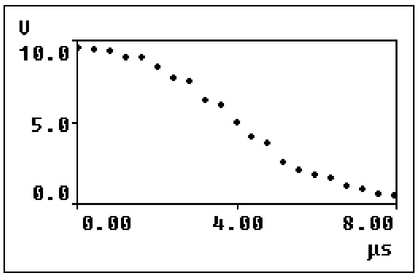
The data set illustrated below is to be sorted. It consists of 21 values:



```
NwUp = Sort(NwData, 1)
NwDown = Sort(NwData, 2)
```

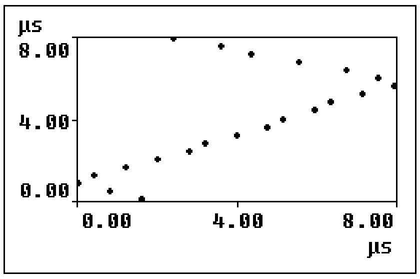
The result of sorting with the parameter [SvOption] specified as 1 (ascending sorting of y-values) is outputted in imc FAMOS as the data set [NDUp], displayed at the left. When the [SvOption] parameter is specified as 2 (descending sorting of y-values), the result is outputted as the data set [NDDown], displayed at the right:

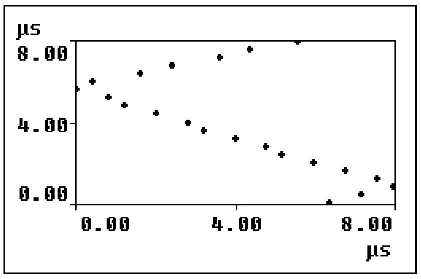




```
NwUpX = Sort(NwData, 3)
NwDownX = Sort(NwData, 4)
```

Selection of 3 as the [SvOption] parameter (associated x-positions of y-values in ascending order) results in the data set [NDUpx], displayed in the illustration at the left. Selecting 4 for the parameter [SvOption] (associated x-positions of the y-values in descending order) results in the data set [NDDownx], displayed in the illustration at the right:





See also:

Mirror, Top

SoundIndex

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Articulation index and others characteristics of a sound signal. Calculation over the entire sound signal.

Declaration:

```
SoundIndex ( Sound signal, Calculation ) -> Result
```

Parameter:

Sound signal	Time-plot of sound signal, the time scaled in seconds
Calculation	Calculation
	0 : Articulation Index (0 .. 100%)
	1 : Open Articulation Index (0 .. 224.78%)
Result	
Result	

Description:

To calculate the articulation index, the input data must already be scaled according to the reference pressure (e.g. 2e-5 Pa) for the purpose of the internal calculation of dBs.

The articulation index (AI) was introduced by Leo L. Beranek for evaluating the quality of telephone equipment. In the early 70s B. Braune modified the algorithm for acoustics in the automotive area, so that it could be calculated from a 1/3-octave spectrum. The 1/3-octave spectrum is compared with two empirically determined reference curves. For each frequency band one percentage is calculated.

The articulation index results from the sum of the individual proportions from 200Hz to 6300Hz. The intelligibility is very good if AI=100% but becomes bad as AI approaches 0%.

For instance, the reference curves values are 44.0 and 74.0 dB(A) at 1000Hz, and 23.1 and 53.1 dB(A) at 200Hz.

The open articulation index differs from the articulation index only in the lower reference curve, which is 0dB for all frequencies.

The articulation index is calculated internally by means of a call of SpecThirds_1 (..., 1).

Examples:

```
AI = SoundIndex( vibration / 2e-5'Pa', 0 )
```

See also:

LoudnessLevel

SoundIntensityThirds

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Sound intensity over time (per 1/3 octave).

Declaration:

```
SoundIntensityThirds ( Sound_pressure_1, Sound_pressure_2, f1, f2, Frequency weighting, Time rating,
OutputInterval, Spacer, AirDensity [, Filter start-up] ) -> Result
```

Parameter:

Sound_pressure_1	Time-plot of sound pressure signal in Pa at Microphone 1; the time scaled in seconds
Sound_pressure_2	Time-plot of sound pressure signal in Pa at Microphone 2; the time scaled in seconds
f1	Center frequency of lowest 1/3 octave in Hz
f2	Center frequency of highest 1/3 octave in Hz
Frequency weighting	Frequency weighting for the result
	0 : linear
	1 : A-weighting
	2 : B-weighting
	3 : C-weighting
	4 : D-weighting
Time rating	Averaging, time rating of sound intensity
	-5 : mean value in interval
	-6 : mean value from start
OutputInterval	Output interval; 1/3 octave spectra are calculated using this time increment. Integer multiple of the sound pressure's sampling time. Specified in seconds.
Spacer	Distance between the microphones, scaled in m; > 0; normally in the range of a few cm
AirDensity	Density of the air stated in kg/m^3. E.g. 1.204 (at 0°C) or 1.2038; > 0
Filter start-up	Treatment of the transients at the beginning of the calculation (optional , Default value: 0)
	0 : During the transients, the result is set to zero in case the 1/3-octave is still experiencing transients.
	1 : The first valid result of each 1/3-octave is extended forward as a constant.
	2 : The region in which any filter is still in transients is truncated. The length is shortened, the starting time point of the result is shifted.
Result	
Result	Sound intensity referenced to time

Description:

Method

Sound intensity is determined by a two-microphone-based technique. For this purpose, an intensity probe is used in which two microphones are mounted at a small and fixed distance from each other.

The course of the sound pressure over time at each microphone is provided as a signal.

The calculation is performed as the multiplicative product of the sound pressure and the sound particle velocity.

The sound pressure used is the average of the sound pressures at the two microphones.

The sound particle velocity is determined by integration of the pressure difference. The pressure difference is an approximation of the derivative of the sound pressure appearing in the Euler equation.

To prevent the calculated integral from drifting away due to initial values and (small) offset errors, and thereby (strongly) distorting the result, a high-pass filter is applied.

For the purpose of the output, an average is taken, for which the parameter Time Weighting sets the averaging region.

All result values are intensities and stated in W/m^2 if the input data are scaled in Pa over s.

Sign

The calculated sound intensity is an average with an arithmetical sign.

The sign expresses the direction.

The sign is positive when the energy first hits the first microphone and next the second. This means it is positive when Microphone 1 is aimed at the source.

Limitations of the procedure

The microphone distance determines what frequency range is possible for the analysis:

When frequency components become very low, they are no longer possible to evaluate precisely because their phase difference is either very difficult or impossible to determine.

When frequency components become very high, they are no longer possible to evaluate precisely because their wavelength approaches the size of the distance between microphones.

The approximation of the derivative by a pressure difference, mentioned above, only works well when the frequency components are significantly less than 1/4 of the sampling frequency.

1/3-octave-based analysis

The calculation is performed for each 1/3-octave: First the 1/3-octave filtering is performed, next the calculation of the sound intensity.

The two frequency boundaries f1 and f2 should be specified as 1/3-octave midband frequencies, e.g. f1 = 63 Hz and f2 = 5000 Hz. f1 <= f2.

The top 1/3-octave along with its frequency band must lie completely within one-half of the sampling frequency.

The top 1/3-octave should lie clearly within 1/4 of the sampling frequency.

The duration of transient oscillations for the 1kHz 1/3-octave is assumed to be 20ms.

This duration is inversely proportional to the 1/3-octave frequency.

For very low 1/3-octaves, this duration becomes substantial.

The duration of the measurement must be set accordingly.

The result is a segmented data set. Each segment contains one 1/3-octave spectrum.

The result?s x-coordinate counts the 1/3-octaves (just like the FAMOS-function OctA).

To achieve a useful display in the curve window, 1/3-octave labeling should be selected there.

The z-coordinate of the result represents the time. The time difference between the segments, delta-z, corresponds to the parameter "OutputInterval"

The 1/3-octave filter and analyses are in accordance with IEC 651 (sound level measurement), DIN 45652 (1/3-octave filter for electro-acoustic measurements) and EN61260-1:2014 or IEC61260-1:2014 (band-pass filter for octaves and fractions of octaves, filter Class 1).

In addition to the signal transients due to the 1/3-octave filter, the high-pass filter used also causes transient oscillations.

If f1=f2, then for this 1/3-octave, a normal time data set is generated, not a segmented one.

Total sound intensity

The input signal?s sound intensity is determined. There is no 1/3-octave filtering. f2=-1 must be specified.

f1 is the frequency used for the high-pass filtering performed internally. f1 lies within the range [SamplingFrequency/1e2] .. [SamplingFrequency/1e6].

Only Frequency Weighting = linear is supported. For instance, if A-weighting is desired, it is necessary to first perform ABCRating() for each of the two microphone signals.

The signal transients are determined by the high-pass filter. Their duration is inversely proportional to the filter?s cutoff frequency.

The result is a time-domain channel. The output interval specified is its sampling interval.

dB

If conversion to dB is desired, it can be performed subsequently.

The mathematical sign which indicates the direction is determined previously and needs to be handled separately. See example.

With the dB calculation, you must consider that the calculation is in terms of 10*log(), as with the power.

With the dB calculation, you must consider that the magnitude of the result can also be < 1E-12, which is equivalent to negative dB-values. It is often helpful to limit these values to 0 by means of UpperValue().

As the reference value for the dB calculation in air, 1E-12W/m^2 is generally used.

Notes

The signal propagation delay between the two microphones is of decisive importance for the analysis. All filters and pre-processing applied to the microphone signals, which affect the signal phase, must always proceed identically for both channels.

The transient oscillation procedure depends on what filters are used.

During the transient oscillation procedure, the intensities are assumed to be 0.0.

Input data are allowed to have events.

If a single value is desired for the sound intensity for the entire measurement, the time weighting selected is =-6. The last value of the result is read.

Examples:

We want to determine the sound intensity from 2 microphone signals p1 and p2. The microphone distance is 12mm.

The sampling frequency is 50kHz. Every 100ms, one result is desired for the 1/3-octaves 50Hz through 5kHz with A-weighting.

The sign representing the direction is to be calculated separately. The sound intensity is to be stated in dB.

```
omega_t = ramp(0,2e-5,50000)*(500*PI2)
p1=sin(omega_t)
p2=sin(omega_t-0.1)
in10c = SoundIntensityThirds(p1, p2, 50, 5000, 1, -5, 0.1, 0.012, 1.204, 0)
Sign = signum(in10c)
in10c_dB = 0.5 * dB(in10c / 1E-12)
```

P1 is nearer to the source. Therefore the sign representing the direction is positive.

The sound intensity is to be determined from the 2 microphone signals p1 and p2. A high-pass is applied at 3Hz; the signal from 50Hz on is of interest.

Subsequently, conversion to dB, where the sign attached to the dB-value represents the direction.

```
omega_t = ramp(0,1e-5,100000)*(1000*PI2)
p1=sin(omega_t)
p2=sin(omega_t-1*PI2/360)
in10c = SoundIntensityThirds(p1, p2, 3, -1, 0, -5, 0.1, 0.012, 1.204, 0)
in10c_dB_sign = signum(in10c) * UpperValue(0.5 * dB(in10c / 1E-12), 0)
```

See also:

SpecThirds

SpeakConfig

Configuration of the voice output using the function SpeakText().

Declaration:

```
SpeakConfig ( TxVoice [, TxRequestedLanguage] [, TxRequestedGender] [, TxRequestAge] [, SvVolumne] [, SvRate] )
-> SvSucess
```

Parameter:

TxVoice	Name of the speaking voice to be selected. For empty text, the speaking voice is selected according to the next three parameters.
TxRequestedLanguage	Name of the preferred output language. Only taken into account if [TxVoice] is empty. For the syntax of the parameter, see below; typical values include "en" or "de" for English or German, respectively. (optional , Default value: "")
TxRequestedGender	Preferred gender of speaking voice. Only taken into account if [TxVoice] is empty. (optional , Default value: 0)
	0 : no preference
	1 : male
	2 : female
	3 : neutral
TxRequestAge	Preferred age of speaking voice. Only taken into account if [TxStimme] is empty. In Windows, by default only voices of normal age timbre (adult) are installed, so that this parameter usually can be left at the setting 0 (=no preference). (optional , Default value: 0)
	0 : no preference
	1 : normal
	2 : older
	3 : youthful
	4 : childlike
SvVolumne	Specifies the desired volume of voice output. Expressed in percent of the system's volume setting for the default audio playback device (permitted range: 0..100). (optional , Default value: 100)
SvRate	Specifies the desired speaking rate. The permitted values range from -10 (very slow) to 10 (very fast).The default value is 0 (normal talking speed). (optional , Default value: 0)
SvSucess	
SvSucess	Success of the function: 1, if the function could be performed successfully; 0 on error. In case of error, the cause can be determined using the function GetLastError().

Description:

This function serves the purpose of configuring the voice output for subsequent calls of the function SpeakText().

You can select the speaking voice as well as adjust the volume and playback speed.

By default, the Windows-Desktop Control Panel's settings for Text-to-Speech conversion are applied ("Control Panel"->"Ease of Access"->"Speech Recognition"->"Text-to-Speech").

Selection of speaking voice

There are two basic options for selecting the speaking voice:

Explicitly specifying the name in the parameter [TxVoice]

Searches for the name among the voices installed; the function's next 3 parameters are then ignored. If there is no voice having this name, the function cancels and returns 0. The list of available voices is found in the Windows-Desktop Control Panel ("Control Panel"->"Ease of Access->"Speech Recognition"->"Text-to-Speech"). The entries in the list "Voice selection" take the form "Name - Language". For example, if a list entry reads "Microsoft Zira Desktop - English (United States)", then the name to specify here is "Microsoft Zira Desktop". It is recommended to use the FAMOS Functions Assistant, which lists the languages available. Enter "Default" in order to restore the default voice selected in the Control Panel.

Automatic setting based on desired voice characteristics

Toward this end, [TxVoice] must be passed as an empty text and the next 3 parameters must be filled accordingly. The language to be used in the parameter [TxRequestedLanguage] has the highest priority . If the desired language can not be obtained, the previous voice setting is retained. If multiple voices in the specified languages are available, an attempt is made then the system attempts to obtain the specified voice gender and age in succession. If this is not possible, the default voice in the language selected is used.

Format of [TxRequestedLanguage]: Enter an empty string to select the current system language. Otherwise, the language is specified by language identifier in the form "xx-yy". The identifier consists of two characters for the language code as per ISO 639 and (optionally) 2 characters for country/region-code as per ISO 3166. Examples:

Identifier	Meaning
"de"	German
"en"	English. No region preference
"en-US"	English, USA pronunciation
"ja"	Japanese
"zh-CN"	Chinese - People's Republic of China
"zh-TW"	Chinese - Taiwan

If all parameters for the language selection are entered as empty text or 0, the language currently set is not changed and only the volume and if applicable the talking speed are changed.

The settings made here remain intact until the next restart of a Top-Level-sequence or the menu item 'Restart'.

Calling this function stops any asynchronous voice output still running, which had previously been started by means of SpeakText().

As an alternative to this command, it is possible to specify the text to be outputted in SpeakText() in the SSML-format. With SSML, you can explicitly govern such aspects as the language, pronunciation, and talking speed by means of tags in the text.

Multithreading: The function has a global effect. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL therefore share a common memory.

Examples:

For texts not expressed in the system language, it is recommended to explicitly specify the language:

```
; English edition:
SpeakConfig( "Microsoft Zira Desktop")
; or e.g.
; SpeakConfig( "", "en")
SpeakText("Sequence is finished")
; German edition:
SpeakConfig( "Microsoft Hedda Desktop")
; or e.g.
; SpeakConfig( "", "de")
SpeakText("Sequenz ist fertig.")
```

Output in English is desired. A female voice is preferred:

```
SpeakConfig( "", "en", 2)
SpeakText("I prefer a female voice")
```

The volums is set to 50% of the system volume; playback slightly slower than normal:

```
SpeakConfig( "", "", 0, 0, 50, -5)
SpeakText("Not so loud and slower, please!")
```

The default settings in the Control Panel are restored:

```
SpeakConfig( "standard")
```

See also:

SpeakText

SpeakText

Voice output of the specified text via the default audio playback device

Declaration:

```
SpeakText ( Text [, Mode] ) -> SvSucess
```

Parameter:

Text	The text to play back
Mode	Output mode (optional , Default value: 0)
	0 : Output is performed asynchronously; the function's return jump is immediate, without waiting for the end of the voice output. If asynchronous output is currently active, the text is placed in a waiting list and outputted once all "older" texts have been processed..
	1 : The function waits until output is concluded. The output begins immediately; any asynchronous output currently running is interrupted.
SvSucess	
SvSucess	Success of the function: 1, if the function could be performed successfully; 0 on error. In case of error, the cause can be determined using the function GetLastError().

Description:

The specified text is played by the default audio playback device. Windows' built-in Text to Speech ('TTS') is used.

By default, the voice specified in the Window's desktop Control Panel is used ("Control Panel"->"Speech"->"Text-to-Speech"). The voice, volume and playback speed can be adjusted using the function SpeakConfig().

An asynchronous playback started here is concluded if:

- SpeakText() is called with an empty text-parameter
- SpeakText() is called with the mode='synchronous'
- SpeakConfig() is called
- a TopLevel-sequence is started
- the menu item "Restart" is selected
- FAMOS is closed

Errors which occur in asynchronous output only after a delay can, of course, not be reported by the function's return value.

The text to be outputted can be restructured or outputted in the **SSML-format** (Speech Synthesis Markup Language, Version 1.0, https://www.w3.org/TR/speech-synthesis). With SSML, you can explicitly govern such aspects as language, pronunciation, volume and talking speed. The function automatically detects SSML by checking the first 100 characters for the string '<speak'.

Example of a typical header of an SSML-string:

```
<?xml version="1.0"?>
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"&#160;xml:lang="en-US">
```

The 'speak'-tag must at least contain the attributes 'version' and 'xml-lang'. The parts not underlined here are optional.

Examples:

The end of a longer evaluation can be announced by an appropriate auto-message, and a calculated result read out loud:

```
_max = ...
...
SpeakText("Sequence is finished.")
SpeakText("The calculated maximum is "+ TForm( _max,""))
```

For texts not expressed in the system language, it is recommended to explicitly specify the language:

```
; English edition:
SpeakConfig( "Microsoft Zira Desktop")
; or e.g.
; SpeakConfig( "", "en")
SpeakText("Sequence is finished.")
; German edition:
SpeakConfig( "Microsoft Hedda Desktop")
; or e.g.
; SpeakConfig( "", "de")
SpeakText("Sequenz ist fertig.")
```

Synchronous output of an English text in accordance with SSML. The text consists of 2 sentences. Between the two sentences, a pause is inserted;

the 2nd sentence is played back at high speed:

```
; SSML start- and end-tags for English language: <speak version="1.0" xml:lang="en">...</speak>
ssmlHeader = "<speak version=~0341.0~034 xml:lang=~034en~034>"
ssmlFooter = "</speak>"
; The tag <p>...</p> marks a paragraph in the text:
ssmlText = "<p>This is the first paragraph. There should be a pause after this text is spoken.</p>"
; The tag <prosody rate="fast">...</prosody> causes higher playback speed:
ssmlText = ssmlText+ "<p><prosody rate=~034fast~034>Speak faster.</prosody></p>"
SpeakText(ssmlHeader + ssmlText + ssmlFooter, 1)
```

Here, SSML is used to output a date in English in long form (month name pronounced):

```
ssmlHeader = "<speak version=~0341.0~034 xml:lang=~034en~034>"
ssmlFooter = "</speak>"
; pronouncement of a date as "January 29, 2009" by means of <say-as type="date"..
ssmlText = "<say-as type=~034date:mdy~034>"+  "1/29/2009" + "</say-as>"
SpeakText(ssmlHeader + ssmlText + ssmlFooter)
```

See also:

SpeakConfig

Spec

Calculates a meaningful spectrum (Spectral lines as amplitudes)

Declaration:

```
Spec ( Data [, SvWindow] [, SvMode] ) -> Spectrum
```

Parameter:

Data	Waveform whose spectrum is to be calculated [NW, CX].
SvWindow	Window-function (optional , Default value: -1)
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman/Harris
	5 : Flat-Top
	-1 : The global preset (dialog: 'Options'/'Functions', SetOption(), FFTOPTION) is used.
SvMode	Procedure (optional , Default value: -1)
	0 : Radix 2: Before the calculation, the length of the data set is reduced (truncated) to the next lower power of 2.
	1 : Radix 2: Before the calculation, the data set's length is increased (by adding zeroes) up to the next higher power of 2.
	2 : Radix 2: Before the calculation, the data set is re-sampled so that the data set's length becomes a power of 2. This corresponds to a previous application of the function Red2().
	3 : Mixed-Radix: The length of the data set needs to be a product of powers of the numbers 2, 3 and 5. Other data set lengths can not be processed.
	-1 : The global preset (dialog: 'Options'/'Functions', SetOption(), FFTOPTION) is used. Compatibe with older versions of FAMOS (< V2022).
Spectrum	
Spectrum	Spectrum of the data set.

Description:

This function calculates a meaningful spectrum in which the spectral lines can be interpreted as amplitudes. If a normal (real) data set is specified, the spectrum is determined as with the FFT() function, except that it is automatically followed by any necessary correction.

The spectrum indicates the amplitudes and phases of the harmonics contained in the signal. The values of the spectrum are sorted by frequency, with the first value corresponding to a frequency of 0, i.e. the mean value of the signal.

The remaining values characterize the basic frequency and harmonics of the signal. A cosine-shaped signal with an amplitude 1 creates an amplitude of 1 and a phase of 0° in the spectrum. A sine-shaped signal also creates an amplitude of 1, but a phase of -90°.

If the signal contains the sum of multiple sine-shaped signals, the amplitudes and phases of the individual components can be recognized at the corresponding frequencies of the spectrum.

When a complex data set is provided, only a correction is performed in which all middle spectral lines are increased by a factor of 2. This generates the same result, content-wise, for all complex data types. However, this requires the last harmonic of the complex data set to actually be real, as is the case when using the FFT on input data with even number lengths, for example. For an FFT having an odd number input length (possible with the Mixed-Radix-procedure), the last harmonic is complex and the last spectral ine must also be doubled.

It is important to note that the calculated spectrum is discrete, not continuous. This means that the spectrum is calculated only for certain discrete frequencies, i.e. all frequencies at which a whole number of periods are contained within the calculation interval. If the FFT is executed on a data set with the length 0.5 s, the spectrum is determined at the frequencies 0, 2 Hz, 4 Hz, 8 Hz ....

An important concept in this context is that a discrete spectrum is the spectrum of a periodic signal. In calculating the spectrum, the signal is interpreted as if it consisted of a long chain of identical signals. If the actual signal represents only one pulse, the spectrum is calculated not for this pulse, but for a signal containing a series of pulse identical to it. This periodic concatenation is noticeable whenever a non-integer number of periods are contained in a signal. For example, a data set contains 3.5 periods of a sine-shaped signal. One single harmonic would be expected as the spectrum. However, since the signal is extended periodically for the spectrum calculation, the signal no longer appears as a sine wave, but as a series of sine-shaped sections, which appear to have been put together incorrectly. The spectrum of this signal will include many harmonics, which are particularly strong in the vicinity of the actual frequency.

This apparent disadvantage of the Spec function can be avoided in two ways.The first option is to specify a section of the signal which contains an integer number of periods before calculating the spectrum, a standard task for periodic signals. The other option is to use a window function. These two options can also be combined.

A window function assigns different significance to different values of a data set. The edges of the signal are especially weak, the center values influence the result very strongly.

Six different window functions are available:

- Rectangle
- Hanning
- Hamming
- Blackman
- Blackman-Harris.
- Flat Top

The rectangular window evaluates all values equally, representing the standard setting for which no extra calculation is required. The Flat Top window at the end of the list represents the other extreme. The "windowing" effect increases from the top to the bottom of this list. The effect of the window function is demonstrated using a 3.5-period sine signal:

The spectrum exhibits additional harmonics around the actual frequency, whose amplitude decreases with distance from the actual frequency. The rectangular window creates a noticeable, sharp peak around 3 or 4, but a distorting influence on the spectrum is noticeable even at greater distances. Using other windows will increase the width of the peak, but the greater the distance from the actual frequency, the less distortion is apparent. Each window function represents a compromise between the two extremes.

To differentiate between harmonics located very close to one another, the rectangular window is appropriate; to ascertain the exact amplitudes and phases of harmonics located further away from one another, use the Blackman window.

Differences from the "FFT" function:

The FFT() function determines a spectrum according to the FFT algorithm, whereby the only automatic processing of the signal is the removal of the negative side of the spectrum.

Now in order to be able to interpret the spectral lines as amplitudes of oscillations, the function Spec() performs a certain correction of the spectrum. Toward this end, all complex harmonics are multiplied by the factor 2.

For an input data set of even-number length, all spectral lines except the first and last are multiplied by the factor 2.

If the length of the input data set is an odd number (only possible with the Mixed-Radix-procedure), the last harmonic is also complex, and all spectral lines except the first are multiplied by 2.

- If further calculations are to be performed with a spectrum, use the FFT() function. Using the inverse FFT or multiplication produce incorrect results for spectra created with the Spec() function.
- The **Spectral package** (included in the Professional/Enterprise edition) contains additional functions which are substantially more powerful and closely adapted to the requirements of signal analysis, for the calculation of spectra. Example: **AmpSpectrumRMS()** calculates an amplitude spectrum (harmonics as RMS-value), with moving windows and linear averaging.
- The optional parameters for the window function and mode are suported as of FAMOS 2022. If these parameters are omitted, then as with earlier versions the global settings set by means of the dialog: 'Options'/'Functions', by means of the command FFTOPTION or by means of the function SetOption() are in force.
- Spec() can be used to process spectra calculated with the FFT() function to correctly interpret the amplitudes.
- The optional parameters for the window function and mode are supported as of FAMOS-Version 2022. If these parameters are omitted, then as with earlier versions the global settings which are set by means of the dialog 'Options'/'Functions', or by the command FFTOPTION, or by the function SetOption() are in force.
- When the length of the data set exceeds 2^27, a warning message is generated and calculation is not possible. Use the functions Leng or Red2 to shorten the data set. The maximum number of points which can be processed is 134.217.728, resulting in a complex data set with a length of 4.194.304.
- The FFT function accessed by the Spec function to process specified equidistant data sets requires temporary working memory for execution. If insufficient memory is available (especially in longer data sets), a warning message is generated.
- The Spec function implemented here only supplies the positive side of the spectrum. When using real data sets, the spectrum is symmetrical, so that the other half does not contain any additional information. representing negative frequencies or frequencies greater than half of the sampling rate is meaningless.
- If the data set provided has N points and if N is even, then the spectrum generated has N/2 + 1 points. Since the first and last value (center value and highest harmonic) are always only real while the other spectral lines are complex, it follows that the spectrum has exactly as many significant digits as the data set contains, so that no information has been lost. If N is odd (only possible with Mixed-Radix procedure), then the spectrum generated has (N-1)/2 + 1 points. The highest harmonic is then complex.
- The **Mixed-Radix-procedure** has the advantage of returning spectra with "round" frequency line distances from data sets having common sampling frequencies (e.g. 1kHz and multiples in the pattern1/2/5) and an appropriate selection of the input length. For a data set sampled at 1kHz, for instance, and a length of 1000 samples (1000 = 2^3 * 5^3), the procedure returns a spectrum having a frequency line distance of 1Hz; for a length of 20000 samples (= 2^5 * 5^4), the frequency line distance is 50mHz.
- The x-unit of the spectrum is always Hz; the y-unit of the magnitude corresponds to that data set. The phase is given in degrees.
- The x-offset of the specified data set should be zero. If it is not, a warning message is generated and the x-offset is treated as zero. To include the influence of an x-offset in the phase, adjust the phase after the spectrum calculation, correcting it with a linear function.

Examples:

```
MPspec = Spec(NWdata, 1, 0)
```

Finds a data set's spectrum in order to be able to derive the signal's upper harmonic amplitudes from the spectrum (Hamming window; truncated if necessary to the next-lower power of 2).

```
First1000Samples = CutIndex(data1kHz, 1, 1000)
Spec_with_1Hz_distance = Spec(First1000Samples, 0, 3)
```

The amplitude spectrum of the first 1000 samples of adata set sampled at 1kHz is calculated. The result has 501 spectral lines at the frequencies 0, 1, 2, ... 500 Hz.

```
; Pay attention when the input length is odd:
First125amples = CutIndex(data1kHz, 1, 125)
Spec_with_8Hz_distance = Spec(First125Samples, 0, 3)
```

The amplitude spectrum of the first 125 samples of a data set sample at 1kHz is calculated. The result has 63 spectral lines at the frequencies 0, 8, 16,... 496 Hz. The last spectral line is complex and is not located at half of the sampling frequency.

```
maxValue = Max(Cmp1(Spec(NWdata, 0, 1)))
```

Determines the greatest harmonic in the spectrum.

```
NDwrong = iFFT( Spec(NDdata))
```

Warning! Serious error! The inverse FFT can only be used on spectra whose amplitudes have not been corrected, i.e. those calculated with the FFT function.

See also:

FFT, iFFT, Red2, AmpSpectrumPeak, AmpSpectrumPeak_1, AmpSpectrumRMS_1

SpecThirds

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

From the time-histories of a vibration, the 1/3 octave spectrum referenced to time is determined.

Declaration:

```
SpecThirds ( Vibration, f1, f2, Frequency weighting, Time rating, Output interval ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal, the time scaled in seconds
f1	Center frequency of lowest 1/3 octave in Hz
f2	Center frequency of highest 1/3 octave in Hz
Frequency weighting	Frequency weighting for the result
	0 : linear
	1 : A-weighting
	2 : B-weighting
	3 : C-weighting
	4 : D-weighting
Time rating	Averaging time, time rating of filtered data. How is the floating RMS calculated?
	-1 : Fast (0.125s)
	-2 : Slow (1s)
	-3 : Pulse
	-4 : Peak
	> 0.0 : Time constant is defined by the user, given in seconds
Output interval	1/3 octave spectra are calculated using this time increment (delta-time, dt). Integer multiple of the vibration's sampling time. Specified in seconds.
Result	
Result	1/3 octave spectrum referenced to time

Description:

The two frequency limits f1 and f2 are to be given as the 1/3-octave center frequencies, e.g. f1 = 8 Hz and f2 = 12500 Hz. f1 < f2. The upper 1/3-oct. with its frequency band must lie entirely within half of the sampling frequency.

The individual 1/3-octave values are stated as root mean square (RMS) values.

While the transients in the individual 1/3-octave (band-pass) filters are subsiding at the start of the measurement, the input signal's values are ignored. The transient time for the 1kHz 1/3-octave is is assumed to be 20ms. This time interval is inversely proportional to the 1/3-octave frequency. For very low 1/3-octaves, this time interval becomes considerable. A correspondingly long measurement must then be expected.

During the settling phase the RMS is assumed to be 0.0.

The result is a segmented data set. Each segment contains one 1/3 octave spectrum. The x-coordinate of the result counts the 1/3-octave bands (just like the Famos-function OctA). For good representation in the curve window, 1/3-octave labeling should be selected.

The z-coordinate of the result represents time. The first spectrum is taken where time is equal to the output interval.

The 1/3-octave filter and analyses are in accordance with IEC 651 (sound level measurement), DIN 45652 (1/3-octave filter for electro-acoustic measurements) and EN61260-1:2014 or IEC61260-1:2014 (band-pass filter for octaves and fractions of octaves, filter Class 1).

Time rating "Pulse": For increasing amplitudes the time constant is 35ms, for decreasing amplitudes 1.5s. Thus impulse-shaped signals are captured quickly, the response decays slowly.

Time rating "Peak": Extreme response for very short impulses; ensuring capture of the peak value. Time constant is zero during increasing amplitude (can be performed exactly by computer, by analog operation only in approximation); during decreasing amplitude 3s.

Examples:

The 1/3 octave spectrum is to be calculated from the time-history of the vibration every 0.1s. The signal is sampled every 0.025 ms.

```
fEval = 1 ; 0 (linear) 1 (A-weighting)
f1 = 10
f2 = 12500
```

```
tEval = -1 ; -1 (Fast)
tInterval = 0.1 ; [s]
Thirds = SpecThirds ( vib, f1, f2, fEval, tEval, tInterval )
```

See also:

OctA, OtrRpmThirds, SpecThirds_1

SpecThirds_1

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

From the time-histories of a vibration, the averaged 1/3 octave spectrum determined.

Declaration:

```
SpecThirds_1 ( Vibration, f1, f2, Frequency weighting ) -> Result
```

Parameter:

Vibration	Time-history of vibration signal, the time scaled in seconds
f1	Center frequency of lowest 1/3 octave in Hz
f2	Center frequency of highest 1/3 octave in Hz
Frequency weighting	Frequency weighting for the result
	0 : linear
	1 : A-weighting
	2 : B-weighting
	3 : C-weighting
	4 : D-weighting
Result	
Result	1/3 octave spectrum

Description:

The two frequency limits f1 and f2 are to be given as the 1/3-octave center frequencies, e.g. f1 = 8 Hz and f2 = 12500 Hz. f1 < f2. The upper 1/3-oct. with its frequency band must lie entirely within half of the sampling frequency.

The individual 1/3-octave values are stated as root mean square (RMS) values.

While the transients in the individual 1/3-octave (band-pass) filters are subsiding at the start of the measurement, the input signal's values are ignored. The transient time for the 1kHz 1/3-octave is is assumed to be 20ms. This time interval is inversely proportional to the 1/3-octave frequency. For very low 1/3-octaves, this time interval becomes considerable. A correspondingly long measurement must then be expected.

During the settling phase the RMS is assumed to be 0.0.

The result is a normal data set (vector). The x-coordinate of the result counts the 1/3-octave bands (just like the Famos-function OctA). For good representation in the curve window, 1/3-octave labeling should be selected.

The 1/3-octave filter and analyses are in accordance with IEC 651 (sound level measurement), DIN 45652 (1/3-octave filter for electro-acoustic measurements) and EN61260-1:2014 or IEC61260-1:2014 (band-pass filter for octaves and fractions of octaves, filter Class 1).

Examples:

The 1/3 octave spectrum is to be calculated from the time-history of the vibration. The signal is sampled every 0.025 ms.

```
fEval = 1 ; 0 (linear) 1 (A-weighting)
f1 = 10
f2 = 12500
Thirds = SpecThirds_1 ( vib, f1, f2, fEval )
```

See also:

OctA, OtrRpmThirds, SpecThirds

sqr

Square

Declaration:

```
sqr ( Parameter ) -> Result
```

Parameter:

Parameter	Parameter. Allowed types: [ND],[XY].
Result	
Result	Square of the parameter

Description:

The parameter passed is squared.

Remarks

- The y-unit is squared.
- The x-coordinate(s) of the parameter and the result are the same.
- The function Sqr() works more effectively than the explicitly expressed product using the operator * (multiplication).
- The parameter may be structured (events/segments).

Examples:

The electral power at an ohmic resistor is the square of the voltage divided by the resistance.

```
power = sqr(voltage) / resistance
```

See also:

sqrt, ^(hoch)

sqrt

Square root

Declaration:

```
sqrt ( Parameter ) -> Result
```

Parameter:

Parameter	Parameter. Allowed types: [ND],[XY].
Result	
Result	Square root of the parameter

Description:

The square root is calculated.

Remarks

- The square root of the unit is taken. Toward this end, the unit must contain at leas squared terms.
- This root is only defined for numbers greater than zero.
- The square root can also be calculated as an exponent: Sqrt(x) is equivalent to (x ^ 0.5).
- The x-coordinate(s) of the parameter and the result are the same.
- The parameter may be structured (events/segments).

Examples:

The square root of 9 is 3:

```
three = sqrt(9)
```

See also:

sqr, ^(hoch)

sRMS

Stat()-function: A data set's RMS-value

Description

sRMS is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Alternatively, the function RMS() can be used.

Examples:

Displaying the RMS-value of a voltage plot:

```
Stat(U)
BoxMessage("RMS Value", "Urms = " + TForm(sRMS, ""), "!1")
```

This is equivalent to:

```
BoxMessage("RMS Value", "Urms = " + TForm(RMS(U), ""), "!1")
```

See also:

Stat, RMS, sMean

sStDev

Stat()-function: A data set's standard deviation

Description

sStDev is a predefined variable and has been calculated by an earlier call of the statictical function Stat().

Alternatively, the function StDev() can be used.

Examples:

Display of a voltage plot's standard deviation:

```
Stat(U)
BoxMessage("Standard deviation", "Us = " + TForm(sStDev, ""), "!1")
```

This is equivalent to:

```
BoxMessage("Standard deviation", "Us = " + TForm(StDev(U), ""), "!1")
```

See also:

Stat, StDev, sMean

Stat

Determines various statistical quantities of a data set and evaluates the predefined variables sMin, sMax, sMean, sRMS, sStDev, sMinPos, sMaxPos.

Declaration:

```
Stat ( Data )
```

Parameter:

Data	Data set from which the statistical characteristic values are to be calculated [ND]

Description:

This function has the form of a procedure; it returns no value and cannot be used with an assignment.

FAMOS contains a number of pre-defined variables calculated by calling the Stat()-function:

sMin	Minimum
sMax	Maximum
sMean	arithmetic mean
sStDev	Standard deviation
sRMS	RMS value, square root of the mean of squares
sMinPos	Position (x-value) of the minimum
sMaxPos	Position (x-value) of the maximum

The statistical function calculates these predefined variables each time it is called. Once it has been called, many important statistical quantities of the data set are available for use in subsequent processing formulas.

- The statistical quantities are created by the statistical function with the corresponding units.
- The predefined variables sMin, sMax, etc. can be used in all formulas, but they do not appear in the variable list. They cannot be changed by hand or assigned any new values; they can only receive the values calculated by the Stat function. The predefined variables are initialized to values of zero when the program starts.
- The predefined variables are recalculated each time the statistical function is called. The old values are lost, so be sure to use any variables of interest before calling the statistical function again.
- The Stat function displays a list containing all of the calculated quantities in the imc FAMOS main window's output box.
- The results of the Stat function can be obtained individually with other imc FAMOS functions.
  Multithreading: The function acts locally on the execution thread in which it was called. The standard execution thread and each sequence function executed in a separate thread using BEGIN_PARALLEL thus each have their own, independent memory.

Examples:

```
Stat(NDdata)
specificValue = (sMax - sMin) / sMean
```

Both of these formulas calculate a quantity from standard quantities of the data set, minimum and maximum. The call is equivalent to:

```
specificValue = (Max(NDdata) - Min(NDdata)) / Mean(NDdata)
```

```
Stat(NDdata)
delta1 = sMax - sMean
Stat(NDdata * 2)
delta2 = sMax - sMean
```

Please note that SvDelta1 and SvDelta2 are different because the statistical function was called a second time for a different data set, one twice as large as the first.

See also:

Min, Max, Mean, Pos, PosiEx, RMS, StDev

StDev

The standard deviation of a data set's numerical values is determined.

Declaration:

```
StDev ( Data ) -> SvStDev
```

Parameter:

Data	Data whose standard deviation is to be calculated [ND].
SvStDev	
SvStDev	Standard deviation of the data set

Description:

Standard deviation is a statistical characteristic of a data set, indicating how far the data are located from their arithmetic mean value. Waveforms with very similar numerical values have a small standard deviation, whereas data sets with very different values exhibit a very large standard deviation.

The standard deviation is calculated according to the following algorithm:

The deviation of each value of the data set from the arithmetic mean is squared and then all of these square values are summed. This value is then divided by the number of values in the data set minus 1; finally, the square root is calculated.

Examples:

A data set with the values 1, 3 and 5 has a standard deviation of 2.

```
data = [1, 3, 5]
SD = StDev(data)
```

See also:

Mean, Min, Max, Stat

STri

Schmitt-trigger with specified upper and lower threshold values

Declaration:

```
STri ( Data, SvUpper, SvLower ) -> Result
```

Parameter:

Data	Data set to be shaped. Allowed types: [ND].
SvUpper	Upper threshold
SvLower	Lower threshold
Result	
Result	Rectangular shaped result of the Schmitt-trigger

Description:

The Schmitt trigger function sTri is used to shape impulses; this function generates ideally shaped rectangular impulses. Only two different function values are generated: +1 and -1. The configuration of this function requires two threshold values, one lower and one upper.

The Schmitt trigger functions according to the following algorithm:

- The Schmitt trigger returns -1 as the initial value if the signal's first value is less than the top threshold value, otherwise the intial value is +1.
- If the Schmitt trigger last returned -1 and the signal is less than the lower threshold, it returns -1 again, otherwise +1.
- If the Schmitt trigger last returned +1 and the signal is greater than the lower threshold, it generates +1 again, otherwise -1.
- The last two steps are repeated until the entire data set has been run through. It is important to note that the Schmitt trigger function has a memory.

This function is used to "clean up" impulses distorted with noise before further processing. All distortion is eliminated, but the shape of the original signal is lost. The greater the difference between the thresholds, the less sensitive the Schmitt trigger reacts to distortion.

It is worthwhile to use the Schmitt trigger when the frequency of and spacing between impulses in a series is important, but the exact shape is insignificant.

- The Schmitt trigger does not have a y-unit, but matches the x-scaling of the specified data set.
- Select the stairstep display mode for graphical display of Schmitt trigger signals or other rectangular curves. See the "Y" menu in the curve window.
- Short but very noisy impulses often cannot be suppressed with the Schmitt trigger if the distance between the threshold values cannot be large enough. Use one of the smoothing functions Smo3, Smo5 or Smo to eliminate this noise, or change the individual values by hand.
- If the upper and lower thresholds are the same, the Schmitt trigger functions as a comparator, establishing whether a signal is greater than or less than the specified value.
- The 2nd and 3rd parameters may be interchanged. The lower value will always be adopted as the lower limit.

Examples:

```
peakCount = Peaks(STri(NDdata, 10 'V', 20 'V'))
```

A signal has a general level of 3V to 8V, but a few important peaks between 25V and 30V. The number of these peaks is to be determined. The threshold is set to between 8V and 25V, somewhat generous, but small enough to suppress noise on the slopes of the peaks. The Schmitt trigger function idealizes the peaks to rectangular impulses, which are counted in the final step.

See also:

Hyst, Peaks, Scale, Smo, Perio

Sub

Time-/x-correct subtraction

Declaration:

```
Sub ( Minuend, Subtrahend, SvOption ) -> Difference
```

Parameter:

Minuend	First parameter, minuend; allowed types: [ND],[XY].
Subtrahend	Second parameter, subtrahend; allowed types: [ND],[XY].
SvOption	Option
	0 : The trigger time of the two summands is ignored.
	1 : Time-correct superposition with regard to trigger-time
Difference	
Difference	Difference; results of the subtraction [XY]

Description:

Two data sets undergo time-correct or x-correct subtraction, meaning that the y-values at each point of the minuend is subtracted from the value at the corresponding point of the subtrahend.

The result is defined only within the x-range which is shared by both data sets. Within this range a resultvalue is determined for every point at which at least one of the data sets possesses a value. If no value exists for the other data set, one is determined by linear interpolation.

The x-tracks of both parameter data sets must be monotonous, i.e. the x-coordinates must increase continuously.

The subtraction operator, by contrast, performs subtraction on the values of data sets.

Examples:

Two channels are measured; one between 11:00 and 13:00, and the other between. 12:00 and 14:00.

```
difference = Sub(voltage11_13h, voltage12_14h, 1)
```

A time-correct subtraction of the two data sets with respect to the trigger time is performed. The result is defined for the time between 12:00 and 13:00 hours.

See also:

-(Subtraktion), Add, Mult, Div, Append

Sum

Sums all of a data set's values

Declaration:

```
Sum ( Data ) -> SvSum
```

Parameter:

Data	Data set whose sum to calculate [ND, XY].
SvSum	
SvSum	Sum of all of a data set's y-values

Description:

The sum of all of the data set's y-values is calculated.

Examples:

Both formulas compute the sum of all of an equidistantly sampled data set's y-values.

```
dataSum = Sum(data)
dataSum = Int(data) / xdel?(data)
```

See also:

Int, MInt

SvToChar

A text with an arbitrary character is generated:

Declaration:

```
SvToChar ( SvChar ) -> TxChar
```

Parameter:

SvChar	The ASCII code of the character to be displayed as text (0..255)
TxChar	
TxChar	Text (of length 1) with the specified character

Description:

The ASCII-code of a character specified as the parameter is used to generate a text containing exactly this one character. This is helpful for special characters such as Tab and Line Break

Tom specify special characters in text constants, it is alternatively possible to use the "tilde" character (~), followed by a three-digit number representing the ASCII code.

- The ASCII-codes permitted are 1..255.
- The ASCII code 0 is returned for an empty text.
- For example, tabulator character's code = 9, carriage return = 13, line feed = 10.
- Not all ASCII codes can be displayed in Windows. Above all be sure that ANSI character set is used. ASCII tables made for the DOS environment display the characters in the OEM character set.

Examples:

A text with the combination Carriage Return/ Line Feed is generated:

```
TxCrlf = SvToChar(13) + SvToChar(10)
```

The same result can be obtained with

```
TxCrlf = "~013" + "~010"
```

or more briefly:

```
TxCrlf = "~013~010"
```

See also:

CharToSv, TtoSv, TReplace, TConv

SWITCH

Initiates a case distinction (multiple branching). Depending on the result of the expression specified here, a maximum of one of multiple subsequent alternative instruction blocks (CASE/DEFAULT) is performed.

Declaration:

```
SWITCH ComparisonValue
```

Parameter:

ComparisonValue	Comparison value for subsequent CASE instructions. The evaluation of the expression specified here must return either be a single value or a text.

Description

After the SWITCH, at least one CASE or DEFAULT command must follow, the end of the case distinction is denoted by the command END.

As the CASE-condition, you can either specify a single value, a comma-separated list of values, or a range (with the keyword 'TO').

The construct

```
SWITCH value
CASE c1
   ...Instructions1
CASE c2,c3
   ...Instructions2
CASE c4 TO c5
   ...Instructions3
DEFAULT
   ...Instructions4
END
```

is equivalent to:

```
IF value = c1
   ...Instructions1
ELSEIF value = c2 OR value = c3
   ...Instructions2
ELSEIF value >= c4 AND value <= c5
   ...Instructions3
ELSE
   ...Instructions4
END
```

Examples:

A descriptive text is formed for a value normally lying between 0 and 100.

```
SWITCH Round(value, 1)
CASE 0
   Tx = "Lower limit"
CASE 1 TO 48
    Tx = "Lower half"
CASE 49,50,51
    Tx = "Center"
CASE 52 To 99
   Tx = "Upper half"
CASE 100
   Tx = "Upper limit"
DEFAULT
   Tx = "Invalid Value"
END
```

The user is prompted to select a file to load. On the basis of the file extension, the file format is recognized and the corresponding function for loading is called.

```
TxFileName = DlgFileName("", "", "",0)
TxFileExt = FsSplitPath(TxFileName, 3)
SWITCH TxFileExt
CASE ".dat", ".raw"
   ; Load imc data file
   fh = FileOpenDSF(TxFileName, 0)
   ;...
CASE ".xls"
   ; Load EXCEL file
   fh = FileOpenXLS(TxFileName, 0)
   ;...
DEFAULT
   PAUSE ==> Invalid file format
END
```

See also:

CASE, DEFAULT, IF, CodeRange

TAdd

Two texts appended together

Declaration:

```
TAdd ( TxFront, TxBack ) -> TxWhole
```

Parameter:

TxFront	Text to which another is appended
TxBack	Text to be appended
TxWhole	
TxWhole	What is returned is the concatenation of the two texts specified.

Description:

The two texts specified are concatenated.

Alternatively, the **addition operator +** can be used.

Examples:

```
Path = TAdd(TAdd("c:\", "imc\"), "dat")
; or:
Path = "c:\" + "imc\" + "dat"
```

After execution, the variable Path contains the text "c:\imc\dat".

See also:

TForm, TConv, TPart, TtoSv

tan

Tangent, trigonometric function

Declaration:

```
tan ( Parameter ) -> Result
```

Parameter:

Parameter	Input data (angle). Allowed types: [ND],[XY].
Result	
Result	Tangent of the parameter

Description:

The trigonometric function tan is calculated using radians or degrees, corresponding to the unit of the specified parameter.

The x-coordinate(s) of the results and the parameter are the same.

Remarks

- The parameter of the cos function should not have a unit or the units 'rad', 'degr', or '° '. Any other units will cause an error message to be generated and the unit will be used for the result.
- The parameter may be structured (events/segments).
- The corresponding inverse function is atan().

Examples:

tan and atan reverse each other:

```
one = tan(atan(1))
```

The tan function can be used to correct a measurement quantity subject to a saturation effect by stretching the saturation range:

```
NDcorr = 3 * tan(NDdata / 10)
```

An angle in degrees is specified as the parameter. Additional valid formulas: tan(PI/4) = 1, tan(0) = 0, tan(-PI/4) = -1 or tan(45'Deg') = 1, etc.

```
one = tan(45'°')
```

See also:

sin, cos, atan2

TComp

Two tests or text arrays are compared with each other.

Declaration:

```
TComp ( TxText1, TxText2 ) -> Result
```

Parameter:

TxText1	The first of the parameters to be compared
TxText2	The second of the two parameters to be compared
Result	
Result	Results of the comparison; single value or data set with the following values:
	-1 : The first text is ordered alphabetically before the second text.
	0 : Both texts are the same
	1 : The first text is ordered alphabetically after the second text.

Description:

The following parameter combinations are allowed:

Text - Text:	The result is a single value with the value -1, 0 or 1.
Textarray - Textarray	An element-by-element comparison of the two text arrays is performed. The result is a data set consisting of -1, 0, 1 and having the length of the shorter of the two text arrays.
Textarray - Text:	Each element of the text array is compared with the 2nd parameter. The result is a data set consisting of -1, 0, 1 and having the length of the text array.

- The function does not differentiate between upper and lower case letters; comparison of the texts "hello" and "Hello" returns a value of 0.
- An empty text is alphabetically ordered before any other texts.
- For the purpose of comparing texts and text arrays, the operators "<>" and "=" can also be used (with text arrays, these operators do not return an element-by-element comparison, but rather check the whole array for equality).

Examples:

```
DatName = "wave0001"
WavVergl = TComp(TPart(DatName, 1, 4), "WAVE")
```

[WavComp] contains the value 0.

```
Txa1 = ["Channel1", "CHANNEL2", "Channel5"]
Txa2 = ["Channel2", "channel2", "Channel4", "Channel7"]
v = TComp(Txa1, Txa2)
v2 = TComp(Txa1, "channel5")
```

v => [-1, 0, 1].

v2 => [-1, -1, 0].

See also:

TLike, TForm, TPart, TReplace, TxWhere, TxRegExMatch

TConv

A text can be converted in various ways, e.g. uppercase/lowercase spelling.

Declaration:

```
TConv ( TxText, SvTask ) -> TxConverted
```

Parameter:

TxText	Text to be converted
SvTask	Specificatino of how to perform conversion
	1 : All uppercase letters are converted to lowercase letters.
	2 : All lowercase letters are converted to uppercase letters
	3 : A text is transferred from ANSI to the OEM character set; files with OEM texts are read correctly by the DOS application. Windows applications expect texts in ANSI format.
	4 : All spaces at the beginning of the textes are deleted.
	5 : All spaces at the end of the text are deleted.
	6 : All spaces are deleted.
	7 : A text is converted from the OEM to the ANSI character set.
TxConverted	
TxConverted	Converted text

Description:

Examples:

```
TxPfad= TConv(" c:\imc\ dat ",6)
```

TxPath contains the contents "c:\imc\dat"

See also:

TForm, TPart, TComp, TtoSv

TForm

A number is converted to a formatted text.

Declaration:

```
TForm ( SvNumber, TxFormat ) -> TxFormatted
```

Parameter:

SvNumber	Real number to be converted
TxFormat	Specification of the designed format
TxFormatted	
TxFormatted	The formatted text

Description:

A number is converted to a formatted text, where a variety of format types can be specified. The parameter [TxFormat] contains the specifications needed for this purpose.

The following formats are available, selected with the bold-print letter codes for the type, followed by the desired numerical values (replacing the normal letters):

TxFormat	Description	Example
"e.N"	Floating point format N: Count of decimal places	3.456 with "e.2" -> "3.46E+00"
"FV.N"	Fixed point format V: Minimum count of pre-decimal digits N: Count of post-decimal digits Excess pre-decimal places are filled with spaces.	3.456 with "f2.2" -> " 3.46"
"gV.N"	Fixed point format V: Count of pre-decimal digits N: Count of post-decimal digits Excess pre-decimal places are filled with zeroes.	3.456 with "g2.2" -> "03.46"
"a.N"	Automatic N: Maximum outputted digit count The shortest possible notation is used, depending on the numerical value. Appended zeroes after the decimal point are omitted. or even the decimal if applicable. Thus, ideal for integers.	3.40056 with "a.2" -> "3.4"
""	Corresponds to "a.6"	
"xG"	Hexadecimal format. G: Total amount of digits	127 with "x2" -> "7F"
"x"	Hexadecimal format (for integer data formats) [SvNumber] muss ganzzahlig und unskaliert sein.	127 (4 Byte unsigned integer) -> "0000007F" -1 (4 Byte signed integer) -> "FFFFFFFF"
"bG"	Binary format. G: Total amount of digits	34.56 with "b4" -> "100011"
"b"	Binary format (for integer data formats) [SvNumber] muss ganzzahlig und unskaliert sein.	127 (1 Byte unsigned integer) -> "01111111" -1 (1 Byte signed integer) -> "11111111"

Comma or period

If the period is replaced by a comma in these format strings, the output features a decimal comma instead of a decimal point. Otherwise, the period can be omitted (Example: "f23" is equivalent to "f2.3"; however, "f2,3" forces the use of a decimal comma).

If the period is replaced by a semicolon in these format strings, the output is formatted in accordance with the global presetting for the preferred decimal separator for real numbers ("Extra"/"Options"/"Display & Curve Window", or SetOption("Display.DecimalSeparator",...).

Special remarks to floating point format:

In floating point format, the exponent is specified with 2 places and an "E" is used as the character for the exponent, followed by the sign.

Special remarks to fixed point format:

In fixed point display, spaces are filled in front of the specified number until the define number of places left of the decimal is reached. If spaces in front of the number are not desired, set the number of places left of the decimal to 1. If the number has more places left of the decimal than specified in the TxFormat, all places left of the decimal are created so that the number is not distorted.

Rounding

If necessary, the outputted value will be rounded off in accordance with the decimal position count.

Special remarks to Binary- or Hexadecimal format:

The number to be converted must be in the numerical range -2^63..2^63-1.

With negative numbers, the two's-complement representation is used.

For "xG" and "bG", the resulting minimum output width is respectively:
Hex-format: range: -2147483648..0: 8 positions, else 16 positions.
Binary format: range: -32768..0: 16 positions; -2147483648..-32769: 32 positions, else 64 positions.

"x" and "b" (without specified width) are to be preferred when the data format of the number to be formatted is an integer. In that case there is no internal data converion, for which reason it is also possible to format 8-Byte numbers without losing precision. The number of positions to output is determined automatically according to the numerical format: for "x", twice the Byte-count; for "b" it is 8 times the Byte-count. The positions are packed with zeroes if appropriate.

Examples:

```
txValue= TForm(1.2345, "f2,3")
; txValue contains: " 1,234"

txValue= TForm(31, "f4.2")
; txValue contains: " 31.00" (2 preceding spaces)

txValue= TForm(31, "g4.0")
; txValue contains: "0031"

txValue= TForm(31.000, "")
; txValue contains: "31"

txValue= TForm(1.200, "a.6")
; txValue contains: "1.2"

txValue= TForm(1.201, "a.6")
; txValue contains: "1.201"

txValue= TForm(31, "x2")
; txValue contains: "1F"

value = 128
SetDataFormat(value, 6) ; 2 Byte unsigned integer
txValue= TForm(value, "x")
; txValue contains: "0080"
txValue= TForm(value, "b")
; txValue contains: ""0000000010000000""

value = -2
SetDataFormat(value, 2) ; 1 Byte signed integer
txValue= TForm(value, "x")
; txValue contains: "FE"
txValue= TForm(value, "b")
; txValue contains: "11111110"
```

See also:

TConv, TPart, TtoSv

ThrowError

Generate error

Declaration:

```
ThrowError ( TxErrorText )
```

Parameter:

TxErrorText	Error text to be posted

Description:

The function generates a user-defined runtime error message with the error text specified. The display and additional handling proceeds in the same way as for errors reported during runtime by the Formula Interpreter - and thus depends on the error handling mode selected and specified by means of OnError().

When the error handling mode is set to "Default", a message box containing the specified text is displayed and execution of the routine is cancelled. With the error modes "Return" and "ResumeEnd", execution resumes accordingly; the error text can be queried later by means of GetLastError(). "ReturnFail" returns the error to the caller. "ResumeNext" usually is not useful in combination with ThrowError().

Examples:

The function Sqrt() included in FAMOS for calculating a square root returns 0 for negative input values and also posts a correpsonding warning. The following sequence function instead returns an error.

```
; Declaration: !Sqrt_Strict(Par) => Result
OnError("ReturnFail")
IF min(Par) < 0
   ThrowError("The parameter contains negative values!")
END
Result = Sqrt(par)
```

The following sequence function checks whether the entire course of a data set's signal is above a specified reference data set's signal course. If the function can not be executed due to inappropriate parameters, this is indicated by a special return value. The caller ca query the error cause by using GetLastError().

```
; Declaration: !CheckAbove(TestData, Lower) => Result
; Result = 1:  OK
; Result = 0: At least 1 value of [TestData] is lower than the corresponding value in [Lower]
; Result = -1: Error: Input data do not match (in terms of x-axis) or have events/segments

OnError("Return", "Unknown error", "Result", -1)
IF NOT(VerifyVar(TestData, "!SegEvn")) OR NOT(VerifyVar(Lower, "!SegEvn"))
   ThrowError("The paramater may not have segments or events!")
END
Verify(Leng?(TestData)=Leng?(Lower) AND xdel?(TestData)=xdel?(Lower) AND xoff?(TestData)=xoff?(Lower), "Parameter: x-scaling incompatible!")
Result = Max(Lower - TestData) < 0
```

See also:

OnError, Verify, VerifyVar, BoxMessage, GetLastError

Time?

Queries a data set's trigger time.

Declaration:

```
Time? ( Data ) -> SvTime
```

Parameter:

Data	Data set whose trigger time is to be determined
SvTime	
SvTime	Trigger time

Description:

The function returns the trigger time of a data set in a special imc FAMOS time format containing date and time.

The numerical value obtained can be analyzed, subjected to calculations, and assigned.

Examples:

A data set's trigger time is queried and displayed in a message box.

```
triggerTime = Time?(data)
Tx = TimeToText(triggerTime, 0)
BoxMessage("Data set time: ", Tx, "!1")
```

See also:

SetTime, TimeSplit, TimeJoin, TimeToText, TimeSystem?

TimeAdd

Adding seconds to time value

Declaration:

```
TimeAdd ( SvTime, SvSeconds ) -> SvTimeSum
```

Parameter:

SvTime	Time expressed in the internal time format.
SvSeconds	Number of seconds to add.
SvTimeSum	
SvTimeSum	Resulting time value.

Description:

A number of seconds is added to a time specification expressed in the imc FAMOS time format.

The result is expressed in the internal time format again.

Examples:

The trigger time of a data set is increase by one minute.

```
triggerTime = Time?(data)
triggerTime = TimeAdd(triggerTime, 60)
SetTime(data, triggerTime)
```

See also:

SetTime, TimeDiff, TimeJoin, TimeSplit, TimeToText

TIMECOPY

Copy trigger time
This command is obsolete, instead of it, the more powerful functions SetTime() and Time?() should be used.

Declaration:

```
TIMECOPY SourceVariable TargetVariable
```

Parameter:

SourceVariable	Variable whose trigger time is to be copied
TargetVariable	Variable which is to receive the new trigger time

Description

The trigger time of the target variable is set to equal that of the source variable.

Examples:

```
TIMECOPY qdat zdat
```

The variable "tDat" contains the trigger time of the variable "sDat".

See also:

Time?, SetTime

TimeDiff

Calculates difference between 2 time values.

Declaration:

```
TimeDiff ( SvTime1, SvTime2 ) -> SvDiffInSeconds
```

Parameter:

SvTime1	First time value expressed in the internal time format
SvTime2	Second time value expressed in the internal time format
SvDiffInSeconds	
SvDiffInSeconds	Resulting difference in seconds

Description:

The difference in seconds between two times in the imc FAMOS time format is calculated.

Examples:

The x-offset of a data set is set to the difference between the trigger times of two other data sets:

```
time1 = Time?(dataA)
time2 = Time?(dataB)
sec = TimeDiff(time1, time2)
XOFFSET dataC sec
```

See also:

SetTime, TimeAdd, TimeJoin, TimeSplit, TimeToText

TimeJoin

A time value in imc FAMOS is created from the individual components.

Declaration:

```
TimeJoin ( SvDay, SvMonth, SvYear, SvHour, SvMinute, SvSeconds ) -> SvTime
```

Parameter:

SvDay	Day (1..31)
SvMonth	Month (1..12)
SvYear	Year (1980..)
SvHour	Hour (0..23)
SvMinute	Minute (0..59)
SvSeconds	Seconsd (0..<60)
SvTime	
SvTime	Resulting time value

Description:

A time value expressed in the imc FAMOS time format is assembled from its individual components.

Examples:

A data set's trigger time is set to 11/2/1995, 12:00 PM:

```
SetTime(data, TimeJoin(2, 11, 1995, 12, 0, 0))
```

See also:

SetTime, Time?, TimeSplit, TimeToText, TimeSystem?

TIMESET

Set trigger time
This command is obsolete, instead of it the more powerful function SetTime() should be used.

Declaration:

```
TIMESET TargetVariable SvDay SvMonth SwYear SvHour SwMinute SvSecond
```

Parameter:

TargetVariable	Variable whose trigger time is to be set
SvDay	Day
SvMonth	Month
SwYear	Year (4-digit)
SvHour	Hour
SwMinute	Minute
SvSecond	Second

Description

The trigger time of the variable "TargetVar" is set according to the new time.

Examples:

```
   TIMESET DestinationVar 1 3 1992 12 0 1
```

The trigger time of the variable "TargetVar" is set to '01.03.1992 12:00:01'

See also:

Time?, SetTime

TimeSplit

Get a component of a time expressed in the imc FAMOS time format.

Declaration:

```
TimeSplit ( SvTime, Selection ) -> SvTimeComponent
```

Parameter:

SvTime	Time expressed in the imc FAMOS time format
Selection	Selection of component
	0 : Day (1..31)
	1 : Month (1..12)
	2 : Year (1980..)
	3 : Hour (0..23)
	4 : Minute (0..59)
	5 : Seconds (0.. <60)
SvTimeComponent	
SvTimeComponent	Desired component

Description:

A component of a time spefication expressed in the imc FAMOS time format is queried.

Examples:

The year in which the data set was created is determined:

```
year = TimeSplit(Time?(data), 2)
```

See also:

SetTime, Time?, TimeToText, TimeJoin

TimeSystem?

Inquiring current system time.

Declaration:

```
TimeSystem? ( ) -> SvTime
```

Parameter:

SvTime	
SvTime	System time in the imc FAMOS time format

Description:

The function returns the current system time in the imc FAMOS time format. The time value obtained can be processed by other time functions.

Examples:

The current system time is converted to text and outputted:

```
tx = TimeToText(TimeSystem?(), 0)
BoxMessage("Current time", tx, "!1")
```

See also:

SetTime, Time?, TimeToText, TimeAdd, TimeJoin, TimeSplit

TimeToText

A time value is converted to a text.

Declaration:

`TimeToText ( SvTime, SvCode ) -> TxTime`

Parameter:

SvTime	Time value in the internal time format
SvCode	Formatting rule
	0 : Date and time, separated by spaces, 2-digit year
	1 : Date only, 2-digit year
	2 : Time only
	3 : Date and time, separated by spaces, 4-digit year
	4 : Date only, 4-digit year
TxTime	
TxTime	Time value as text

Description:

The function converts an existing time statement in imc FAMOS format to a text. The time and date format in the Windows Control Panel are applied.

Examples:

The current system time is converted to text and outputted:

```
tx = TimeToText(TimeSystem?(), 0)
tx = TimeToText(TimeSystem?(), 0)
BoxMessage("Current time", tx, "!1")
```

See also:

SetTime, Time?, TimeSystem?, TimeAdd, TimeJoin, TimeSplit

TLeng

Finds the length of a text, meaning how many characters it contains. With a text array, the length of each element is found.

Declaration:

```
TLeng ( TxText ) -> Length
```

Parameter:

TxText	Text/Text array, whose length is to be found
Length	
Length	Single value (for Text), or data set (for Text array) with the texts' respective character counts. -1 for other data types

Description:

The number of characters in a text (including spaces) is determined and returned.

With a text array, the length of the individual elements is found and the data set is returned.

The length of an empty text is 0.

If the parameter's data type is neither Text nor Text Array, then -1 is returned.

Examples:

```
text = "18 characters long"
Leng_ = TLeng(text)
```

Length contains the value 18.

```
txa = ["current1", "RPM2", "", "rpm1", "Current2"]
Leng_ = TLeng(txa)
```

Leng_ contains the values [8, 4, 0, 4, 8].

See also:

TAdd, TForm, TPart, TComp, TConv

TLike

A text or text array is compared for matching to a pattern. The pattern can contain wildcards.

Declaration:

```
TLike ( TxText, TxPattern, SvOption ) -> Result
```

Parameter:

TxText	Text or text array to inspect
TxPattern	The pattern according to which the comparison is drawn. The wildcard characters '?' and '*' are allowed.
SvOption	Determines whether upper- and lower case spelling are distinguished.
	0 : Upper- and lower case spelling are regarded as identical.
	1 : Upper- and lowercase spelling distinguishes letters.
Result	
Result	Results of the comparison; single value or data set with the following values:
	0 : The text does not match the pattern specified.
	1 : The text matches the pattern specified.

Description:

In the pattern (2nd parameter), the wildcard characters "?" and "*" can be used. "?" stands for exactly one character, "*" for a string of any length.

If the first parameter is a text array, each element of the array is compared with the pattern. The result is a data set containing only the values [0, 1] and having the length of the text array.

Only the first 259 characters of both texts are considered.

For complex pattern comparisons, the function TxRegexMatch() can be used.

Examples:

Check whether TxVar begins with the letter 'a' or 'A':

```
ok = TLike(TxVar, "a*", 0)
```

Check whether TxVar contins the string "voltage" at any position. No distinction is made between upper- and lower-case spelling:

```
ok = TLike(TxVar, "*voltage*", 0)
```

Check whether TxVar begins with the string "A0" and contains exactly two more characters. Upper- and lower-case spelling is distinguished:

```
ok = TLike(TxVar, "A0??", 1)
```

Checks whether TxVar begins with the string "Channel" and ends with "_1". Upper- and lowercase spelling are not distinguished:

```
ok = TLike(TxVar, "Channel*_1", 1)
```

```
txa = [ "current1", "RPM2", "rpm1", "Current2", "voltage", "rpm11"]
; test whether the array elements end with 1:
res = TLike(txa, 0, "*1")
```

res => [1, 0, 1, 0, 0, 1]

```
; tests whether the array elements begin with a lowercase 'r' and have exactly 4 characters:
res = TLike(txa, 1, "r???")
```

res => [0, 0, 1, 0, 0, 0]

See also:

TComp, TxRegExMatch

ToInt

Conversion to the data format '8-Byte Integer (signed)'

Declaration:

```
ToInt ( Parameter ) -> Converted
```

Parameter:

Parameter	Single value or data set to convert
Converted	
Converted	Converted single value or data set

Description:

The single value or data set passed as the parameter is converted to the data format "Integer (8-Byte signed)". In the process, every value is rounded to the next closest integer.

Remarks:

- The maximum value range of the resulting data format is [-2^63-1 ... 2^63]; if the range is violated in either direction, the value is set to the respective range boundary value.
- All of the parameter's other properties are applied.
- The parameter may be segmented and event-based.

The call

```
   integer = ToInt(parameter)
```

is functionally equivalent to

```
   integer = parameter
   SetDataFormat(integer, 12)
```

Examples:

The Python-function "round" is used to round PI to 3 decimal places. Without the ToInt()-function having been called, the constant would be passed to Python as a real number and would cause a runtime error, since the function expects an integer argument at this position.

```
PI_rounded = PyCallFunction("", "round", PI, ToInt(3))
```

See also:

SetDataFormat

Supported since:

Version 2022

Top

Returns the x-positions of all data points where the y-value exceeds a threshold value.

Declaration:

```
Top ( Data, SvfLimit ) -> XTop
```

Parameter:

Data	Data set examined. Allowed types: [ND],[XY].
SvfLimit	Threshold
XTop	
XTop	X-coordinates of y-values from NDData which are greater than [SvThresh]

Description:

The x-coordinates of all data points of NwData greater than the threshold SvThresh form the result.

- The Value() function can be used to determine the associated y-coordinates.
- An empty data set is returned if no values fulfill the threshold conditions.
- As an alternative, use the SearchLevel function, which is also suitable for XY-data sets.

Examples:

```
NDxPeak = Top(NDdata, "5V")
```

The result is all x-coordinates of all points in the data set NDData, whose values are greater than 5 volts.

```
NDyPeak = Value(NDdata, Top(NDdata, topValue))
```

The result is all y-coordinates of the data set NwData greater than SvTop.

```
meanValue = Mean(Value(NDimpulse, Top(NDimpulse, 0.1)))
```

The data set NDImpulse is passed, which contains a number of impulses whose noise is a maximum of 0.1. The result is the mean value of all impulses.

See also:

SearchLevel, Value, All0, RangeSet, xMax

TPart

A excerpt of a text is copied.

Declaration:

```
TPart ( TxText, SvStart, SvLength ) -> TxPart
```

Parameter:

TxText	Text from which a excerpt is to be copied
SvStart	Position as of which the copy excerpt begins
SvLength	Number of characters to copy
TxPart	
TxPart	The excerpted text

Description:

A section is cut out of a specified text string. This section is defined by the position of the first character to be cut out and the number of characters to be cut out.

- The position of the first character in the text and thuse the smallest possible value for [Start] is 1.
- If [Start] is greater than the text length, an empty text is returned.
- If [Start] + [Length] is greater than the text length, the excerted portion of text ends at the text end.

Examples:

```
extension = TPart( "wave.dat", 6, 3 )
```

The variable extension contains "dat".

See also:

TReplace, TAdd, TForm, TLeng, TtoSv

TransposeMatrix

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

A matrix (a segmented waveform) is transposed. In other words, the rows and columns exchange places. The matrix' rows correspond to the waveform's segments. Input data and result are segmented. The input data have a single component.
This function is only included for the purpose of compatibility with imc FAMOS 6.0 and its predecessors. Please use instead MatrixTranspose().

Declaration:

```
TransposeMatrix ( Matrix ) -> Result
```

Parameter:

Matrix	Input matrix to be transposed.
Result	
Result	Transposed matrix

Examples:

```
Spectra = AmpSpectrumRMS ( Channel, 1000, 0, 50, 1, 0, 0 )
tm = TransposeMatrix ( Spectra )
```

The matrix consists of amplitude values, plotted over time and frequency. The function re-arranges the time axis and the frequency axis.

See also:

MatrixTranspose

TransRec

Data reduction according to the Transitional-Recording procedure

Declaration:

```
TransRec ( Data, SvTolerance ) -> Reduced
```

Parameter:

Data	Data set whose data are to be reduced [ND]
SvTolerance	Permitted tolerance (in phys. units)
Reduced	
Reduced	Reduces data set [XY]

Description:

An equidistantly sampled data set is interpolated to an XY-data set. The tolerance (in physical units) specifies the permissible discrepancy between source data and interpolated results. The greater the tolerance, the shorter the resulting data set will be.

With analog source data, the results should be represented in "Lines" mode in the curve window, rather than in "Steps".

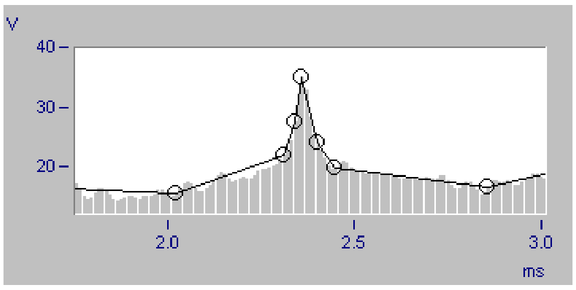
With digital source data, [SvTolerance] should be set to = 0. That way only changes in the signal will be recorded. For pseudo-analog data reduction it is possible to use [SvTolerance] = 1e-20.

he function does not change the data format of the source data (float, double, integer..). For best data reduction results, therefore, first use a function such as "SetDataFormat(..)" to set the y-component's data format to 1 or 2-byte integers.

Examples:

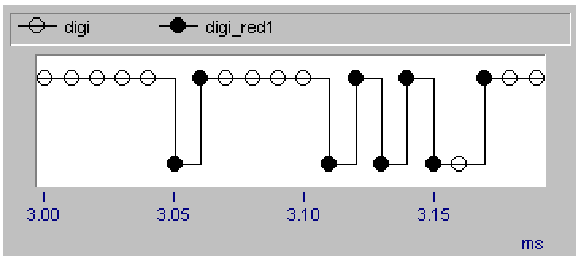
```
slope_2 = TransRec(slope, 2)
```

Data reduction for analog data. The gray bars show the original signal, the black lines and circles show the course of the reduced signal and its points.



```
digi_red1= TransRec(digi, 0)
```

Data reduction for digital data. The gray bars show the original signal, the black lines and circles show the course of the reduced signal and its points.



See also:

SetDataFormat, Red, XYdt

TReplace

A selected string in a text is replaced with another string.

Declaration:

```
TReplace ( TxWhole, TxPart, TxReplacement ) -> TxResult
```

Parameter:

TxWhole	The whole text in which a section is to be replaced by another
TxPart	The text excerpt to be replaced
TxReplacement	Text to replace [TxPart] in [TxWhole]
TxResult	
TxResult	Accordingly changed text

Description:

The function searches for a section of text; when it is found it is replaced by the specified text.

- If [TxPart] is not found in [TxWhole], the latter is returned unchanged
- Even if the text is present several times in the text, only the section found first is replaced.
- The function does not differentiate between upper and lower case letters, e.g. "X" and "x" are treated the same when searching for the section TxPart in TxWhole.

Examples:

```
TxFile = "measurement.txt"
BoxMessage("Hint", TReplace("File %s is saved!", "%s", TxFile), "!1")
```

The message "File Measurement.txt saved!" is displayed.

See also:

TxWhere, TAdd, TPart, TComp, TConv

TsaAppend

Scope: TimestampASCII data

A new element is appended to the end of a time-stamp-ASCII channel.

Declaration:

```
TsaAppend ( TsaChannel, Data, Time )
```

Parameter:

TsaChannel	TsaChannel
Data	Data
Time	Time

Description:

The contents and time of the new element are to be specified. The contents are to be expressed as a waveform. The waveform is transfered byte by byte, no matter in which data format it is expressed. It makes sense for the waveform to take the Byte numerical format. An element's maximum size (see time-stamp-ASCII documentation) must not be exceeded. The time may not be less than that of its predecessor and not more than that of its successor. Times must increase monotonously but not strictly monotonously; i.e., adjacent elements' times may be the same. The time may be rounded, since the time is expressed internally as x0+i*dx, where i >= 0 is an integer. The time is entered in relative terms: as the time after the trigger release, without the absolute trigger time, date or time of the measurement start. The parameter channel is changed in the process. If the waveform contains events, the element is appended to the last event.

Examples:

An element containing ten values from 48 to 57 is appended.

```
Content = ramp ( 48, 1, 10 )
SetDatFormat ( Content, 5, 0, 255) ; Makes the numerical format Byte (0..255).
Time = 20 ;

TsaAppend ( TsaChannel, Content, Time )
```

TsaAppendText

Scope: TimestampASCII data

A new element is appended to the end of a time-stamp-ASCII channel.

Declaration:

```
TsaAppendText ( TsaChannel, Text, Time )
```

Parameter:

TsaChannel	TsaChannel
Text	Text
Time	Time

Description:

The contents and time of the new element are to be specified. The contents are to be expressed as text. The time may not be less than that of its predecessor and not more than that of its successor. Times must increase monotonously but not strictly monotonously; i.e., adjacient elements' times may be the same. The time may be rounded, since the time is expressed internally as x0+i*dx, where i >= 0 is an integer. The time is entered in relative terms: as the time after the trigger release, without the absolute trigger time, date or time of the measurement start. The parameter channel is changed in the process. If the waveform contains events, the element is appended to the last event.

Examples:

; An element with fixed text is appended.

```
Time = 20
Text = "hello"

TsaAppendText ( TsaChannel, Text, Time )
```

TsaCreateEmpty

Scope: TimestampASCII data

Creates an empty channel in time-stamp-ASCII format.

Declaration:

```
TsaCreateEmpty ( x0, dx, xUnit ) -> TsaChannel
```

Parameter:

x0	x0
dx	dx
xUnit	xUnit
TsaChannel	
TsaChannel	TsaChannel

Description:

x0: smallest possible time-stamp dx: time-stamp resolution All possible tmes which elements of this channel may take later follow the pattern x = x0+i*dx, i>=0, where i is an integer. The absolute time (trigger time) will be set to 0.0 (which is the first of January, 1980).

Examples:

An empty channel is created. The times entered later are >= 0 and have a resolution of 1ms.

```
TsaChannel = TsaCreateEmpty ( 0.0, 1e-3, "s" )
```

TsaDataToText

Scope: TimestampASCII data

A waveform is converted byte by byte into a text.

Declaration:

```
TsaDataToText ( Data, ZeroReplace ) -> Text
```

Parameter:

Data	Data
ZeroReplace	ZeroReplace
Text	
Text	Text

Description:

The numerical format of the waveform does not matter. It makes sense for the waveform to take the Byte numerical format. All bytes with the value zero are replaced by ZeroReplace. ZeroReplace: instead of a zero-byte: 0..255. If the value is 0, the text is ended at a 0-byte.

Examples:

A waveform is converted to text. All zero-bytes are replaced with a space (ASCII) so that the text is not truncated.

```
Text = TsaDataToText ( Data, 32 )
```

TsaDecode

Available in: Professional Edition and above

A time-stamped-ASCII channel is interpreted as a series of time-stamped frames, from the CAN-Bus for example. From these frames, a channel is generated by finding a value from each frame at a certain position.

Declaration:

```
TsaDecode ( TsaChannel, Analysis [, Startbyte] [, Startbit] [, BitCount] [, Data format] [, ByteOrder] [,
Factor] [, Offset] [, Unit] ) -> Channel
```

Parameter:

TsaChannel	TsaChannel
Analysis	What is to be analysed?
	"" : General frame of any format and any origin. If the frames in question are CAN-messages, then the ID is located in the first 4 Bytes, followed by the data bytes.
	"CAN.data" : CAN-message, data. The frames in this case are CAN-messages. The start byte is counted as of the beginning of the CAN-message's data region.
	"CAN.ID" : CAN-message, ID. The frames in this case are CAN-messages. Only the ID is desired. The MSB (Bit 31) which is = 1 to indicate an extended identifier is not read. Thus it is no longer possible to distinguish between standard and extended identifiers.
	"CAN.ID.msb" : CAN-message, ID. The frames in this case are CAN-messages. Only the ID is desired. May include the MSB (Bit 31) which is = 1 to indicate an extended identifier.
	"CAN.len" : CAN-message, length. The frames in this case are CAN-messages. Only the length (data byte count) of the CAN-message's data region is desired.
	"len" : Length of the message. The total length of the message is found. In case of CAN-messages, the length including the 4 byte identifier is determined.
Startbyte	The value is read from this Byte-position in the frame onwards. Startbyte = 0 for the first Byte. The Byte is counted as of the beginning of the frame. Any existing ID must be taken into account. For "CAN.data", the Startbyte = 0 is the first data value of the CAN frame. (optional , Default value: 0)
Startbit	Start Bit. The number to be read is located within the start byte from this bit position onwards. The LSB is Bit 0. The MSB is Bit 7. 0 <= Startbit <= 7. For numbers of more than 56 bits in length, the Startbit must be = 0 (Intel) or = 7 (Motorola). (optional , Default value: 0)
BitCount	Bit count. The number to be read is this many bits long. 1 <= Bit count <= 64 (optional , Default value: 0)
Data format	Data format of the number to be read (optional , Default value: "signed")
	"signed" : Signed integer in the two's complement
	"unsigned" : Unsigned integer
	"real" : Real (float, double). Only 32 or 64 bits with Startbit = 0 (Intel) or = 7 (Motorola) are possible.
ByteOrder	Byte order. Which (highest, lowest) is stated first. (optional , Default value: "intel")
	"intel" : Intel, lowest value (least significant) Byte first , little-endian
	"motorola" : Motorola, highest value (most significant) Byte first, big-endian
Factor	The value read is multiplied by this scaling factor. Only for integer formats (optional , Default value: 1)
Offset	The scaling offset is added to the value multiplied by the scaling factor. Only for integer formats (optional , Default value: 0)
Unit	Unit. The scaled value receives this unit. (optional , Default value: "")
Channel	
Channel	Channel in the XY-format.

Description:

From each frame, one value is extracted in the format specified. If the frame is long enough, the extraction is successful, in which case the value is written to the result.

If the frame is too short, the value cannot be read (completely). In this case the value is not written to the result.

The data set may not contain events.

The definition of the numerical format is compatible with imc DEVICES and imc STUDIO.

In case of CAN-frames, the following consideration applies for the CAN-ID: Standard identifiers have a size of 11 bits, thus a range of 0 to 2047. Extended identifers have a size of 29 bits. Addionally, the MSB of the 32-bit number is set. The CAN-ID is located in the first 4 Bytes and is represented INTEL Byte order. That applies, if data were written by imc DEVICES or imc STUDIO.

The function is not actually used for filtering. The purpose is to write one value to the result for each frame. If however the value cannot be determined successfully, then it cannot be written to the result. This situation can be avoided by proper filtering performed beforehand. On filtering, please refer to TsaFilter().

Scaling

The scaling factor and scaling offset are governed by this formula:

[Physical value] = [integer number] * scaling factor + scaling offset

The scaling factor and scaling offset are only available with integer formats.

With the default values for the scaling factor and the scaling offset, the system preferentially generates a suitable data format in the result.

Example: Intel Byte order

Startbit = 3, BitCount = 15, MSB appears first in each Byte; LSB last. Bit=X belongs to the number, bit=0 does not:

XXXXX000 XXXXXXXX 000000XX

In the 1st Byte, the 5 highest bits are set. This is where the number's lowest value bits are located.

In the 3rd Byte, the lowest 2 bits are set. This is where the number's highest value bits are located.

Startbit = 3 signifies that the 3 lowest bits (Bits 0, 1, 2) do not yet belong to the number.

Example: Intel Byte order

Startbit = 0, BitCount = 16, MSB in each Byte displayed first, LSB last. Bit=X belongs to the number, bit=0 does not:

XXXXXXXX XXXXXXXX

In the 1st Byte, all bits are set. This is where the number's lowest value bits are located.

In the 2nd Byte, all 8 bits are set. This is where the number's highest value bits are located.

Example: Motorola Byte order

Startbit = 2, BitCount = 15, MSB in each Byte displayed first, LSB last. Bit=X belongs to the number, bit=0 does not:

00000XXX XXXXXXXX XXXX0000

In the 1st Byte, the lowest 3 bit are set. This is where the number's highest-value bits are located.

In the 3rd Byte, the highest 4 bits are set. This is where the number's lowest-value bits are located.

Startbit = 2 signifies that the 3 lowest bits (Bits 0, 1, 2) belong to the number.

Example: Motorola Byte order

Startbit = 7, BitCount = 16, MSB appears first in each Byte; LSB last. Bit=X belongs to the number, bit=0 does not:

XXXXXXXX XXXXXXXX

In the 1st Byte, all 8 bits are set. This is where the number's highest value bits are located.

In the 2nd Byte, all 8 bits are set. This is where the number's lowest-value bits are located.

Examples:

generate xy channel with all CAN-IDs

```
ids = TsaDecode( tsa, "CAN.ID")
```

Both standard and extended IDs can occur. The differentiating MSB is to be included in the chanel.

```
ids = TsaDecode( tsa, "CAN.ID.msb")
```

Starting at the 3rd Byte of a CAN-message, there is a 16-bit signed integer led by the least significant Byte. 1 LSB = 5 N.

```
N = TsaDecode( tsa, "CAN.data", 2, 0, 16, "signed", "intel", 5.0, 0.0, "N")
```

Generates a channel with the message's 1st Byte

```
c = TsaDecode( tsa, "", 0, 0, 8, "unsigned", "intel")
```

Starting at the 3rd Byte of a CAN-message having the standard ID 22 Hex, there is a 16-bit unsigned integer led by the most significant Byte. 1 LSB = 5 N.

```
tsaFil = TsaFilter ( tsa, "CAN.ID.std", 0x22, 0x22)
N = TsaDecode( tsaFil, "CAN.data", 2, 7, 16, "unsigned", "motorola", 5.0, 0.0, "N")
```

Numerical example

```
tsa = TsaCreateEmpty(1, 0.1, "s")
b01 = [0x31, 0x32, 0x33, 0x81, 0x77, 0x55] ; ext id
setdatFormat( b01, 5, 0, 255 )
TsaAppend (tsa, b01, 1.7 )
c = TsaDecode( tsa, "CAN.ID.msb") ; c.y = 0x81333231
c = TsaDecode( tsa, "CAN.ID") ; c.y = 0x01333231
```

```
c = TsaDecode( tsa, "CAN.len") ; c.y = 2
c = TsaDecode( tsa, "len") ; c.y = 6
c = TsaDecode( tsa, "",0,0,8,"unsigned","intel") ; c.y = 0x31
c = TsaDecode( tsa, "",0,0,16,"unsigned","intel") ; c.y = 0x3231
c = TsaDecode( tsa, "",0,7,16,"unsigned","motorola") ; c.y = 0x3132
c = TsaDecode( tsa, "CAN.data",0,0,16,"unsigned","intel") ; c.y = 0x5577
```

See also:

TsaFilter, TsaGetFirst, SetDataFormat

TsaDelete

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element is deleted from the channel.

Declaration:

```
TsaDelete ( TsaChannel, Position )
```

Parameter:

TsaChannel	TsaChannel
Position	Position

Description:

This changes the positions of the subsequent elements.

Examples:

The first element is deleted

```
Position = TsaFindFirst ( TsaChannel )
TsaDelete ( TsaChannel, Position )
```

TsaFilter

Available in: Professional Edition and above

A time-stamp-ASCII channel is interpreted as a series of time-stamped frames, for example from the CAN-bus. By means of filtering, a subset of these frames is generated. E.g. all frames having a particular CAN-ID.

Declaration:

```
TsaFilter ( TsaChannel, Analysis, Min, Max [, Mask] [, Startbyte] [, Startbit] [, BitCount] [, Data format] [, ByteOrder] ) -> Filtered
```

Parameter:

TsaChannel	TsaChannel
Analysis	What is to be analysed?
	"" : General frame of any format and any origin. If the frames in question are CAN-messages, then the ID is located in the first 4 Bytes, followed by the data bytes.
	"CAN.data" : CAN-message, data. The frames in this case are CAN-messages. The start byte is counted as of the beginning of the CAN-message's data region.
	"CAN.ID" : CAN-message, ID. The frames in this case are CAN-messages. Only the ID is desired. The MSB (Bit 31) which is = 1 to indicate an extended identifier is not read. Thus it is no longer possible to distinguish between standard and extended identifiers.
	"CAN.ID.msb" : CAN-message, ID. The frames in this case are CAN-messages. Only the ID is desired. May include the MSB (Bit 31) which is = 1 to indicate an extended identifier.
	"CAN.ID.std" : CAN-message, standard identifier. The frames in this case are CAN-messages. Only standard identifiers are desired. Extended identifiers are ignored.
	"CAN.ID.ext" : CAN-message, extended identifier. The frames in this case are CAN-messages. Only extended identifiers are desired. The MSB (Bit 31) which is then = 1 is not read. Standard identifiers are ignored.
	"CAN.len" : CAN-message, length. The frames in this case are CAN-messages. Only the length (data byte count) of the CAN-message's data region is desired.
	"len" : Length of the message. The total length of the message is found. In case of CAN-messages, the length including the 4 byte identifier is determined.
Min	Minimum value. The value read is checked against this value after application of the bitmask. The filter condition can only be met if Value >= Min.
Max	Maximum value. The value read is checked against this value after application of the bitmask. The filter condition can only be met if Value <= Max.
Mask	The value read is combined with this bitmask in an AND logical expression. Only if the bitmask <> 0. Only up to BitCount <= 32. (optional , Default value: 0)
Startbyte	The value is read from this Byte-position in the frame onwards. Startbyte = 0 for the first Byte. The Byte is counted as of the beginning of the frame. Any existing ID must be taken into account. For "CAN.data", the Startbyte = 0 is the first data value of the CAN frame. (optional , Default value: 0)
Startbit	Start Bit. The number to be read is located within the start byte from this bit position onwards. The LSB is Bit 0. The MSB is Bit 7. 0 <= Startbit <= 7. For numbers of more than 56 bits in length, the Startbit must be = 0 (Intel) or = 7 (Motorola). (optional , Default value: 0)
BitCount	Bit count. The number to be read is this many bits long. 1 <= Bit count <= 64 (optional , Default value: 0)
Data format	Data format of the number to be read (optional , Default value: "signed")
	"signed" : Signed integer in the two's complement
	"unsigned" : Unsigned integer
	"real" : Real (float, double). Only 32 or 64 bits with Startbit = 0 (Intel) or = 7 (Motorola) are possible.
ByteOrder	Byte order. Which (highest, lowest) is stated first. (optional , Default value: "intel")
	"intel" : Intel, lowest value (least significant) Byte first , little-endian
	"motorola" : Motorola, highest value (most significant) Byte first, big-endian
Filtered	
Filtered	Filtered channel in the time-stamped ASCII format

Description:

One value in the format specified is extracted from each frame. That value then is tested for the filter condition.

The filter condition consists of a check of the value range after combining with the bitmask.

If the filter condition is true, the frame will be written to the result.

This is the regular filter condition: Value >= Min AND Value <= Max. In that case Min <= Max holds true. If for example all values equal to 7 are desired, then Min = 7 and Max = 7 with Value >= 7 AND Value <= 7.

If however Min > Max, then that will mean: Value >= Min OR Value <= Max. If for example all values <> 7 are desired, then Min = 8 and Max = 6, with Value >= 8 OR Value <= 6.

If the frame is too short, the value cannot be read (completely). Then the frame is not written to the result.

The data set may not contain events.

The definition of the numerical format is compatible with imc DEVICES and imc STUDIO.

In case of CAN-frames, the following consideration applies for the CAN-ID: Standard identifiers have a size of 11 bits, thus a range of 0 to 2047. Extended identifers have a size of 29 bits. Addionally, the MSB of the 32-bit number is set. The CAN-ID is located in the first 4 Bytes and is represented INTEL Byte order. That applies, if data were written by imc DEVICES or imc STUDIO.

If a frame is written to the result, it is copied unchanged including its original time-stamp.

Frames are also called messages.

Example: Intel Byte order

Startbit = 3, BitCount = 15, MSB appears first in each Byte; LSB last. Bit=X belongs to the number, bit=0 does not:

XXXXX000 XXXXXXXX 000000XX

In the 1st Byte, the 5 highest bits are set. This is where the number's lowest value bits are located.

In the 3rd Byte, the lowest 2 bits are set. This is where the number's highest value bits are located.

Startbit = 3 signifies that the 3 lowest bits (Bits 0, 1, 2) do not yet belong to the number.

Example: Intel Byte order

Startbit = 0, BitCount = 16, MSB in each Byte displayed first, LSB last. Bit=X belongs to the number, bit=0 does not:

XXXXXXXX XXXXXXXX

In the 1st Byte, all bits are set. This is where the number's lowest value bits are located.

In the 2nd Byte, all 8 bits are set. This is where the number's highest value bits are located.

Example: Motorola Byte order

Startbit = 2, BitCount = 15, MSB in each Byte displayed first, LSB last. Bit=X belongs to the number, bit=0 does not:

00000XXX XXXXXXXX XXXX0000

In the 1st Byte, the lowest 3 bit are set. This is where the number's highest-value bits are located.

In the 3rd Byte, the highest 4 bits are set. This is where the number's lowest-value bits are located.

Startbit = 2 signifies that the 3 lowest bits (Bits 0, 1, 2) belong to the number.

Example: Motorola Byte order

Startbit = 7, BitCount = 16, MSB appears first in each Byte; LSB last. Bit=X belongs to the number, bit=0 does not:

XXXXXXXX XXXXXXXX

In the 1st Byte, all 8 bits are set. This is where the number's highest value bits are located.

In the 2nd Byte, all 8 bits are set. This is where the number's lowest-value bits are located.

Examples:

Starting at the 3rd Byte of a CAN-message having the standard ID 22 Hex, there is a 16-bit unsigned integer led by the most significant Byte. 1 LSB = 5 N.

```
tsaFil = TsaFilter ( tsa, "CAN.ID.std", 0x22, 0x22)
N = TsaDecode( tsaFil, "CAN.data", 2, 0, 16, "unsigned", "motorola", 5.0, 0.0, "N")
```

Get all CAN messages with standard ID 100

```
tsaFil = TsaFilter ( tsa, "CAN.ID.std", 100, 100)
```

Extract all CAN frames with extended ID 0x374500 according to J1939, whith PF=0x37 and PS=0x45. The lowest 8 bits are the "source address" and are to be ignored. The highest 3 bits of the 29-bit ID are the "priority" and are also to be ignored.

```
tsaFil = TsaFilter ( tsa, "CAN.ID.ext", 0x374500, 0x374500, 0x3ffff00 )
```

Get all CAN messages having standard ID 33 Hex as well as a 7 in the first data bytes (Intel, unsigned).

```
tsaFil2 = TsaFilter ( tsa, "CAN.ID.std", 0x33, 0x33)
tsaFil = TsaFilter ( tsaFil2, "CAN.data", 7, 7, 0, 0, 0, 16, "unsigned", "intel")
```

Extract all CAN messages whose standard ID does not lie in the range 100 to 110.

```
tsaFil = TsaFilter ( tsa, "CAN.ID.std", 111, 99)
```

Numerical example. These filters write the specified frame to the result.

```
tsa = TsaCreateEmpty(1, 0.1, "s")
b01 = [0x31, 0x32, 0x33, 0x81, 0x77, 0x55] ; ext id
setdatFormat( b01, 5, 0, 255 )
TsaAppend (tsa, b01, 1.7 )
tsaFil = TsaFilter ( tsa, "CAN.ID.ext", 0x01333231, 0x01333231)
tsaFil = TsaFilter ( tsa, "CAN.ID.msb", 0x81333231, 0x81333231)
tsaFil = TsaFilter ( tsa, "CAN.ID", 0x01333231, 0x01333231)
tsaFil = TsaFilter ( tsa, "CAN.ID.ext", 0, 0x1fffffff, 0)
tsaFil = TsaFilter ( tsa, "CAN.ID.msb", 0, 0x9fffffff, 0)
tsaFil = TsaFilter ( tsa, "CAN.ID", 0, 0x1fffffff, 0)
tsaFil = TsaFilter ( tsa, "CAN.len", 2, 2)
tsaFil = TsaFilter ( tsa, "CAN.len", 0, 8)
tsaFil = TsaFilter ( tsa, "len", 6, 6)
tsaFil = TsaFilter ( tsa, "CAN.data", 0x5577, 0x5577, 0, 0, 0, 16, "unsigned", "intel")
tsaFil = TsaFilter ( tsa, "CAN.data", 0x5577, 0x5577, 0xffff, 0, 0, 16, "unsigned", "intel")
tsaFil = TsaFilter ( tsa, "CAN.data", 0x0570, 0x0570, 0x0ff0, 0, 0, 16, "unsigned", "intel")
tsaFil = TsaFilter ( tsa, "CAN.data", 0x7755, 0x7755, 0, 0, 7, 16, "unsigned", "motorola")
tsaFil = TsaFilter ( tsa, "", 0x7755, 0x7755, 0, 4, 7, 16, "unsigned", "motorola")

tsa = TsaCreateEmpty(1, 0.1, "s")
b02 = [0x31, 0x01, 0x0, 0x0, 0x77, 0x55] ; std id
setdatFormat( b02, 5, 0, 255 )
TsaAppend (tsa, b02, 1.7 )
tsaFil = TsaFilter ( tsa, "CAN.ID.std", 0x131, 0x131)
tsaFil = TsaFilter ( tsa, "CAN.ID.msb", 0x131, 0x131)
tsaFil = TsaFilter ( tsa, "CAN.ID", 0x131, 0x131)
```

See also:

TsaDecode, TsaGetFirst, SetDataFormat

TsaFindBefore

Scope: TimestampASCII data

This function is called to continue the enumeration of time-stamp-ASCII channels.

Declaration:

`TsaFindBefore ( TsaChannel, Position ) -> ResultPosition`

Parameter:

TsaChannel	TsaChannel
Position	Position
ResultPosition	
ResultPosition	ResultPosition

Description:

The parameter to specify is the valid position of an element. The position of the preceeding element within the channel is determined. Return value: Valid position, if >= 0. Otherwise, there is no other element.

Examples:

```
; Initialisation
Position = TsaFindLast ( TsaChannel )
while Position >= 0
   ; read the element here, e.g.
   text = TsaGetText ( TsaChannel, Position )

   ; here to the preceeding element
   Position = TsaFindBefore ( TsaChannel, Position )
end
```

TsaFindFirst

Scope: TimestampASCII data

The position of the 1st channel element is determined.

Declaration:

```
TsaFindFirst ( TsaChannel ) -> Position
```

Parameter:

TsaChannel	TsaChannel
Position	
Position	Position

Description:

This function is called to begin an enumeration of the elements of a time-stamp ASCII channel. Return value: the valid position, if >= 0. Otherwise, there are no elements.

Examples:

```
; Initialisation
Position = TsaFindFirst ( TsaChannel )
while Position >= 0
   ; read the element here, e.g.
   text = TsaGetText ( TsaChannel, Position )

   ; here to the next element
   Position = TsaFindNext ( TsaChannel, Position )
end
```

TsaFindLast

Scope: TimestampASCII data

The position of the last element within the channel is determined.

Declaration:

```
TsaFindLast ( TsaChannel ) -> Position
```

Parameter:

TsaChannel	TsaChannel
Position	
Position	Position

Description:

This function is called to begin an enumeration of the elements of a time-stamp ASCII channel. Return value: the valid position, if >= 0. Otherwise, there are no elements.

Examples:

```
; Initialisation
Position = TsaFindFirstLast ( TsaChannel )
while Position >= 0
   ; read the element here, e.g.
   text = TsaGetText ( TsaChannel, Position )

   ; here to the next element
   Position = TsaFindBefore ( TsaChannel, Position )
end
```

TsaFindNext

Scope: TimestampASCII data

This function is called to continue the enumeration of time-stamp-ASCII channels.

Declaration:

```
TsaFindNext ( TsaChannel, Position ) -> ResultPosition
```

Parameter:

TsaChannel	TsaChannel
Position	Position
ResultPosition	
ResultPosition	ResultPosition

Description:

The parameter to specify is the valid position of an element. The position of the next element in the channel is determined. Return value: Valid position, if >= 0. Otherwise, there is no other element.

Examples:

```
; Initialisation
Position = TsaFindFirst ( TsaChannel )
while Position >= 0
   ; read the element here, e.g.
   text = TsaGetText ( TsaChannel, Position )

   ; here to the next element
   Position = TsaFindNext ( TsaChannel, Position )
end
```

TsaFindTime

Scope: TimestampASCII data

At a specified time, the position of the first time-stamp-ASCII element whose time is greater than or equal to the specified time, is returned.

Declaration:

```
TsaFindTime ( TsaChannel, Time ) -> Position
```

Parameter:

TsaChannel	TsaChannel
Time	Time
Position	
Position	Position

Description:

The time is entered in relative terms: as the time after the trigger release, without the absolute trigger time, date or time of the measurement start. Adjacent elements can well have the same time. Return value: Valid position, if >= 0.

Examples:

```
Time = 45.0 ; searching for element at >= 45s.
Position = TsaFindTime ( TsaChannel, Time )
; e.g. query the text
text = TsaGetText ( TsaChannel, Position )
; query the actual time:
Time = TsaGetTime ( TsaChannel, Position )
```

TsaFindValidPos

Scope: TimestampASCII data

At a specified position in a time-stamp-ASCII channel, next equal or larger valid position of an element is determined.

Declaration:

```
TsaFindValidPos ( TsaChannel, Position ) -> ResultPosition
```

Parameter:

TsaChannel	TsaChannel
Position	Position
ResultPosition	
ResultPosition	ResultPosition

Description:

Return value: Valid position if >= 0. Otherwise no other element exists. Normally, the valid positions should be located using TsaFindFirst() and TsaFindNext().

Examples:

The time for a sample at position >= 100000 is to be determined.

```
Position = TsaFindValidPos ( TsaChannel, 100000 )
if Position >= 0
   Time = TsaGetTime ( TsaChannel, Position )
end
```

TsaGetCount

Scope: TimestampASCII data

For a time-stamp-ASCII channel, the number of constituent elements is determined.

Declaration:

```
TsaGetCount ( TsaChannel ) -> Data
```

Parameter:

TsaChannel	TsaChannel
Data	
Data	Data

Description:

Examples:

```
Count = TsaGetCount ( TsaChannel )
```

TsaGetData

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element's contents are returned as a waveform.

Declaration:

```
TsaGetData ( TsaChannel, Position ) -> Data
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Data	
Data	Data

Description:

The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). The waveform returned takes the Byte data format.

Examples:

```
Position = TsaFindFirst ( TsaChannel )
Content = TsaGetData ( TsaChannel, Position )
```

TsaGetText

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element's contents are returned as text.

Declaration:

```
TsaGetText ( TsaChannel, Position ) -> Text
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Text	
Text	Text

Description:

The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). TsaGetData() may be preferable, if the element contains binary data.

Examples:

```
Position = TsaFindFirst ( TsaChannel )
Text = TsaGetText ( TsaChannel, Position )
```

TsaGetTime

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element's time is returned.

Declaration:

```
TsaGetTime ( TsaChannel, Position ) -> Time
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Time	
Time	Time

Description:

The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). The returned time is the relative time from the measurement's start.

Examples:

```
Position = TsaFindFirst ( TsaChannel )
Time = TsaGetTime ( TsaChannel, Position )
```

TsaInsert

Scope: TimestampASCII data

At a specified position in a time-stamp-ASCII channel, a new element is inserted.

Declaration:

```
TsaInsert ( TsaChannel, Position, Data, Time )
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Data	Data
Time	Time

Description:

The contents and time of the new element are to be specified. The contents are to be expressed as a waveform. The waveform is transfered byte by byte, no matter in which data format it is expressed. It makes sense for the waveform to take the Byte numerical format. An element's maximum size (see time-stamp-ASCII documentation) must not be exceeded. The time may not be less than that of its predecessor and not more than that of its successor. Times must increase monotonously but not strictly monotonously; i.e., adjacent elements' times may be the same. The time may be rounded, since the time is expressed internally as x0+i*dx, where i >= 0 is an integer. The time is entered in relative terms: as the time after the trigger release, without the absolute trigger time, date or time of the measurement start. The parameter channel is changed in the process. The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). As a consequence of this function, the subsequent elements' positions generally change also. Also as a result of this function, the element originally at the specified position is moved to after the newly inserted element.

Examples:

An element containing ten values from 48 to 57 is inserted at the beginning.

```
Content = ramp ( 48, 1, 10 )
SetDatFormat ( Content, 5, 0, 255) ; Makes the numerical format Byte (0..255).
Time = 20 ;

Position = TsaFindFirst ( TsaChannel )
TsaInsert ( TsaChannel, Position, Content, Time )
```

TsaInsertText

Scope: TimestampASCII data

At a specified position in a time-stamp-ASCII channel, a new element is inserted.

Declaration:

```
TsaInsertText ( TsaChannel, Position, Text, Time )
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Text	Text
Time	Time

Description:

The contents and time of the new element are to be specified. The contents are to be expressed as text. The time may not be less than that of its predecessor and not more than that of its successor. Times must increase monotonously but not strictly monotonously; i.e., adjacent elements' times may be the same. The time may be rounded, since the time is expressed internally as x0+i*dx, where i >= 0 is an integer. The time is entered in relative terms: as the time after the trigger release, without the absolute trigger time, date or time of the measurement start. The parameter channel is changed in the process. The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). As a consequence of this function, the subsequent elements' positions generally change also. Also as a result of this function, the element originally at the specified position is moved to after the newly inserted element.

Examples:

An element with a fixed text is inserted at the beginning.

```
Text = "hello"
Time = 20 ;

Position = TsaFindFirst ( TsaChannel )
TsaInsertText ( TsaChannel, Position, Text, Time )
```

TsaJoin

Scope: TimestampASCII data

A second time-stamp ASCII channel is appended to one time-stamp ASCII channel.

Declaration:

```
TsaJoin ( TsaChannel, TsaChannelJoin, UseTriggerTime, DoMerge, AllowDuplicate )
```

Parameter:

TsaChannel	TsaChannel
TsaChannelJoin	The time-stamp ASCII channel to be appended. Or 0 in the special case described below.
UseTriggerTime	Apply the trigger time
	0 : The trigger time is ignored. Only the relative time-stamp (time from start/trigger) of the elements is taken into account.
	1 : Trigger time taken into account. The absolute time-stamp (date and time) of the elements is taken into account.
DoMerge	Should the values of the channel to be appended also be inserted?
	0 : Append only. Only such elements are appended whose time-stamp is not less than the last one in TsaChannel. All other elements are ignored.
	1 : The elements of the channel to be appended are inserted and/or appended at the appropriate locations among the elements of TsaChannel.
AllowDuplicate	If any elements are identical in both TsaChannel and the channel to be appended, they can be inserted either only once or in duplicate.
	0 : Don't allow duplicates
	1 : Allow duplicates

Description:

The function allows appending/merging of data sets in correct time/x-coordinates.

The result applies the time resolution (dx) of TsaChannel. If the value deviates in the appended channel, its times are rounded appropriately and thus altered.

If TsaChannel is not to be altered, it is necessary to work with a copy.

Neither of the channels to be joined may have events.

In a special application, the function can work analogously to EventJoin(): In that case, TsaChannel is a channel with events and TsaChannelJoin is set to the value 0. All event are joined together. Afterwards, TsaChannel has no more events.

Examples:

Two measurements in succession are joined:

```
TsaJoin ( Tsa1, Tsa2, 1, 0, 0 )
```

All events are joined:

```
TsaJoin ( TsaEvents, 0, 1, 0, 0 )
```

TsaSaveAscii

Scope: TimestampASCII data

A time-stamp-ASCII channel is saved to a file in ASCII format.

Declaration:

```
TsaSaveAscii ( TsaChannel, Filename, Separator, Header, AbsTime, DecimalComma, Columns )
```

Parameter:

TsaChannel	TsaChannel
Filename	Filename is the files complete and valid name of the newly created file.
Separator	Separator: The Ascii-code for the separator between columns, e.g. 9 for a TAB, 44 for a comma, 32 for a space.
Header	Header
	0 : no header desired
	1 : a comment, trigger time and unit are written to the file's beginning
AbsTime	AbsTime
	0 : Time specification in the Time column in relative time
	1 : Time specification in the Time column in absolute time with date and time of day
DecimalComma	DecimalComma
	0 : Decimal point
	1 : Decimal comma
Columns	Columns
	0 : Text without time column
	1 : Text with time column (default)
	3 : Hex display with time column
	5 : CAN, LIN message with time column
	7 : Flexray message with time column

Description:

The channel may also have events.

Examples:

The channel TsaChannel is written to the specified file 1.txt with a header and tabs as column separators. As is typical, it is expessed using decimal points and relative time.

```
TsaSaveAscii ( TsaChannel, "c:\imc\dat\1.txt", 9, 1, 0, 0, 1 )
```

TsaSetData

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element's contents are reset. The contents are represented as a waveform.

Declaration:

```
TsaSetData ( TsaChannel, Position, Data )
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Data	Data

Description:

The waveform is transfered byte by byte, no matter in which data format it is expressed. It makes sense for the waveform to take the Byte numerical format. An element's maximum size (see time-stamp-ASCII documentation) must not be exceeded. The parameter channel remains unchanged. The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). When an element's position is changed, the subsequent elements' positions generally change also.

Examples:

The 1st element's contents are set to a ramp.

It will contain 100 bytes, increasing from 1 to 100.

```
Content = ramp ( 1, 1, 100 )
SetDatFormat ( Content, 5, 0, 255) ; Makes the numerical format Byte (0..255).

Position = TsaFindFirst ( TsaChannel )
TsaSetData ( TsaChannel, Position, Content )
```

TsaSetText

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element's contents are reset. The contents are represented as text.

Declaration:

```
TsaSetText ( TsaChannel, Position, Text )
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Text	Text

Description:

The parameter channel is changed in the process. The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos(). When an element's position is changed, the subsequent elements' positions generally change also.

Examples:

The contents of the 1st element are set to a fixed text.

```
Text = "hello"
Position = TsaFindFirst ( TsaChannel )
TsaSetText ( TsaChannel, Position, Text )
```

TsaSetTime

Scope: TimestampASCII data

For a specified position in a time-stamp-ASCII channel, the corresponding element's time is reset.

Declaration:

```
TsaSetTime ( TsaChannel, Position, Time )
```

Parameter:

TsaChannel	TsaChannel
Position	Position
Time	Time

Description:

The time may not be less than that of its predecessor and not more than that of its successor. Times must increase monotonously but not strictly monotonously; i.e., adjacient elements' times may be the same. The time may be rounded, since the time is expressed internally as x0+i*dx, where i >= 0 is an integer. The time is entered in relative terms: as the time after the trigger release, without the absolute trigger time, date or time of the measurement start. The parameter channel remains unchanged. The position must be valid e.g. must be determined using TsaFindTime(), TsaFindFirst(), TsaFindNext() or TsaFindValidPos().

Examples:

The time of the 1st element is set tto 3s.

```
Position = TsaFindFirst ( TsaChannel )
Time = 3.0
TsaSetTime ( TsaChannel, Position, Time )
```

TsaTextToData

Scope: TimestampASCII data

A text is converted byte by byte into a waveform. This waveform will take the Byte numerical format.

Declaration:

```
TsaTextToData ( Text ) -> Data
```

Parameter:

Text	Text
Data	
Data	Data

Description:

Examples:

```
Data = TsaTextToData ( Text )
```

TtoSv

A number is imported in various formats from a text.

Declaration:

`TtoSv ( TxText, TxFormat ) -> SvNumber`

Parameter:

TxText	Text from which the number is to be exported.
TxFormat	Format specification
	"e" : A real or integer decimal number is read. As the decimal separator, a period is expected. Examples: "375" or "-1.3e-4" or "3.124".
	"f" : Identical with "e".
	"a" : A decimal number is read. Decimal period or comma according to global presettings for preferred decimal separator for real numbers ('Extra'/'Options'/'Display' or SetOption("Display.DecimalSeparator",...)
	"b" : A number in binary notation is imported, e.g. "100100".
	"x" : A number in hexadecimal notation is imported, e.g. "9B" or "a21E".
SvNumber	
SvNumber	The return value is the imported number.

Description:

If the text contains no permitted numerical notation and thus no number can be determined, an error message is posted.

The format text is case insensitive.

Examples:

```
x = TtoSv("22", "x")
b = TtoSv("001001", "b")
```

x contains the value 34 (= 22Hex). b contains the value 9.

See also:

TForm, TPart, TConv, SvToChar

TxArrayClean

In a text array, all elements are deleted which meet a specified condition.

Declaration:

`TxArrayClean ( TxaTextArray, SvCondition [, SvUpperLowerCase] [, TxPattern] ) -> SvCountDeleted`

Parameter:

TxaTextArray	Text array
SvCondition	Specifies which elements are to be deleted.
	0 : After running the function, the text array no longer contains any duplicates.
	1 : All empty texts are deleted (length 0).
	2 : All empty texts as well as all texts which contain only blank spaces are deleted. Blank spaces include the space character (0x20) as well as the carriage return character, new-line character, vertical tabulator character, and linefeed character (0x09 through 0x0D).
	3 : All texts are deleted which conform to the pattern specified in the 4th parameter.
	4 : All texts are deleted which do not conform to the pattern specified in the 4th parameter.
	5 : All texts are deleted which conform to the regular expression specified in the 4th parameter.
	6 : All texts are deleted which do not conform to the regular expression specified in the 4th parameter.
SvUpperLowerCase	Specifies whether when testing for duplicates or when comparing an element with the comparison pattern, a distinction is made between the uppercase and lowercase versions of a letter. With [SVCondition] = 1 or 2, not used. (optional , Default value: 0)
	0 : Uppercase and lowercase of the same letter are considered identical.
	1 : Uppercase and lowercase of the same letter are considered different.
TxPattern	For [SVCondition] = 3 or 4, a simple comparison pattern (wildcards "*","?"). For [SVCondition] = 5 or 6, a regular expression. Else, not used. (optional)
SvCountDeleted	
SvCountDeleted	Count of elements deleted (optional)

Description:

In the simple comparison pattern ([SVCondition] = 3 or 4), it is possible to use the wildcard characters "*" and "?" in their respective customary meanings. "?" stands for exactly one arbitrary character, "*" for arbitrary many arbitrary characters.

For ([SVCondition] = 5 or 6), a "regular expression" is used for the comparison pattern. In this way it is possible to formulate complex comparisons. A summary of the syntax of regular expressions is presented in the online help on the function TxRegexMatch().

Examples:

Delete duplicates

```
txa = [ "RPM", "Current", "rpm", "current", "voltage", "rpm"]
; delete all duplicates, case insensitive
count = TxArrayClean(txa, 0)
; txa contains [ "RPM", "Current", "voltage"]; count has the value 3

txa = [ "RPM", "Current", "rpm", "current", "voltage", "rpm"]
; delete all duplicates, case sensitive
TxArrayClean(txa, 0, 1)
; txa contains [ "RPM", "Current", "rpm", "current", "voltage"]

txa = [ "RPM", "rpm"]
; delete all duplicates, case sensitive
count = TxArrayClean(txa, 0, 1)
; txa contains [ "RPM", "rpm"]; count has the value 0
```

Delete empty texts

```
txa = [ "", "   ", "rpm", "current", "", "rpm"]
; delete all texts of length 0
TxArrayClean(txa, 1)
; txa contains [ "   ", "rpm", "current", "rpm"]

txa = [ "", "   ", "rpm", "current", "", "rpm"]
```

```
; delete all texts having the length 0 or consisting only of spaces
TxArrayClean(txa, 2)
; txa contains [ "rpm", "current", "rpm"]
```

Simple pattern comparison

```
txa = [ "current1", "RPM2", "rpm1", "Current2", "voltage"]
; delete all texts beginning with "1"
TxArrayClean(txa, 3, 0, "*1")
; txa enthält [ "RPM2", "Current2", "voltage"]

txa = [ "current1", "RPM2", "rpm1", "Current2", "voltage"]
; delete all texts which do not consist of 3 arbitrary characters followed by a 1.
TxArrayClean(txa, 4, 0, "???1")
; txa contains [ "rpm1"]

txa = [ "current1", "RPM2", "rpm1", "Current2", "voltage"]
; deletes all texts beginning with an uppercase or lowercase 'C'.
TxArrayClean(txa, 3, 0, "C*")
; txa contains [ "RPM2", "rpm1", "voltage"]

txa = [ "current1", "RPM2", "rpm1", "Current2", "voltage"]
; delete all texts which begin with an uppercase  'C'.
TxArrayClean(txa, 3, 1, "C*")
; txa enthält [ "current1", "RPM2", "rpm1", "voltage"]
```

Regular expressions

```
txa = ["asin3.dat", "sin.dat", "SIN11.DAT", "SIN2.raw", "sin1.txt"]
; deletes all filenames whose name consists of "sin" + number and the extension "dat" or "raw".
TxArrayClean(txa, 5, 0, "sin\d+\.(dat|raw)")
; txa contains ["asin3.dat", sin.dat", "sin1.txt"]

txa = ["1", "s", "1e2", "+2e-3", "2,3", "2a", "-1.235"]
; only retain texts which represent a number
TxArrayClean(txa, 6, 0, "[-+]?([0-9]+\.?[0-9]*|\.[0-9]+)([eE][-+]?[0-9]+)?")
; txa contains ["1", "1e2", "+2e-3", "-1.235"]
```

See also:

TxArrayDelete, TLike, TComp, TxRegexMatch

TxArrayCombine

Two text arrays are combined according to a specified criterion.

Declaration:

`TxArrayCombine ( Txa1, Txa2, SvCondition [, SvUpperLowerCase] ) -> TxaResult`

Parameter:

Txa1	First text array
Txa2	Second text array
SvCondition	Specifies the conditions according to which the text arrays are combined.
	0 : OR: The result contains all elements which are contained in either [Txa1] or [Txa2]. The result starts with the elements from [Txa1] and then the elements from [Txa2], each in their original order.
	1 : AND: The result contains elements which are contained both in [Txa1] and in [Txa2]. The order of the result matches the order in [Txa1].
	2 : Exclusive-OR (XOR): The result contains all elements which are contained in [Txa1] or in [Txa2]. The result starts with the elements from [Txa1] and next the elements from [Txa2], each in their original order.
SvUpperLowerCase	Case sensitive (optional , Default value: 0)
	0 : The upper and lower cases of the same letter are regarded as identical, e.g. 'A' = 'a'.
	1 : The upper and lower cases of the same letter are regarded as different, e.g. 'A' <> 'a'.
TxaResult	
TxaResult	Results of the combination of the two text arrays

Description:

The text array of results contains no duplicates.

Examples:

```
Txa1 = [ "current", "Voltage", "rpm" ]
txa2 = [ "RPM", "velocity", "current"]

; all entries which are contained in Txa1 OR txa2:
; -----------------------------------------------------
txaResult = TxArrayCombine(Txa1, txa2, 0, 0)
; txaResult contains ["current", "Voltage","rpm", "velocity" ]

;... case sensitive:
txaResult = TxArrayCombine(Txa1, txa2, 0, 1)
; txaResult contains ["current", "Voltage","rpm", "RPM", "velocity"]

; all entries which are contained in Txa1 AND txa2:
; ----------------------------------------------------
txaResult = TxArrayCombine(Txa1, txa2, 1, 0)
; txaResult contains ["current", "rpm"]

;... case sensitive:
txaResult = TxArrayCombine(Txa1, txa2, 1, 1)
; txaResult contains ["current"]

; all entries which are contained in EITHER Txa1 OR txa2:
; -----------------------------------------------------------------
txaResult = TxArrayCombine(Txa1, txa2, 2, 0)
; txaResult contains ["Voltage", "velocity"]

;... case sensitive:
txaResult = TxArrayCombine(Txa1, txa2, 2, 1)
; txaResult contains ["Voltage", "rpm", "RPM", "velocity"]
```

See also:

TxArrayInsert, TxArrayClean, TxArraySort, TComp, AND

TxArrayCreate

The function creates a text array having the specified initial amount of elements.

Declaration:

```
TxArrayCreate ( SvDimension ) -> TxaNewTextArray
```

Parameter:

SvDimension	Amount of text elements included
TxaNewTextArray	
TxaNewTextArray	Text array having the specified amount of elements

Description:

The function creates a text array variable. The text array is created having the specified amount of elements, which are all initially empty.

A text array variable describes a list of texts which are grouped together under one name and which can be adressed by means of their index.

For setting and getting the individual texts, use the index notation with square brackets, for instance

```
FourTexts = TxArrayCreate(4)
FourTexts[2] = "Just another text"
SecondText = FourTexts[2]
```

You can subsequently change the amount of elements in a text array using the function TxArraySetSize().

If when assigning a text, the specified index is 1 greater than the current size, then internally the size is adjusted automatically. This makes it convenient to use, if for instance the text array in a loop is to be filled up step-by-step and the ultimate size is not yet known.

```
TxArray = TxArrayCreate(0) ; Initial size = 0
Index = 1
WHILE ...
   ...
   TxArray[Index] = AnyText  ; TxArray grows automatically
   Index = Index + 1
END
```

However, as soon as the final value is known, it is more efficient to state is explicitly in TxArrayCreate() or by means of TxArraySetSize().

For enumerating the elements of a text array, the loop FOREACH ELEMENT is well suited.

Alternatively, (short) text arrays can also be conveniently genreated with an initialization list:

```
TxArrayAllUsers = ["Joe", "John", "Jack"]
```

Examples:

In a text array, various information on a spectrum analysis is stored, along with the actual result data in one file.

```
Result1 = AmpSpectrumRMS(Channel1, 1024, 1, 50, 1, 0)
Result2 = AmpSpectrumRMS(Channel2, 1024, 1, 50, 1, 0)
CalculationParameters = TxArrayCreate(3)
CalculationParameters[1] = "Window length: 1024"
CalculationParameters[2] = "Overlapping: 50%"
CalculationParameters[3] = "Window function: Hamming"
FileSave("c:\tmp\result", "", 0, Result1, Result2, CalculationParameters)
```

In a specified folder, all files having the extension .dat are listed. The path name of all files whose creation time is newer than 1/1/2012 are saved in a text array.

```
FileListID = FsFileListNew("c:\copy", "*.dat", 0, 0, 0)
n = FsFileListGetCount(FileListID)
time_fence = TimeJoin(1, 1, 2012, 0, 0, 0)
FileNames = TxArrayCreate(n)
j = 1
FOR i = 1 TO n
   IF FsFileListGetTime(FileListID, i) >= time_fence
      FileNames[j] = FsFileListGetName(FileListID, i)
      j = j + 1
   END
TxArraySetSize(FileNames, j-1) ; reduce size to actual element count
END
```

The content of a multi-channel file is read completely. The names of the channels loaded are recorded in a text array.

```
fh = FileOpenDSF("Samplefile.dat", 0)
IF fh > 0
   count = FileObjNum?(fh)
   LoadedVarNames = TxArrayCreate(count)
   FOR i = 1 TO count
      name = FileObjName?(fh, i)
      <name> = FileObjRead(fh, i)
      LoadedVarNames[i] = name
   END
   FileClose(fh)
END
```

See also:

TxArraySetSize, TxArrayGetSize

TxArrayDelete

Deletes an element of a text array.

Declaration:

```
TxArrayDelete ( TxArray, Index ) -> TxArrayResult
```

Parameter:

TxArray	Text array
Index	Index of the element to be deleted
TxArrayResult	
TxArrayResult	Result text array

Description:

By means of this function, elements of a text array can be deleted.

- The index may only lie in a range between 1 and the text array size.
- If the index is outside of the text array's range boundaries, the sequence will be cancelled.

Examples:

The text array's 3rd element is deleted

```
txArray=TxArrayCreate(5)
txArray[1]="One"
txArray[2]="Two"
txArray[3]="Three"
txArray[4]="Four"
txArray[5]="Five"
txArrayResult = TxArrayDelete(txArray,3)
```

See also:

TxArrayInsert

TxArrayGetSize

The function returns the current size of a text array (meaning the amount of text-elements it contains).

Declaration:

```
TxArrayGetSize ( TxaTextArray ) -> SvDimension
```

Parameter:

TxaTextArray	Variable of the type Text Array to query
SvDimension	
SvDimension	Size, amount of elements

Description:

The amount of elements a text array has is specified when it is created using the function TxArrayCreate() and can be changed subsequently using the function TxArraySetSize().

Examples:

A text box having the name [Log] is loaded from a file and entered into a list in the Panel currently open.

```
FileLoad("c:\logfiles\Log.dat", "", 0)
n = TxArrayGetSize(Log)
FOR i = 1 TO n
   PnInsertItem("list", 0, Log[i], 0)
END
```

Remark: For an enumeration of this sort, it is also possible to use a FOREACH ELEMENT-loop.

A text array having the name [Log] is loaded from a file and expanded with the entries of an existing text array [CurrentLog]. Subsequently, the updated text array is saved again.

```
FileLoad("c:\logfiles\Log.dat", "", 0)
size = TxArrayGetSize(Log)
current_size = TxArrayGetSize(CurrentLog)
TxArraySetSize(log, size + current_size)
FOR i = 1 TO current_size
   Log[size+i] = CurrentLog[i]
END
FileSave("c:\logfiles\Log.dat", "", 0, log)
```

TxArrayInsert

Inserts a text or text array into a text array.

Declaration:

```
TxArrayInsert ( TxArray1, TextOrTxArray2, Index ) -> TxArrayResult
```

Parameter:

TxArray1	Either the text or Text Array 2 is inserted into this text array.
TextOrTxArray2	Text or text array which is inserted into Text Array 1.
Index	Index in Text Array 1. The text or Text Array 2 is inserted ahead of this index.
TxArrayResult	
TxArrayResult	Result text array

Description:

By means of this funciton, text arrays can be joined.

- The 2nd parameter may be either a text or a text array.
- The text or Text Array 2 is inserted ahead of the index specified.
- If the index is -1, then the text or Text Array 2 is appended to Text Array 1.
- The index may only lie within the range from 1 to the text array size.
- If the index is outside of Text Array 1's range boundaries, the sequence will be cancelled.

Examples:

TextArray2 will be appended to TextArray 1.

```
txArrayResult=TxArrayInsert(txArray1,txArray2,-1)
```

TextArray2 will be appended to Text Array 1 in front of the index=1.

```
txArrayResult=TxArrayInsert(txArray1,txArray2,1)
```

A text will be appended to Text Array 1 before the index=3.

```
txArrayResult=TxArrayInsert(txArray1,"Hello",3)
```

See also:

TxArrayDelete

TxArrayPart

A section of a text array is copied to a new text array.

Declaration:

```
TxArrayPart ( TxaSource, SvIndex, SvCount ) -> TxaPartCopy
```

Parameter:

TxaSource	Text array whose elements are to be copied.
SvIndex	Index of the first element to be copied. The first element has the index 1.
SvCount	Count of elements to be copied. -1, if all elements down to the last are to be copied.
TxaPartCopy	
TxaPartCopy	Newly generated text arry with copies of the specified elements

Description:

This function creates a new text array which represents the copy of a section of the parameter text array.

When a -1 is specified for [SvIndex], or when ([SvIndex]+[SvCount]) is greater than the count of elements in the source text array, all elements are copied all the way to the end.

Examples:

The last 3 elements of a text array are copied to a new text array.

```
n = TxArrayGetSize(txaSource)
txaSourcePart = TxArrayPart(txaSource, n-2, 3)
```

Suppose a text array contains a large amount of filenames. We want to load and evaluate the associated files.

```
allFiles = FsGetFileNames("c:\data", "*.raw", 0, 1, 0)
FOREACH ELEMENT file in allFiles
   !WorkWithFile(file) ; loads the file and carries out the evaluation
END
```

This routine is to be accelerated by means of parallel execution. For this purpose, the text array is subdivided into 4 parts which are all executed in parallel.

```
allFiles = FsGetFileNames("c:\data", "*.raw", 0, 1, 0)
count = TxArrayGetSize(allFiles)
chunksize = floor(count/4)
BEGIN_PARALLEL
   !WorkWithFiles(TxArrayPart(allFiles, 1,              chunksize))
   !WorkWithFiles(TxArrayPart(allFiles, 1+ chunksize,   chunksize))
   !WorkWithFiles(TxArrayPart(allFiles, 1+ chunksize*2, chunksize))
   !WorkWithFiles(TxArrayPart(allFiles, 1+ chunksize*3, count-3*chunksize))
END_PARALLEL
```

For this purpose, the sequence function !WorkWithFiles is defined as follows:

```
; Declaration: !WorkWithFiles(Par1 [Data type: Textarray])
FOREACH ELEMENT file in Par1
   !WorkWithFile(file)
END
```

See also:

TxArrayInsert, TxArrayGetCount

Supported since:

Version 2022

TxArraySetSize

The function changes a text array's size (meaning the amount of text-elements it contains).

Declaration:

`TxArraySetSize ( TxaTextArray, SvDimension )`

Parameter:

TxaTextArray	Variable of type Text Array to be changed
SvDimension	New size, amount of elements

Description:

The amount of elements a text array has is specified when it is created using the function TxArrayCreate() and can be changed subsequently using the function TxArraySetSize().

When the size is reduced, the content of the remaining texts remains unchanged. When increasing the size, the new elements are initialized with an empty text.

When a text is assigned, if the index specified is 1 higher than the momentary size, the size is automatically adapted internally. This is a convenience for operation, for instance if the text array is to be filled step-by-step in a loop and the ultimate size is not yet known. However, as soon as the final size is known, it is more efficient to state it explicitly using TxArrayCreate() or TxArraySetSize().

Examples:

A text array having the name [Log] is loaded from a file and expanded with the entries of an existing text array [CurrentLog]. Subsequently, the updated text array is saved again.

```
FileLoad("c:\logfiles\Log.dat", "", 0)
size = TxArrayGetSize(Log)
current_size = TxArrayGetSize(CurrentLog)
TxArraySetSize(Log, size + current_size)
FOR i = 1 TO current_size
   Log[size+i] = CurrentLog[i]
END
FileSave("c:\logfiles\Log.dat", "", 0, log)
```

See also:

TxArrayCreate, TxArrayGetSize

TxArraySort

The elements in a text array are sorted.

Declaration:

`TxArraySort ( TxaTextArray, SvMode, SvOrder )`

Parameter:

TxaTextArray	Text array
SvMode	The mode according to which the elements are to be sorted.
	0 : Alphabetic sorting. Upper and lower cases of the same letter are regarded as identical, e.g. 'A' = 'a'.
	1 : Alphabetical sorting. Upper and lower cases of the same letter are regarded as different, e.g. 'A' < 'a'.
	2 : Natural sorting in ascending order. With the natural sorting order, alphabetical sorting applies in principle, however digits apearing in texts are sorted according to their value. Thus for example, 'Channel_2' appears before 'Channel_10'. Upper and lower cases of the same letter are regarded as identical.
SvOrder	Order in which the elements are to be sorted
	0 : Increasing order, the "smallest" element first
	1 : Descending order, the "largest" element first.

Examples:

```
txa = [ "channel_10", "Channel_2", "channel_1"]

; alphabetic; case insensitive
TxArraySort(txa , 0, 0)
; txa = [ "channel_1", "channel_10", "Channel_2"]

; alphabetic; case sensitive
TxArraySort(txa , 1, 0)
; txa = [ "Channel_2", "channel_1", "channel_10"]

; natural
TxArraySort(txa , 2, 0)
; txa = [ "channel_1", "Channel_2", "channel_10"]
```

See also:

TxArrayClean, TComp, Sort

TxArrayToChannel

The elements of a text array are converted to numbers and a numerical data set is constructed from these.

Declaration:

```
TxArrayToChannel ( TxaTextArray, SvFormat [, SvReplacementValue] ) -> Data
```

Parameter:

TxaTextArray	Text array from whose elements the numbers are to be read out.
SvFormat	Format specification
	0 : The numbers are expected to be expressed as either integers or real decimal numbers. Either a point or comma is allowed as a decimal separator. Examples: "375" or "-1.3e-4" or "3,124" or "2E11".
	1 : The numbers are expected to be expressed in binary representation, e.g. "100100".
	2 : The numbers are expected to be expressed in hexadecimal representation, e.g. "9B" or "a21E". The prefix '0x' or '0X' is allowed but not necessary.
SvReplacementValue	Optional substitute value to be used when an element can not be converted. (optional)
Data	
Data	Data set with the values read.

Description:

How the function behaves when a text element contains no permitted numerical representation, and thus no number can be determined, depends on the optional parameter [SVSubValue].

- If this optional parameter is specified, it is applied in the result as a substitute value. The result always has the exact same number of values as the text array has elements.
- If omitted, the conversion is cancelled and the values previously read successfully are returned. The length of the result matches the array index of the last succesfully converted text.

When converting from the decimal format, the result's data format is '8-Byte Real' (double). As the decimal separator, a point or a comma can be used; a separator character for thousands is not allowed.

When converting from the binary or hexadecimal format, the result's data format is '4-Byte unsigned integer'; thus numbers in the range 0..4294967295 can be converted.

Initial and final spaces in the texts are allowed.

In order to convert a text containing a series of numbers, you can initially disassemble the text with TxSplit() to a text array and then use the function TxArrrayToChannel().

Examples:

```
txa = ["-1", " 1.2e3", "1,23"]
data = TxArrayToChannel(txa, 0)
; data has the value [-1, 1200, 1.23]
```

```
tx = "-1,  1.2e3, 1.23"
data = TxArrayToChannel(TxSplit(tx, ","), 0)
; data has the value [-1, 1200, 1.23]
```

```
txa = ["11.2", "", "---", "1,23"]
data = TxArrayToChannel(txa, 0)
; data has the value [11.2]
data = TxArrayToChannel(txa, 0, -1)
; data has the values [11.2, -1, -1, 1.23]
```

```
txa = [" 1", "a  ", "FFFFFFFF", "10"]
data = TxArrayToChannel(txa, 2)
; data has the values [1, 10, 4294967295, 16]
```

```
txa = [" 01", "-a", "", "FFFFFFFFF", "10"]
data = TxArrayToChannel(txa, 2)
; data has the value [1]
data = TxArrayToChannel(txa, 2, 0)
; data has the values [1, 0, 0, 0, 16]
```

```
txa = [" 1", "010", "101"]
data = TxArrayToChannel(txa, 1)
; data has the values [1, 2, 5]
```

```
txa = ["-1", "12"]
data = TxArrayToChannel(txa, 1)
; data is empty
data = TxArrayToChannel(txa, 1, 0)
; data has the values [0,0]
```

The first column of a multi-column text file containing numerical values (columns separated by commas) is to be imported:

```
idFile = FileOpenASCII("z:\000.txt", 0)
TxaLines = TxArrayCreate(0)
ret = FileLineRead(idFile, TxaLines, 0) ; read all text lines from file
IF ret = 0
   TxaLines = TxRegexMatch(TxaLines, "^[^,]+", " ", 0, 0) ; split 1st column (comma separated) from line
   ; To get the 3rd column, you could use the following line (note the '{2}' for the the zero based column index):
   ; TxaLines = TxRegexMatch(TxaLines, "^(?:[^,]+,){2}([^,]+)", " ", 0, 1)
   channel = TxArrayToChannel(TxaLines, 0);
END
FileClose(idFile) ; close file processing
```

Remarks: Such import tasks can often be accomplished more conveniently using the ASCII-Import-Assistant.

See also:

TxFind

The system searches for a text in the text array or text.

Declaration:

```
TxFind ( TextOrTxArray, TxFindText, Occurrence, Options ) -> Result
```

Parameter:

TextOrTxArray	Text or text array in which to find the text
TxFindText	Text to search for
Occurrence	The n-th occurrence of the text for which the system searches
	-1 : Finds the last occurrence of the search text.
	1... : Finds the n-th occurrence of the search text.
	0 : Finds all occurrences of the search text (not for Textarrays).
Options	Search criteria options
	0 : Case sensitive
	1 : Case insensitive
Result	
Result	Data set with the result

Description:

The system searches for a text or text array for the occurence of a text.

This function's result is a normal data set.

The data set length corresponds to the text array's dimensions, i.e. there is onle value in the data set for each element in the text array.

This value denotes the first position of the n-th occurrence of the search text in the text array element.

If a value is = 0, then the search text is not found in this element of the text array.

The parameter Occurrence specifies at which occurrence of the search text to determine the position.

If the 1st parameter is a simple text (no array), then Occurence = 0 is also allowed. In this case, the positions of every occurrence of the search text are returned as as data set (length of 0, when the search text can't be found).

- Invalid options cause the sequence to be cancelled.
- An empty search text causes the sequence to be cancelled.

Examples:

The system searches for the term "Form" in a text array.

```
txArray=TxArrayCreate(3)
txArray[1]="Decimal format and binary format"
txArray[2]="Conversion"
txArray[3]="The form is valid"
result=TxFind( txArray,"form",1,1)
```

The data set "result" contains the values 9,0,5

```
result=TxFind( txArray,"form",2,1)
```

The data set "result" contains the values 27,0,0

See also:

TxReplace, TxRegexMatch, TxRegexReplace

TxFormatEx

Function for formatting texts

Declaration:

`TxFormatEx ( TxFormat, Parameter1 [, Parameter2] [, Parameter3] [, Parameter4] [, Parameter5] [, Parameter6] [, Parameter7] [, Parameter8] ) -> TxResult`

Parameter:

TxFormat	Formatting string
Parameter1	1st parameter for the formatting
Parameter2	2nd parameter for the formatting (optional)
Parameter3	3rd parameter for the formatting (optional)
Parameter4	4th parameter for the formatting (optional)
Parameter5	5th parameter for the formatting (optional)
Parameter6	6th parameter for the formatting (optional)
Parameter7	7th parameter for the formatting (optional)
Parameter8	8th parameter for the formatting (optional)
TxResult	
TxResult	Result text

Description:

The function receives the formatting string with text and formatting elements, as well as parameters. The text is returned with the parameters inserted into the respective formatting.

The parameters may be texts or single values.

The formatting string consists of text, format elements and optionally a culture element. The formatting elements are replaced with the parameters. A format element determines which parameter is used and how it is formatted.

The formatting string consists of text, format elements and optionally a culture element. The formatting elements are replaced with the parameters. A format element determines which parameter is used and how it is formatted.:**{[T]Index[Alignment][:FormatString]}**

The matching curved brackets ("{" and "}") are required and interpreted as the start and finish of a format element. In order to output an opening or closing curved bracket as actual characters, they must be entered in duplicate.

The optional statement **T** at the start of a format element denotes the associated parameter as a time object. It interpretes the affected parameter as Date/Time-value in imc Time-format (Seconds since 1980/1/1). A format string representing the date and time must be entered. A parameter can not simultaneously be formatted as a time object and as a numerical value.

The obligatory **Index** denotes the parameter to be used. It begins at 1 and ends with 8 (since only a maximum of 8 parameters can be passed). Multiple format elements can access the same parameter, by using the same index in the format elements. For the indices in the format elements, the corresponding parameters must be passed. If an index is specified in a format element, for which no parameter was passed, the sequence is canceled.

The optional statement of the **alignment** is a signed integer. This indicates the total length of the field into which the argument is inserted. For a positive number, the alignment is right-justified, otherwise left-justified. As fill characters, spaces are used. If the alignment is less than the length of the formatted characters, then the alignment is ignored. The comma is required for specifying an alignment.

The optional **Format-String** governs the formatting. It must match the parameter. If the parameter is a numerical value, then a numerical format string must be used. If the parameter is a time value (date/time), then a format string for date and time must be supplied. If the format The colon is required when a format string is supplied.

If the format string is missing, then the format string "G" is used. The the most compact output with 7 significant places(input data 4 byte real, 1/2 byte integer) or 15 significant places (8 byte real, 4/8 byte integer) is choosen.

The tables below are a compilation of the valid format characters for numeric and date-/time values. These tables are not guaranteed to be exhaustive. The examples provided are all based on the setting en-US for the culture.

Default format-strings for numerical values:

Format character	Description
E or e	Output in exponential format. By appending a number it is possible to specify the amount of decimal places. Example: 1052.0329112756 ("E") -> 1.052033E+003
F or f	Output of a number in decimal format. By appending a number it is possible to specify the amount of decimal places. By default, two decimal places. Example: -1234.56 ("F4") -> -1234.5600

G or g	The most compact fixed-point or scientific notation. By appending a number, it is possible to define the amount of significant digits. Example: 123.4546 ("G4") -> 123.5
N or n	Integers or decimal numbers with group separator characters, a decimal separator character, and with an optional minus sign. By appending a number it is possible to specify the amount of decimal places. The output follows the format ddd.ddd.ddd,dd. Example: -1234.56 ("N3") -> -1,234.560
P or p	The number is multiplied by 100 and associated with a percent character. By appending a number it is possible to specify the amount of decimal places. Example: 1 ("P") -> 100.00 %
C or c	Output in the currency format (including the currency symbol for the current country setting). By appending a number, the amount of decimal places can be set. Example: 123.456 ("C") -> $123.46

User-defined format-strings for numerical values

Format character	Description
0	The number 0 serves as a placeholder for a number. Non-significant zeros are represented by the number 0. Example: 1234.5678 ("00000") -> 01235
#	Replaces the "#"-symbol with a corresponding digit if available, else no digit in the result. Example: 0.45678 ("#.##") -> .46
.	The first period character in the format-string determines the position of the decimal separator in the formatted value. Example: 0.45678 ("0.00") -> 0.46
,	Serves to separate the thousands digits. Example: 2147483647 ("##,#") -> 2,147,483,647
%	This character causes multiplication by 1000. The per-mill sign is included in the output. Example: 0.3697 ("%#0.00") -> %36.97
E0, E+0, E-0, e0, e+0, e-0	These format characters cause a number to be expressed in exponential notation. With E+0 and e+0, the positive sign is always displayed; with all others, always only the negative. The amount of zeroes determines the minimum number of digits in the exponent. Example: 987654 ("#0.0e0") -> 98.8e4
\	This character, following the Escape-character, is interpreted as an actual literal slash and not as a user-defined format character. Example: 987654 ("\###00\#") -> #987654#

Default format strings for date-/time values

Format character	Description
d	Short date Example: 2016-06-02T14:50:09 ("d")-> 6/2/2016
D	Long date Example: 2016-06-02T14:50:09 ("D")-> Thursday, June 02, 2016
f	Long date with short time indication Example:: 2016-06-02T14:50:09 ("f")-> Thursday, June 02, 2016 2:50 PM
F	Long date with long time indication Example: 2016-06-02T14:50:09 ("F") -> Thursday, June 02, 2016 2:50:09 PM
g	Short date with short time indication Example: 2016-06-02T14:50:09 ("g") -> 6/2/2016 2:50 PM
G	Short date with long time indication Example: 2016-06-02T14:50:09 ("G") -> 6/2/2016 2:50:09 PM
M or m	Day and month Example: 2016-06-02T14:50:09 ("M") -> June 02
R or r	Date according to pattern of RFC 1123 Example: 2016-06-02T14:50:09 ("R")-> Thu, 02 Jun 2016 14:50:09 GMT
t	Short time indication Example: 2016-06-02T14:50:09 ("t") -> 2:50 PM
T	Long time indication Example: 2016-06-02T14:50:09 ("T") -> 2:50:09 PM
Y or y	Month and year Example:: 2016-06-02T14:50:09 ("Y") -> June, 2016

User-defined format strings for date-/time values

Format character	Description

dd / d	Day of the month, with/without preceding zero Example: 2016-06-02T14:50:09 ("dd") -> 02
ddd	Abbreviation of weekday Example: 2016-06-02T14:50:09 ("ddd") -> Thu
dddd	Complete name of weekday Example: 2016-06-02T14:50:09 ("dddd") -> Thursday
f / ff / .. / fffffff	Indication of 1/10-second, 1/100 seconds ... 1/10-microseconds Example: 2016-06-02T14:50:09.123456 ("fff ") -> 123
HH / H	Hour indication in 24-hour notation, with/without preceding zero Example: 2016-06-02T14:50:09 ("HH") -> 14
hh / h	Hour indication in 12-hour notation, with/without preceding zero Example: 2016-06-02T14:50:09 ("hh") -> 02
MM / M	Month with/without preceding zero Example: 2016-06-02T14:50:09 ("MM") -> 06
MMM	Abbreviation of the month name Example: 2016-06-02T14:50:09 ("MMM") -> Jun
MMMM	Complete month name Example: 2016-06-02T14:50:09 ("MMMM") -> June
mm / m	Indication of minutes with/without preceding zero Example: 2016-06-02T14:50:09 ("mm") -> 50
ss / s	Indication of seconds with/without preceding zero Example: 2016-06-02T14:50:09 ("ss") -> 09
tt	The AM/PM indicator Example: 2016-06-02T14:50:09 ("tt") -> PM
yy / y	Two-digit year number with/without preceding zero Example: 2016-06-02T14:50:09 ("yy") -> 16
yyyy	Complete year number Example: 2016-06-02T14:50:09 ("yyyy") -> 2016
zz / z	Time zone indication (+ or -, followed by the hour indication; with/without preceding zero) Example: 2016-06-02T14:50:09 ("zz") -> +02
zzz	Four-digit time zone indication Example: 2016-06-02T14:50:09 ("zz") -> +02:00
:	Output of time separator Example: 2016-06-02T14:50:09 (": ") -> :
/	Output of date separator Example: 2016-06-02T14:50:09 ("/ ") -> /

A **culture element** may be positioned at the start of the formatting string

If no culture element is provided, then the current culture in Windows will be applied. This can be found in the imc FAMOS-settings (see "Settings for Display/ Curve window").

If in those settings, the decimal period was the preferred decimal separator selected, then real numbers, currencies or statements of percentage are always expressed with a period as the decimal separator.

If in those settings, the decimal comma was the preferred decimal separator selected, then real numbers, currencies or statements of percentage are always expressed with a comma as the decimal separator. The grouping separator character (thousands separator of magnitude) is suppressed in both cases.

If the formatting is to comply with a certain culture, it is possible to specify a culture element. The culture element has the following structure: **{xx-yy}**. xx-yy stands for the culture name. It consists of 2 characters for the language-code according to ISO 639-1 and 2 characters for the Country/Region-code according to ISO 3166.

Examples:

Cultur elements	Meaning
{}	Invariant culture - No culture is applied.
{en}	Neutral culture English This culture is associated with English language, but no specific country.
{en-US}	Culture: English and Country: USA
{en-GB}	English - Great Britain
{de-DE}	German - Germany
{ja-JP}	Japanese - Japan

{zh-CN}	Chinese - PR China
{zh-TW}	Chinese - Taiwan

If the resources for a particular culture are not available in the operating system, the resources for the associated neutral culture are used.

Examples:

Output of the same parameter in exponential- and decimal representation.

```
para2=2341052.0329112756
result=TxformatEx("Value, exponential= {1:e6}, Value, decimal={1:F6}",para2
```

result="Value exponential= 2.341052e+006, Value decimal=2341052.032911"

Create a channel name with fix prefix and appended number.

```
index = 12
ChannelName=TxformatEx("Chan{1:#}", index)
```

ChannelName has the content "Chan12".

Create a channel name with fixed prefix and appended number (4 digits, preceding zeroes.

```
index = 12
ChannelName=TxformatEx("Chan{1:0000}", index)
```

ChannelName has the content "Chan0012".

An error message is constructed:

```
TxChanname = "u1"
TxMsg = TxFormatEx( "The frequency of {1} is too low, {1}_result is imprecise", TxChanname)
```

TxMsg has the content "The frequency of u1 is too low, u1_result is imprecise"

Output of time information with long date and time in US-format

```
time=TimeJoin(02,06,2016,14,50,09)
result=TxFormatEx("{en-US}{T1:F}",time)
```

result="Thursday, June 02, 2016 2:50:09 PM"

Output of time information with user-defined formatting in German format

```
time=TimeJoin(02,06,2016,14,50,09)
time=time+0.123456
result=TxFormatEx("{de}{T1:dd.MM.yyyy HH:mm:ss ffffff }",time)
```

result="02.06.2016 14:50:09 123456"

See also:

TxGetValidVarName

The function forms a valid variable name from a text

Declaration:

```
TxGetValidVarName ( TxText ) -> TxVarName
```

Parameter:

TxText	String
TxVarName	
TxVarName	Variable name

Description:

Converts an arbitrary text into a valid FAMOS variable name.

- Toward this end, all invalid characters are replaced with an underline "_".
- Invalid characters include:
  - All characers with an ASCII-code < 32
  - < > + - ( ) * / ^ = { } [ ] | . @ ' : , Comma, Semicolon, Spaces and quotation marks
- If the 1st character is a digit, it is prefixed by an underline.
- If the text represents a FAMOS-constant, an underline is prefixed.
- If the text has more than 255 characters, the text is truncated after 255 characters.
- If an empty text is provided, the sequence is cancelled.

Examples:

```
TxGetValidVarName("ValidName") => "ValidName"
tab="~009"
TxGetValidVarName("Valid"+tab+"Name") => "Valid_Name"
TxGetValidVarName("1"+tab+"ValidName") => "_1_ValidName"
TxGetValidVarName("<Valid>.Name") => "_Valid__Name"
TxGetValidVarName("pi") => "_pi"
```

See also:

TForm

TxRegexMatch

Finds texts in a text array or in a text by means of a regular expression.

Declaration:

```
TxRegexMatch ( TextOrTxArray, TxPattern, TxResultSeparator, Options [, GroupIndex] ) -> TextOrTxArrayResult
```

Parameter:

TextOrTxArray	Text or text array in which to search for the pattern
TxPattern	Search pattern
TxResultSeparator	String for separating results
Options	Options for the search. The individual values may be added.
	0 : No option
	1 : IgnoreCase
	2 : Multiline
	4 : ExplicitCapture
	16 : Singleline
	32 : IgnorePatternWhitespace
GroupIndex	Group index, Parameter ist optional, Default value = 0 (optional)
TextOrTxArrayResult	
TextOrTxArrayResult	Contains the texts found

Description:

If the function's first parameter is a text, the return value is also a text.

If the first parameter is a text array, then a text array with the same dimension is returned.

The meanings of the options:

- IgnoreCase: The matching is case insensitive
- Multiline: Changes the interpretation of ^ and $ to match the beginning and end of any line and not the beginning and end of the entire input string. Circumflex and Dollar-sign match line break.
- ExplicitCapture: With this option, the familiar parentheses (...) which normally bracket text become non-capturing brackets. Thus, they have the same effect as (?:...) and only serve the purpose of grouping. With named bracketed expressions (?<Name>...), it is still possible to capture text fragments as before.
- SingleLine: This option modifies the behavior of the period character (.) . In this case the period matches every character; normally the Newline is exempted. With this option, line breaks are also recognized with the period character.
- IgnorePatternWhitespace: When this option is selected, all white space in the expression pattern which is either unescaped or within character classes is ignored. This allows expressions to be formed in a clearer manner. Furthermore, all characters (including the next line break) from outside of a character class which are located between two unescaped #-characters are ignored. In this way, it is possible to insert comments into the regular expression.
- Invalid options cause the sequence to be cancelled.

If the regular expression finds multiple texts in the input text, then they are all returned in the result text, separated by the separator specified in TxResultSeparator.

An empty search pattern causes the sequence to be cancelled.

The regular expression pattern can include subexpressions, which are defined by a part of the pattern in parentheses () is included. That subexpression forms a group. The group with the index = 0 contains the string that matches the entire regular expression (default setting).

Should return the string that corresponds to a expression of of part of, as a result, the desired group index is so specify.

Example: The following pattern is used to identify a date in American format.

```
pattern="\b(\d{1,2})/(\d{1,2})/(\d{2,4})\b"
```

The pattern includes 3 subexpressions. The month is recognized with the first subexpression. If one only wants to receive this information, the group index must be 1.

```
str=TxFormatEx("{en-US}The date in American format is {T1:d}",currentTime)
; -> The date in American format is 10/11/2016
result= TxRegexMatch(str,pattern,";",0); -> 10/11/2016
result= TxRegexMatch(str,pattern,";",1); -> 10 (result of the 1st subexpression)
```

Short overview: Regular expressions

"Regular Expressions" are patterns with a standardized syntax used in describing the structure of a string. The function uses the package implemented in the Microsoft .NET Framework for regular expressions. The composition and most important components of the pattern are presented below. For a complete list, see the Microsoft .NET- Framework documentation.

When searching for the pattern, the string is scanned character by character from left to right. Each new position is compared with the pattern. If the partial string matches the pattern from this position on, then this is counted as a success. But then the search continues onward from the next character after the find position.

The patterns can consist of normal characters, Escape-sequences, character classes, quantifiers, groupings and additional special characters such as alternatives.

The following characters are called **metacharacters**. They fulfill special functions in the search pattern. If a metacharacter is to be treated as a normal character and not as a metacharacter, that character must be preceded by a backslash.

Metacharacters	
.	Stands for an arbitrary character except the Newline
^	The search pattern must be located at the start of the text to be searched, or negates the pattern at the start of a character class.
$	Search pattern must be located at the end of the text to be searched.
|	Denotes an alternative in the search pattern
?	Denotes an amount of repetitions or: introduces a modifier or: denotes a look-ahead assertion
+	Denotes an amount of repetitions
*	Denotes an amount of repetitions
()	Serves the purpose of grouping
[]	Denotes a character class
{}	Denotes a quantifier
-	Character range in character classes
\	Reverses the special meaning of the metacharacters in order to search for them as plain characters

A **Character class** represents one or more characters. It is bracketed by the metacharacters [ ]. Within a character class, the following applies

\	for masking
^	for negating the character class, when it is the first character
-	for denoting a range

Character classes	
.	matches any character except Newline. The option 'Singleline' sets whether newlines are also included.
\d	arbitrary decimal digit; identical with [0-9]
\D	all characters except digits; identical with [^0-9]
\s	whitespace-characters (Space, Tabulator, Carriage Return, Line Feed)
\S	stands for a character which is not a whitespace-character; identical with [^\s]
\w	every alphanumerical character and the underline; identical with [a-zA-Z_0-9]
\W	all characters except alphanumerical characters; identical with [^\w]
[abc]	a,b or c
[a-z]	a through z
[a-z&&[^bc]]	a through z , without b and c

The **quantifiers** stand after simple characters or character classes and determine the number of characters. Quantifiers determine how often the character appearing before them appear.

Quantifier	
{n}	Specifies that the character appearing to its left must occur exactly n times in succession.
{n,}	Specifies that the character must appear at least n times.
{n,m}	Specifies that the character must occur at least n times but no more than m times.
*	Specifies that the character appearing to the left of it must occur any arbitrary amount of time, including zero, in succession.
*?	Specifies that the character appearing to the left of it must occur any arbitrary amount of time, including zero, in succession.

+	Specifies that the character appearing to the left of it must occur once, and may appear any arbitrary amount of times in succession. Identical with {1,}. Example: The regular expression \w+ recognizes a word.
?	Specifies that the character appearing to the left of it may occur either one time or zero times. Identical with {0,1}.

Ordinarily, quantifiers are greedy. They cause the regular expression engine to match as many occurrences of particular patterns as possible. Appending the ? character to a quantifier (e.g. *?) makes it lazy. It causes the regular expression engine to match as few occurrences as possible.

Regular expressions can contain both the normal characters and **Escape-sequences**. Escape-sequences begin with a Backslash. Next comes a character possessing special meaning and which is not be interpreted as a normal character.

Escape-sequences	
\\	Backslash
\b	Backspace
\e	Escape
\t	Horizontal tabulator
\r	Carriage Return
\v	Vertical tabulator
\f	Form Feed
\ n	New Line
\a	Alarm (Bell)
\xhh	specifies a character by means of the associated two-digit hexadecimal ASCII-character code
\uhhhh	specifies a character by means of the associated two-digit hexadecimal Unicode-character code
\cz	ASCII-control character, corresponding to z

With **anchors**, it is possible to limit the possible number of hits for a regular expression. An anchor determines at what location(s) in the search text the hit must occur, while giving special significance to the start and end of the search text.

Anchor	
^	Line start. The ^-anchor specifies that the subsequent pattern must begin at the first position of the string. Can be modified by the modifier /m.
$	Line end. The $-anchor specifies that the preceding pattern must be located at the end of the input string, or before at the end of the input string. Can be modified by the modifier /m.
\b	Word boundary
\B	Not at a word boundary
\A	Start of a string. The \A-anchor specifies that there must be a match at the start of the input string. This is identical to the ^-anchor, except \A ignores the option Multiline. For this reason, in a multi-line input string, the system can only search for a match at the end of the first line. Cannot be modified with any modifier
\Z	End of a string. The \Z-anchor specifies that a match must be located at the end of the input string or before \ n at the end of the input string. This is identical to the $-anchor, except \Z ignores the option Multiline. For this reason, in a multi-line input string, the system can only search for a match at the end of the last line or the last line before \ n. Cannot be modified with any modifier.
\z	The \z-anchor specifies that a match must be located at the end of the input string. In contrast to the $-language element, \z ignores the option Multiline. In contrast to the \Z-language element, \z does not correspond to a \ n -character at the end of a string. For this reason, it can only be used to search for a match with the last line of the input string. Cannot be modified with any modifier.
\G	The \G-anchor specifies that a match must be located at the location where the preceding match ends.

The anchors **^** and **\A** on the one hand, and respectively **$** and **\Z** ont he other hand, only differ when working in the multi-line mode, or when ·the search text contains more than one line.

To search for **alternative** strings, the character "|" is used. If the partial expression is not found before the | character, then the system search for the partial expression behind the | character.

Alternatives	
x|y	Alternative. x|y stands for x or y. The alternative normally refers to the complete left-side/right-side partial expression . Thus, abc|xyz means -> abc or xyz. Parentheses can be used to constrain alternatives. Thus, ab(x|y)cd means ab, followed by x or y , followed by cd.

Regular expressions enable **grouping** of partial expressions. They delimit partial expressions and record the partial strings of a input string. A group is bracketed by the metacharacters ( ).

Grouping

(x)	Matching partial expression x. This grouping has two meanings: On one hand, the partial pattern is grouped, which enables you to apply quantifiers to the partial pattern or refine it by means of the alternative | character. (\w\d){1,}, for instance, stands for a combination of a decimal digit and a word character, which may occur either once or any arbitrary amount of times. x(\d{3}|###) stands for "x followed by a three-digit number or followed by ###". On the other hand, the find location is written to an internal array, so that you are able to refer back to it in backward references.
(?:x)	Non-captured group, groups a partial pattern. However, the find location of the partial pattern x is not entered into the internal array. The partial pattern can not be used with backward references.
(?<name>x)	Captures a matching partial pattern x and enables access via a name or number. In this context, the name is a group name and the partial pattern x is an arbitrary pattern of a regular expression. With \k<name>, it is possible to access the partial expression captured. Alternatively, it is possible to access it using _number,_ where _"number_" is the ordinal number of the captured partial expression x.
x (?=y)	Positive lookahead-assertion. The effect of this pattern is that the partial pattern x only achieves success when directly followed by the partial pattern y. Thus, you can located partial strings, to which another string is directly appended. The pattern in the brackets is not contained in the search results. The pattern \d{3}(?=x), for instance, finds all three-digit numbers which are followed by an x. In the string "123a 456x 789x", for example, the partial strings "456" and "789" would be found, but not "123".
x(?!y)	Negative lookahead-assertion. The effect of the pattern is that only such find locations are included in the results which match the pattern x and which are not followed by a string matching the pattern y. The pattern in the brackets is not contained int he search results.
(?<=x) y	Positive Lookbefore-assertion. The pattern y after the brackets is only recognized if the pattern x in the brackets preceding it is found first. The pattern in the brackets is not contained in the search results. With the pattern (?<=\b20)\d{2}\b, it is possible to find the last two digits of the year in the 21st century. With an input string "2010 1999 1861 2140 2009", the results would be 10 and 09.
(?<!x) y	Negative Lookbefore-assertion. The pattern y after the brackets is only recognized if the patten x in the preceding brackets was not found. The pattern in the brackets is not contained in the search results. The pattern (?<!halt)bar finds the "bar" only if there is no "halt" preceding it.
(?>x)	The language construct (?>x) deactivates the backtracking. The module for regular expressions matches as many characters in the input string as possible. If no further matching is possible, there is no backtracking, in order to attempt alternative pattern matching. (I.e., the partial expression only finds matches for strings which would match the partial expression alone. The system does not try to find any match for a string on the basis of the partial expression and arbitrary subsequent partial expressions.)
(?(x)y|z)	Conditional expression If the expression x is found, the expression y is used. If the expression x was not found, the expression z is used.

Patterns found grouped by means of () can be used either in the search itself or again later in the replacement action, as a **backward reference**. Up to a maximum of 9 backward references can be used.

Backward reference	
number	With _number_, it is possible to access the captured partial expression, where _number_ denotes the partial expression's ordinal number. (\w)(\w) The system searches for 2 alphanumeric characters. For instance, with \2 the second partial expression can be re-used.
\k<_name_>	If the partial expression has a name (?<_name_>partial pattern), then by using \k<_name_> it is possible to access the captured partial expression
	With (?:x), it is possible to prevent the formation of a backward reference, which conserves backward references and memory and increases the system's speed.

The behavior of a regular expression can be governed by **modifiers**. Some modifiers can also be replaced with options (e.g. the option IgnoreCase corresponds to the modifier i, except that the option affects the entire expression, while the modifier can be applied to partial expressions. A modifier is stated in the form (?i). Instead of the modifier i, the modifiers m, s, and x are also allowed.

Modifier	
(?i)	The modifier i is activated, which means that matching is case-insensitive.
(?-i)	Modifier i is deactivated
(?i-msx)	Modifier i is deactivated
(?m)	If m is active, the string is considered to be a multi-line string. "^" and "$" find the start and end of each internal line. If m is deactivated, then "^" and "$" find the beginning and end of the entire string.
(?n)	If n is active, the normal parentheses(...), which normally bracket text, become non-capturing brackets. They thus function like (?:...) and only serve to group elements together.
(?s)	If s is active, the string is considered to have only one line. The period (.) finds all characters, including line breaks.
?(x)	If x is active, comments and whitespaces (e.g. space characters) within the pattern are ignored.

Examples:

In a text array containing channel names, the system is to find all channel names beginning with "sintest", followed by an arbitrary number and ending with the extension "dat" or "raw".

The search pattern (?i)^sintest\d*\.(dat|raw)$ signifies:

(?i)	Modifier: Case insensitive matching. Instead of using this modifier, it is also possible to work with the option IgnoreCase.
^sintest	The entered text must start with "sintest".
\d*	Next, any arbitrary amount of digits, or none, can follow.
\.	The period character must be found.
(dat|raw)$	The extension may be either "dat" or "raw".

```
txArray= TxArrayCreate(5)
txArray[1]="sin3.dat"
txArray[2]="SINTEST1.DAT"
txArray[3]="SINTEST2.DAT"
txArray[4]="SINTEST3.RAW"
txArray[5]="SPANNUNG.DAT"
separator=";"
pattern = "(?i)^sintest\d*\.(dat|raw)$"
erg= TxRegexMatch(txArray,pattern,separator,0)
```

The text array "res" contains the following strings:

```
[1]
[2] SINTEST1.DAT
[3] SINTEST2.DAT
[4] SINTEST3.RAW
[5]
```

From a text array containing arbitrary texts, the floating point numbers are to be extracted from each element.

The search pattern [-+]?([0-9]+\.?[0-9]*|\.[0-9]+)([eE][-+]?[0-9]+)? signifies:

[+-]?	A plus/minus sign may or may not appear one time.
(	Start of Group 1 Start of the 1st part of the alternative in Group 1
[0-9]+	A decimal digit must occur at least once and then arbitrarily often in succession.
\.?	The decimal period may or may not appear one time.
[0-9]*	A decimal digit may appear any number of times including zero.
	End of the 1st part of the alternative
|	Or
	Beginning of the alternative's 2nd part
\.	The decimal period must appear.
[0-9]+	A decimal digit must occur at least once and then arbitrarily often in succession.
	End of the alternative's 2nd part
)	End of Group 1
(	Beginning of Group 2
[eE]	The exponential character must appear
[-+]?	A plus/minus sign may or may not appear one time.
[0-9]+	A decimal digit must occur at least once and then arbitrarily often in succession.
)	End of Group 2
?	Group 2 may or may not appear one time.

```
txArray= TxArrayCreate(3)
txArray[1]="The minimum is at -123.8 and the maximum at +1234."
txArray[2]=At 1.9e3°C, the temperature has exceeded the cutoff limit.
txArray[3]="This line contains no number."
pattern ="[-+]?([0-9]+\.?[0-9]*|\.[0-9]+)([eE][-+]?[0-9]+)?"
erg= TxRegexMatch(txArray,pattern,";",0)
```

The text array "res" contains the following strings:

```
[1] -123.8;+1234.
[2] 1.9e3
[3]
```

By means of the call below, the specified longitude and latitude values are used to return the associated address (reverse geocoding).

For this purpose, the search machine of OpenStreetMap is used. The parameter 'format' specifies the desired format for the answer (here XML; an alternative would be JSON); the parameters 'lat' (latitude) and 'lon' (longitude) specify the latitude and longitude.

```
answer = HttpGetText( "http://nominatim.openstreetmap.org/reverse?format=xml&lat=52.541861&lon=13.3869693")
```

The text returned includes the following section:

```
<road>Voltastraße</road>
<suburb>Gesundbrunnen</suburb>
<city_district>Mitte</city_district>
<state>Berlin</state>
<postcode>13355</postcode>
```

In order to extract individual fields from such answers, the function TxRegexMatch() can be recommended:

```
NameOfTheRoad = TxRegexMatch( answer, "<road>(.*?)</road>", ",", 0, 1)
; NameOfTheRoad now has the content 'Voltastraße'.
```

If the response is returned in JSON-format instead:

```
answer = HttpGetText( "http://nominatim.openstreetmap.org/reverse?format=json&lat=52.541861&lon=13.3869693")
;answer contains (besides other items): ...{... "road":"Voltastraße","suburb":"Gesundbrunnen", ...}
NameOfTheRoad = TxRegexMatch( answer, "~034road~034:~034(.*?)~034", ",", 0, 1)
```

See also:

TxFind, TxReplace, TxRegexReplace

TxRegexReplace

Replaces texts in a text array or in a text by means of a regular expression.

Declaration:

```
TxRegexReplace ( TextOrTxArray, TxPattern, TxReplace, Options ) -> TextOrTxArrayResult
```

Parameter:

TextOrTxArray	Text or textarray in which the pattern is replaced
TxPattern	Pattern
TxReplace	Text to be replaced
Options	Options for replacing. The individual values may be added.
	0 : No option
	1 : IgnoreCase
	2 : Multiline
	4 : ExplicitCapture
	16 : Singleline
	32 : IgnorePatternWhitespace
TextOrTxArrayResult	
TextOrTxArrayResult	Contains the replaced texts

Description:

The meanings of the options:

- IgnoreCase: The matching is case insensitive
- Multiline: Changes the interpretation of ^ and $ to match the beginning and end of any line and not the beginning and end of the entire input string. Circumflex and Dollar-sign match line break.
- ExplicitCapture: With this option, the familiar parentheses (...) which normally bracket text become non-capturing brackets. Thus, they have the same effect as (?:...) and only serve the purpose of grouping. With named bracketed expressions (?<Name>...), it is still possible to capture text fragments as before.
- SingleLine: This option modifies the behavior of the period character (.) . In this case the period matches every character; normally the Newline is exempted. With this option, line breaks are also recognized with the period character.
- IgnorePatternWhitespace: When this option is selected, all white space in the expression pattern which is either unescaped or within character classes is ignored. This allows expressions to be formed in a clearer manner. Furthermore, all characters (including the next line break) from outside of a character class which are located between two unescaped #-characters are ignored. In this way, it is possible to insert comments into the regular expression.
- Invalid options cause the sequence to be cancelled.
- An empty search pattern causes the sequence to be cancelled.

A **short overview of syntax of regulare expressions** is presentedd in the online help for the function TxRegexMatch()

The parameter **TxReplace**specifies the string that is to replace each match in input. It can consist of any combination of literal text and **substitutions**.

Substitution	**Description**
$number	Includes the last substring matched by the capturing group that is identified by number, where number is a decimal value, in the replacement string.
${name}	Includes the last substring matched by the named group that is designated by (?<name>) in the replacement string.
$$	Includes a single '$' literal in the replacement string
$&	ncludes a copy of the entire match in the replacement string.
$`	Includes all the text of the input string before the match in the replacement string.
$'	Includes all the text of the input string after the match in the replacement string.
$+	Includes the last group captured in the replacement string.
$_	Includes the entire input string in the replacement string.

Examples:

Replaces all whitespace characters in a text with an underscore:

```
TxPatched = TxRegexReplace( "Text with spaces", "\s", "_", 0)
```

TxPatched has the content "Text_with_spaces"

Removes all round brackets and whitespace characters from a text:

```
TxPatched = TxRegexReplace( "(0190) 1234567", "[\s()]", "", 0)
```

TxPatched has the content "01901234567"

Removes all leading zeroes from a text.

```
TxPatched = TxRegexReplace( "0001023", "^0*", "", 0)
```

TxPatched has the content "1023"

Conversion of the current date from American format to German format. To do this, in the first code line, the system gets the current system time. In the second line, the time is formatted as date in the American format.

The search pattern \b(?<month>\d{1,2})/(?<day>\d{1,2})/(?<year>\d{2,4})\b means:

\b	Pattern comparison begins at a word boundary
(?<month>\d{1,2})	One or two decimal places are cached in the group named 'month'.
/	A slash is obligatory.
(?<day>\d{1,2})	One or two decimal places are cached in the group named 'day'.
/	A slash is obligatory.
(?<year>\d{2,4})	One or two decimal places are cached in the group named 'year'.
\b	Pattern comparison ends at a word boundary

In the text to be replaced, the specified groups are accessed by means of the $ character. A period is inserted between the groups.

```
currentTime=TimeSystem?()
strDate=TxFormatEx("{en-US}{T1:d}",currentTime); -> 6/30/2016
pattern="\b(?<month>\d{1,2})/(?<day>\d{1,2})/(?<year>\d{2,4})\b"
replace="${day}.${month}.${year}"
germanDate=TxRegexReplace( strDate,pattern,replace,0); -> 30.06.2016
```

With the following expression, a text is to be converted to a valid imc FAMOS variable name. All invalid characters are to be replaced with an underline "_". Invalid characters include all characters with an ASCII-code < 32. Also < >+ - ( ) * / ^ = { } [ ] | . @ , ; ? ? as well as spaces. If the 1st character is a digit, then an underline is prefixed to it.

The search pattern (^\d)|[^\w!$%&?~#] means:

(^\d)	If the first character is a digit, it is cached in a group.
[^w!$%&?~#]	The content of the character class finds all characters which do not match 0?9, a-z,_, A-Z nor the characters !$%&?~#.

In the replacement text, _$1 is entered, i.e. each character found by the search pattern is replaced with an underline plus the content of the first group $1. The first group is only chosen if the first character is a digit.

```
pattern ="(^\d)|[^\w!$%&?~#]"
varname=TxRegexReplace( "Valid Name?",pattern,"_$1",0);-> Valid_Name?
varname=TxRegexReplace( "1Valid Name+",pattern,"_$1",0);-> _1Valid_Name_
```

Remark: The function TxGetValidVarName() is more suitable for the present task.

See also:

TxFind, TxReplace, TxRegexMatch

TxReplace

The function searches for a text excerpt in a text or text array, and replaces it with different text.

Declaration:

```
TxReplace ( TextOrTxArray, TxFindText, TxReplaceText, Occurrence, Options ) -> TextOrTxArrayResult
```

Parameter:

TextOrTxArray	Text or text array in which to find and replace the text excerpt
TxFindText	Text to search for
TxReplaceText	Text to be replaced
Occurrence	The n-th occurrence of the text for which the system searches
	-1 : The last occurrence of the search text is replaced.
	0 : All occurrences of the search text are replaced.
	1... : The n-th occurrence of the search text is replaced.
Options	Options for Find & Replace
	0 : Case sensitive
	1 : Case insensitive
TextOrTxArrayResult	
TextOrTxArrayResult	Contains the replaced texts

Description:

The system searches for a text in a text or text array and replaces it with a different one.

If the first parameter is a text array, then the function's result is a text array. Its dimensions are the same as the one submitted.

If the first parameter is a text, then the function's result is a text.

In the resulting text or text array, the texts found are replaced with the substitute text.

The parameter Occurrence specifies at which occurrence of the search text to replace the text.

- Invalid options cause the sequence to be cancelled.
- An empty search text causes the sequence to be cancelled.

Examples:

Replaces all whitespace characters in a text with an underscore:

```
TxPatched = TxReplace( "Text with spaces", " ", "_", 0, 0)
```

TxPatched has the content "Text_with_spaces"

Remove all hyphens from a text:

```
TxPatched = TxReplace( "FF-1A-13-2E", "-", "", 0, 0)
```

TxPatched has the content "FF1A132E""

From one filename, a new filename for the calculated result is derived:

```
TxInputFileName = "c:\Measurements.raw\Test1.raw"
TxOutputFileName = TxReplace(TxInputFileName, ".raw", "_processed.dat", -1, 1)
```

TxOutputFileName has the content 'c:\Measurements.raw\Test1_processed.dat'

In a text box with prepared error messages, the placeholder "%1" is replaced with a variable name.

```
txArray=TxArrayCreate(4)
txArray[1]="%1 has the wrong data type"
txArray[2]="Not enough memory"
txArray[3]="Acceptable range of %1 is exceeded"
txArray[4]="Sample rate of to %1 ist too low, %1_result is imprecise"
txArrayResult=TxReplace( txArray,"%1","Channel1",0,0)
```

The text array "txArrayResult" contains the following strings:

```
txArray[1]="Channel1 has the wrong data type"
```

```
txArray[2]="Not enough memory"
txArray[3]="Acceptable range of Channel1 is exceeded"
txArray[4]="Sample rate of to Channel1 ist too low, Channel1_result is imprecise"
```

See also:

TxFind, TxRegexMatch, TxRegexReplace

TxSplit

Function for splitting a text

Declaration:

```
TxSplit ( TxText, TxSeparator ) -> TxArrayResult
```

Parameter:

TxText	String to be split
TxSeparator	String with the separator character
TxArrayResult	
TxArrayResult	Result text array

Description:

The text TxText is subdivided according to the separator strings in TxSeparator.

The split strings are returned in a text array.

- The text TxSeparator can contain multiple separator strings.
- The individual separator strings are to be separated by a semicolon ;.
- If the semicolon is to be used as a separator, then \; to specify.
- The elements of the resulting text arra to not contain separator characters.
- If the TxSeparator is empty, then the text is split at the white spaces (spaces, tabs, carriage return).
- This function is case sensitive.

Examples:

The text is separated at the semicolon, comma or period.

```
txArray1=TxSplit("This is a text. It is to be subdivided; in other words split." ,"\;;,;.;")
```

Results in txArray1:

```
[1] This is a text
[2]  It is to be subdivided
3]  in other words split.
```

.

The text is separated at the white spaces.

```
tab="~009"
cr="~010"
txArray2=TxSplit("This"+tab+"is the text"+cr+";It will be split." ,"")
```

Results in txArray2:

```
[1] This
[2] is
[3] the
[4] text
[5] ;It
[6] will
[7] be
[8] split.
```

See also:

TxFind, TxRegexMatch

TxToClipboard

This function stores the contents of a Text or TextArray on the Windows Clipboard.

Declaration:

`TxToClipboard ( TxOrTxArray ) -> SvSucess`

Parameter:

TxOrTxArray	Text/Textarray to store on the Clipboard.
SvSucess	
SvSucess	Success of the function: 1, if the function could be performed successfully; 0 on error. In case of error, the cause can be determined using the function GetLastError().

Description:

The content of the parameter is copied to the Windows Clipboard (ANSI-Text-Format).

With TextArrrays, each element constructs a new line.

Examples:

The complete content of a table in a panel is copied to the Windows Clipboard.

Columns are separated by a tabulator.

```
cols = PnTableColumns?("Table1")
rows = PnTableRows?("Table1")
tx = ""
FOR r = 1 TO rows
   FOR c = 1 TO cols
      tx = tx + PnTableGetCellText("Table1", c, r)
      IF c < cols
         tx = tx + "~009"  ; Tabulator
      END
   END
   tx= tx + "~013~010"   ; Carriage return/Line feed
END
TxToClipboard(tx)
```

See also:

TxWhere

The position of a string in a text is determined.

Declaration:

```
TxWhere ( TxText, TxSearch ) -> SvPosition
```

Parameter:

TxText	Text to be examined
TxSearch	String to be searched for. The search is case insensitive.
SvPosition	
SvPosition	Position of the string searched for in the text. The first possible position is 1. If the string is not found, 0 is returned.

Description:

This function searches for a specified string in a text. If the string is found, its position is returned.

- The function searches for the string from the beginning to the end of the text.
- If the string occurs more than once in this text, only the first occurrence is returned.
- The return value should always be checked for 0 to make sure that the string was actually found.

Examples:

```
Position = TxWhere("Das ist ein Text", "ist")
```

This code example searches for the word "text" and finds it at position 11.

See also:

TLike, TReplace, TComp, TPart

UNCERTAINTY_LOOP

Available in: Professional Edition and above

Loop for determining the measurement uncertainty results from an algorithm using the Monte-Carlo method (MCM).

Declaration:

```
UNCERTAINTY_LOOP SvTrialCount SwInit
```

Parameter:

SvTrialCount	Number of Monte-Carlo trials to be performed. The parameter is frequently in the range 100..10000.
SwInit	The random number generator for the function UncertaintyModify() is (re)initialized with this value. If the parameter is omitted, a fixed internal value is used.
	0 : This is selected when you wish to work with different random values in a new run of the entire loop.
	>0 : Integer value for the reinitialization of the random number generator. Ensures reproducible behavior of the entire loop.

Description

The UNCERTAINTY_LOOP is a loop which serves the purpose of determining the measurement uncertainty of the results of an algorithm by means of the Monte-Carlo method. It forms the framework for calls to the functions UncertaintyModify() and UncertaintyCalc(). Once the loop has been run through, the measurement uncertainty is set as the result as a user-defined property.

The loop is run (M+2) times.

In the first run, the algorithm is performed with uninfluenced input data. Subsequently, the algorithm is run M times in M Monte-Carlo trials on input data superimposed with noise (typically generated by the function UncertaintyModify). In this case, a noise-based result is calculated. From this result, the measurement uncertainty is determined by means of the function UncertaintyCalc. Finally, another run is performed with the uninfluenced input data for restoration purposes.

The loop defined with the command UNCERTAINTY_MODIFY is concluded by the command END.

Within an UNCERTAINTY_LOOP, no additional UNCERTAINTY_LOOPs can be started.

For further information on determining the measurement uncertainty in FAMOS, see the user's manual/online help in the section "Determining the measurement uncertainty". Here, the foundational standards and methods (GUM, Monte-Carlo method) are also explained in detail.

This command is not available in the FAMOS Standard Edition.

Examples:

From a voltage channel Input1 with a measurement range of 10V and measurement uncertainty of 0.1V, a result named "Result" is to be obtained by means of an algorithm (low-pass filter, maximum value). The algorithm without determining the measurement uncertainty:

```
_In1 = Input1
Result = Max(FiltLp(_In1, 0, 0, 1, 0.02))
```

To determine the measurement uncertainty for the result, the algorithm must be expanded as follows:

```
; stating the measurment uncertainty of the input data:
UncertaintySet(Input1, "Uncertainty", 0.1)
; run all Monte-Carlo trials, here the trial count M = 1000
UNCERTAINTY_LOOP 1000
   ; adding noise to the input data
   _In1 = UncertaintyModify(Input1)
   ; caling the actual algorithm and calculating the result
   Result = Max(FiltLp(_In1, 0, 0, 1, 0.02))
   ; calculating the measurement uncertainty of the result
   UncertaintyCalc(Result)
END
; querying the measurement uncertainty of the result
uc = UncertaintyGet(Result, "Uncertainty")
```

See also:

UncertaintyModify, UncertaintyCalc, UncertaintySet, UncertaintySnapshot

UncertaintyCalc

Available in: Professional Edition and above

Determining the measurement uncertainty by means of the Monte-Carlo method

Declaration:

```
UncertaintyCalc ( Variable [, SvCoverageProbability] [, Null] [, TxExtended1] [, TxExtended2] [, TxExtended3] [,
TxExtended4] )
```

Parameter:

Variable	The variable with the result of an algorithm. The measurement uncertainty of this variable is to be determined. The variable must be specified directly; temporary or indexed data are not permitted.
SvCoverageProbability	Confidence Level in percent. If 0, or not specified, the expanded measurement uncertainty is not determined. If non-zero, the coverage interval is determined, and from it the expanded measurement uncertainty. Typical values are 95 or 99. At the specified probability, the true value lies somewhere in the range given by the expanded measurement uncertainty around the uninfluenced result. (optional)
Null	Reserved; set to 0. (optional)
TxExtended1	1st expanded analysis result (optional)
	"uc" : The measurement uncertainty is determined for each measurement point of the result.
	"mean" : For each measurement point, the mean value of all Monte-Carlo trials is determined.
	"min/max" : For each measurement point of the result, the minimum and maximum of all Monte-Carlo trials is determined.
	"pdf0" : Distribution density of the deviations from the uninfluenced result, centered around an ostensible result of 0.0.
	"pdf" : Distribition density of the deviations from the uninfluenced result
TxExtended2	2nd expanded analyis result (optional)
	"uc" : The measurement uncertainty is determined for each measurement point of the result.
	"mean" : For each measurement point, the mean value of all Monte-Carlo trials is determined.
	"min/max" : Envelope: for each measurement point of the result, the minimum and maximum of all Monte-Carlo trials is determined.
	"pdf0" : Distribution density of the deviations from the uninfluenced result, centered around an ostensible result of 0.0.
	"pdf" : Distribution density (only for single values) of the deviations from the uninfluenced result
TxExtended3	3rd expanded analysis result (optional)
	"uc" : The measurement uncertainty is determined for each measurement point of the result.
	"mean" : For each measurement point, the mean value of all Monte-Carlo trials is determined.
	"min/max" : For each measurement point of the result, the minimum and maximum of all Monte-Carlo trials is determined.
	"pdf0" : Distribution density of the deviations from the uninfluenced result, centered around an ostensible result of 0.0.
	"pdf" : Distribition density of the deviations from the uninfluenced result
TxExtended4	4th expanded analysis result (optional)
	"uc" : The measurement uncertainty is determined for each measurement point of the result.
	"mean" : For each measurement point, the mean value of all Monte-Carlo trials is determined.
	"min/max" : For each measurement point of the result, the minimum and maximum of all Monte-Carlo trials is determined.
	"pdf0" : Distribution density of the deviations from the uninfluenced result, centered around an ostensible result of 0.0.
	"pdf" : Distribition density of the deviations from the uninfluenced result

Description:

This procedure is typically called for the result variable of an algorithm toward the end of an UNCERTAINTY_LOOP. The result of a Monte-Carlo trial is provided to it, in order to improve the estimate of the measurement uncertainty.

Measurement uncertainty and expanded measurement uncertainty (if determined) are appended to the result variable passed as user-defined properties. They can be queried using the function UncertaintyGet().

With the optional parameters 4 through 7, it is possible to request additional expanded analysis results. With the expanded analysis, all previously performed Monte-Carlo trials are taken into account. The order of the additionally specified analyses does not matter; "pdf" and "pdf0" can not be used at the same time. The results of the expanded analysis are written in a group, which is only created if at least one expanded analysis is desired. The group variable is created, initialized and updated as a side effect of this procedure.

The name of the group variables created is formed from the name of the result variable by appending "_uc_result", and if the associated option is selected, contains the following channels:

original:	The uninfluenced result of the first run of the UNCERTAINTY_LOOP (always included)
uc:	Measurement uncertainty in accordance with the option "uc"
min:	Lower envelope curve in accordance with the option "min/max"
max:	Upper envelope curve in accordance with the option "min/max"
mean:	Mean value in accordance with the option "mean"
pdf0:	Zero-based distribution density of the deviations in accordance with the option "pdf0"
pdf:	Distribution density for single values in accordance with the option "pdf"

Example: The call

```
UncertaintyCalc(x, 99, 0, "uc", "pdf0", "min/max")
```

creates the group "x_uc_result" with the following structure:

```
x_uc_result
   |_ original
   |_ uc
   |_ min
   |_ max
   |_ pdf0
```

If the result variable itself is a channel belonging to a group, the name of the group with the expanded results is formed according to the pattern "[GroupName]_[ChannelName]_uc_result". Any existing measurement association is applied.

The procedure can only be used within the UNCERTAINTY_LOOP. Upon the first call, it initializes the group and user-defined properties. In the last run, the results are finalized if appropriate.

The procedure may be called exactly one time per run of the loop for each result. The exception is the very first run, in which the procedure call, along with its parameters, is activated.

Calculation of the measurement uncertainty is supported for the following data types:

- Equidistant data
- y-component of xy-data
- Magnitude in complex data

If there are other data types, it is necessary to first insert an assignment to an auxiliary variable for naming, for example, A_P = A.P and next UncertaintyCalc(A_P).

After elapse of the UNCERTAINTY_LOOP, the user deletes the group of expanded analysis, if needed.

Information on determining the measurement uncertainty in FAMOS is available in the user's manual/online-help, in the section "Determining the measurement uncertainty". Here, the foundational standards and methods (GUM, Monte-Carlo method) are explained in more detail.

This function is not available in the FAMOS Standard Edition.

Examples:

From a voltage channel Input1 having a 10V measurement range and measurement uncertainty of 0.1V, a result named "Result" is to be determined by means of an algorithmus (low-pass filter, maximum value). Additionally, the distribution density is to be determined.

```
; stating the measurment uncertainty of the input data:
UncertaintySet(Input1, "Uncertainty", 0.1)
; run all Monte-Carlo trials, here the trial count M = 1000
UNCERTAINTY_LOOP 1000
   ; adding noise to the input data
   _In1 = UncertaintyModify(Input1)
   ; caling the actual algorithm and calculating the result
   Result = Max(FiltLp(_In1, 0, 0, 1, 0.02))
   ; calculating the result's measurement uncertainty and the expanded measurement uncertainty
   UncertaintyCalc(Result, 95 , 0 , "pdf")
   ; also creates the group "Result_uc_result" with the channels "original" and "pdf"!
END
; querying the result's measurement uncertainties
uc = UncertaintyGet(Result, "Uncertainty")
U = UncertaintyGet(Result, "Expanded uncertainty")
```

See also:

UNCERTAINTY_LOOP, UncertaintyModify, UncertaintySet, UncertaintyGet, UncertaintySnapshot

UncertaintyGet

Available in: Professional Edition and above

Queries a user-defined property on the topic measurement uncertainty.

Declaration:

```
UncertaintyGet ( Variable, TxPropName ) -> SvValue
```

Parameter:

Variable	Variable to be queried
TxPropName	Name of the property
	"Uncertainty" : Standard measurement uncertainty. Expressed in the y-unit of the data set. It is calculated by the function UncertaintyCalc(). The function UncertaintyModify() adds an influencing variable of normal distribution.
	"Expanded uncertainty" : Expanded measurement uncertainty. Stated in the data set's y-unit. Calculated by the function UncertaintyCalc(). The function UncertaintyModify() adds a normalmally distributed influence variable, if the coverage probability or expansion factor is provided.
	"Coverage probability" : The coverage probability, stated in percent. E.g. 95 or 99.7. This is the probability that the true value lies within the coverage (confidence) interval. Is set by UncertaintyCalc(). The function UncertaintyModify() takes the value into account in conjunction with the expanded measurement uncertainty.
	"Coverage factor" : Ratio of expanded measurement uncertainty to standard measurement uncertainty, typically in the range 2 to 3. The function UncertaintyModify() takes the value into account in conjunction with the expanded measurement uncertainty.
	"Uncertainty Source.Rectangular" : UncertaintyModify() adds a symmetrical uniform distribution. Half of the distributions width is specified.
	"Uncertainty Source.LSBs" : Number of LSBs determining the bit noise. Only for integers. UncertaintyModify() add bit noise of the standard deviation specified in terms of LSBs.
	"Uncertainty Source.Amplitude" : Gain deviation around the value 1.0. UncertaintyModify() adds a random gain deviation which is however fixed for the measurement. E.g. 0.1, if the gain is to vary by 10%; to be interpreted as standard deviation.
	"Uncertainty Source.Offset" : Constant offset deviation in physical units of the channel. UncertaintyModify() adds a random offset deviation which however is fixed for the measurement. The standard deviation is given.
	"Uncertainty Source.Drift Offset" : Drift fluctuating over time. The height of the drift-offset error is stated in terms of standard deviation (in physical units of the channel). UncertaintyModify() adds an offset drift.
	"Uncertainty Source.Drift Time" : Drift fluctuating over time. If UncertaintyModify() adds an offset drift, this sets a time dimension for the change.
	"Uncertainty Source.Gain Drift" : Gain deviation drifting over time stated as a factor around the value 1.0. E.g. 0.01, if the gain is to vary by 1%, to be interpreted as the standard deviation. UncertaintyModify() adds this drift.
	"Uncertainty Source.Gain Drift Time" : If UncertaintyModify() adds a gain deviation drifting over time, this specifies a time dimension for the change.
	"Uncertainty Source.Hum Amplitude" : UncertaintyModify() adds a mains hum signal. The amplitude is selected at random and is fixed for a measurement. The stated value is the standard deviation of the selected amplitudes.
	"Uncertainty Source.Hum Frequency" : If UncertaintyModify() adds a mains hum signal, this sets the fixed frequency.
	"Uncertainty Source.Hum Harmonics" : If UncertaintyModify() adds a mains hum signal, this specifies the ration of the upper harmonics' power to the fundamental oscillation, e.g. 0.01, if the ratio is 1%.
	"Uncertainty Source.Spikes Max" : Spikes; UncertaintyModify() adds interference signal pulses with the specified maximum value (in y-units of the data set).
	"Uncertainty Source.Spikes Min" : Spikes; UncertaintyModify() adds interference signal pulses with the specified minimum value.
	"Uncertainty Source.Spikes Width" : If UncertaintyModify() adds interference signal pulses, then this sets the maximum width of the pulses.
	"Uncertainty Source.Spikes Time" : If UncertaintyModify() adds interference signal pulses, then this specifies after what amount of time at the latest a new pulse appears.
	"Uncertainty Source.Noise RMS" : UncertaintyModify() adds a noise signal, which primarily contains high frequency components. The standard deviation is given.
	"Uncertainty Source.Noise Frequency" : If UncertaintyModify() adds a noise signal, then this specifies the lower cutoff frequency.

	"Uncertainty Source.Temp Off" : Offset increasing in proportion to the temperature, stated as standard deviation per Kelvin (unit of the temperature channel influenced). UncertaintyModify() adds such an offset deviation with a random proprtionality factor which is however fixed for the particular measurement, if a temperature channel is specified. E.g. 0.01, if the gain deviation is 1% per Kelvin; to be interpreted as the standard deviation. The added deviation is proportional to the measurement value and to the temperature channel's deviation from the reference temperature. The reference temperature must be specified.
	"Uncertainty Source.Temp Gain" : Gain deviation increasing in proportion to the temperature, per Kelvin (unit of the temperature channel influenced) from the value 1.0. UncertaintyModify() adds such a gain deviation with a random proprtionality factor which is however fixed for the particular measurement. E.g. 0.01, if the gain deviation is 1% per Kelvin; to be interpreted as the standard deviation. The added deviation is proportional to the measurement value and to the temperature channel's deviation from the reference temperature. The reference temperature must be specified.
	"Uncertainty Source.Temp Ref" : The reference temperature used in UncertaintyModify() for the temperature-dependent influences, in the unit of the temperature channel influenced. If not specified, 20 is assumed. If the temperature channel is stated in °C and the specified value is 23, then this is interpreted as 23&deg;C.
SvValue	
SvValue	Value of the property (single value)

Description:

The properties defined here directly reflect characteristic parameters regarding the topic measurement uncertainty, which are defined either by the user of by the measurement system, and are calculated for output quantities by the function UncertaintyCalc - or they describe influencing quantities which cause measurement uncertainty ("Uncertainty Source.*") and are taken into account by the function UncertaintyModify().

There is an additional characteristic parameter concerning the topic measurement uncertainty, which is not listed here: "Uncertainty evaluation". This contains the method by which the measurement uncertainty is determined, is set by the function UncertaintyCalc() and is of the data type [Text], e.g. "MCM, M=100". To query this property, use the function UserPropText?(.., "Uncertainty evaluation").

Further information on determining the measurement uncertainty in FAMOS is presented in the user's manual/online help, in the section "Determining the measurement uncertainty". There, the characteristic parameters and fundamental standards and methods (GUM, Monte-Carlo method) are described in greater detail.

This function is not available in the FAMOS Standard Edition.

Examples:

From a voltage channel Input1 with a 10V measurement range and measurement uncertainty of 0.1V, a result named "Result" is to be determined by means of an algorithm (low-pass filter, maximum value).

```
; stating the measurment uncertainty of the input data:
UncertaintySet(Input1, "Uncertainty", 0.1)
; run all Monte-Carlo trials, here the trial count M = 1000
UNCERTAINTY_LOOP 1000
   ; adding noise to the input data
   _In1 = UncertaintyModify(Input1)
   ; caling the actual algorithm and calculating the result
   Result = Max(FiltLp(_In1, 0, 0, 1, 0.02))
   ; calculating the result's measusrement uncertainty and the expanded measurement uncertainty
   UncertaintyCalc(Result, 95)
END
; querying the result's measurement uncertainties
uc = UncertaintyGet(Result, "Uncertainty")
U = UncertaintyGet(Result, "Expanded uncertainty")
```

Definition of an offset drift:

```
UncertaintySet(Input1, "Uncertainty Source.Drift Offset", 0.01) ; [mV]
UncertaintySet(Input1, "Uncertainty Source.Drift Time", 1000) ; [s]
UNCERTAINTY_LOOP 1000
...
```

See also:

UNCERTAINTY_LOOP, UncertaintySet, UncertaintyModify, UncertaintyCalc, UncertaintySnapshot

UncertaintyModify

Available in: Professional Edition and above

Adds noise to a signal in the framework of determining the measurement uncertainty according to the Monte-Carlo method.

Declaration:

```
UncertaintyModify ( Variable [, Temperature] ) -> Result
```

Parameter:

Variable	Signal to which to add noise. The variable must be specified directly; temporary or indexed data are not permitted.
Temperature	Optional temperature channel, if temperature-dependent deviations are to be taken into account. (optional)
Result	
Result	Modified signal

Description:

A signal is distorted with noise. If the input channel has user-defined properties from which a signal interference can be calculated, this calculation is performed. Such properties include "Uncertainty" or "Uncertainty Source.Offset" and are defined using the function UncertaintySet().

If the input signal possesses the property "Uncertainty", then a normally distributed distorting influence is added.

The signal dsitorted with the corersponding interference is returned. The function adds noise and other interference.

This function can only be used within the UNCERTAINTY_LOOP. On the first and last runs, the function returns the unchanged input signal without inspection. In the loop runs in the interim, the input signal is distorted with a random interference signal, in accordance with the Monte-Carlo method.

The random number generator is initialized by the loop frame UNCERTAINTY_LOOP and runs independently of the generator which the random() function uses.

The function may also be called for signals which (currently) have no appropriate user-defined properties. The function then only returns a copy.

The function only affects the y-coordinate of equidistant and XY-data, as well as the magnitude of complex data. If there are other data types, then splitting into individual partial variable is necessary. If for instance the phase of a complex data set is to be distorted with noise, a separate variable with the phase must be introduced before the beginning of the UNCERTAINTY_LOOP, e.g. A_P = A.P. With UncertaintySet(), it is provided with appropriate specifications.

The optional temperature channel must have either the same sampling interval, or one which is an integer multiple of, the sampling interval of the signal to be distorted with noise. If the temperature channel extends further in time, it is truncated accordingly. If its time range is smaller, it is lengthened by repeating its last value the corresponding number of times.

Information on determining the measurement uncertainty in FAMOS is available in the user's manual/online-help, in the section "Determining the measurement uncertainty". Here, the foundational standards and methods (GUM, Monte-Carlo method) are explained in more detail.

This function is not available in the FAMOS Standard Edition.

Examples:

From a voltage channel Input1 having a 10V measurement range and measurement uncertainty of 0.1V, a result named "Result" is to be determined by means of an algorithmus (low-pass filter, maximum value). Additionally, the distribution density is to be determined.

```
; stating the measurment uncertainty of the input data:
UncertaintySet(Input1, "Uncertainty", 0.1)
; run all Monte-Carlo trials, here the trial count M = 1000
UNCERTAINTY_LOOP 1000
   ; adding noise to the input data
   _In1 = UncertaintyModify(Input1)
   ; caling the actual algorithm and calculating the result
   Result = Max(FiltLp(_In1, 0, 0, 1, 0.02))
   ; calculating the result's measusrement uncertainty and the expanded measurement uncertainty
   UncertaintyCalc(Result, 95 , 0 , "pdf")
   ; also creates the group "Result_uc_result" with the channels "original" and "pdf"!
END
; querying the result's measurement uncertainties
uc = UncertaintyGet(Result, "Uncertainty")
U = UncertaintyGet(Result, "Expanded uncertainty")
```

From a voltage channel U1 having a measurement range of 10V and measurement uncertainty of 0.1V, and a current chanel I1 (2A, 0.01A), the power is to be determined. The voltage channel is additionally affected by mains hum.

```
UncertaintySet(U1, "Uncertainty", 0.1)
UncertaintySet(U1, "Uncertainty Source.Hum Amplitude", 0.01)
UncertaintySet(I1, "Uncertainty", 0.01)
UNCERTAINTY_LOOP 1000
```

```
   U1_mod = UncertaintyModify(U1)
   I1_mod = UncertaintyModify(I1)
   P = Mean(U1_mod * I1_mod)
   UncertaintyCalc(P)
END
uc = UncertaintyGet(W, "Uncertainty")
```

Given: A voltage channel 'Channel' scaled in V and a temperature channel 'Temperature' in &deg;C. The sensor's spec sheet states a temperature-dependent zero point offset of 0.001V/K and a temperature-dependent gain offset of 50ppm/K, based on a reference value of 23&deg;C.

```
UncertaintySet(Channel, "Uncertainty Source.Temp Off", 0.001)
UncertaintySet(Channel, "Uncertainty Source.Temp Gain", 0.000050)
UncertaintySet(Channel, "Uncertainty Source.Temp Ref", 23)
UNCERTAINTY_LOOP 100
   _Channel = UncertaintyModify(Channel, Temperature)
   ...
```

See also:

UNCERTAINTY_LOOP, UncertaintyCalc, UncertaintySet, UncertaintyGet, UncertaintySnapshot

UncertaintySet

Available in: Professional Edition and above

Sets a user-defined property regarding the topic of measurement uncertainty.

Declaration:

```
UncertaintySet ( Variable, TxPropName, SvValue )
```

Parameter:

Variable	Variable to be changed
TxPropName	Name of the property to be changed
	"Uncertainty" : Standard measurement uncertainty. Stated in the data set's y-unit. Calculated by the function UncertaintyCalc(). The function UncertaintyModify() adds a normally distributed influencing variable, meaning a white noise with standard deviation/RMS-value equal to the measurement uncertainty.
	"Expanded uncertainty" : Expanded measurement uncertainty. Stated in the data set's y-units. Calculated by the function UncertaintyCalc(). The function UncertaintyModify() adds a normally distributed influencing variable (noise), if the coverage probability or expansion factor is additionally specified.
	"Uncertainty evaluation" : The method according to which the measurement uncertainty/expanded measurement uncertainty is determined. E.g. "MCM, M=100". It is set by UncertaintyCalc(). (Data type: Text)
	"Coverage probability" : The coverage probability, stated in percent. E.g. 95 or 99.7. This is the probability that the true value lies within the coverage (confidence) interval. Is set by UncertaintyCalc(). The function UncertaintyModify() takes the value into account in conjunction with the expanded measurement uncertainty.
	"Coverage factor" : Ratio of expanded measurement uncertainty to standard measurement uncertainty, typically in the range 2 to 3. The function UncertaintyModify() takes the value into account in conjunction with the expanded measurement uncertainty.
	"Uncertainty Source.Rectangular" : UncertaintyModify() adds a symmetrical uniform distribution. Half of the distributions width is specified.
	"Uncertainty Source.LSBs" : Number of LSBs determining the bit noise. Only for integers. UncertaintyModify() add bit noise of the standard deviation specified in terms of LSBs.
	"Uncertainty Source.Amplitude" : Gain deviation around the value 1.0. UncertaintyModify() adds a random gain deviation which is however fixed for the measurement. E.g. 0.1, if the gain is to vary by 10%; to be interpreted as standard deviation.
	"Uncertainty Source.Offset" : Constant offset deviation in physical units of the channel. UncertaintyModify() adds a random offset deviation which however is fixed for the measurement. The standard deviation is given.
	"Uncertainty Source.Drift Offset" : Drift fluctuating over time. Hight of the drift-offset deviation, states as the standard deviation (in the channel's physical units). UncertaintyModify() adds an offset drift.
	"Uncertainty Source.Drift Time" : Drift fluctuating over time. If UncertaintyModify() adds an offset drift, this sets a time dimension for the change.
	"Uncertainty Source.Gain Drift" : Gain deviation drifting over time stated as a factor around the value 1.0. E.g. 0.01, if the gain is to vary by 1%, to be interpreted as the standard deviation. UncertaintyModify() adds this drift.
	"Uncertainty Source.Gain Drift Time" : If UncertaintyModify() adds a gain deviation drifting over time, this specifies a time dimension for the change.
	"Uncertainty Source.Hum Amplitude" : UncertaintyModify() adds a mains hum signal. The amplitude is selected at random and is fixed for a measurement. The stated value is the standard deviation of the selected amplitudes.
	"Uncertainty Source.Hum Frequency" : If UncertaintyModify() adds a mains hum signal, this sets the fixed frequency.
	"Uncertainty Source.Hum Harmonics" : If UncertaintyModify() adds a mains hum signal, this specifies the ration of the upper harmonics' power to the fundamental oscillation, e.g. 0.01, if the ratio is 1%.
	"Uncertainty Source.Spikes Max" : Spikes; UncertaintyModify() adds interference signal pulses with the specified maximum value (in y-units of the data set).
	"Uncertainty Source.Spikes Min" : Spikes; UncertaintyModify() adds interference signal pulses with the specified minimum value.
	"Uncertainty Source.Spikes Width" : If UncertaintyModify() adds interference signal pulses, then this sets the maximum width of the pulses.
	"Uncertainty Source.Spikes Time" : If UncertaintyModify() adds interference signal pulses, then this specifies after what amount of time at the latest a new pulse appears.
	"Uncertainty Source.Noise RMS" : UncertaintyModify() adds noise primarily containing higher frequency components. The standard deviation is specified. If regular (white) noise is desired, the property "Uncertainty" must be used.

	"Uncertainty Source.Noise Frequency" : If UncertaintyModify() adds a noise signal, this states the lower cutoff frequency (greater than 0). If this property is not specified, a value of 20% of the sampling frequency is assumed.
	"Uncertainty Source.Temp Off" : Offset increasing in proportion to the temperature, stated as standard deviation per Kelvin (unit of the temperature channel influenced). UncertaintyModify() adds such an offset deviation with a random proprtionality factor which is however fixed for the particular measurement, if a temperature channel is specified. E.g. 0.01, if the gain deviation is 1% per Kelvin; to be interpreted as the standard deviation. The added deviation is proportional to the measurement value and to the temperature channel's deviation from the reference temperature. The reference temperature must be specified.
	"Uncertainty Source.Temp Gain" : Gain deviation increasing in proportion to the temperature, per Kelvin (unit of the temperature channel influenced) from the value 1.0. UncertaintyModify() adds such a gain deviation with a random proprtionality factor which is however fixed for the particular measurement. E.g. 0.01, if the gain deviation is 1% per Kelvin; to be interpreted as the standard deviation. The added deviation is proportional to the measurement value and to the temperature channel's deviation from the reference temperature. The reference temperature must be specified.
	"Uncertainty Source.Temp Ref" : The reference temperature used in UncertaintyModify() for the temperature-dependent influences, in the unit of the temperature channel influenced. If not specified, 20 is assumed. If the temperature channel is stated in °C and the specified value is 23, then this is interpreted as 23&deg;C.
	"*" : Deletes all properties pertaining to the topic measurement uncertainty. For the 2nd parameter, an empty text must be specified.
SvValue	New value to which the property is to be set. For "Uncertainty evaluation" and "*" data type Text, else a number. If an empty text is entered, the property will be deleted.

Description:

The properties defined here directly reflect characteristic parameters regarding the topic measurement uncertainty, which are defined either by the user of by the measurement system, and are calculated for output quantities by the function UncertaintyCalc - or they describe influencing quantities which cause measurement uncertainty ("Uncertainty Source.*") and are taken into account by the function UncertaintyModify().

Further information on determining the measurement uncertainty in FAMOS is presented in the user's manual/online help, in the section "Determining the measurement uncertainty". There, the characteristic parameters and fundamental standards and methods (GUM, Monte-Carlo method) are described in greater detail.

The call UncertaintySet("*", "") deletes all properties associated with the topic of measurement uncertainty.

This function is not available in the FAMOS Standard Edition.

Examples:

From a voltage channel Input1 with a 10V measurement range and measurement uncertainty of 0.1V, a result named "Result" is to be determined by means of an algorithm (low-pass filter, maximum value).

```
; stating the measurment uncertainty of the input data:
UncertaintySet(Input1, "Uncertainty", 0.1)
; run all Monte-Carlo trials, here the trial count M = 1000
UNCERTAINTY_LOOP 1000
   ; adding noise to the input data
   _In1 = UncertaintyModify(Input1)
   ; caling the actual algorithm and calculating the result
   Result = Max(FiltLp(_In1, 0, 0, 1, 0.02))
   ; calculating the measurement uncertainty of the result
   UncertaintyCalc(Result)
END
; querying the measurement uncertainty of the result
uc = UncertaintyGet(Result, "Uncertainty")
```

Definition of an offset drift:

```
UncertaintySet(Input1, "Uncertainty Source.Drift Offset", 0.01) ; [mV]
UncertaintySet(Input1, "Uncertainty Source.Drift Time", 1000) ; [s]
UNCERTAINTY_LOOP 1000
...
```

See also:

UNCERTAINTY_LOOP, UncertaintyGet, UncertaintyModify, UncertaintyCalc, UncertaintySnapshot

UncertaintySnapshot

Available in: Professional Edition and above

Collection of a variable's trial-variations produced by the Monte-Carlo method within an UNCERTAINTY_LOOP.

Declaration:

```
UncertaintySnapshot ( Variable [, Option] )
```

Parameter:

Variable	Variable whose variations are to be collected. For example, a noise-affected input channel returned by UncertaintyModify(), or an intermediate result in the algorithm, or even a final result which is ultimately passed to UncertaintyCalc(). The variable must be specified directly; temporary or indixed data are not allowed.
Option	Type of result (optional)
	0 : For each Monte-Carlo trial, a copy of the data set generated is appended to a group. The group name is assigned automatically, the data sets are consecutively numbered. Often used when subsequent comparison of all variations across all trials is desired.
	1 : Each Monte-Carlo trial is interpreted as a measurement. In accordance with the FAMOS measurement concept, copies of the data sets which are passed are assigned to an automatically numbered measurement name. Often used when there are multiple calls of UncertaintySnapshot() (e.g. for input parameters, intermediate results and the final result) and a comparison is to be made upon every Monte-Carlo trial (e.g. to assess the propagation of the input error). This method then allows convenient visual display and comparison by means of the Variables list's Measurement view.

Description:

This procedure can only be called within an UNCERTAINTY_LOOP and serves the purpose of advanced assessment of how the measurement uncertainty was determined. For example, all results of the function UncertaintyModify() can be collected and thus the influence of the measurement uncertainty property on the generation of the 'noisy signal' can be checked.

As a side effect, this procedure generates variables and initializes, updates, and even deletes them. An "UNCERTAINTY_LOOP M" loop is run (M+2) times - (Initialization, M Monte-Carlo trials, Finalization). In Initialization and Finalization, the function does not do anything; when calling within the first Monte-Carlo trial, all existing variables matching the specified target name are deleted and the first copy (according to the event type selected) is created.

Note that the function generally creates a very large number of data sets (corresponding to the trial count specified for UNCERTAINTY_LOOP) and thus has a very great influence on the sequence's execution time and memory requirements. After setting and commissioning of a sequence, any no longer needed calls should therefore be deleted before applying the sequence productively.

Information on determining the measurement uncertainty in FAMOS is available in the user's manual/online-help, in the section "Determining the measurement uncertainty". Here, the foundational standards and methods (GUM, Monte-Carlo method) are explained in more detail.

This function is not available in the FAMOS Standard Edition.

Principle of name formation with the option = 0 (Generate group)

The name of the group generated is formed from the original name by appending "_uc_shots". The channels generated in the group contain the name "L[loop iteration number]", where the loop iteration number is filled with enough preceding zeroes to match the width of the last loop number.

Example: The following code generates a group "x_uc_shots" having the channels "L001", "L002" ... "L100":

```
UNCERTAINTY_LOOP 100
   x = UncertaintyModify(input)
   UncertaintySnapshot(x, 0)
   ...
END
```

If the parameter is a channel belonging to a group, the nomenclature follows the pattern "[Group name]_[Channel name]_uc_shots". Any measurement association is retained.

Principle of name formation with option = 1 (Generate measurements)

The variable's name is retained, the respective measurement is assigned the name "uc_loop[loop iteration number]", where the loop iteration number is filled with enough zeroes to match the width of the last loop number.

Example: The following code generates the variables "x@uc_loop001", "x@uc_loop002" ... "x@uc_loop100":

```
UNCERTAINTY_LOOP 100
   x = UncertaintyModify(input)
   UncertaintySnapshot(x, 1)
   ...
END
```

If the parameter is a channel or a group, the name formation follows the pattern "[GroupName]_[ChannelName]@uc_loop###"

If the parameter itself is already assigned to a measurement, the name formation follows the pattern "[VariableName]@[MeasurementName]_uc_loop###" or "[GroupName]_[ChannelName]@[MeasurementName]_uc_loop###"

Examples:

From a voltage channel Input1 having a measurement range of 10V and measurement uncertainty of 0.1V, a result named "Result" is to be determined by means of an algorithm (low-pass filter, maximum value). The following code saves all 1000 variations of the input channel (the results of the function UncertaintyModify) in the group "x_uc_snapshots".

```
UncertaintySet(Input1, "Uncertainty", 0.1)
UNCERTAINTY_LOOP 1000
   x = UncertaintyModify(Input1)
   UncertaintySnapshot(x, 0)
   Result = Max(FiltLp(x, 0, 0, 1, 0.02))
   UncertaintyCalc(Result)
END
```

Like previously, but additionally the 1000 variations of the algorithms result are collected. The variables "x@uc_loop0001", "Result@uc_loop0001" through "x@uc_loop1000", "Result@uc_loop1000" are generated.

```
UncertaintySet(Input1, "Uncertainty", 0.1)
UNCERTAINTY_LOOP 1000
   x = UncertaintyModify(Input1)
   UncertaintySnapshot(x, 1)
   Result = Max(FiltLp(x, 0, 0, 1, 0.02))
   UncertaintySnapshot(Result, 1)
   UncertaintyCalc(Result)
END
```

See also:

UNCERTAINTY_LOOP, UncertaintyModify, UncertaintyCalc, UncertaintySet, UncertaintyGet

Unit?

Queries a data set's unit.

Declaration:

```
Unit? ( Data, SvCode ) -> TxUnit
```

Parameter:

Data	Data set whose unit is to be determined
SvCode	Specification of which unit to query
	0 : X-unit for single-component data. For XY-data, the unit of the X-component. For complex data, unit of the phase/imaginary part.
	1 : Y-unit for single-component data. For XY-data, the unit of the Y-component. For complex data, unit of the magnitude/real part.
	2 : Z-unit
	3 : Unit of the parameter for 2-component data
TxUnit	
TxUnit	The unit determined

Description:

Examples:

A data set's Y-unit is queried. If the unit is "W", it is converted to "VA":

```
unitY = Unit?(data, 1)
cmp = TComp(unitY, "W")
IF cmp = 0
   SetUnit(data, "VA", 1)
END
```

See also:

SetUnit, ConvertUnit, XUNIT, YUNIT

UpperValue

Returns the greater value of the two parameters.

Declaration:

```
UpperValue ( Parameter1, Parameter2 ) -> Result
```

Parameter:

Parameter1	First single value or data set to be compared.
Parameter2	Second single value or data set to be compared.
Result	
Result	The respective greater value of the two parameters.

Description:

The function has two practical applications. When the parameters are a data set/ single value combination, the effect is to set an upper limit on the data set values at the second parameter's value. If both parameters are data sets, the result is the upper envelope curve.

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/segments); but the respective counterpart parameter must then either have exactly the same structure (same segment length, event-count and -length) or it must be a single value.

Examples:

The input channel is converted to decibels and limited to an upper limit of -100 dB.

```
Channel_01_dB = UpperValue(db(Channel_01), -100)
```

The lower and upper envelope curves of two data sets are determined.

```
Env_L = LowerValue(Channel_01, Channel_02)
Env_H = UpperValue(Channel_01, Channel_02)
```

See also:

LowerValue, >, RangeSet

UserPropCopy

Copying of user-defined properties

Declaration:

```
UserPropCopy ( Destination, Source, TxPropName )
```

Parameter:

Destination	The target variable to which the properties are to be copied
Source	Variable with the properties to copy
TxPropName	

Description:

The property's name may also be specified containing the wildcard characters '*' (representing a string of any length) and '?' (representing exactly one character). In that case, all properties are copied whose name matches the pattern specified.

With the names of user-defined properties, there is no distinction between upper- and lowercase spelling.

With the permanent properties already predefined by imc, either the internal name ("imc??") or the display name can be specified (e.g. "imc33" or "Uncertainty").

Examples:

A variable is high-pass filtered. All user-defined properties of the source, whose names begin with 'Env.', are copied into the result.

```
Channel1_Filtered =  FiltHP(Channel1, 0, 0, 4, 100)
UserPropCopy(Channel1_Filtered, Channel1, "Env.*")
```

See also:

UserPropSet, UserPropText?, UserPropValue?

UserPropCount?

Gets the amount of a variable's user-defined properties.

Declaration:

```
UserPropCount? ( Variable ) -> SvCount
```

Parameter:

Variable	The variable queried
SvCount	
SvCount	Amount of user-defined properties

Description:

The function is used together with the function UserPropName?() in order to count all of a variable's user-defined properties.

Examples:

For the variable 'Channel', all user-defined properties are enumerated and displayed together with their current contents in the output box.

```
n = UserPropCount?(Channel)
FOR i = 1 TO n
   TxName = UserPropName?(Channel, i)
   TxValue = UserPropText?(Channel, TxName)
   BoxOutput(TxName + " : " + TxValue, EMPTY, "", 1)
END
```

See also:

UserPropName?

UserPropDel

Deletion of user-defined properties

Declaration:

```
UserPropDel ( Variable, TxPropName )
```

Parameter:

Variable	Variable from which the property is to be deleted
TxPropName	Specifies the property(-ies) to be deleted. Empty string or "*" to delete all properties.

Description:

- The properties' name may also be specified containing wildcard characters '*' (representing a string of any length) and '?' (representing exactly one character). Then all properties are deleted whose name matches the specified pattern.
- Pre-defined properties with the attribute 'Write-protected' can not be deleted.
- With the names of user-defined properties, there is no distinction between upper- and lowercase spelling.
- With the permanent properties already predefined by imc, either the internal name ("imc??") or the display name can be specified (e.g. "imc33" or "Uncertainty").

Examples:

All properties are deleted from the variable 'Channel1', whose names begin with 'Env.':

```
UserPropDel(Channel1, "Env.*")
```

See also:

UserPropSet, UserPropText?

UserPropInfo?

Get information on a user-defined property

Declaration:

```
UserPropInfo? ( Variable, TxPropName, TxChoice ) -> SvResult
```

Parameter:

Variable	
TxPropName	The name of the property queried
TxChoice	Selection of information to obtain
	0 : Present? - Returns 1, if the property for this variable is defined. Else 0.
	1 : Valid? - Returns 1, if the property is defined for this variable and, for numerical types, is also initialized (meaning not empty). Else 0.
	2 : Data type - Returns the den property's data type (1: Text, 2: Integer, 3: Real, 4: Boole, 5: Enumeration type, 6: Time) or -1, if the property is not defined.
	3 : Temporary? - Returns 1, if the property has the attribute 'temporary' for this variable, meaning it is ignred when the variable is saved. Else 0, if the attribute is not set, -1 if the property is not defined.
	4 : Write-protected? - Returns 1, if the property has the attribute 'write-protected' for this variable, meaning it is ignred when the variable is saved. Else 0, if the attribute is not set, -1 if the property is not defined.
SvResult	
SvResult	Result according to [Selection].

Description:

With the names of user-defined properties, there is no distinction between upper- and lowercase spelling.

With the permanent properties already predefined by imc, either the internal name ("imc??") or the display name can be specified (e.g. "imc33" or "Uncertainty").

Examples:

All variables currently present in FAMOS, for which the property 'UserComment' is defined, are displayed in the output box together with this property's content.

```
Count = VarGetInit2("*", 2)
n = 1
WHILE n <= Count
   TxVarName = VarGetName?(n)
   IF UserPropInfo?(<TxVarName>, "UserComment", 0)
      TxValue = UserPropText?(<TxVarName>, "UserComment")
      BoxOutput(TxVarName + " : " + TxValue, EMPTY, "", 1)
   END
   n = n+1
END
```

See also:

UserPropSet, UserPropText?

UserPropName?

Gets the name of a user-defined property of a variable.

Declaration:

`UserPropName? ( Variable, SvIndex ) -> TxName`

Parameter:

Variable	The variable queried
SvIndex	Index (beginnng at 1) of the property queried
TxName	
TxName	Name of the property

Description:

The function is used together with the function UserPropName?() in order to count all of a variable's user-defined properties.

Examples:

For the variable 'Channel', all user-defined properties are enumerated and displayed together with their current contents in the output box.

```
n = UserPropCount?(Channel)
i = 1
WHILE i <= n
   TxName = UserPropName?(Channel, i)
   TxValue = UserPropText?(Channel, TxName)
   BoxOutput(TxName + " : " + TxValue, EMPTY, "", 1)
   i = i+1
END
```

See also:

UserPropCount?

UserPropSet

Creates/sets a user-defined property

Declaration:

```
UserPropSet ( Variable, TxPropName, Content, SvType, SvAttribute )
```

Parameter:

Variable	
TxPropName	Name of the property to alter
Content	New content or value to which the property is to be set
SvType	The property's data type
	0 : Automatic/Retain: When a new property is created, the type is automatically determined based on the parameter supplied. For Text, the type 'Text' is selected; for a number the type 'Real'. When an existing property is changed, the existing type is retained.
	1 : Arbitrary text
	2 : A whole number
	3 : A real number
	4 : A property whose value can only be either 0 or 1
SvAttribute	Additional attribute for the property to be set
	0 : Automatic/retain: When a new property is created, it is assumed to be 'permanent', meaning the property is saved when the variable is saved with the file.
	1 : Temporary: The property is regarded as temporary and ignred when the variable is saved.
	2 : Permanent: The property is included when the variable is saved (imc file format).

Description:

The type for the property's new content (text/number), which the user supplys as the 3rd parameter, must match the property's specified data type.

Properties having names beginning with 'imc' are reserved for internal purposes. It is not possible to create new properties with such a name, and when exporting existing properties, only the content but not the data type or attribute may be changed. Thus, [SvAttribute] and [SvType] are ignored. Furthermore, internal properties may also be supplied with the attribute 'write-protected'. With properties already predefined by imc, either the internal name ("imc??") or the display name can be specified (e.g. "imc33" or "Uncertainty").

If the name for a property contains a period, then the part before the period is regarded as a category designation. Thus, it is possible to group properties having similar significance. The category is used, for example, for hierarcical listing of properties in the dialog 'Variable'/'Properties'.

The following constraints apply to names for new properties:

- The first character may not be a period.
- Neither the first or last character may be a space.
- The following characters are not allowed: <>{}[]()&%$§|
- Special characters with an ASCII-code < 32 (e.g. Tabulator) may not be included.

With the names of user-defined properties, there is no distinction between upper- and lowercase spelling.

Examples:

A data set is amended with various information specific to the measurement, which is written to the file when the data set is saved in imc file format:

```
UserPropSet(data, "TestNumber", 212, 2, 0)        ; integer
UserPropSet(data, "Env.UserName", "Smith", 1, 0)  ; Text
UserPropSet(data, "Env.Temperature", 21.5 , 3, 0) ; real
UserPropSet(data, "Env.CalibrationOK", 1, 4, 0)   ; bool
```

See also:

UserPropText?, UserPropValue?, UserPropCopy

UserPropText?

Gets a user-defined property

Declaration:

```
UserPropText? ( Variable, TxPropName ) -> TxContent
```

Parameter:

Variable	The variable queried
TxPropName	Name of the property to query
TxContent	
TxContent	Content of the property queried

Description:

If the property does not exist, an empty text is returned. You can use the function UserPropInfo?() with the option 0 beforehand, in order to verufy the property's existence.

With the names of user-defined properties, there is no distinction between upper- and lowercase spelling.

With the permanent properties already predefined by imc, either the internal name ("imc??") or the display name can be specified (e.g. "imc33" or "Uncertainty").

Examples:

For the variable 'Channel', all user-defined properties are enumerated and displayed together with their current contents in the output box.

```
n = UserPropCount?(Channel)
i = 1
WHILE i <= n
   TxName = UserPropName?(Channel, i)
   TxValue = UserPropText?(Channel, TxName)
   BoxOutput(TxName + " : " + TxValue, EMPTY, "", 1)
   i = i+1
END
```

The second column of a table in the active Panel is filled with a data set's values. The first row contains the name, the 2nd and 3rd rows the content of two user-defined properties. The numerical values start at the 4th row.

```
PnTableSetCell("Tab1", 2, 1, "channel1")
PnTableSetCell("Tab1", 2, 2, UserPropText?(Kanal1,  "Env.UserName"))
PnTableSetCell("Tab1", 2, 3, UserPropValue?(Kanal1, "Env.Temperature"))
PnTableSetColumn("Tab1", 2, 4, channel1)
```

See also:

UserPropSet, UserPropValue?

UserPropValue?

Gets a numeric user-defined property

Declaration:

```
UserPropValue? ( Variable, TxPropName ) -> SvValue
```

Parameter:

Variable	
TxPropName	Name of the property to query
SvValue	
SvValue	Value of the property queried

Description:

If the property does not exist or is not initialized, 0 is returned. You can use the function UserPropInfo?() with the option 1 beforehand in order to verify the property's validity.

The function can only be called for property with a numeric type (real, integer, boole).

With the names of user-defined properties, there is no distinction between upper- and lowercase spelling.

With the permanent properties already predefined by imc, either the internal name ("imc??") or the display name can be specified (e.g. "imc33" or "Uncertainty").

Examples:

A multi-channel file in imc-format is opened. All channels contain the ambient temperature during measurement in the property 'Env.Temperature'. All channels are displayed whose ambient temperature exceeded 30 degrees.

```
idFile = FileOpenDSF("c:\imc\dat\xxx.dat", 0)
IF idFile >= 1
   count = FileObjNum?(idFile)
   index = 1
   WHILE index <= count
      TxName  = FileObjName?(idFile, index)
       <TxName>= FileObjRead(idFile, index)
      envTemp = UserPropValue?(<TxName>, "Env.Temperature")
      IF envTemp > 30
         SHOW <TxName>
      END
      index = index + 1
   END
   FileClose(idFile)
END
```

See also:

UserPropSet, UserPropText?

Value

Returns a data set's y-values at specified x-position.

Declaration:

```
Value ( Data, XPositions ) -> Values
```

Parameter:

Data	Data set queried; allowed types: [ND],[XY].
XPositions	X-coordinates whose associated Y-values are to be determined.
Values	
Values	Y-values determined [ND]

Description:

This function returns the value of the specified data set at a predefined x-coordinate, i.e. the ycoordinate corresponding to an x-coordinate is determined.

Data type NW: If the x-coordinate does not exactly coincide with any of the data set's sample values, the x-coordinate is rounded to the preceding sample value.

Data type XY: The function can only be applied to data sets with a strictly monotonic X-track. If the x-coordinate does not exactly coincide with a sample value in the data set, the y-coordinate is interpolated linearly.

This function is obsolete; the more powerful funciton Value2() is generally preferable.

- The result has the same y-unit as the input data set.
- The x-position of the value to be determined should be specified as an x-coordinate, not as the index of a point in the data set.
- If the specified x-position lies outside the range of the data set, the corresponding beginning or end value of the data set is returned.
- The rounding of normal data type data sets to the preceding sample value is subject to some slight imprecision, so that a specified x-value is considered to exactly coincide with a sample value if it is located less than 1/100 of a sampling step before it. Thus if the value for which the function is searching is only slightly less than a sample value, the value is returned at this position anyway. This behavior is intended to compensate for numerical discrepancies where X-coordinates are previously calculated.
- The specified second parameter should indicate the unit of the x-coordinate.
- When several x-coordinates are specified, the corresponding number of y-values are created and returned as a data set.
- If the numerical values of a data set are desired, but no further calculations are to be performed with them, use the crosshairs in the measurement value window in the corresponding Curve Window. Use the stairstep display for more convenient operation. The Data Editor can also be used to view the values in a table.
- Alternatively, you can use the function ValueIndex, or indexing by means of square brackets [..] if you wish to specify the desired positions via the indices of the data set's points.

Examples:

The y-coordinate of a data set with an expansion in the x-direction from 2 s to 10 s is determined for the x-coordinate 5 s:

```
yValue = Value(NDdata, 5 's')
```

The last value of the data set is determined. The expansion of the data set is less than 1E20.

```
yLast = Value(NDdata, 1E20)
```

The value of a data set is doubled at the x-coordinate 6:

```
NDdata = Set(NDdata, 6, 2 * Value(NDdata, 6))
```

See also:

Value2, ValueIndex, PosiEx2, Set

Value2

Returns a data set's respective y-values for specified x-positions.

Declaration:

```
Value2 ( Data, XPositions, SvInterpolation ) -> YValues
```

Parameter:

Data	Data set examined. Allowed types: [ND],[XY]. The time axis/x-axis must be monitonic.
XPositions	X-coordinates for which the associated Y-values are to be determined
SvInterpolation	If the x-coordinate does not exactly coincide with a data set's sample value, the result is interpolated as follows:
	0 : Linear. The input data set is interpolated linearly.
	1 : Constant, preceding value. The input data set is subject to level interpolation, i.e. each value is kept constant until a new sample value becomes effective. Thus, the result value is the input data set's value whose x-coordinate is immediately BEFORE the x-coordinate examined.
	2 : Constant, closest value. The result value is the input data set's value whose x-coordinate is CLOSEST to the x-coordinate examined.
YValues	
YValues	The Y-values found

Description:

This function returns the values of the data set supplied by the user at specified x-coordinates, i.e., the y-corrdinates associated with the specified x-coordinaten are found.

- If the x-position specified is outside of the data set's domain, the data set's first/last value is returned.
- Linear interpolation is generally used for continuous input signals.
- The constant interpolation styles are most sensible to use when the input signal is comprised of pre-defined discrete values. This applies to digital data , or measured data which by nature can only take integer values (e.g. the current gear in a transmission system). The resulting data set consists only of values which also exist in the input data set, and has the same data format.
- Linear interpolation for digital input data is not possible, in which case the interpolation parameter may automatically be corrected to the value 1 (constant interpolation).
- Alternatively, you can work with the function ValueIndex() if you wish to specify the desired positions by the points' indeces rather than by their x-coordinates in the data set.

Special case of SvInterpolation = 1 (constant, preceding):
In this case there is some slight imprecision, so that a specified x-value is considered to exactly coincide with a sample value if it is located less than 1/100 of a sampling step before it. Thus if the value for which the function is searching is only slightly less than a sample value, the value is returned at this position anyway. This behavior is intended to compensate for numerical discrepancies where X-coordinates are previously calculated

Examples:

A data set's Y-value at the position X=5 is determined. If needed, linear interpolation is applied.

```
SignalAt5 = Value2(Signal, 5 's', 0)
```

A vehicle's speed <velocity> and transmission stage (<gear> comprising only the values 1 - 5) are measured. All times are determined at which the vehicle was accelerated beyond 50 km/h and in each such case the gear engaged is recorded.

```
t = PosiEx2(velocity, 50, 1, 0)
GearAt50 = Value2(Gear, t, 1)
GearAt50XY = XYof(t, GearAt50)
```

See also:

ValueIndex, PosiEx2, MatrixGet, RSampEx

ValueIndex

Determines the y-values of a data set at positions specified by indices.

Declaration:

```
ValueIndex ( Data, Indices ) -> Values
```

Parameter:

Data	Data set queried; allowed types: [ND],[XY].
Indices	Indices of points, whose associated y-values are to be determined
Values	
Values	Y-values determined [NW]

Description:

For the data set [Data], at all specified positions (indices), the associated y-values are returned.

The indices specified in [Indices] must have values between 1 and the length of [NDData].

For indices < 1, the first y-value , for indices > (data set length) the last value is returned.

To query a single value in imc FAMOS by means of its index in the data set ,you can also address its index within the data set in a formula:

```
singleValue = Data[ Index ]
```

- There first point in a data set has the index 1; the index of the last point matches the data set length.
- Alternatively, you can use the function Value2(), in which the positions of data pointsare specified in terms of their x-coordinates.
- If you specify multiple indices, the function returns a corresponding amount of y-values as a data set.
- To determine numerical values in a data set without intending to process them any further, you can use the measurement cursors in the measurement value window of the data set concerned. Note that using the "Steps" display option offers special convenience for this purpose. Another possibility is to use the imc FAMOS Data Editor to view the values in a table.

Examples:

The last value of a data set is determined:

```
LastValue = ValueIndex(Data, Leng?(Data))
; also:
LastValue = Data[Leng?(Data)]
```

Every second y-value of the data set is doubled:

```
Indizes = Ramp(1, 2, Leng?(Data)/2) ;Indizes = 1,3,5..
NewY= ValueIndex(Data, Indizes) * 2
DataNew = SetIndex(Data, Indizes, NewY)
```

See also:

Value2, CutIndex, SamplesGate, ReplIndex, SetIndex, MatrixGet

VarExist?

Queries whether a variable of a certain name exists in imc FAMOS.

Declaration:

```
VarExist? ( TxVariableName ) -> SvExist
```

Parameter:

TxVariableName	Name of the variable to be found
SvExist	
SvExist	0: Not present / 1: Present

Description:

Checks whether a variable having the specified name exists in imc FAMOS at the time the function is called.

Data sets belonging to groups can be specified in the customary form 'GroupName:ChannelName'.

Local variables are not found.

Examples:

```
FileLoad("c:\imc\dat\multi.dat", "", 0) ; Multichannel-File
IF VarExist?("channel012")
   SHOW channel012
END
```

A multi-channel file is loaded. It is checked for the presence of a channel bearing a specified name. The particular channel is displayed, if it exists. If so, it is displayed.

See also:

VarGetInit2, VarGetInit, VarGetName?, Name?

VarGetInit

Gets the FAMOS variables which are present when the sequence execution starts. Initialization for subsequent calls of VarGetName?().

Declaration:

```
VarGetInit ( SvOption ) -> SvCount
```

Parameter:

SvOption	Options parameter
	0 : All entries
	1 : Only selected entries
	2 : All entries; extended syntax
	3 : Only selected entries; extended syntax
	4 : Number of sequence parameters passed. This call is deprecated, newly created sequences should use the ParametersPassed?() function (available since version 2024) instead.
SvCount	
SvCount	Amount of variables found.

Description:

This function (VarGetInit) returns the contents of the Standard Variables List at the start of the sequence execution. The function VarGetInit2, by contrast, returns the variables which are present when the function is called.

During execution of a sequence, the status of the variables list is actually undefined. The exception is the case where the sequence is being executed in single-step mode; in this case, the status of the Variables list at the moment the function is called is used.

VarGetInit is used to prepare for subsequent inquiry of the variables list with the function VarGetName?().

Local variables are ignored.

In order that a variable name can be entered in formulas, it must comply with certain rules (e.g. no spaces, the first character may not be a digit etc.). If this is not the case, the name must additionally appear inside curly brackets {...}. In options 2 and 3, the name returned by VarGetName? may be automatically amended with the curly brackets, for which reason these options are preferable for this type of application.

Using these two functions in concert, sequences can be written which, for instance, manipulate all selected variables.

The function with the options 0-3 returns the actually visible status of the variable list, so it also takes into account a set display filter, for example, and can only be used if the FAMOS main window with variable list is visible at all (e.g. not guaranteed when called by Runtime or when remotely controlled by imc STUDIO). If all existing variables are required, the VarGetInit2() function should be used.

The option '4' does not apply to the Variables list, but has a special meaning:If the currently executed sequence has been called with the command SEQUENCE, this call returns the number of sequence parameters passed to it; else -1.

Reference: The BoxVarSelector() function shows a list of the variables available at the time of the call, from which you can then make an interactive selection.

Examples:

For all entries selected in the Variables list, the mean value is taken:

```
count = VarGetInit(1)
FOR index = 1 TO count
   TxName = VarGetName?(index)
   TxNameMean = TxName + "_M"
   <TxNameMean> = Mean(<TxName>)
END
```

A sub-sequence expects 2 parameter. If only 1 parameter is given, a default value is used for the 2nd parameter.

```
ParCount = VarGetInit(4)
; or, since FAMOS 2024: ParCount = ParametersPassed?()
SWITCH ParCount
   CASE 2
      ; ok, PA1 and PA2 are specified
   CASE 1
      ; PA2 is missing, use default value
      PA2 = 2
   DEFAULT
      BoxMessage("Error", "Calling error", "!1")
      EXITSEQUENCE
END
```

See also:

VarGetInit2, VarGetName?, ParametersPassed?, BoxVarSelector, VarExist?

VarGetInit2

Gets all FAMOS variables existing at the moment when called. Initialization for subsequent calls of VarGetName?().

Declaration:

```
VarGetInit2 ( TxNamePattern, SvOption ) -> SvCount
```

Parameter:

TxNamePattern	Name pattern for the listing of variables. The interpretation depends on the parameter's value [option]. The wildcards "*" and "?" can be used in their customary meanings.
SvOption	Options parameter
	0 : All variables are accepted whose names have a match with the specified name pattern. To list all exisitng variable, enter an empty text or "*".
	1 : All variables are accepted whose names DO NOT have a match with the specified name pattern..
	2 : All variables are accepted whose names match the specified name pattern. Variable names with extended syntax are prepared for direct use in formulas (see Remarks).
	3 : All variables are accepted whose names DO NOT match the specified name pattern. Variable names with extended syntax are prepared for direct use in formulas (see Remarks).
SvCount	
SvCount	Amount of variables found.

Description:

This function is used to initialize a subsequent count of available imc FAMOS variables using the function VarGetName?().

The function VarGetInit2() lists the variables existing at the moment of the function call. The function VarGetInit(), by contrast, returns the existing variables at the beginning of the (sequence-)run.

To get the variable names, use the function VarGetName?() next.

Local variables are ignored.

In order that a variable name can be entered in formulas, it must comply with certain rules (e.g. no spaces, the first character may not be a digit etc.). If this is not the case, the name must additionally appear inside curly brackets {...}. In Options 2 and 3, the name returned by VarGetName?() may be automatically amended with the curly brackets, for which reason these options are preferable for this type of application.

Reference: The BoxVarSelector() function shows a list of the variables available at the time of the call, from which you can then make an interactive selection.

Examples:

All variables whose names are prefaced with the string "channel" are displayed in a curve window.

```
Count = VarGetInit2("channel*", 2)
FOR n = 1 TO Count
   TxVarName = VarGetName?(n)
   SHOW <TxVarName>
END
```

All variables whose names DON'T begin with the character "$" are saved together in a file.

```
$fh = FileOpenDSF("z:\allvars.dat",1)
$Count = VarGetInit2("$*", 3)
FOR $i = 1 TO $Count
   $TxVarName = VarGetName?($i)
   FileObjWrite($fh, <$TxVarName>)
END
FileClose($fh)
```

See also:

VarGetInit, VarGetName?, BoxVarSelector, VarExist?

VarGetName?

Gets an entry from the FAMOS Variables list

Declaration:

```
VarGetName? ( SvIndex ) -> TxName
```

Parameter:

SvIndex	Index of the entry (1..)
TxName	
TxName	Name of the desired Variables entry

Description:

Before using this function, either of the functions **VarGetInit** or **VarGetInit2** must be called in order to initialize it.

The function VarGetInit2 lists the variables available at the moment it is called. By contrast, the function VarGetInit returns the content of the imc FAMOS variables list at the beginning of the (sequence-) run.

Examples:

For all entries selected in the Variables list, the mean value is taken:

```
count = VarGetInit(1)
FOR index = 1 TO count
   TxName = VarGetName?(index)
   TxNameMean =  TxName + "_M"
   <TxNameMean> = Mean(<TxName>)
END
```

All variables whose names DON'T begin with the character "$" are saved together in a file.

```
$fh = FileOpenDSF("z:\allvars.dat",1)
$Count = VarGetInit2("$*", 3)
FOR $i = 1 TO $Count
   $TxVarName = VarGetName?($i)
   FileObjWrite($fh, <$TxVarName>)
END
FileClose($fh)
```

See also:

VarGetInit, VarGetInit2, VarExist?, Name?

Verify

Available in: Professional Edition and above

Checks whether a value is true, i.e. not equal to zero. Returns an error message otherwise. In that case, the running of the sequence is also cancelled.

Declaration:

```
Verify ( Value [, Error text] )
```

Parameter:

Value	This value is checked against 0.0.
Error text	Text outputted when an error occurs (optional)

Description:

Verify() is used as a compact way to incorporate verifications of inputs or self-verification into sequences.

Examples:

Displays an error because A and B are not equal and Equal() returns 0.

```
A=2
B=sqrt(2)^2 ; A <> B !
verify( equal( A, B ))
```

See also:

Equal

VerifyVar

Checks whether a variable meets specified conditions.

Declaration:

```
VerifyVar ( Variable, TxCondition1 [, TxCondition2] [, C3] [, C4] [, C5] [, C6] [, C7] [, C8] [, C9] [, C10] [, C11] [, C12] [, C13] [, C14]
) -> OK
```

Parameter:

Variable	Variable to be checked
TxCondition1	1st condition
	"SV" : Single value (normale data set of length 1)
	"ND" : Normal data set
	"ND(->)" : Normal data set; no events, no segments
	"ND(..)" : Normal data set, Length > 1
	"ND(..,->)" : Normal data set; no events, no segments, Length > 1
	"CX" : Complex data set
	"!CX" : Data set, not complex
	"CX(->)" : Complex data set; no events, no segments
	"RI" : Complex data set in Real-/Imaginary-part representation
	"MP" : Complex data set in Magnitude/Phase-representation
	"DP" : Complex data set in Decibel/Phase-representation
	"XY" : XY-data set
	"!XY" : Data set; not XY
	"XY(->)" : XY-data set; no events, no segments
	"XY(/)" : XY-data set with monotonically increasing x-track
	"XY(->,/)" : XY-data set with monotonically increasing x-track; no events, no segments
	"TSA" : Data set, TimeStamp-ASCII
	"!TSA" : Data set; not TimeStamp-ASCII
	"TSA(->)" : TimeStamp-ASCII; no events, no segments
	"DS" : Data set (any type), therefore no text, text array or data group
	"Evn" : Data set (any type) with events
	"Seg" : Data set (any type) with segments
	"SegEvn" : Data set (any type) with segments and events
	"!Evn" : Data set (any type) without events
	"!Seg" : Data set (any type) without segments
	"!SegEvn" : Data set (any type); no events, no segments
	"->" : Abbreviation for "!SegEvn"; data set (any type), no events, no segments
	"Monotone" : Single value, normal data set or XY-data set with monotonically increasing x-track
	"/" : Abbreviation for "Monotonic". Single value, normal data set or XY-data set with monotonically increasing x-track
	"Digital" : Digital data set
	"!Digital" : Data set; not digital
	"TXT" : Text
	"!TXT" : No text; therefore data set, text array or data group
	"TXA" : Text array
	"!TXA" : No text array; therefore data set, text or data group
	"TX*" : Text or text array
	"!TX*" : No text or text array; therefore data set or data group
	"Group" : Data group
	"!Group" : No data group
	"GrMember" : Element of a data group
	"!GrMember" : No element of a data group
	"Empty" : Variable is empty. Length 0 for data sets; text length 0 for texts; element count 0 for text arrays and TSA.
	"!Empty" : Variable is not empty
	"@M" : The variable is assigned to a measurement.
	"!@M" : The variable is not assigned to any measurement.
	"ReadOnly" : Write protection

	"!ReadOnly" : No write protection
	"V74Compat" : Variable has no properties which are introduced in future versions of FAMOS and which might possibly impair compatibility with existing sequences.
TxCondition2	2nd condition (optional)
C3	3rd condition (optional)
C4	4th condition (optional)
C5	5th condition (optional)
C6	6th condition (optional)
C7	7th condition (optional)
C8	8th condition (optional)
C9	9th condition (optional)
C10	10th condition (optional)
C11	11th condition (optional)
C12	12th condition (optional)
C13	13th condition (optional)
C14	14th condition (optional)
OK	
OK	The value 1 if the variable meets all specified conditions; else 0.

Description:

The function checks whether a variable meets the specified conditions. It is helpful for the purpose of verifying already at the beginning of an analysis whether input variables meet requirements. This can prevent certain errors from occurring in the subsequent execution of the sequence, e.g if a variable takes an unexpected data type and it can not be processed by the functions which call it.

The specified conditions are in an AND conjunction, so the result is exactly 1 whenever all conditions without exception are met.

Special case: When "Group" is specified as the **first** condition, the following special feature applies:

If the variable is of the type "Data Group", all subseqent conditions are applied to all the channels included (AND-conjunction). This provides a convenient way to check whether all of a data group's elements meet a given requirement.

Test of whether the parameter is a data group and all elements are single values:

```
ok = VerifyVar(MyGroup, "Group", "SV")
```

Test of whether the parameter is a data group and all elements are empty:

```
ok = VerifyVar(MyGroup, "Group", "Empty")
```

Test of whether the parameter is a data group without elements:

```
ok = VerifyVar(MyGroup, "Empty" "Group")
```

Examples:

A sequence determines the moving maximum of the 1st parameter.

At the beginning of the sequence, the system checks whether the parameter passed is an unstructured data set (no events, no segments). There is also a test of whether it is an equidistantly sampled data set or an XY-data set with a monotonically increasing x-axis. If not, an error message is displayed and the sequence exited.

If the data set is an XY-data set, it is sampled equidistantly before processing resumes. This ensures that the subsequent algorithm is performed without errors.

```
par1 = PA1
IF NOT(VerifyVar(par1, "!SegEvn", "Monotone"))
   ThrowError("Invalid data type of the 1st parameter.")
END

IF VerifyVar(par1, "XY")
   par1 = XYdt(Cmp1(par1), Cmp2(par1), 0.1)
END

par1 = Smo5(par1)
result = MvMax(par1, 1, 1)
```

The following sequence function checks whether the entire course of a data set's signal is above a specified reference data set's signal course. If the function can not be executed due to inappropriate parameters, this is indicated by a special return value. The caller ca query the error cause by using GetLastError().

```
; Declaration: !CheckAbove(TestData, Lower) => Result
; Result = 1:  OK
; Result = 0: At least 1 value of [TestData] is lower than the corresponding value in [Lower]
; Result = -1: Error: Input data do not match (in terms of x-axis) or have events/segments

OnError("Return", "Unknown error", "Result", -1)
IF NOT(VerifyVar(TestData, "!SegEvn")) OR NOT(VerifyVar(Lower, "!SegEvn"))
   ThrowError("The paramater may not have segments or events!")
END
Verify(Leng?(TestData)=Leng?(Lower) AND xdel?(TestData)=xdel?(Lower) AND xoff?(TestData)=xoff?(Lower), "Parameter: x-scaling incompatible!")
Result = Max(Lower - TestData) < 0
```

The following routine checks whether the variable [test] is either a normal data set or an XY-data set having at least 10 samples:

```
ok = VerifyVar(test, "ND")) OR VerifyVar(test, "XY"))
ok = ok AND (Leng?(test) >= 10)
```

The following routine checks whether the variable [gr] represents a data group which exclusively contains normal data sets or complex data sets in Magnitude/Phase-representation:

```
ok = VerifyVar(gr, "Group", "ND")) OR VerifyVar(gr, "GROUP", "BP"))
```

See also:

Verify, OnError, ThrowError, DataFormat?

VFAppendCwSnapshot

The content of the currently selected curve window is exported to a bitmap, and this is appended to a video file previously opened using VFOpen().

Declaration:

```
VFAppendCwSnapshot ( File-ID ) -> Success
```

Parameter:

File-ID	ID of the opened video file. Matches the return value of the function VFOpen()
Success	
Success	Success of the function: 1, if the function was executed successfully; 0 at fault condition. At fault condition, the cause can be queried by means of the function GetLastError().

Description:

The curve window must be generated and selected prior to calling this function. To do this, the functions of the curve window kit (CwSelectWindow() etc.) can be used.

The image is generated on the screen at the current size of the curve window content.

If the size of the resulting bitmap does not match the video's image size, the bitmap is positioned centered in the video image.

Examples:

A folder contains various files of measured data resulting from an experiment. In the order in which they were generated, all files matching a specific name pattern are loaded, displayed in a fixed configuration, and their curve plots are appended to a video. The video image size matches the curve imgae size on screen; the video format is MP4 in default quality with 2 frames per second.

```
CwSelectMode("variable")
id = VFOpen( "c:\video\test_2019_04_18.mp4", 0, 0, 0, 0, 2)
IF id > 0
   CwLoadCCV(id, "data.ccv") ;curve configuration with fixed channel name "data"
   CwSelectWindow(id)
   fileNames = FsGetFileNames("c:\test\2019_04_18","ch*.raw", 0, 0, 2)
   FOREACH ELEMENT fileName in fileNames
      fh = FileOpenDSF(fileName, 0)
      IF fh > 0
         data = FileObjRead(fh, 1)
         FileClose(fh)
          VFAppendCwSnapshot(id )
      END
   END
   VFClose(id)
END
```

See also:

VFOpen, VFAppendFrame, VFAppendPanelSnapshot, VFAppendRGBData, VFClose

VFAppendFrame

A bitmap is loaded from an image file and appended to a video file previously opened with VFOpen().

Declaration:

```
VFAppendFrame ( File-ID, ImageFile ) -> Success
```

Parameter:

File-ID	ID of the opened video file. Matches the return value of the function VFOpen()
ImageFile	Complete path to a bitmap image file
Success	
Success	Success of the function: 1, if the function was executed successfully; 0 at fault condition. At fault condition, the cause can be queried by means of the function GetLastError().

Description:

The following file types are supported: BMP, GIF, JPEG, PNG, TIFF with color depths of 24 or 32 bits per pixel, and 8-bit bitmaps with color table.

If the size of the resulting bitmap does not match the video's image size, the bitmap is positioned centered in the video image.

Examples:

A folder contains various PNG-files which are jpined together to make a video (MP4; with resolution: 1280x720, 1fps).

```
 id = VFOpen( "d:\data.mp4", 0, 0, 1280, 720, 1)
IF id > 0
   fileNames = FsGetFileNames("d:\report_exports","*.png", 0, 0, 2)
   FOREACH ELEMENT fileName in fileNames
      VFAppendFrame(id, fileName)
   END
   VFClose(id)
END
```

See also:

VFOpen, VFAppendPanelSnapshot, VFAppendCwSnapshot, VFAppendRGBData, VFClose

VFAppendPanelSnapshot

A page of the active Panel is exported as a bitmap and this is appended to a video file previously opened with VFOpen().

Declaration:

```
VFAppendPanelSnapshot ( File-ID, PageSelection, Size ) -> Success
```

Parameter:

File-ID	ID of the opened video file. Matches the return value of the function VFOpen()
PageSelection	Selection of the page to be exported.
	-1 : The active page is exported.
	>=1 : Page number of the page to be exported
Size	Size specification for the exported bitmap. Only used for Report-pages; for dialog-pages, this value is ignored and the current size is always used on the screen.
	0 : The dimensions corresponding to the current display are used. The resulting size is thus dependent on the current zoom level.
	>=72 : The value states the resolution to select in 'dpi' (dots per inch). Typical values include 150dpi or 300dpi. The value must lie within the range 72 - 600dpi. Example: For a Report-page in A4 landscape format (297x210mm) with a 10mm margin (resulting size thus 277x190mm) and 150dpi, the resulting bitmap size is thus (277*150/25.4, 190*150/25.4) = (1636x1124) pixels.
Success	
Success	Success of the function: 1, if the function was executed successfully; 0 at fault condition. At fault condition, the cause can be queried by means of the function GetLastError().

Description:

If the size of the resulting bitmap does not match the video's image size, the bitmap is positioned centered in the video image.

For dialog-pages, as exact a replica of the screen output as possible is generated. All visible widgets are outputted; the widget-property 'Print/Export' is ignored.

For Report-pages, as exact a replica of the printout (without margins) as possible is generated. The widget-property 'Print/Export' is implemented.

Examples:

A Panel has a "Take Snapshot" button. When this button is pressed, a snapshot is taken of the current Panel page and is appended to a video file which is opened upon loading the Panel. When the Panel is closed, the Video-file is also closed.

Event-sequence 'Init':

```
; Opening the video-file in the WMV-format with 2 fps. The video image size matches the size of the first image added.
videoID = VFOpen( "d:\evaluation.wmv", 0, 0, 0, 0, 2)
```

Event-sequence 'Button pressed' for the 'Take Snapshot'-button:

```
; appending the current Panel-image (Page 1) in the original size
VFAppendPanelSnapshot( videoID, 1, 0)
```

Event-sequence 'End':

```
VFClose(videoID)
```

See also:

VFOpen, VFAppendFrame, VFAppendCwSnapshot, VFAppendRGBData, VFClose

VFAppendRGBData

An RGB-data set is converted to a bitmap or a sequence of bitmaps, and these are appended to a video file previously opened using VFOpen().

Declaration:

```
VFAppendRGBData ( File-ID, RGBDataSet ) -> Success
```

Parameter:

File-ID	ID of the video file opened. Matches the return value of the function VFOpen().
RGBDataSet	Data set with RGB-attribute. The individal values contain the color information for one pixel.
Success	
Success	Success of the function: 1, if the function was executed successfully; 0 at fault condition. At fault condition, the cause can be queried by means of the function GetLastError().

Description:

The data set passed must be interpreted as an image frame, where each sample contains the color information for 1 pixel in RGB- or grayscale format. Such data sets are denoted by a special Image attribute. By default, the Image attribute is only set by import filters for image files, special functions such as VpGetImages(), or explicitly by the function SetFlag().

The data set may have events, where each event is interpreted as one frame.

If the size of the resulting bitmap does not match the video's image size, the bitmap is positioned centered in the video image.

Examples:

A Panel with a Video Player widget is loaded and a video file is displayed. Starting at the 400th frame, 200 frames are extracted to a new MP4-file.

```
PnLoad( "videoplayer.panel")
VpVideoLoad("c:\videos\sample.avi",1)
VpSetPosFrames(400,1)
images = VpGetImages(200)
id = VFOpen( "c:videos\clipped.mp4", 0, 0, 0, 0, 25, 2)
IF id > 0
   VFAppendRGBData(id, images)
   VFClose(Id)
END
```

See also:

VFOpen, VFAppendPanelSnapshot, VFAppendCwSnapshot, RGB, VpGetImages, Flag?

VFClose

A video file is closed and its contents written to the data carrier.

Declaration:

`VFClose ( File-ID ) -> Success`

Parameter:

File-ID	ID of the opened video file. Matches the return value of the function VFOpen(). By specifying 0, all video file currently open are closed.
Success	
Success	Success of the function: 1, if the function was executed successfully; 0 at fault condition. At fault condition, the cause can be queried by means of the function GetLastError().

Description:

Examples:

A Panel with a Video Player widget is loaded and a video file is displayed. Starting at the 400th frame, 200 frames are extracted to a new MP4-file.

```
PnLoad( "videoplayer.panel")
VpVideoLoad("c:\videos\sample.avi",1)
VpSetPosFrames(400,1)
images = VpGetImages(200)
id = VFOpen( "c:\videos\clipped.mp4", 0, 0, 0, 0, 25, 2)
IF id > 0
   VFAppendRGBData(id, images)
   VFClose(Id)
END
```

See also:

VFOpen, VFAppendPanelSnapshot, VFAppendCwSnapshot, VFAppendFrame, VFAppendRGBData

VFOpen

A videofile is opened and prepared for subsequent appending of individual frames.

Declaration:

```
VFOpen ( Filename, Mode, Format, Width, Height, FrameRate [, Quality] ) -> File-ID
```

Parameter:

Filename	Complete path name of the video file
Mode	
	0 : The file is created from scratch. Any already existing file of the same name is overwritten.
	1 : If the file already exists, it is opened for appending. In this case, all video parameters of the existing file are retained and the function's other parameters are ignored. However, these must still have values which are permitted.
Format	Video-format
	0 : Automatic determining of the format based on the file extension. Supprted formats: ".mp4", ".wmv" and ".avi".
Width	Width of the video image in pixels. Must be a multiple of 2 and lie in the range 100..4096.
Height	Height of the video image in pixels. Must be a multiple of 2 and lie in the range 100..4096. The width and height may also both be 0; in this case the width and height of the video are determined automatically by the dimensions of the first image added (see the functions VFAppend*).
FrameRate	Frame rate in fps (frames per second).
Quality	Determines the quality (compression) of the encoded video. Smaller values mean higher quality, but also a larger file size. (optional , Default value: 3)
	1 :
	2 :
	3 :
	4 :
	5 :
File-ID	
File-ID	ID of the opened video file (>=1) or 0 at fault condition. At fault condition, the cause can be queried using the function GetLastError().

Description:

The following video formats are supported:

File extension	Format
.mp4	Container: MP4 (MPEG-4 Base Media). Videocodec: MPEG-4 Visual (Advanced Simple@L1).
.avi	Container: AVI (Audio Video Interleaved). Videocodec: MPEG-4 Visual (Simple@L1).
.wmv	Container: WMV (Windows Media). Videocodec: WMV2 (Windows Media Video 8).

When appending an existing file ([Mode]=1, note that the existing file must first be decoded and split up into individual images before new images can be appended. This demands lots of computation resources and memory. Wherever possible, therefore, video files should be generated in one run and any closing of files in the meantime should be avoided.

Every file opened with VFOpen() must be closed again with a call of VFClose().

The ID of the opened file must be specified each subsequent time the file is accessed by a video-file function (VF*). It is valid until VFClose() is called.

A maximum of 10 video file can be open simultaneously.

If calling VFClose() is omitted, this number can be reached quickly. You are then provided with an appropriate error message.

Calling VFClose(0) closes all files open at the present time. This can be useful, for example, if some files remained open while testing sequences in single-step-mode, whose ID's you no longer know.

All open video files are also automatically closed in consequence of the menu command 'File'/'Restart' and upon restarting a sequence in the Editor window (e.g. by the keyboard combination F7).

Multithreading:The File-ID returned by the function is only valid for the current execution thread. Files are automatically closed at the end of the thread (e.g. at the end of a sequence function executed using BEGIN_PARALLEL).

Examples:

A folder contains various files of measured data resulting from an experiment. In the order in which they were generated, all files matching a specific name pattern are loaded, displayed in a fixed configuration, and their curve plots are appended to a video. The video image size matches the curve imgae size on screen; the video format is MP4 in default quality with 2 frames per second.

```
CwSelectMode("variable")
id = VFOpen( "c:\video\test_2019_04_18.mp4", 0, 0, 0, 0, 2)
```

```
IF id > 0
   CwLoadCCV(id, "data.ccv") ;curve configuration with fixed channel name "data"
   CwSelectWindow(id)
   fileNames = FsGetFileNames("c:\test\2019_04_18","ch*.raw", 0, 0, 2)
   FOREACH ELEMENT fileName in fileNames
      fh = FileOpenDSF(fileName, 0)
      IF fh > 0
         data = FileObjRead(fh, 1)
         FileClose(fh)
          VFAppendCwSnapshot(id )
      END
   END
   VFClose(id)
END
```

A Panel has a "Take Snapshot" button. When this button is pressed, a snapshot is taken of the current Panel page and is appended to a video file which is opened upon loading the Panel. When the Panel is closed, the Video-file is also closed.

Event-sequence 'Init':

```
; Opening the video-file in the WMV-format with 2 fps. The video image size matches the size of the first image added.
videoID = VFOpen( "d:\evaluation.wmv", 0, 0, 0, 0, 2)
```

Event-sequence 'Button pressed' for the 'Take Snapshot'-button:

```
; appending the current Panel-image (Page 1) in the original size
VFAppendPanelSnapshot( videoID, 1, 0)
```

Event-sequence 'End':

```
VFClose(videoID)
```

See also:

VFClose, VFAppendPanelSnapshot, VFAppendCwSnapshot, VFAppendRGBData, VFAppendFrame

VibrationFilter

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

Filtering for the evaluation of vibration. The filtering is performed in accordance with a specified frequency-weighting. Subsequently, calculation of the moving exponential RMS value (time weighting) is performed on the results. Finally, resampling is performed, reducing the data volume by a certain factor.

Declaration:

```
VibrationFilter ( InputChannel, FrequencyWeighting, TimeConstant, Reduction ) -> Result
```

Parameter:

InputChannel	The waveform to be filtered, time scaled in seconds.
FrequencyWeighting	Frequency weighting
	10 : Wk, z direction and for vertical recumbent direction, except head. As per ISO 2631-1:1997
	11 : Wd, x and y directions and for horizontal recumbent direction. As per ISO 2631-1:1997
	12 : Wf, motion sickness. As per ISO 2631-1:1997
	13 : Wc, seat-back measurement. As per ISO 2631-1:1997
	14 : We, measurement of rotational vibration. As per ISO 2631-1:1997
	15 : Wj, vibration under the head of a recumbent person. As per ISO 2631-1:1997
	16 : Hx, whole body vibration, standing, sitting position: x- and y-directions. Recumbent position: y- and z-directions. As per DIN 45671-1:1990-09
	17 : Hz, whole body vibration, standing, sitting position: z-direction. As per DIN 45671-1:1990-09
	18 : Hxl, whole body vibration, recumbent position: x-direction. As per DIN 45671-1:1990-09
	19 : Hb, whole body vibration, body posture not specified. As per DIN 45671-1:1990-09
	20 : Hh, hand-arm-vibration, for all directions. As per DIN 45671-1:1990-09
	20 : hand transmitted vibration, weighting filter. As per ISO 7505:1986-05
	21 : Weighting factors for transverse (x, y) vibrations, see table 3. As per withdrawn ISO 2631-1:1985
	22 : Weighting factors for longitudinal (z) vibrations, see table 3. As per withdrawn ISO 2631-1:1985
	23 : Wb (passenger and crew comfort in fixed-guideway transport systems). As per ISO 2631-4:2001
	24 : Wm (human exposure to vibration in buildings). As per ISO 2631-2:2003
	25 : Acceleration input. As per ISO 6954:2000
	26 : Velocity input. As per ISO 6954:2000
	27 : Wh (hand transmitted vibration, weighting filter). As per ISO 5349-1:2001
	28 : Wb (passenger and crew comfort in fixed-guideway transport systems). As per ISO 8041:2005
	29 : Wc (seat-back measurement). As per ISO 8041:2005
	30 : Wd (x and y directions and for horizontal recumbent direction). As per ISO 8041:2005
	31 : We (measurement of rotational vibration). As per ISO 8041:2005
	32 : Wf (Whole body low frequency, motion sickness). As per ISO 8041:2005
	33 : Wh (hand transmitted vibration). As per ISO 8041:2005
	34 : Wj (vibration under the head of a recumbent person). As per ISO 8041:2005
	35 : Wk (z direction and for vertical recumbent direction, except head). As per ISO 8041:2005
	36 : Wm (human exposure to vibration in buildings). As per ISO 8041:2005
	37 : Wb (railway applications, passenger traveling comfort, Z ground, Z seat shell). As per EN 12299:2009
	38 : Wc (railway applications, passenger traveling comfort, X seat backrest). As per EN 12299:2009
	39 : Wd (railway applications, passenger traveling comfort, X ground, Y ground, Y seat shell). As per EN 12299:2009
	40 : Wp (railway applications, passenger traveling comfort, Y ground, phi ground). As per EN 12299:2009

TimeConstant	The time constant for finding the exponentially weighted RMS. Specified in seconds, >= 0.0. E.g. 1.0s for SLOW weighting, 0.125s for FAST weighting. If = 0, no RMS value calculated. In this case, only the filtered signal is returned.
Reduction	Only every n-th spectrum is returned. Use of a large TimeConstant can provide useful reduction of the data volume.
Result	
Result	Filtered waveform

Description:

The sampling time must be small enough so that the relevant bends or resonances of the frequency responce will be reflected.

Examples:

```
wc = VibrationFilter ( Vibration, 10, 0.125, 20 )
```

Wk weighting as per ISO 2631. Moving RMS with TimeConstant 0.125 (FAST). The input signal Vibration is an acceleration and is sampled at 1 ms. Only every 20th value is reflected in the results.

```
a = diff ( v )
hx = VibrationFilter ( a, 16, 0, 1 )
```

Hx weighting as per DIN 45671-1:1990-09 is applied. Since an acceleration is to be treated, but only a velocity was measured, the first derivative is taken. Only filtering is conducted; not the calculation of the RMS value.

See also:

FilterAnalog, FiltLP, dFilt, ExpoRms

VpBackStep

Available in: Professional Edition and above (Video-Kit)

Displays the frame, which is followed by the present frame.

Declaration:

```
VpBackStep ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The video file's last frame is frozen. If the file reaches the start, the function posts an error.

Examples:

In the following example, a file named EXAMPLE.AVI is opened in the Plug-in window. The second frame is displayed and after a 10s pause, the first frame.

```
VpVideoLoad ("C:\VIDEO\EXAMPLE.AVI", 0)
VpSingleStep(0)
sleep(10)
VpBackStep(0)
```

See also:

VpSingleStep

VpContinue

Available in: Professional Edition and above (Video-Kit)

Resumes playback of the video file fom the current position.

Declaration:

```
VpContinue ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

If the video was halted by means of the Pause button or of the Kit function VpPause(), playback can be resumed using the function VpContinue(). Multiple running of this function doesn't stop the video, nor is an error code < 0 posted. If the video wasn't stopped in the middle of playback, but rather is in the resetted state (right after opening or having been stopped), calling VpContinue() has no effect.

Examples:

The following example assumes a file EXAMPLE.AVI located in the folder C:\VIDEO. This file is loaded, played back for 10 seconds, then paused for 10 seconds, after which playback resumes.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
sleep(10)
VpPause(0)
sleep(10)
VpContinue(0)
```

The video freezes for 10 seconds at the frame which was reached after 10 seconds, before playback resumes.

See also:

VpPause

VpDelLink

Available in: Professional Edition and above (Video-Kit)

Deletes a link to a curve window

Declaration:

```
VpDelLink ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video Player for which the function is to be performed. Reserved; always set it to zero. The function can only be applied to the Video Player in the FAMOS Plug-in window.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

If a link to a curve window or the Data Editor exists, it is closed. This doesn't affect the Video-Player's state.

Examples:

In the following example, the file EXAMPLE.AVI from the folder C:\VIDEO is opened and a link is set up to a curve window, which displays the time behavior of the variable Test. The video is played back for 10 seconds, then the linkage is cut.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
id=FileOpenDSF("TEST.DAT", 0)
test = FileObjRead(id, 1)
CvConfig(test, "TEST.CCV")
VpSetLink(test, 0)
VpPlay(0)
sleep(10)
VpDelLink(0)
```

See also:

VpSetLink, VpLinkExists

VpGetAbsStartTime

Available in: Professional Edition and above (Video-Kit)

Returns the date and time when the currently opened video file was created.

Declaration:

`VpGetAbsStartTime ( Player ) -> Time point`

Parameter:

Player	Selection of the Video Player for which the function is to be performed. Reserved; always set it to zero. The function can only be applied to the Video Player in the FAMOS Plug-in window.
Time point	
Time point	Date and time of creation.

Description:

If absolute time statements are used in a linked curve window, it's necessary to take the video recording's absolute starting time into account. By default, the video's creation time is used for this. The function VpGetAbsStartTime() returns the absolute start time set.

The absolute start time is stated as text in the form "tt.mm.yy hh:mm:ss" (Day.Month.Year Hours:Minutes:Seconds).

The function has no error codes. If an error occurs, an empty text is returned.

Examples:

In the following example, the file EXAMPLE.AVI from the folder C:\VIDEO is opened and its creation time is stored in the variable "build".

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
build=VpGetAbsStartTime(0)
```

See also:

VpSetAbsStartTime, VpGetXOffset, VpSetXOffset

VpGetAbsStartTime2

Available in: Professional Edition and above (Video-Kit)

Returns the date and time when the currently opened video file was created.

Declaration:

```
VpGetAbsStartTime2 ( Player ) -> Time point
```

Parameter:

Player	Selection of the Video-Player for which the function is to be performed. Reserved; always set ot 1. The function can only be applied to the video player selected in the current Panel of the FAMOS Data Browser. If there are multiple players available, it is possible to select the player desired by previously calling the function VpSelect().
Time point	
Time point	Creation time (FAMOS time format).

Description:

If the time is stated in absolute terms in a linked curve window, it is necessary to take the video recording's absolute start time into account. By default, the video's creation time is used. The function VpGetAbsStartTime2() returns the absolute start time set.

The return value in the FAMOS time format can be subjected to further processing using the functions of the group '18> Date, Time'.

If an error occurs, an error code (always negative) is returned.

Both the Video Player display element and an open video file can each have an explicitly assigned start time. Depending on its type, the video file's start time can be entered either in the file header or in a parallel configuration file (*.ivi), either directly when it is integrated into the system or subsequently by the user (dialog: 'Video-Properties'). The Video Player-element itself is associated with a corresponding property which can be determined during the Design process.

When determining the operative start time, the display element's setting is dominant, and only if this setting is on 'automatic' will any private start time for the video displayed be observed. If there isn't any such start time either, the resulting start time is the same as the video file's creation time.

Examples:

In the following example, the file EXAMPLE.AVI is loaded from the folder C:\VIDEO and its creation time is queried.

```
VpVideoLoad("C:\VIDEO\BEISPIEL.AVI", 1)
StartTime = VpGetAbsStartTime2(1)
TxStartTime = ZeitInText(StartTime, 0)
```

See also:

VpSetAbsStartTime2, VpGetXOffset, VpSetXOffset

VpGetErrorText

Available in: Professional Edition and above (Video-Kit)

Returns the error test belonging to the specified error code.

Declaration:

```
VpGetErrorText ( Error code ) -> Error text
```

Parameter:

Error code	The number returned by the Video-Kit functions at fault condition.
Error text	
Error text	Error text for the error code.

Description:

Many Video-Player functions return a number as the return value, which in case of an error provides information on the error cause. If the function is successful, a 0 is returned. For every error code there is a text stating the error cause. This text can be obtained using VpGetErrorText().

Specifying the error code's negative sign is not necessary for this function.

Examples:

The following sequence is faulty, because it attempt to skip to a too high position. This assumes that the video file EXAMPLE.AVI is located in the folder C:\VIDEO.

```
VpVideoLoad( "C:\VIDEO\EXAMPLE.AVI", 0)
erg  = VpGetLengthFrames(0)
erg = erg + 1
num = VpSetPosFrames(erg, 0)
txt = VpGetErrorText(num)
```

This sequence sets txt = "Unable to skip to specified position".

See also:

VpGetFileName

Available in: Professional Edition and above (Video-Kit)

Returns the name and directory path of the currently open video file.

Declaration:

```
VpGetFileName ( Player ) -> Name
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Name	
Name	Name and path of current file

Description:

The function VpGetFileName() returs the name and path of the currently open video-file.

This function has no error codes. If an error occurs, an empty text is returned.

Examples:

In the following example, it is asumeed that the video file EXAMPLE.AVI was opened by the user from the folder C:\VIDEO before the sequence was run. After the sequence has been run, the variable "name" contains the string "C:\VIDEO\EXAMPLE.AVI".

```
name = VpGetFileName(0)
```

See also:

VpGetImages

Available in: Professional Edition and above (Video-Kit)

Retrieves one or more frames from the video file currently displayed in the active der Panel Video Player.

Declaration:

```
VpGetImages ( Frame count ) -> Image data
```

Parameter:

Frame count	Specifies the amount of frames to be extracted from the currently loaded video file.
Image data	
Image data	Data set with the requested image data

Description:

The function continually exports the amount of video frames specified by the parameter "Frame count", starting from the current position. If the video is being played back at the moment the function is called, playback is stopped without any warning message and not resumed.

The result data set's data format is "4 Byte unsigned integer". Each pixel is represented by one 4-Byte value, using the "RGB" format. The most significant Byte is always 0, followed by the values for the color components blue, green and red, in the value range 0 ... 255. A 4-Byte vlaue of 0x00000000 corresponds to black; a value of 0x00FFFFFF (decimal => 16777215) corresponds to white. The resulting data set is segmented; the segment length corresponds to the width of a video frame, expressed in pixels. The segment count in a frame corresponds to the video frame's height, expressed in pixels. The first segment represents the bottom line of the frame. The first sample of the first segment represents the bottom left pixel.

If only one frame is exported, then a simple data set is created; otherwise an event-based data set is created in which each event represents one frame. If more frames are specified for export than the amount of frames which follow the current position, then only the still-available number of frames is exported, without any warning message appearing. Thus, the total length of the data set is given by the image width x height (pixels) x count of frames returned.

On error, a data set of length 1 is returned, containing the error code. The associated error text can be retreived using the function "VpGetErrorText".

The function can only be applied to the video player selected in the current Panel of the FAMOS Data Browser. If there are multiple players available, it is possible to select the player desired by previously calling the function VpSelect().

Examples:

In the following example, the file SAMPLE.AVI is loaded into the current Panel video player. VpSetPosFrames() is used to set the video to the position which corresponds to the frame number (100) provided. Subsequently, five frames are extracted from the video file which is currenly loaded in the Video-Player. Extraction of the first of the five frames starts from the current time. In conclusion, the individual color components of the first frame's first pixel (bottom left) are extracted.

```
VpVideoLoad("c:\SAMPLE.AVI",1)
VpSetPosFrames(100,1)
images = VpGetImages(5)
image1 = images[1]
row1 = image1[1]
pixel1 = row1[1]
redValue = BitAnd(Pixel1, 0x000000FF, 32)
greenValue = BitShift(BitAnd(Pixel1, 0x0000FF00, 32), -8, 32)
blueValue = BitShift(BitAnd(Pixel1, 0x00FF0000, 32), -16, 32)
```

See also:

RGB

VpGetLengthFrames

Available in: Professional Edition and above (Video-Kit)

Reterns the number of frames iin the currently open video file.

Declaration:

```
VpGetLengthFrames ( Player ) -> Length
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Length	
Length	Length of the video in frames
	>= 0 : Frame number
	< 0 : Error code

Description:

The function VpGetLengthFrames() returns the number of frames in the currently opened video-file.

Examples:

The file EXAMPLE.AVI is opened from the folder C:VIDEO. Since the count of frames starts at 1, VpGetLengthFrames() can be used to determine and display the last frame.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
last = VpGetLengthFrames(0)
VpSetPosFrames(last, 0)
```

See also:

VpGetLengthSeconds

VpGetLengthSeconds

Available in: Professional Edition and above (Video-Kit)

Returns the length of the currently open video-file, in seconds.

Declaration:

```
VpGetLengthSeconds ( Player ) -> Length
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Length	
Length	The video's length in seconds
	>= 0 : The video's length
	< 0 : Error code

Description:

The function VpGetLengthSeconds() returns the length of the currently open video file in seconds.

Examples:

The file EXAMPLE.AVI from the folder C:\VIDEO is opened, and its length in seconds is retrieved.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
length = VpGetLengthSeconds(0)
```

See also:

VpGetLengthFrames

VpGetPlayRate

Available in: Professional Edition and above (Video-Kit)

Retunrs the currently oped video's playback speed.

Declaration:

```
VpGetPlayRate ( Player ) -> Playback speed
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Playback speed	
Playback speed	The video's playback speed
	>= 0 : Playback speed
	< 0 : Error code

Description:

The currently opened video is played back at the defualt speed. This speed is stated in frames per second, FPS; it is saved in the video file. The playback speed can be changed temporarily; see VpSetPlayRate(). The function VpGetPlayRate() returns returns the currently set playback speed.

Examples:

In the following example, the video EXAMPLE.AVI is opened from the folder C:\VIDEO, and its default playback speed is determined.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
rate = VpGetPlayRate(0)
```

See also:

VpSetPlayRate, VpGetRecordRate, VpSetRecordRate

VpGetPosFrames

Available in: Professional Edition and above (Video-Kit)

Returns the number of the currently displayed frame.

Declaration:

`VpGetPosFrames ( Player ) -> Frame number`

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Frame number	
Frame number	Number of the current frame
	>= 0 : Frame number
	< 0 : Error code

Description:

Videos are saved as (compressed) single frames in a fixed order. In the video-player, the count starts at 1; Frame 1 is thus the curren frame if the position (in seconds) lies in the range from 0.0000s to (video length in seconds/ number of frames). VpGetPosFrames() returns the current frame's number.

Examples:

In the following example the video EXAMPLE.AVI is loaded from the folder C:\VIDEO into the FAMOS Plug-in window, played for 10s and then the current frame number at the current time is determined.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
sleep(10)
position = VpGetPosFrames(0)
```

If the recording speed was 10 frames per second (FPS), the variable "position" holds the value 101 after the sequence has been run.

See also:

VpSetPosFrames, VpGetPosSeconds, VpSetPosSeconds

VpGetPosSeconds

Available in: Professional Edition and above (Video-Kit)

Returns the position of the currently displayed frame.

Declaration:

```
VpGetPosSeconds ( Player ) -> Position
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Position	
Position	Position of the current frame in seconds

Description:

When a video is recorded, the individual frames are saved at a certain recording speed. A frame's position in seconds is then the frame number divided by the recording speed. The functionn VpGetPosSeconds() then returns the current frame's position in seconds.

The function has no error codes. If an error occurs, 0.000 is returned.

Examples:

In the following example, the video EXAMPLE.AVI is opened from the folder C:\VIDEO, played back up to frame number 20, and the frame's position in seconds is then returned.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlaySync(20, 0)
position = VpGetPosSeconds(0)
```

If the recording speed was 10 frames per second (FPS), the variable "position" holds the value 1.9 after the sequence has been run.

See also:

VpSetPosSeconds, VpGetPosFrames, VpSetPosFrames

VpGetRecordRate

Available in: Professional Edition and above (Video-Kit)

Determines the recording speed of the currently loaded Video-file in frames per second.

Declaration:

```
VpGetRecordRate ( Player ) -> Rate (FPS)
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Rate (FPS)	
Rate (FPS)	Recording speed in frames per second
	>= 0 : Recording speed
	< 0 : Error code

Description:

The video file consists of (compressed) single frames, which were recorded at a certain speed. This speed is stated in frames per second (FPS). By default, it is assumed that the recording speed and the playback speed are the same. However, they can take different values, e.g. if high-speed recordings are converted to .AVI-files. Using the function VpGetRecordRate(), the recording speed currently set can be retrieved.

Video Player-element in the Data Browser:

Both the Video Player display element and an open video file can each have an explicitly assigned recording rate. Depending on its type, the video file's recording rate can be entered either in the file header or in a parallel configuration file (*.ivi), either directly when it is integrated into the system or subsequently by the user (dialog: 'Video-Properties'). The Video Player-element itself is associated with a corresponding property which can be determined during the Design process.

When determining the operative rate, the display element's setting is dominant, and only if this setting is on 'automatic' will any private recording rate for the video displayed be observed. If there isn't any such rate either, the resulting rate is the same as the video file's original rate.

Examples:

The file EXAMPLE.AVI is opened from the folder C:\VIDEO, and its recording speed is determined. Once the sequece has been run, the variable rate contains the value specifying the default playback speed.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
rate = VpGetRecordRate(0)
```

See also:

VpSetRecordRate, VpGetPlayRate, VpSetPlayRate

VpGetState

Available in: Professional Edition and above (Video-Kit)

Returns a number which describes the Video-Player's state.

Declaration:

```
VpGetState ( Player ) -> State
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
State	
State	Number describing the Video-Player's state

Description:

The Video-Player's state can change in response to Kit functions or to manipulation of the control elements. The function VpGetState() returns a number reflecting the current state. The following values are possible:

0: No video is open.

1: The Video-Player is reset; the video is at its initial position (newly loaded or Stop).

3: The video is being played back (Play).

5: The video was stopped, without having been played back before. (This state occurs if the picture position after loading or stopping is adjusted manually).

7: The video has been halted (Pause).

Examples:

The function VpGetState() is used to verify whther a video is open in the Plug-in window. If so, i is closed and then EXAMPLE.AVI is loaded from the folder C:\VIDEO and played back.

```
erg = VpGetState(0)
IF erg > 0
   VpVideoClose(0)
END
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
```

Note: This manner of proceeding isn't necessary; see VpVideoLoad().

See also:

VpGetStateText

VpGetStateText

Available in: Professional Edition and above (Video-Kit)

Returns a text describing the Video-Player's current state.

Declaration:

```
VpGetStateText ( Player ) -> State
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
State	
State	State of the Video-Player

Description:

The Video-Player's state can change in response to Kit functions or to manipulation of the control elements. The function VpGetStateText() returns a number text the current state. The following values are possible for the text variables:

"No video opened."

"Video is reset (newly opened or stopped)."

"Video being played back (Play)."

"Video is reset and halted (New/Stop or Pause)."

"Video has been halted (Pause)."

The function has no error codes. If an error occurs, an empty text is returned.

Examples:

The video EXAMPLE.AVI in the folder C:\VIDEO is opened in the Plug-in window and played for 30 seconds. Then the function VpGetStateText() is used to check whether or not the video is still playing.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
sleep(30)
erg = VpGetStateText(0)
```

Depending on the length of the video, the variable erg contains either the text "Video being played back (Play)." or "Video was halted (Pause).".

See also:

VpGetState

VpGetXOffset

Available in: Professional Edition and above (Video-Kit)

Returns the time offset of the video file currently opened, in seconds.

Declaration:

```
VpGetXOffset ( Player ) -> Time offset
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Time offset	
Time offset	The video's time offset in seconds.

Description:

By default, playback of a video file begins at the time 0.000s. The function VpGetXOffset() returns the video's starting time. This relative starting time determines the position (in seconds) displayed for the video's first frame.

The function has no error codes. If an error occurs, the value 0.000 is returned.

Video Player-element in the Data Browser:

Both the Video Player display element and an open video file can each have an explicitly assigned offset. Depending on its type, the video file's offset can be entered either in the file header or in a parallel configuration file (*.ivi), either directly when it is integrated into the system or subsequently by the user (dialog: 'Video-Properties'). The Video Player-element itself is associated with a corresponding property which can be determined during the Design process.

When determining the operative offset, the display element's setting is dominant, and only if this setting is on 'automatic' will any private offset for the video displayed be observed. If there isn't any such offset either, the resulting offset is 0.

Examples:

The file EXAMPLE.AVI from the folder C:\VIDEO is opened; after running the sequence, the file's relative start time is reflected in the variable xoff.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
xoff=VpGetXOffset(0)
```

See also:

VpSetXOffset, VpGetAbsStartTime, VpSetAbsStartTime

VpLinkExists

Available in: Professional Edition and above (Video-Kit)

States whether a linkage to a curve window or to the FAMOS Data Editor exists.

Declaration:

```
VpLinkExists ( Player ) -> Value
```

Parameter:

Player	Selection of the Video Player for which the function is to be performed. Reserved; always set it to zero. The function can only be applied to the Video Player in the FAMOS Plug-in window.
Value	
Value	1, if linkage exists, else 0.

Description:

States whether a linkage to a curve window or to the FAMOS Data Editor exists. If a linkage exists, the function VpLinkExists() returns the value 1, otherwise 0.

The function doesn't have any error codes. When an error occurs, it returns 0.

Examples:

In the following example, the system checks whether there is a link to a curve window; in this case it is deleted.

```
link = VpLinkExists(0)
IF link > 0
   VpDelLink(0)
END
```

See also:

VpSetLink, VpDelLink

VpPause

Available in: Professional Edition and above (Video-Kit)

Pauses the video file at the current position.

Declaration:

```
VpPause ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

Playback of the video file is interrupted at the current position; the current frame is frozen. In contrast to the use of the buttons, multiple runs of the function function VpPause() doesn't cause the videos to be resumed; even an error code < 0 is not posted.

Examples:

In the following example, a Panel having a Video Player is opened, the video EXAMPLE.AVI is opened, played for 10 seconds and then paused.

```
DbLoadPanel(" c:\VIDEO\VIDEO.PANEL", 0)
VpVideoLoad("C:\VIDEO\BEISPIEL.AVI", 1)
VpPlay(1)
sleep(10)
VpPause(1)
```

The video halts at the frame reached after 10 seconds.

See also:

VpContinue

VpPlay

Available in: Professional Edition and above (Video-Kit)

Plays the open Video file.

Declaration:

```
VpPlay ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The video file currently open is played back. For this purpose, the setting for the playback speed is observed. The settings for the start time and the recording speed are entered in the display of the position (in seconds). Playback is performed asynchronously, which means that is even possible during playback to operate the program.

How the function behaves depends on the Player selected. With the Plug-in, playback continues as of the current position. With the Player in the Panel, playback always starts at the beginning of the video; for this case, use VpContinue() to resume playback from the current position.

Note: Calling the function VpPlay(0) corresponds to clicking the Play-button.

Examples:

In the following example, a file named EXAMPLE.AVI is loaded from the folder C:\VIDEO and played back from the first frame at the default playback speed.

```
VpVideoLoad ("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
```

In the next example, a Panel having a Video Player is opened in the Data Browser, the specified video is loaded and playback is started.

```
DbLoadPanel(" c:\VIDEO\VIDEO.PANEL", 0)
VpVideoLoad ( "C:\VIDEO\BEISPIEL.AVI", 1)
VpPlay(1)
```

In the following example, sleep() is used, in order to delay execution of VpPause() for 10 seconds. However, playback cannot be interrupted during these 10 seconds, since the sleep()-command prevents it.

```
VpVideoLoad ( "C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
sleep(10)
VpPause(0)
```

See also:

VpPlaySync

VpPlaySync

Available in: Professional Edition and above (Video-Kit)

Plays the video back until the specified position has been reached

Declaration:

`VpPlaySync ( Frame number, Player ) -> Error code`

Parameter:

Frame number	Frame number at which playback is to end
Player	Selection of the Video-Player for which the function is to be performed. Reserved; it must always be set to zero. The function can only be applied to the Video Player in the FAMOS Plug-in window.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The video file currently open is played back. This takes the current position and the playback speed into account. Playback is synchronous, i.e. during playback it's not possible to issue other commands; even the display and the linkage to the curve window aren't updated during playback. VpPlaySync() expects the specification of a position (as frame number), at which playback is to be ended.

Examples:

In the following example, a file named EXAMPLE.AVI is loaded from the folder C:\VIDEO and played back from the first frame at the default playback speed. Playback cannot be interrupted until the last frame has been reached.

```
VpVideoLoad ("C:\VIDEO\EXAMPLE.AVI", 0)
length = VpGetLengthFrames(0)
VpPlaySync(length, 0)
```

See also:

VpPlay

VpSelect

Available in: Professional Edition and above (Video-Kit)

Selection of the Video Player in the current Data Browser Panel, upon which the subsequent calls to Video-Kit functions are to apply.

Declaration:

`VpSelect ( TxVideoplayerName )`

Parameter:

TxVideoplayerName	Name of the Video Player element to be selected.

Description:

The Video Kit's functions have a parameter which specifies whether the Video Player in the integrate FAMOS Plug-in (0) or a Video Player in the current Panel (1) is meant. If the Panel has multiple Video Player elements, this function is used to select the element affected. If this functino is not called, the first Video Player found is affected.

The Video Player can be selected either in the form [PageName].[ElementName] or only by the element name. If the page is not explicitly specified, the system searches for the Video Player as follows:

- If the function is called within an event sequence, and the event can be assigned to a page (e.g. a button's 'Pressed'-event), then the system searches for Video Player with the specified name on this page.
- Otherwise the search is performed on the active (visible) page.

If no panel is loaded in the data browser, the current panel is in design mode or there is no video player with the specified name, the function aborts with an error message.

Multithreading: The functions of the Video kit can be called anywhere and have a global effect. The Player selected here is therefore valid for all execution threads.

Examples:

On a Panel page, there are 2 Video Players and one button. When the button is pushed, 2 video files are opened and played back.

The 'Start'-button's Event-Sequence 'Pressed'

```
VpSelect( "Player1")
VpVideoLoad( "c:\video\sample1.avi", 1)
VpPlay(1)
VpSelect( "Player2")
VpVideoLoad( "c:\video\sample2.avi", 1)
VpPlay(1)
```

VpSetAbsStartTime

Available in: Professional Edition and above (Video-Kit)

Sets the time and date of creation for the currently open video-file.

Declaration:

```
VpSetAbsStartTime ( Time point, Player ) -> Error code
```

Parameter:

Time point	Creation time
Player	Selection of the Video Player for which the function is to be performed. Reserved; always set it to zero. The function can only be applied to the Video Player in the FAMOS Plug-in window.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

If absolute time statements are used in a linked curve window, it's necessary to take the video recording's absolute starting time into account. By default, the video's creation time is used for this. With the function VpSetAbsStartTime(), the video recording's start date and time can be set. The format is a text in the form "tt.mm.yy hh:mm:ss" (Day.Month.Year Hours:Minutes:Seconds).

Examples:

The file EXAMPLE.AVI from the folder C:\VIDEO is opened, its absolute start time is set to January 1, 2000, 0:00 hours.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpSetAbsStartTime("01.01.00 00:00:00", 0)
```

See also:

VpGetAbsStartTime, VpGetXOffset, VpSetXOffset

VpSetAbsStartTime2

Available in: Professional Edition and above (Video-Kit)

Sets the time and date of creation for the currently open video-file.

Declaration:

```
VpSetAbsStartTime2 ( Time point, Player ) -> Error code
```

Parameter:

Time point	Creation time point (FAMOS-time format). -1 for automatic.
Player	Selection of the Video-Player for which the function is to be performed. Reserved; always set ot 1. The function can only be applied to the video player selected in the current Panel of the FAMOS Data Browser. If there are multiple players available, it is possible to select the player desired by previously calling the function VpSelect().
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

If the time readout in a linked curve window indicates absolute time, it is necessary to take the video recording's absolute start time into account. By default, the video's creation time is used. Using the function VpSetAbsStartTime2(), it is possible to adapt video recording's start date and time.

The time specification in the FAMOS time format can be generated using functions belonging to the group '18> Date, Time'.

The function sets the Video Player's corresponding property, which is dominant over any already existing private video file property (saved in the file header or in a parallel *.ivi configuration file). In order to reset the Video Player property to 'automatic' (and thus to put any video file setting back into effect), enter the value -1.

Examples:

The file EXAMPLE.AVI from the folder C:\VIDEO is opened, its absolute start time is set to January 1, 2000, 0:00 hours.

```
VpVideoLoad("C:\VIDEO\BEISPIEL.AVI", 1)
time = TimeJoin( 1, 1, 2000, 0, 0, 0)
VpSetAbsStartTime2(time, 1)
```

See also:

VpGetAbsStartTime2, VpGetXOffset, VpSetXOffset

VpSetLink

Available in: Professional Edition and above (Video-Kit)

Establishes a link to an existing curve window or the FAMOS Data Editor.

Declaration:

```
VpSetLink ( Reference data set, Player ) -> Error code
```

Parameter:

Reference data set	Channel with which the link is established
Player	Selection of the Video Player for which the function is to be performed. Reserved; always set it to zero. The function can only be applied to the Video Player in the FAMOS Plug-in window.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

Any given data set whose x-axis can be interpreted as the time axis can be displayed in a curve window or in the FAMOS Data Editor. Using the function VpSetLink() it's possible to establish a link between a video and this reference data set. As a result, the video and the data set can be synchronized, i.e. playback of the video or dragging of the position cursor in the x-direction leads to corresponding adjustment of the position in the linked window. The starting time in such a case is determined by the linking window. If absolute timedisplay is selected for the data set, this reflected in the video's playback. The relative starting time (x-offset) is observed in any case.

Examples:

The file EXAMPLE.AVI from the folder C:\VIDEO is opened and linked with a variable named "test", which is generated from the waveform TEST.DAT with a curve window configuration TEST.CCV. Then, the video is played; the curve window displays the assciated measured values.

```
id=FileOpenDSF("TEST.DAT", 0)
test = FileObjRead(id, 1)
CvConfig(test, "TEST.CCV")
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpSetLink(test, 0)
VpPlay(0)
```

See also:

VpDelLink, VpLinkExists

VpSetPlayRate

Available in: Professional Edition and above (Video-Kit)

Sets the playback speed to a different value.

Declaration:

```
VpSetPlayRate ( Playback speed, Player ) -> Error code
```

Parameter:

Playback speed	Sets the playback speed to a different value.
Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The currenly open video is played back at the default speed. This speed is stated in frames per second frames per second, FPS; its stored in the video file. With the function VpSetPlayRate(), the playback speed can be set to another value. This value is valid as long as the video is open; it isn't saved.

The maximum playback speed is 200-times the default value. However, this value depends on the video- or audio-CODEC; with most videos having audio, a maximum playback speed of twice the pre-set value is possible.

Note: Entering a zero for th edesired playback speed sets it to the default speed.

Examples:

In the following example, the video EXAMPLE.AVI is opened from the folder C:\VIDEO and its playback speed is set to the value 5 FPS.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpSetPlayRate(5, 0)
VpPlay(0)
```

If the file's default playback speed is 10 FPS, for instance, then after the sequence has been run, it is played back at half speed.

See also:

VpGetPlayRate, VpGetRecordRate, VpSetRecordRate

VpSetPosFrames

Available in: Professional Edition and above (Video-Kit)

Displays the frame with the number given.

Declaration:

```
VpSetPosFrames ( Frame number, Player ) -> Error code
```

Parameter:

Frame number	The number of the frame to whose position the system is to skip
Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

Videos are saved as (compressed) single frames in a fixed order. In the video-player, the count starts at 1; Frame 1 is thus the curren frame if the position (in seconds) lies in the range from 0.0000s to (video length in seconds/ number of frames). VpSetPosFrames() sets the video to the position which corresponds to the frame number given. The video is halted if it was being played back at the time the command is run.

Examples:

In the following example, a Panel having a Video Player is opened, the video EXAMPLE.AVI from the folder C:\VIDEO is played back for 10s, after which the frame with the number 5 is displayed.

```
DbLoadPanel("C:\VIDEO\VIDEO.PANEL", 0)
VpVideoLoad("C:\VIDEO\BEISPIEL.AVI", 1)
VpPlay(1)
sleep(10)
VpSetPosFrames(5, 1)
```

See also:

VpGetPosFrames, VpGetPosSeconds, VpSetPosSeconds

VpSetPosSeconds

Available in: Professional Edition and above (Video-Kit)

Displays the frame which is the current frame at the specified time.

Declaration:

```
VpSetPosSeconds ( Seconds, Player ) -> Error code
```

Parameter:

Seconds	Position of the frame to be displayed, in seconds.
Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

When the video is recorded, the individual frame are saved at a certain recording speed. The position of a frame in seconds is then the frame number divided by the recording speed. The function VpSetPosSeconds() sets the video at the frame which was recorded at the specified time.

Examples:

In the following example, a Panel having a Video Player is opened, the video EXAMPLE.AVI is loaded from the folder C:\VIDEO and it is played back as of the frame recorded at 10s.

```
DbLoadPanel("C:\VIDEO\VIDEO.PANEL", 0)
VpVideoLoad("C:\VIDEO\BEISPIEL.AVI", 1)
VpSetPosSeconds(10, 1)
VpContinue(1)
```

See also:

VpGetPosSeconds, VpGetPosFrames, VpSetPosFrames

VpSetRecordRate

Available in: Professional Edition and above (Video-Kit)

Sets the recording speed of the currently open video-file in frames per second.

Declaration:

```
VpSetRecordRate ( Rate, Player ) -> Error code
```

Parameter:

Rate	Original speed at which the video is recorded.
Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The video file consists of (compressed) single frames recorded at a certain speed. This speed is stated in frames per second (frames per second, FPS). By default, it is assumed that the recording speed and the playback speed are the same. However, they can also take different values, e.g. if high-speed recordings are converted to .AVI-files. With the help of the function VpSetRecordRate(), the recording speed can be set. Time specifications for the position and length are adjusted automatically. However, the playback speed is independent of the recording speed.

Video Player-element in the Data Browser:

The function sets the Video Player's corresponding property, which is dominant over any already existing private video file property (saved in the file header or in a parallel *.ivi configuration file) or over the original data recording rate. In order to reset the Video Player property to 'automatic' (and thus to put any video file setting back into effect), enter the value 0.

Examples:

The file EXAMPLE.AVI is opened from the folder C:\VIDEO and its recording speed is set to 1000 frames per second.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpSetRecordRate(1000, 0)
length=VpGetLengthSeconds(0)
```

Assuming that the file consists of 1000 frames, then the variable length has the value 1 after execution of the sequence. If the playback speed is set to 10 FPS, complete playback of the video lasts 100s. Thus, playback speed versus recording speed is 1/100.

See also:

VpGetRecordRate, VpGetPlayRate, VpSetPlayRate

VpSetXOffset

Available in: Professional Edition and above (Video-Kit)

Sets the currently opened video file's time offset in seconds.

Declaration:

```
VpSetXOffset ( Rate, Player ) -> Error code
```

Parameter:

Rate	The video's starting time stated in seconds
Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

By default, playback of a video file begins at the time 0.000s. With the help of the function VpSetXOffset(), the time can be set to a different value.

Video Player-element in the Data Browser:

The function sets the Video Player's corresponding property, which is dominant over any already existing private video file property (saved in the file header or in a parallel *.ivi configuration file). In order to reset the Video Player property to 'automatic' (and thus to put any video file setting back into effect), enter the value 1e35.

Examples:

The file EXAMPLE.AVI from the folder C:\VIDEO is opened and linked with a variable named "test", which is generated from the waveform TEST.DAT with a curve window configuration TEST.CCV. Um In order to compensate for a time offset between the starting times of these two files, the video's relative starting time is set to 1s, so that the video's first frame coincides with the measurement value recorded after one second.

```
id=FileOpenDSF("TEST.DAT", 0)
test = FileObjRead(id, 1)
CvConfig(test, "TEST.CCV")
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpSetLink(test, 0)
VpSetXOffset(1, 0)
```

See also:

VpGetXOffset, VpGetAbsStartTime, VpSetAbsStartTime

VpSingleStep

Available in: Professional Edition and above (Video-Kit)

Displays the frame following the current frame.

Declaration:

```
VpSingleStep ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The video-file's next frame is frozen on screen. If the file has reached the end, the function posts an error.

Examples:

In the following example, a file named EXAMPLE.AVI is opened in the Plug-in window. The second frame is displayed and after a 10s pause, the third frame.

```
VpVideoLoad ("C:\VIDEO\EXAMPLE.AVI", 0)
VpSingleStep(0)
sleep(10)
VpSingleStep(0)
```

See also:

VpBackStep

VpStop

Available in: Professional Edition and above (Video-Kit)

Halts the video file and sets it back to its start.

Declaration:

```
VpStop ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The video is halted irrespective of its current state; the first frame is shown. Settings for the recording and playback speed, as well as the relative and absolute start times (in the info-dialog) remain intact.

Examples:

In the following example, the file EXAMPLE.AVI is assumed to be located in the folder C:\VIDEO. This file is opened in the Plug-in window, played for 10 seconds, and then set back to the start.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
sleep(10)
VpStop(0)
```

See also:

VpVideoClose

Available in: Professional Edition and above (Video-Kit)

Closes the video file which is currently open..

Declaration:

```
VpVideoClose ( Player ) -> Error code
```

Parameter:

Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The Video-Player can have only one video file open at any time. This file is then closed using VpVideoClose(), and the control elements (except for the button for loading a video file) are returned to the active state.

Examples:

The following example assumes that the folder C:\VIDEO contains a file with the name EXAMPLE.AVI. This file is opened in the Plug-in window, played for 10 seconds and then closed.

```
VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
VpPlay(0)
sleep(10)
VpVideoClose(0)
```

See also:

VpVideoLoad

VpVideoLoad

Available in: Professional Edition and above (Video-Kit)

Loads a video file

Declaration:

```
VpVideoLoad ( Filename, Player ) -> Error code
```

Parameter:

Filename	Name and path of the video file to be loaded
Player	Selection of the Video-Player, for which the function is to be performed
	0 : Video Player in the FAMOS Plug-in window
	1 : Video Player in the FAMOS-Data Browser's current Panel. If there are multiple Players, it is possible to previously call the function VpSelect() to set the desired Player.
Error code	
Error code	Success of the function (optional).
	0 : Function executed successfully
	< 0 : Error code

Description:

The parameter "Filename" expects the name and the path of a file which the Video-Player is able to play back. The ending (.avi, .mpg, .mov or other) must be contained in the name. Then the Video-Player opens this file and skips to the start of the video. This can be playd back using the control elements or other Video-Kit functions.

If another video file is open at the time when VpVideoLoad() is called, the file is closed.

Multithreading: The functions of the Video kit can be called anywhere and have a global effect. So the video loaded here is valid for all Execution threads.

Examples:

The following example assumes that in the path C:\VIDEO ther is an AVI file named EXAMPLE.AVI. This file is opened in the Video Plug-in.

```
erg = VpVideoLoad("C:\VIDEO\EXAMPLE.AVI", 0)
```

In the following example, a Panel having a Video Player is opened, a video is loaded to the Player and playback is started.

```
DbLoadPanel("C:\VIDEO\VIDEO.PANEL", 0)
VpVideoLoad("C:\VIDEO\BEISPIEL.AVI", 1)
```

See also:

VpVideoClose

WFTLOAD

Loads a Nicolet WFT-file (Copyright 1988 Nicolet Instrument Corporation).

Declaration:

```
WFTLOAD SvSegment SvTimeBase Filename VariableName
```

Parameter:

SvSegment	Segment to be loaded (1..1024)
SvTimeBase	Time base to be loaded (1..3)
Filename	Name of the file to be loaded
VariableName	Variable in to which the file excerpt is entered

Description

A WFT file from the Nicolet Instrument Corporation is loaded in FAMOS. A WFT file can contain from 1 to 1024 segments and 1 to 3 time bases. The parameters "Segment" and "TBase" can be used to read a defined segment and time base. The parameter "Segment" can assume values between 1 and 1024, the parameter "TBase" between 1 and 3.

The selected file excerpt is loaded with the designation specified under [VariableName].

For additional file formats belonging to the company Nicolet, such as "Nicolet Team" and "Nicolet Frequency Domain (WFF)", external import filters are available. Such files can be loaded with the function FileLoad() or FileOpenFAS():

```
FileLoad("c:\data\channel1.wff", "Nicolet_FreqDomain.FAS", 0)
```

- The filename can be a complete pathname included folder and filename extension, but can also be specified without either. Then, the system searches for the file in the folder from which FAMOS loads data. The extension is selected automatically.
- The filename can also be specified to contain quotation marks. This can be necessary, if, for instance, the path contains spaces.

Examples:

```
WFTLOAD 4 1 WAVE0001 seg4
WFTLOAD 4 1 c:\data\WAVE0001 seg4
```

In both cases, the first time base of the fourth segment is read from the file "WAVE0001.WFT" and entered into imc FAMOS under the designation "seg4".

```
seg = 4
bas = 1
WFTLOAD seg bas WAVE01 data
```

Two variables define the desired range of the file "WAVE01". It is entered under the designation "Data".

```
WFTLOAD 4 1 "c:\My Data Files\WAVE0001" seg4
```

The pathname contains spaces and must therefore be written inside of quotation marks.

See also:

FileLoad, FileOpenFAS, LDIR

WHILE

Conditonal loop. The following instructions are run cyclically as long as the expression specified here returns a value > 0.

Declaration:

```
WHILE Condition
```

Parameter:

Condition	As long as the condition is met (evaluation returns a value >0), the instructions of the subsequent block are repeated.

Description

The end of the loop is denoted using the command END.

As the condition, it is possible to specify, for example, a single value variable or a complex expression using logical operators (AND, OR..) and/or comparison operators (<, =, ...).

In the loop, it is possible to use the commands BREAK and CONTINUE in order to interrupt execution of the instructions prematurely.

Instead of a WHILE loop, in many applications it is easier to use a FOR or FOREACH-loop.

Examples:

Smoothes a data set until its standard deviation is no longer greater than 0.2.

```
WHILE StDev(data) > 0.2
   Data = Smo5(data)
END
```

The first text object located in a file containing multiple data objects is read out.

```
fh = FileOpenDSF("test.dat", 0)
IF fh > 0
   n = FileObjNum?(fh)
   i = 1
   WHILE i <= n
      IF FileObjType?(fh, i) = 2
         text = FileObjRead(fh, i)
         BREAK ; exit loop
      END
      i = i + 1
   END
   FileClose(fh)
END
```

Tip: The task intended is more elegantly accomplished by means of a FOR loop.

See also:

FOREACH, FOR

XDel

Specifies a data set's increment in the x-direction (Delta-X, sampling interval).

Declaration:

```
XDel ( Data, SvXDelta ) -> Result
```

Parameter:

Data	Data set whose x-increment is to be set
SvXDelta	New Delta-X (>0)
Result	
Result	Data set copy with new Delta-X.

Description:

A copy of the input data is generated and the specified x-increment is entered. All other numbers and characteristic values remain unaffected.

Delta-X represents the difference between the x-coordinates of adjacent points in the data set.

For measurement data, which have been sampled equidistant over time, this corresponds to the **sampling rate**.

- The new Delta-X should have the data set's x-unit.
- The size of the new Delta-X should not be too many orders of magnitude less than the x-offset. Otherwise, the resolution may not be sufficient to represent differences between the x-coordinates of the data set's data points.

Examples:

The sampling interval of a data set which was imported in the form of ASCII data without time tbase information is set to 2ms:

```
SetUnit(NDdata,0, "s")
NDdata = XDel(NDdata, 2e-3)
```

See also:

XDel?, XOff, Leng, XDELTA

XDel?

Determines a data set's increment in x-direction (Delta-X, sampling interval).

Declaration:

```
XDel? ( Data ) -> SvXDelta
```

Parameter:

Data	Data set whose Delta-X is to be determined
SvXDelta	
SvXDelta	Delta-X

Description:

Delta-X represents the difference between the x-coordinates of adjacent points in the data set.

For measurement data, which have been sampled equidistant over time, this corresponds to the **sampling rate**.

- The result has the x-unit of the data set passed.

Examples:

The duration of a data set is the product of its length and sampling interval:

```
duration = XDel?(NDdata) * Leng?(NDdata)
```

The sampling frequency is the reciprocal of the sampling interval:

```
freq = 1 / XDel?(NDdata)
```

See also:

XDel, XOff?, Leng?

XDELTA

Sampling interval; set x-Delta

Declaration:

```
XDELTA VariableName SvDeltaX
```

Parameter:

VariableName	Name of the variable whose x-axis is to be re-apportioned.
SvDeltaX	Disance between 2 data points in x-units

Description

The command XDELTA is obsolete; instead in newly created sequences, the function XDel() is used.

The divisions of the x-axis (sampling intervals) are redefined.

Examples:

```
XUNIT Voltage s
XDELTA Voltage 60
```

The variable "Voltage" first receives the x-unit "s", then the distance between two points is defined to "60", so that the new sampling rate is 60 s .

See also:

XDel, XDel?, XOff

XlBuildA1Ref

Scope: Excel remote control

Forms a cell-/range reference in the A1-style from the column and row numbers.

Declaration:

```
XlBuildA1Ref ( Row, Column, RangeHeight, RangeWidth [, ReferenceCell] ) -> TxA1Reference
```

Parameter:

Row	Row number (1..)
Column	Column number (1..)
RangeHeight	Height (1..) of range
RangeWidth	Width (1..) of range
ReferenceCell	Cell reference for the reference cell in the A1-style. If this is specified, then [Row] and [Column] are not interpreted as absolute specifications but rather as offsets relative to this cell. (optional)
TxA1Reference	
TxA1Reference	Cell/range-reference in A1-style.

Description:

Some of this kit's funtions require specification of a cell with a reference in the style 'A1' (or of a range in the form 'A1:B2'), where the letters represent the column (A=1, Z = 26, AA = 27 etc.) and the numbers represent the row number.

With this function, a cell reference can be constructed from the column-/row-number and the size of the desired range.

If a reference cell is specified as the 5th parameter, then the colummn and row numbers are interpreted as offsets to this cell.

Examples:

```
ref = XlBuildA1Ref( 1, 1, 1, 1)           ; Result: "A1"
ref = XlBuildA1Ref( 3, 2, 1, 1)           ; "B3"
ref = XlBuildA1Ref( 1, 2, 100, 1)         ; "B1:B100"
ref = XlBuildA1Ref( 2, 2, 255, 100)       ; "B2:CW256"
ref = XlBuildA1Ref( 65536, 256, 1, 1)     ; "IV65536"    (maximum table size in XLS-format)
ref = XlBuildA1Ref( 1048576, 16384, 1, 1) ; "XFD1048567" (maximum table size in XLSX-format)
ref = XlBuildA1Ref( 1, 1, 1, 1, "A1")     ; "B2"
ref = XlBuildA1Ref( 0, -1, 1, 1, "B2")    ; "A2"
```

A new Excel table is created and all channels belonging to a data group 'MyGroup' are transferred point-by-point into a column's cells. Subsequently, the output format for all filled cells is set to a maximum of 2 decimal places (it is assumed that all channels are of the same length).

```
XlWbNew("")
ChanNum =  GrChanNum?(MyGroup)
FOR I = 1 TO ChanNum
   ref = XlBuildA1Ref( 1, I, 1, 1)
   XlSetValues( ref, 0, MyGroup:[I], 0)
END

count = leng?(MyGroup:[1])
range = XlBuildA1Ref( 1, 1, count, ChanNum)
XlSetCellFormat( range, "0.##")
XlWbSave( "c:\results\report", 0)
XlQuit()
```

In a table, a time track is created and formatted accordingly for output.

```
time = XlCreateTimeLine(channel1, 1, 0)
XlSetValues( "A1", 0, time, 1)
Count = Leng?(time)
Range = XlBuildA1Ref(1, 1, count, 1)
XlSetCellFormat(Range, "dd.mm.yy hh:mm:ss.0")
```

See also:

XlSetText, XlSetValue, XlGetValues

XlCellMerge

Scope: Excel remote control

Joins the specified cells in the active worksheet.

Declaration:

`XlCellMerge ( TxRange )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.

Description:

The specified cells in the active worksheet are merged. If the cells are already filled, EXCEL posts a message. In order to avoid interrupting the sequence run, the cells should be empty before merging.

Examples:

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = MatrixInit( 10, 10, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 3, I+2, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ; merging cells
   XlCellMerge("A1:B14")
   XlCellMerge("C1:L2")
   XlCellMerge("C13:L14")
   XlCellMerge("M1:N14")
   ; drawing red crosses right and left in the merged cells
   XlSetBorderStyle("A1:B14", 48, "continous")
   XlSetBorderColor("A1:B14", 48, RGB(255,0,0))
   XlSetBorderStyle("M1:N14", 48, "continous")
   XlSetBorderColor("M1:N14", 48, RGB(255,0,0))
END
```

XlCreateTimeLine

Scope: Excel remote control

Constructs an explicit time track for an equidistant data set for subsequent transfer to Excel.

Declaration:

```
XlCreateTimeLine ( DataSet, Type, Reserved ) -> TimeTrack
```

Parameter:

DataSet	Data set to be generated for the time track
Type	
	0 : Relative timetrack
	1 : Absolute time track
Reserved	Reserved parameter, always 0
TimeTrack	
TimeTrack	Constructed time track

Examples:

A new Excel file with a worksheet is created, and a data set is transferred into the table. The first column is configured as a time column with absolute Date/Time and 1ms resolution; the second column contains the data set's actual values.

The file created in this way is then saved.

```
XlWbNew("")
time = XlCreateTimeLine(channel1, 1, 0)
XlSetValues( "A1", 0, time, 1)
Count = Leng?(time)
Range = XlBuildA1Ref(1, 1, count, 1)
XlSetCellFormat(Range, "dd.mm.yy hh:mm:ss.0")
XlSetValues( "B1", 0, channel1, 0)
XlWbSave( "c:\results\report", 0)
XlQuit()
```

Like above, however here the time column is configured in relative terms (seconds since triggering). Output begins from the 2nd row; the data set's trigger time is entered in the cell A1.

```
XlSetValue("A1", Time?(Channel1), 1)
XlSetCellFormat("A1", "dd.mm.yy hh:mm:ss.0")
time = XlCreateTimeLine(channel1, 0, 0)
XlSetValues( "A2", 0, time, 0)
Count = Leng?(time)
Range = XlBuildA1Ref(2, 1, count, 1)
XlSetCellFormat(Range, "0.000")
XlSetValues( "B2", 0, channel1, 0)
```

See also:

XlSetValues

XlFind

Scope: Excel remote control

Searches through the current sheet for text.

Declaration:

`XlFind ( TxSearch, SvWhere, SvWholeCell, SvUpperLower, SvDirection ) -> Search results`

Parameter:

TxSearch	The text for which to search
SvWhere	Specifies in which content to search.
	0 : Values
	1 : Formulas
	2 : Comments
SvWholeCell	Specifies whether the entire cell content is to be compared.
	0 : The text to find must be contained in the cell content.
	1 : The text to find must exactly match the row content.
SvUpperLower	Groß-/Kleinschreibung beachten
	0 : Groß-/Kleinschreibung NICHT beachten
	1 : Groß-/Kleinschreibung beachten
SvDirection	Sets the search direction.
	0 : Column-by-column
	1 : Row-by-row
Search results	
Search results	Text box containing all locations found. The hits are returned in 'A1'-reference style.

Description:

This command replicates the functionality of the Excel-dialog "Find...".

If the current selection comprises more than one cell, the search is limited to the selected region. Otherwise, the entire table sheet is searched.

If the text to be found is not located, an empty text array is returned.

Examples:

A workbook is loaded and all table sheets are searched for the text 'ENGINE_0124'. In case of success, the value in the adjacent cell to the right is read.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   sheetcount = XlSheetGetCount()
   FOR i = 1 TO sheetcount
      XlSheetActivate(i)
      hits = XlFind("ENGINE_0134", 0, 1, 0, 0)
      IF TxArrayGetSize(hits) = 1
         cell = XLBuildA1Ref(0, 1, 1, 1, hits[1])
         val = XlGetValue(cell, 0)
         BREAK
      END
   END
   XlQuit()
END
```

A workbook is loaded and in the 2nd column of the table sheet 'Table', the system searches for all cells containing the text '_Engine_'. The entire content of the cells flound is entered into a new text array.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   XlSheetActivate("Table2")
   XlSelectRange("B:B")
   hits = XlFind("_Engine_", 0, 0, 0, 0)
   foundNames = TxArrayCreate(0)
   FOR i = 1 TO TxArrayGetSize(hits)
      foundNames[i] = XlGetText(hits[i])
   END
   XlQuit()
END
```

See also:

XlGetText, XlSelectRange, XlBuildA1Ref

XlGetSelectedRange

Scope: Excel remote control

Excel: Get selected region

Declaration:

```
XlGetSelectedRange ( ) -> TxRange
```

Parameter:

TxRange	
TxRange	Returns the currently selected cell-region as an 'A1'-reference. Empty text, if no cell region is selected.

Description:

This function affects the current sheet of the active workbook.

Examples of [TxRange]-specifications in the 'A1'-style:

Selection of an individual cell	"C2"	Cell C2 (3rd column, 2nd row)
Selection of a contiguous region	"C2:E3"	Cells C2 through E3 (3rd through 5th columns, 2nd and 3rd rows)
Selection of a column	"C:C"	The 3rd column is selected.
Selection of multiple columns	"C:E"	The 3rd, 4th and 5th columns are selected.
Selection of a row	"2:2"	The 2nd row is selected.
Selection of multiple rows	"2:4"	The 2nd, 3rd and 4th rows are selected.
Multi-selection	"A1:A3;C4"	The cells A1,A2,A3 And C4 are selected.

As demonstrated in the last example, it is also possible for multiple references (separated by semicolons) to be returned if a non-contiguous region is selected.

Examples:

The currently selected cell is moved down by one row:

```
sel = XlGetSelectedRange()
sel = XlBuildA1Ref(1, 0, 1, 1, sel)
XlSelectRange(sel)
```

See also:

XlSetSelectedRange, XlBuildA1Ref

XlGetText

Scope: Excel remote control

Gets the content of the specified cell in the active worksheet as a text.

Declaration:

```
XlGetText ( TxCell ) -> TxContent
```

Parameter:

TxCell	Cell to be read. Either a cell reference in the stype 'A1' or the name of a designated cell.
TxContent	
TxContent	Content of the table cell

Examples:

The user selects an Excel file to open. The values contained in the first column of the file's 2nd worksheet are read out, where the channel's name appears in the first row.

```
filename = DlgFileName("c:\results", "xlsx", "", 0)
IF XlWbOpen(filename)
   XlSheetActivate(2)
   TxName = XlGetText("A1")
   <TxName> = XlGetValues( "A2", 0, 0, 0)
END
```

See also:

XlSetText, XlGetValue, XlSetValue, XlGetValues

XlGetTextArray

Scope: Excel remote control

Specifies the content of a column or row in the active worksheet.

Declaration:

```
XlGetTextArray ( TxStartCell, SvRowOrColumn, SvCount ) -> Content
```

Parameter:

TxStartCell	Reading begins at this cell. Either a cell reference in the 'A1' style, or the name of a specified cell.
SvRowOrColumn	Specifies whether to read in the vertical direction (column) or horizontally (row).
	0 : Vertical (column)
	1 : Horizontal (row)
SvCount	Specifies the maximum count of cells to be read. Reading concludes once the count specified here has been reached.
Content	
Content	Text array with the cell content imported

Examples:

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   XlSetText("A1","Text 1")
   XlSetText("A2","Text 2")
   XlSetText("B1","Text 3")
   XlSetText("B2","Text 4")
   aColumnA = XlGetTextArray("A1",0,2)
   aRow1    = XlGetTextArray("A1",1,2)
END
```

XlGetValue

Scope: Excel remote control

Determines the content of the specified cell in the active worksheet and returns it as a number.

Declaration:

```
XlGetValue ( TxCell, Format ) -> Content
```

Parameter:

TxCell	Cell to be read. Either a cell reference in the stype 'A1' or the name of a designated cell.
Format	Data format of cell to be read
	0 : The cell value is read without conversion.
	1 : The value read is interpreted as an Excel Date/Time (amount of days since 1.1.1900) and converted to the imc time format (amount of seconds since 1.1.1980). For data < 1.1.1980, a 0 is returned.
Content	
Content	Content of the table cell

Description:

At fault condition, for instance if the content of a cell can not be converted to a number, a 0 is returned.

Examples:

A data set is read from an Excel file's first column. The first cell contains the name; the triggering time and sampling time are next. The actual data begin in the 4th row.

```
IF XlWbOpen("c:\results\report.xlsx")
   name = XlGetText("A1")
   time  = XlGetValue( "A2", 1)
   dx = XlGetValue( "A3", 0)
   <name> = XlGetValues("A4", 0, 0, 0)
   SetTime( <name>, time)
   XDELTA <name> dx
END
```

See also:

XlSetValue, XlSetText, XlGetText, XlGetValues

XlGetValues

Scope: Excel remote control

Gets the (numerical) content of a column or cell in the active worksheet.

Declaration:

```
XlGetValues ( TxStartCell, RowOrColumn, Format, Amount ) -> Content
```

Parameter:

TxStartCell	Reading begins at this cell. Either a cell reference in the 'A1' style, or the name of a specified cell.
RowOrColumn	Specifies whether to read in the vertical direction (column) or horizontally (row).
	0 : Vertical (column)
	1 : Horizontal (row)
Format	Data format of the range to be read
	0 : The value of the cells is read without conversion.
	1 : The values read are interpreted as Excel-Date/Time (amount of days since 1.1.1900) and converted to the imc time format (amount of seconds since 1.1.1980). With data < 1.1.1980, a 0 is returned.
Amount	Returns the maximum amount of cells to be read. Reading is concluded once the amount specified here has been reached or at the first cell whose contents not be converted to a number. If you enter 0, reading continues automatically up until the first non-numeric cell.
Content	
Content	Data set with the imported content of the cells.

Examples:

A data set is imported from an Excel file. The first column contains the time stamp of the values, stated with Date/Time. The second column contains the associated numerical values.

In FAMOS, a XY-waveform 'channel' is created, to which the first sample's time is assigned as the trigger time. The X-track then contains the relative differences from the start time.

```
IF XlWbOpen("c:\results\report.xlsx")
   time = XlGetValues("A1", 0, 1, 0)
   data = XlGetValues("B1", 0, 0, 0)
   channel = XYof( time - time[1], data)
   SetTime( channel, time[1])
END
```

See also:

XlGetValues2, XlSetValues, XlGetValue, XlGetText

XlGetValues2

Scope: Excel remote control

Gets the (numerical) content of a column or cell in the active worksheet.

Declaration:

`XlGetValues2 ( TxStartCell, RowOrColumn, Format, Amount, SubstituteValue, Reserved ) -> Content`

Parameter:

TxStartCell	Reading begins at this cell. Either a cell reference in the 'A1' style, or the name of a specified cell.
RowOrColumn	Specifies whether to read in the vertical direction (column) or horizontally (row).
	0 : Vertical (column)
	1 : Horizontal (row)
Format	Data format of the range to be read
	0 : The value of the cells is read without conversion.
	1 : The values read are interpreted as Excel-Date/Time (amount of days since 1.1.1900) and converted to the imc time format (amount of seconds since 1.1.1980). With data < 1.1.1980, a 0 is returned.
Amount	States the amount of cells to be read
SubstituteValue	This value is used if cell contents can not be converted to a numerical value.
Reserved	Reserved parameter, always set to 0
Content	
Content	Data set with the imported content of the cells.

Description:

Data series in EXCEL-tables sometiimes contain gaps to mark invalid values. Such invalid values are denoted by, for instance, empty cells or a special text (e.g. 'ZERO').

In contrast to the function XlGetValues(), which would end reading at such cells, the function XlGetValues2() skips such cells which can not be converted to a number and instead inserts a selectable substitute value into the data set generated.

Later if needed these can be extracted from the data set in FAMOS, for which purpose such functions as SearchLevel() are provided.

Examples:

An Excel file contains 2 data sets. The first column contains the common time column stating Date/Time. The second and third column contain the corresponding numerical values, where empty cells are possible if only one channel has a valid value at a particular time stamp.

The 3rd column is read and saved in FAMOS as an XY-waveform 'channel'. The invalid values are subsequently deleted from the data set.

```
IF XlWbOpen("c:\results\report.xlsx")
   time  = XlGetValues("A1", 0, 1, 0)
   count = leng?(time)
   data = XlGetValues2("C1", 0, 0, count, -1e30, 0)
   channel = XYof( time - time[1], data)
   channel = SearchLevel(channel, 2, -1e29, 0, 0, 0, 0, 0)
   SetTime( channel, time[1])
END
```

See also:

XlGetValues, XlSetValues, XlGetValue, XlGetText

XlPaste

Scope: Excel remote control

Inserts the content of the Clipboard into the active sheet.

Declaration:

`XlPaste ( TxInsertPosition ) -> Success`

Parameter:

TxInsertPosition	Determines the position at which the content of the Clipboard is to be inserted. Either a cell-reference in 'A1'-style or the name of a named cell. If empty, the currently selected cell is used.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

A curve window is displayed and copied to the Clipboard. A new Excel-document is generated, the curve plot inseted at the position 'B2', the workbook saved.

```
CwLoadCCV(channel1, "layout.ccv")
CwAction("clipboard.copy")
XlWbNew("")
XlPaste("B2")
XlWbSave("z:\tmp\channel1.xlsx", 0)
XlQuit()
```

See also:

XlSetText, XlSetValue

XlQuit

Scope: Excel remote control

Closes the linked Excel instance

Declaration:

`XlQuit ( )`

Parameter:

Description:

This function closes the Excel instance started explicitly with XlStart() or implicitly with XlWbOpen()/XlWbNew().

Any files still open are closed, any changes are discarded.

Examples:

An Excel file is opened and data inserted. The updated file is printed out, after which Excel is closed.

```
IF XlWbOpen("c:\Templates\Template.xlsx")
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlWbPrint()
   XlQuit()
END
```

See also:

XlStart, XlVisible

XlRunMacro

Scope: Excel remote control

Runs the specified Excel macro.

Declaration:

```
XlRunMacro ( Macroname ) -> Success
```

Parameter:

Macroname	Name of the macro to run.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

An Excel file is opened and data are inserted. Subsequently, a macro is run, which is savedd with the template. The updated file is saved under the new name, after which Excel is closed again.

```
IF XlWbOpen("c:\Templates\Template.xlsx")
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlRunMacro("Calculation"))
   XlWbSave("c:\results\report", 0)
   XlQuit()
END
```

Excel is started, made visible and a macro is run, which is defined in the workbook 'MyMacros.xlsm'. Subsequently, Excel is closed again.

```
IF XlStart()
   XlVisible(1)
    IF NOT( XlRunMacro("'c:\XLSTemplates\MyMacros.xlsm'!Macro2"))
      BoxMessage("Error", GetLastError(), "!1")
   END
   XlQuit()
END
```

See also:

XlWbOpen

XlSelectRange

Scope: Excel remote control

Selects a cell or a cell region.

Declaration:

```
XlSelectRange ( TxRange ) -> Success
```

Parameter:

TxRange	Cell or cell region to be selected. Either a cell refernce in 'A1'-style or the name of a named cell/cell region.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

This function affects the current sheet of the active workbook.

Examples of [TxRange]-specifications in the 'A1'-style:

Selection of an individual cell	"C2"	Cell C2 (3rd column, 2nd row)
Selection of a contiguous region	"C2:E3"	Cells C2 through E3 (3rd through 5th columns, 2nd and 3rd rows)
Selection of a column	"C:C"	The 3rd column is selected
Selection of multiple columns	"C:E"	The 3rd, 4th and 5th columns are selected.
Selection of a row	"2:2"	The 2nd row is selected
Selection of multiple rows	"2:4"	The 2nd, 3rd and 4th rows are selected
Multi-selection	"A1:A3,C4"	The cells A1,A2,A3 and C4 are selected.

As demonstrated in the last example, you can also specify multiple references (separated by semicolons), in order to select non-contiguous regions.

Examples:

In the current sheet of the current workbook, all cells containing the text 'Overflow' are shaded red. To do this, an Excel-macro 'MakeCellRed' is used, which is defined in the workbook 'MyMacros.xlsm' and takes effect on the cell selected.

```
hits = XlFind("Overflow", 0, 0, 0, 0)
FOR i = 1 TO TxArrayGetSize(hits)
   XlSelectRange(hits[i])
   XlRunMacro("'c:\XLSTemplates\MyMacros.xlsm'!MakeCellRed")
END
```

VBA-source text of the macro:

```
Sub MakeCellRed()
   With Selection.Interior
      .Pattern = xlSolid
      .PatternColorIndex = xlAutomatic
      .Color = 255
    End With
End Sub
```

The currently selected cell is moved down by one row:

```
sel = XlGetSelectedRange()
sel = XlBuildA1Ref(1, 0, 1, 1, sel)
XlSelectRange(sel)
```

See also:

XlGetSelectedRange, XlBuildA1Ref

XlSetBorderColor

Scope: Excel remote control

Sets the color of frame lines for the specified cells in the active worksheet.

Declaration:

```
XlSetBorderColor ( TxRange, SvFrameLine, SvLineColor )
```

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvFrameLine	Frame lines of the cell or group
	1 : Line: left
	2 : Line: right
	4 : Line: top
	8 : Line: bottom
	15 : Cell frame
	16 : Line: diagonally down
	32 : Line: diagonally up
	48 : Cross
	64 : Line: cell group left
	128 : Line: cell group right
	256 : Line: cell group top
	512 : Line: cell group bottom
	960 : Cell group frame
	1024 : Vertical line within the cell group
	2048 : Horizontal line within the cell group
	3072 : Horizontal + vertical line within the cell group
SvLineColor	Color of the frame line

Examples:

The colors for a cell's frame lines are generated individually.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = MatrixInit(5, 5, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues(ref, 0, Matrix[I], 0)
   END
   LOCAL colred   = 0
   LOCAL colgreen = 0
   FOR I = 1 TO count
      FOR J = 1 TO count
         LOCAL ref = XlBuildA1Ref( I, J, 1, 1)
         ; defining the line type and thickness for all 4 sides of the cell
         XlSetBorderStyle(ref,15,"continous")
         XlSetBorderThickness(ref,15,"thick")
         ; specifying the colors for the left and right lines
         XlSetBorderColor(ref,1+2,RGB(colred,0,0))
         ; specifying the colors for the top and bottom lines
         XlSetBorderColor(ref,4+8,RGB(0,colgreen,0))
         ; increasing color values
         colgreen = colgreen + 8
         colred = colred + 8
      END
   END
```

```
END
```

XlSetBorderStyle

Scope: Excel remote control

Sets the style of frame lines for the specified cells in the active worksheet.

Declaration:

`XlSetBorderStyle ( TxRange, SvFrameLine, TxLineType )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvFrameLine	Frame lines of the cell or group
	1 : Line: left
	2 : Line: right
	4 : Line: top
	8 : Line: bottom
	15 : Cell frame
	16 : Line: diagonally down
	32 : Line: diagonally up
	48 : Cross
	64 : Line: cell group left
	128 : Line: cell group right
	256 : Line: cell group top
	512 : Line: cell group bottom
	960 : Cell group frame
	1024 : Vertical line within the cell group
	2048 : Horizontal line within the cell group
	3072 : Horizontal + vertical line within the cell group
TxLineType	Type of the line
	"noline" : no line
	"continous" : line with strikethrough
	"dash" : dashed line
	"dot" : dotted line
	"dashdot" : dashed/dotted line
	"dashdotdot" : dashed/double-dotted line
	"double" : double-line
	"slanteddashdot" : hashed line

Description:

The line types are binary-encoded. Each line has its own code. The individual codes can be summed in order to achieve different frames.

Examples:

Various frame lines and line types are placed in and around the cells.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;removing all frame lines
   XlSetBorderStyle("B1:J10",0x3FF,"noline")
   ;frame line left, solid
   XlSetBorderStyle("B1:J1",1,"continous")
   ;frame line top, bottom, solid
   XlSetBorderStyle("B3:J3",12,"continous")
   ;frame line of whole frame dashed
   XlSetBorderStyle("B5:J5",15,"dash")
   ;frame line around cell group, solid
   XlSetBorderStyle("B7:J10",960,"continous")
   ;frame line diagonally downward, dotted
   XlSetBorderStyle("B12:J12",16,"dot")
   ;cross
   XlSetBorderStyle("B14:J14",48,"continous")
   ;frame line diagonally upward, dotted
   XlSetBorderStyle("B16:J16",32,"dot")
END
```

XlSetBorderThickness

Scope: Excel remote control

Ses the frame line for the specified cells in the active worksheet.

Declaration:

`XlSetBorderThickness ( TxRange, SvFrameLine, TxFrameThickness )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvFrameLine	Sets the frame line of the corresponding thickness.
	1 : Line: left
	2 : Line: right
	4 : Line: top
	8 : Line: bottom
	15 : Cell frame
	16 : Line: diagonally down
	32 : Line: diagonally up
	48 : Cross
	64 : Line: cell group left
	128 : Line: cell group right
	256 : Line: cell group top
	512 : Line: cell group bottom
	960 : Cell group frame
	1024 : Vertical line within the cell group
	2048 : Horizontal line within the cell group
	3072 : Horizontal + vertical line within the cell group
TxFrameThickness	Sets the frame line of the corresponding thickness.
	"hairline" : Hair line
	"thin" : thin
	"medium" : medium
	"thick" : thick

Description:

The line types are binary-encoded. Each line has its own code. The individual codes can be summed in order to achieve different frames.

Examples:

Various frame lines of various thicknesses and line types are placed around the cells.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;removing all frame lines
   XlSetBorderStyle("B1:J10",0x3FF,"noline")
   ;frame line around cell group, solid
   XlSetBorderStyle("B2:J10",960,"continous")
   ;thick frame line around the cell group
   XlSetBorderThickness("B2:J10",960,"thick")
   ;frame line around the cell group, dashed
   XlSetBorderStyle("C4:I8",960,"dash")
   ;thin frame line around the cell group
   XlSetBorderThickness("C4:I8",960,"thin")
END
```

XlSetCellFormat

Scope: Excel remote control

Sets the output format for the specified cells in the active worksheet.

Declaration:

```
XlSetCellFormat ( TxRange, TxFormat )
```

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxFormat	Format code

Description:

This function sets the output format for numerical values in Excel tables. This is especially crucial for Date/Time - for example, the number '36526.5' in the Excel time format (amount of days since 1.1.1900) means '1.1.2000 12:00'. In order for this time to be displayed accordingly and not siimply as a real number, the output format for the cell must explicitly be set to a date-format.

This function corresponds to the menu command 'Format Cells/Numbers' in Excel. An overview of the available format codes is found in this dialog in the category 'User-defined' or in Excel's help under the heading 'Numerical format codes'. Note that the format codes depend on the language of the Excel version used; this applies particularly to the frequently required codes for Date/Time components and the decimal separator for numbers (period vs. comma).

Selected format codes:

Display	German code	English code (if different)
Decimal digits (significant)	"#"	
Decimal digits (possibly non-significant zero)	"0"	
Decimal separator	"," (comma)	"." (period)
Scientific notation	"E-", "e-", "e+", "e-"	
Month	"M" (1-12) / "MM" (01-12)	"m" / "mm"
Day	"T" (1-31) / "TT" (01-31)	"d" / "dd"
Year	"JJ" (00-99) / "JJJJ" (1900-9999)	"yy" / "yyyy"
Hour	"h" (0-23) / "hh" (00-23)	
Minute	"m" (0-59) / "mm" (00-59)	
Second	"s" (0-59)/ "ss" (00-59)	
Time elapsed in minutes	, e.g. 63:46: [mm]:ss	
Fractions of a second	h:mm:ss,00	h:mm:ss.00

Examples:

Number	Format code (German Excel)	Display
38710.5036	"0,00"	"38710,50"
38710.5036	"0,##"	"38710,5"
38710.5036	"0.000E+0"	"3.871E+4"
38710.5036	"dd.mm.yy hh:mm"	"24.12.05 12:05"
38710.5036	"dd.mm.yyyy hh:mm:ss.000"	"24.12.2005 12:05:11,040"

Examples:

A new Excel file with a worksheet is created and a data set is transferred to this table. The first column is configured as a time column with specification of absolute Date/Time, the second column contains the data set's actual values.

The output format of both columns is specified explicitly. The file thus created is then saved.

```
XlWbNew("")
time = XlCreateTimeLine(channel1, 1, 0)
XlSetValues( "A1", 0, time, 1)
Count = Leng?(time)
Range = XlBuildA1Ref(1, 1, count, 1)
XlSetCellFormat(Range, "dd.mm.yy hh:mm:ss.0")
XlSetValues( "B1", 0, channel1, 0)
Range = XlBuildA1Ref(1, 2, count, 1)
```

```
XlSetCellFormat(Range, "0.00")
XlWbSave( "c:\results\report", 0)
XlQuit()
```

Like above, however here the time column is configured in relative terms (seconds since triggering). The time columns is outputted with 1ms resolution; the data column with a maximum of 2 decimal places.

```
XlWbNew("")
time = XlCreateTimeLine(channel1, 0, 0)
XlSetValues( "A1", 0, time, 0)
Count = Leng?(time)
Range = XlBuildA1Ref(1, 1, count, 1)
XlSetCellFormat(Range, "0.000")
XlSetValues( "B1", 0, channel1, 0)
Range = XlBuildA1Ref(1, 2, count, 1)
XlSetCellFormat(Range, "0.##")
XlWbSave( "c:\results\report", 0)
XlQuit()
```

See also:

XlSetText, XlSetValue

XlSetColor

Scope: Excel remote control

Sets the colors for display of the specified cells in the active worksheet.

Declaration:

`XlSetColor ( TxRange, TxCellProperty, SvColor )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxCellProperty	Property for the specified cells
	"background" : Color of the cell's background
	"text" : Color of the text in the cell
SvColor	RGB-value

Examples:

Specifies the text color and background color of a specified area.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = MatrixInit( 10, 10, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ; setting background color
   XlSetColor("A1:J10","background",RGB(250,250,150))
   ; setting text color
   XlSetColor("A1:J10","text",RGB(0,0,250))
END
```

XlSetColumnWidth

Scope: Excel remote control

Defines the width of the columns as a multiple of a basis width. For proportional fonts, the basis width is the width of the character "0", else the width of a character in Normal format.

Declaration:

`XlSetColumnWidth ( TxRange, SvColumnWidth )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvColumnWidth	Defines the column width. The value -1 sets the width to "automatic".

Examples:

The height and width of table cells are specified.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ; text output
   XlSetText("A1", "example text 1")
   XlSetText("B1", "example text 2")
   XlSetText("C1", "example text 3")
   ; cell width as a multiple of the 1st character's width
   XlSetColumnWidth("A1:C1", 20)
   ; spcifying cell height in points (pixels)
   XlSetRowHeight("A1:C1", 50)
END
```

XlSetConditionColor

Scope: Excel remote control

Sets the color for display of the specified cells in the active worksheet, dependent on a condition.

Declaration:

```
XlSetConditionColor ( TxRange, TxCellProperty, TxConditionType, TxExpression, TxConditionOperator, SvParameter1, SvParameter2, SvColor )
```

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxCellProperty	Cell property for which the color is used when the condition is met.
	"background" : Color of the cell's background
	"text" : Color of the text in the cell
	"border" : Color of the cell frame
TxConditionType	Refernce of the condition
	"cellcontent" : The condition refers to the cell's content.
	"expression" : The condition is the evaluated expression.
	"reset" : All conditions for the cells are reset.
TxExpression	Expression which is evaluated for the condition (e.g. A1>0).
TxConditionOperator	Operator for evaluating the cell contents
	"between" : The cell contents lie between the parameters 1 and 2
	"notbetween" : The cell contents do not lie between the parameters 1 and 2
	"equal" : The cell contents equal the Parameter 1
	"notequal" : The cell contents do not equal the Parameter 1
	"greater" : The cell contents are greater than the Parameter 1
	"less" : The cell contents are less than the Parameter 1
	"greaterequal" : The cell contents are greater than or equal to the Parameter 1
	"lessequal" : The cell contents are less than or equal to the Parameter 1
SvParameter1	Comparison value or lower boundary value
SvParameter2	Upper boundary value
SvColor	Color to be used when condition is met

Examples:

The colors for the text, background and frame of a specified area are specified according to a condition.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = MatrixInit( 10, 10, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ; setting background color
   XlSetConditionColor("A1:J10","background","cellcontent","","equal",1,0,RGB(250,250,0))
   ; setting text color
   XlSetConditionColor("A1:J10","text","cellcontent","","equal",1,0,RGB(250,0,0))
   ; setting the frame color
   XlSetConditionColor("A1:J10","border","expression","=A1=1","",0,0,RGB(250,0,0))
END
```

XlSetFontSize

Scope: Excel remote control

Sets the font size for the specified cells in the active worksheet.

Declaration:

`XlSetFontSize ( TxRange, SvSize )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvSize	Font size for the cell in pixels

Examples:

The system magnifies the text incrementally.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = MatrixInit(5, 5, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues(ref, 0, Matrix[I], 0)
   END
   LOCAL TextSize = 8
   FOR I = 1 TO count
      FOR J = 1 TO count
         LOCAL ref = XlBuildA1Ref( I, J, 1, 1)
         XlSetFontSize(ref, TextSize)
         TextSize = TextSize + 1;
      END
   END
END
```

XlSetFontStyle

Scope: Excel remote control

Sets the font style for the specified cells in the active worksheet.

Declaration:

```
XlSetFontStyle ( TxRange, TxStyle )
```

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxStyle	Font style
	"reset" : Restores the original state
	"bold" : Bold text
	"italic" : 'Italic' text
	"underlined" : Text with single underline
	"underlineddouble" : Text double-underlined
	"underlineddoubleclosed" : Text double-underlined directly below the text
	"subscript" : subscript
	"superscript" : superscript
	"strikethrough" : Text with strikethrough

Description:

This function sets a style for the text in a cell. Certain styles can be combined. "Reset" resets all styles.

Examples:

The font size and text style are specified.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = MatrixInit( 10, 10, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ; setting font size
   XlSetFontSize("A1:J1",10)
   XlSetFontSize("A5:J5",20)
   ; setting font style
   XlSetFontStyle("A2:J2","italic")
   XlSetFontStyle("A3:J3","bold")
   XlSetFontStyle("A4:J4","strikethrough")
   XlSetFontStyle("A5:J5","underlined")
END
```

XlSetRowHeight

Scope: Excel remote control

Defines the height of the rows in pixels for the specified cells in the active worksheet.

Declaration:

```
XlSetRowHeight ( TxRange, SvRowHeight )
```

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvRowHeight	Defines the row height.

Examples:

The height and width of table cells are specified.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ; text output
   XlSetText("A1", "example text 1")
   XlSetText("B1", "example text 2")
   XlSetText("C1", "example text 3")
   ; cell width as a multiple of the 1st character's width
   XlSetColumnWidth("A1:C1", 20)
   ; spcifying cell height in points (pixels)
   XlSetRowHeight("A1:C1", 50)
END
```

XlSetText

Scope: Excel remote control

Fills the specified cell in the active worksheet with a text.

Declaration:

`XlSetText ( TxCell, TxContent ) -> Success`

Parameter:

TxCell	Cell to which to write. Either a cell reference in the style 'A1' or the name of a designated cell.
TxContent	New content
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

An Excel file is opened and transferred to 2 columns. The first row contains the respective channel's name. The updated file is printed and saved under a new name.

```
IF NOT (XlWbOpen("c:\Templates\Template.xlsx"))
   BoxMessage("Can't open file", GetLastError(), "!1")
ELSE
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlSetText("C1", "Channel2")
   XlSetValues("C2", 0, Channel2, 0)
   XlWbPrint()
   IF NOT(XlWbSave("c:\Results\Report.xlsx", 0))
      BoxMessage("Can't save file", GetLastError(), "!1")
   END
   XlQuit()
END
```

See also:

XlGetText, XlGetValue, XlSetValue, XlGetValues

XlSetTextAlignment

Scope: Excel remote control

Sets the orientation of the texts of the specified cells in the active worksheet.

Declaration:

`XlSetTextAlignment ( TxRange, TxOrientation )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxOrientation	Position of the text within the cell
	"left" : left
	"hcenter" : centered horizontally
	"right" : right
	"top" : top
	"vcenter" : centered vertically
	"bottom" : bottom

Examples:

Various orientations and directions are specifed for the cell texts.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = 100*MatrixInit( 4, 4, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ; centering all cell texts horizontally and vertically
   XlSetTextAlignment("A1:D4", "hcenter")
   XlSetTextAlignment("A1:D4", "vcenter")
   ; rotate all cell texts of the diagonals
   XlSetTextOrientation("A1", 45)
   XlSetTextOrientation("B2", 90)
   XlSetTextOrientation("C3", -90)
   XlSetTextOrientation("D4", -45)
END
```

XlSetTextArray

Scope: Excel remote control

Sets the content of a column or row in the active worksheet.

Declaration:

`XlSetTextArray ( TxStartCell, SvRowOrColumn, TaTextArray ) -> Success`

Parameter:

TxStartCell	Writing begins at this cell. Either a cell-reference in the style 'A1' or the name of a designated cell.
SvRowOrColumn	Specifies whether to write in either the vertical direction (column) or horizontal direction (row).
	0 : Vertical (column)
	1 : Horizontal (row)
TaTextArray	Text data to be written
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

Text arrays are created as column and row vectors.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   LOCAL headlines = TxArrayCreate(3)
   headlines[1]="1.Headline"
   headlines[2]="2.Headline"
   headlines[3]="3.Headline"
   LOCAL data = TxArrayCreate(3)
   data[1]="1.Dataset"
   data[2]="2.Dataset"
   data[3]="3.Dataset"
   XlSetTextArray("B1",1,headlines)
   XlSetTextArray("A2",0,data)
   ;generating a matrix for the table
   LOCAL Matrix = MatrixInit( 3, 3, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   LOCAL I
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 2, I+1, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   XlSetColumnWidth("A1:D1",15)
   XlSetTextAlignment("B1:D4","hcenter")
END
```

XlSetTextOrientation

Scope: Excel remote control

Sets the orientation of the text in the specified cells in the active worksheet.

Declaration:

`XlSetTextOrientation ( TxRange, SvOrientation )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
SvOrientation	Orientation of the text within the cell

Examples:

Various orientations and directions are specifed for the cell texts.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;generating a matrix for the table
   Matrix = 100*MatrixInit( 4, 4, "I")
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      LOCAL ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ; centering all cell texts horizontally and vertically
   XlSetTextAlignment("A1:D4", "hcenter")
   XlSetTextAlignment("A1:D4", "vcenter")
   ; rotate all cell texts of the diagonals
   XlSetTextOrientation("A1", 45)
   XlSetTextOrientation("B2", 90)
   XlSetTextOrientation("C3", -90)
   XlSetTextOrientation("D4", -45)
END
```

XlSetTextShrinkToFit

Scope: Excel remote control

Activates text shrinking to fit to the cell size for the specified cells in the active worksheet.

Declaration:

`XlSetTextShrinkToFit ( TxRange, TxShrink )`

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxShrink	Fitting the text's to the cell's size.
	"noshrink" : Do not fit text
	"shrink" : Fit text

Examples:

Defines various methods for fitting a cell text.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ; text output
   XlSetText("A1", "example text 1")
   XlSetText("B1", "example text 2")
   XlSetText("C1", "example text 3")
   ; cell size as a multiple of the 1st character's width
   XlSetColumnWidth("A1:C1", 8)
   ; defining a text line break
   XlSetTextWrap("A1", "wrap")
   ; specifying a text fitting method
   XlSetTextShrinkToFit("B1", "shrink")
END
```

XlSetTextWrap

Scope: Excel remote control

Activates the text line break mode for the specified cells in the active worksheet.

Declaration:

```
XlSetTextWrap ( TxRange, TxLineBreak )
```

Parameter:

TxRange	Range to be formatted. Either a range reference in the style 'A1:B2' or the name of a named range.
TxLineBreak	Defines the line break of the text within the cell.
	"nowrap" : Do not line break text
	"wrap" : Line break text

Examples:

Defines various methods for fitting a cell text.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ; text output
   XlSetText("A1", "example text 1")
   XlSetText("B1", "example text 2")
   XlSetText("C1", "example text 3")
   ; cell size as a multiple of the 1st character's width
   XlSetColumnWidth("A1:C1", 8)
   ; defining a text line break
   XlSetTextWrap("A1", "wrap")
   ; specifying a text fitting method
   XlSetTextShrinkToFit("B1", "shrink")
END
```

XlSetValue

Scope: Excel remote control

Fills the specified cell in the active worksheet with a number.

Declaration:

```
XlSetValue ( TxCell, Value, Format ) -> Success
```

Parameter:

TxCell	Cell to which to write. Either a cell reference in the style 'A1' or the name of a designated cell.
Value	Value to be transferred
Format	Data format of the value to be transferred
	0 : The value is written without conversion.
	1 : The value read is interpreted as a Date/Time (amount of seconds since 1.1.1980) in imc format and converted to the time Excel format (amount of days since 1.1.1900).
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

When Option 1 (Date/Time) is specified, the target cell's current cell formatting is checked. If it is set to 'Defaul', the output format is set for Date/Time.

Examples:

The data set 'Channel1' is written to a new Excel file's first column. The first row contains the name; the trigger time and the sampling time are next. The actual data begin in the 4th row.

```
XlWbNew("")
XlSetText( "A1", "Channel1")
XlSetValue("A2", Time?(Channel1), 1)
XlSetCellFormat("A2", "dd.mm.yy hh:mm:ss.000")
XlSetValue("A3", xdel?(Channel1), 0)
XlSetValues("A4", 0, Channel1, 0)
XlWbSave( "c:\results\report", 0)
XlQuit()
```

See also:

XlGetValue, XlGetText, XlSetText, XlGetValues

XlSetValues

Scope: Excel remote control

Sets the content of a column or row in the active worksheet.

Declaration:

```
XlSetValues ( TxStartCell, RowOrColumn, Data, Format ) -> Success
```

Parameter:

TxStartCell	Writing begins at this cell. Either a cell-reference in the style 'A1' or the name of a designated cell.
RowOrColumn	Specifies whether to write in either the vertical direction (column) or horizontal direction (row).
	0 : Vertical (column)
	1 : Horizontal (row)
Data	Data to be written
Format	Data format of the data to be transferred.
	0 : The values are written without conversion.
	1 : The values are interpreted as Date/Time in the imc time format (amount of seconds since 1.1.1980) and converted to the Excel Date/Time-format (amount of days since 1.1.1900).
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

The data set to be transferred must be sampled equidistantly, and unstructured. With structured waveforms (components (XY, complex), segments, events), the individual elemente may have to be transferred sequentially.

When Option 1 (Date/Time) is specified, the current cell formatting of the target range is checked. If it is set to 'Default', the output format is set to Date/Time.

To write Excel files, you can also use the functions FileOpenXLS() and in particular FileOpenXLS2(). In general, these are more powerful and faster, but in return less flexible than the corresponding functions in this Kit.

Examples:

A new Excel file with a worksheet is created, and a data set is transferred into the table. The second column has the header 'Data', followed by the data set's values. The first column has the header 'Time', followed by the corresponding specifications of the Date/Time. The output format for both columns is set explicitly. The file thus created is then saved.

```
XlWbNew("")
; Fill column with Date/Time
XlSetText("A1", "Time")
time = XlCreateTimeLine(channel1, 1, 0)
XlSetValues( "A2", 0, time, 1)
Count = Leng?(time)
Range = XlBuildA1Ref(2, 1, count, 1)
XlSetCellFormat(Range, "dd.mm.yy hh:mm:ss.0")
; fill data column
XlSetText("B1", "Data")
XlSetValues( "B2", 0, channel1, 0)
Range = XlBuildA1Ref(2, 2, count, 1)
XlSetCellFormat(Range, "0.00")
XlWbSave( "c:\results\report", 0)
XlQuit()
```

A event-based data set's events are sequentially transferred to consecutive columns of an Excel table.

```
XlWbNew("")
count = EventNum?(channel)
;       In case the data set is segmented:
;       count = Leng?(Channel) / SegLen?(Channel)
FOR I = 1 TO count
   ref = XlBuildA1Ref( 1, I, 1, 1)
   XlSetValues( ref, 0, channel[I], 0)
END
XlWbSave("c:\results\report", 0)
XlQuit()
```

See also:

XlGetValues2, XlGetValue, XlGetText

XlSheetActivate

Scope: Excel remote control

Activates a sheet belonging to the current workbook.

Declaration:

```
XlSheetActivate ( TxTitleOrIndex ) -> Success
```

Parameter:

TxTitleOrIndex	Title or index of the sheet to be activated. The first sheet's index is 1.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Many functions belonging to this Kit affect the active sheet of the active workbook. The active sheet can change if a sheet is created or deleted with XlSheetAdd()/XlSheetDelete(), if a new sheet is explicitly activated using XlSheetActivate(), or if the user manually activates another page (e.g. clicking on the corresponding tab in the workbook).

Examples:

An Excel file is opened and data are transferred into 2 worksheets. The updated file is printed out and saved under a new name.

```
IF NOT (XlWbOpen("c:\Templates\Template.xlsx"))
   BoxMessage("Can't open file", GetLastError(), "!1")
ELSE
   XlSheetActivate("Table1")
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlSheetActivate("Table2")
   XlSetText("B1", "Channel2")
   XlSetValues("B2", 0, Channel2, 0)
   XlWbPrint()
   IF NOT(XlWbSave("c:\Results\Report.xlsx", 0))
      BoxMessage("Can't save file", GetLastError(), "!1")
   END
   XlQuit()
END
```

See also:

XlSheetAdd, XlSheetGetActive

XlSheetAdd

Scope: Excel remote control

A worksheet is inserted into the current workbook.

Declaration:

```
XlSheetAdd ( TxTitle, SvPos, TxTemplate ) -> Success
```

Parameter:

TxTitle	Title of the new worksheet
SvPos	States the position at which the new worksheet is inserted. Enter a 0 if you wish to append the worksheet at the last position. Entering 1 means that the new worksheet is the workbook's first worksheet, etc.
TxTemplate	If you specify an empty text, an empty standard worksheet is created. You can also specify the complete pathname of an Excel file to be used as the template for the new sheet.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

A new Excel file is created and filled with the data from a FAMOS data group called 'MyGroup'. The first sheet is filled with some general specifications, then a new worksheet is created for each of the group's channels and the first column is filled with the channel's values.

For cover sheets and data sheets each, a pre-made template is used which comes with fixed texts and cell formatting.

```
IF XlWbNew("c:\templates\firstpage.xlsx")
   XlVisible(1)
   XlSetText( "A3", "Name: Mike Smith")
   XlSetText( "A4", "Date: " + TimeToText( TimeSystem?(),0))
   FOR I = 1 TO GrChanNum?(MyGroup)
      XlSheetAdd( GrChanName?(MyGroup, I), 0, "c:\templates\datapage.xlsx")
      XlSetValues( "A1", 0, MyGroup:[I], 0)
   END
   XlWbSave( "c:\results\report", 0)
   XlQuit()
END
```

See also:

XlSheetDelete

XlSheetDelete

Scope: Excel remote control

A worksheet is deleted from the active workbook.

Declaration:

```
XlSheetDelete ( TxTitleOrIndex ) -> Success
```

Parameter:

TxTitleOrIndex	Title or index of the worksheet to be deleted. The first sheet's index is 1.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

An Excel template with 2 worksheets is opened. However, for the current task, only the first worksheet is needed, the second sheet is thus deleted before saving the updated file.

```
XlWbOpen("c:\TemplateWith2Sheets.xlsx")
;...updating first worksheet...
XlSheetDelete(2)
; or e.g. : XlSheetDelete("Table2")
XlWbSave("c:\results\report", 0)
```

See also:

XlSheetAdd

XlSheetExist

Scope: Excel remote control

Checks whether the active workbook contains a sheet with the specified title.

Declaration:

```
XlSheetExist ( TxTitle ) -> Result
```

Parameter:

TxTitle	Title of the sheet to be found
Result	
Result	1 if such a sheet exists; else 0. -1 at fault condition; the error cause can be found by using the function GetLastError().

Examples:

An Excel-file is loaded and is checked for the presence of a spreadsheet having a particular title. If it is not present, an error message is posted.

```
XlWbOpen("c:\results\report.xlsx")
ok = XlSheetExist("Data from 20.07.2018")
IF NOT ok
   BoxMessage( "Error", "Unexpected excel file", "!1")
END
```

See also:

XlSheetGetActive, XlSheetGetTitle, XlSheetGetCount, XlWbExist

XlSheetGetActive

Scope: Excel remote control

Finds the active sheet of the active workbook

Declaration:

`XlSheetGetActive ( ) -> TxTitle`

Parameter:

TxTitle	
TxTitle	Title of the active sheet

Description:

Many functions belonging to this Kit affect the active sheet of the active workbook. The active sheet can change if a sheet is created or deleted with XlSheetAdd()/XlSheetDelete(), if a new sheet is explicitly activated using XlSheetActivate(), or if the user manually activates another page (e.g. clicking on the corresponding tab in the workbook).

The title which this finds matches the name displayed on the corresponding tab in the workbook.

At fault condition, an empty text is returned. The cause can be determined by means of the function GetLastError().

Examples:

The user selects an Excel file to open. The values in the first column of the file's 2nd worksheet are read out. The data set generated contains the worksheet's name.

```
filename = DlgFileName("c:\results", "xlsx", "", 0)
IF XlWbOpen(filename)
   XlSheetActivate(2)
   SheetName = XlSheetGetActive()
   <SheetName> = XlGetValues( "A1", 0, 0, 0)
END
```

See also:

XlSheetAdd, XlSheetActivate

XlSheetGetCount

Scope: Excel remote control

Gets the count of sheets in the active workbook.

Declaration:

`XlSheetGetCount ( ) -> Amount`

Parameter:

Amount	
Amount	Number of sheets. -1 at fault condition; the error cause can be found by using the function GetLastError().

Description:

Examples:

An Excel-file is loaded and the titles of all table sheets is outputted in the output window.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   sheetCount = XlSheetGetCount()
   FOR i = 1 TO sheetCount
      BoxOutput( XlSheetGetTitle(i), EMPTY, "", 1)
   END
END
```

A workbook is opened and a copy of the last page appended.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   sheetCount = XlSheetGetCount()
   XlSheetInsertCopy("", sheetCount, 0, "Tabelle " + TForm(sheetCount+1, ""))
   ;...
END
```

See also:

XlSheetGetActive, XlSheetGetTitle, XlSheetExist

XlSheetGetTitle

Scope: Excel remote control

The title of a sheet in the active workbook is found.

Declaration:

`XlSheetGetTitle ( Index ) -> TxTitle`

Parameter:

Index	Index of the sheet. Lies between 1 and the count of sheets in the active workbook.
TxTitle	
TxTitle	Title of the sheet. Empty text at fault condition; the error cause can be found by using the function GetLastError().

Description:

The count of sheets and thus the maximum value for [Index] can be found by using the function XlSheetGetCount().

Examples:

An Excel-file is loaded and the titles of all table sheets is outputted in the output window.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   sheetCount = XlSheetGetCount()
   FOR i = 1 TO sheetCount
      BoxOutput( XlSheetGetTitle(i), EMPTY, "", 1)
   END
END
```

See also:

XlSheetGetActive, XlSheetGetCount, XlSheetExist

XlSheetInsertCopy

Scope: Excel remote control

The copy of a sheet is inserted into the active workbook.

Declaration:

```
XlSheetInsertCopy ( TxSourceWorkbook, SourceSheetTitleOrIndex, SvPos [, TxSheetTitle] ) -> Success
```

Parameter:

TxSourceWorkbook	Name of the workbook containing the sheet to be copied. Empty string, if the current workbook is to be used.
SourceSheetTitleOrIndex	Title or index of the sheet to be copied. The first sheet's index is 1.
SvPos	States the position at which the new worksheet is inserted. Enter a 0 if you wish to append the worksheet at the last position. Entering 1 means that the new worksheet is the workbook's first worksheet, etc.
TxSheetTitle	Title of the new sheet. If empty, the name automatically assigned by Excel is used. (optional , Default value: "")
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Wenn das einzufügende Blatt aus einer anderen Mappe stammt (Parameter [TxQuellArbeitsMappe] ist nicht leer), so muß auch diese Vorlagenmappe vorher mit XlWbOpen() geladen worden sein.

Examples:

A workbook is opened and a copy of the last page appended.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   sheetCount = XlSheetGetCount()
   XlSheetInsertCopy("", sheetCount, 0, "Tabelle " + TForm(sheetCount+1, ""))
   ;...
END
```

The first sheet of a template file "templates.xlsx" is appended to the workbook 'results.xlsx' angehängt and the 2nd column is filled with the values of 'channel', a previously loaded data set. The new workbook is assigned the data set's trigger date as its title. Subsequently, the updated workook is saved.

```
IF XlWbOpen( "z:\tmp\templates.xlsx")
   IF XlWbOpen( "z:\tmp\results.xlsx")
      XlSheetInsertCopy( "templates.xlsx", 1, 0, TimeToText(Time?(channel), 1))
      XlSetValues("B2", 0, channel, 0)
      XlWbSave("", 0)
   END
   XlQuit()
END
```

See also:

XlSheetDelete, XlSheetAdd, XlSheetMove

XlSheetMove

Scope: Excel remote control

The sheet's position within the active workbook is changed, or a sheet is moved from another workbook to the current workbook.

Declaration:

`XlSheetMove ( TxSourceWorkbook, SourceSheetTitleOrIndex, SvPos [, TxSheetTitle] ) -> Success`

Parameter:

TxSourceWorkbook	Name of the workbook containing the sheet to be moved. Empty string, if the current workbook is to be used.
SourceSheetTitleOrIndex	Title or index of the sheet to be moved. The first sheet's index is 1.
SvPos	Specifies the moved sheet's new position within the active workbook. Enter 0 to append the sheet at the last position. Entering 1 makes the sheet the first sheet in the workbook, etc.
TxSheetTitle	Title of the new sheet. If empty, the name automatically assigned by Excel is used. (optional , Default value: "")
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Wenn das einzufügende Blatt aus einer anderen Mappe stammt (Parameter [TxQuellArbeitsMappe] ist nicht leer), so muß auch diese Mappe vorher mit XlWbOpen() geladen worden sein.

Examples:

A workbook is opened and the last page is moved to the first position.

```
IF XlWbOpen("z:\tmp\results.xlsx")
   sheetCount = XlSheetGetCount()
   XlSheetMove("", sheetCount, 1)
   ;...
END
```

The first sheet of a workbook 'results1.xlsx' is moved into the workbook 'results2.xlsx' at hte last position. Subsequently, both workbooks are saved.

```
IF XlWbOpen("z:\tmp\results1.xlsx")
   IF XlWbOpen("z:\tmp\results2.xlsx")
      XlSheetMove("results1.xlsx", 1, 0)
      XlWbSave("", 0)
      XlWbActivate("results1.xlsx")
      XlWbSave("", 0)
   END
   XlQuit()
END
```

See also:

XlSheetDelete, XlSheetAdd, XlSheetInsertCopy

XlSheetPrint

Scope: Excel remote control

The active sheet of the active workbook is printed out.

Declaration:

```
XlSheetPrint ( ) -> Success
```

Parameter:

Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

An Excel file is opened and the 2nd and 3rd sheet of the workbook are printed out.

```
filename = "c:\results\report.xlsx"
IF XlWbOpen(filename)
   XlSheetActivate(2)
   XlSheetPrint()
   XlSheetActivate(3)
   XlSheetPrint()
END
```

See also:

XlWbOpen, XlWbPrint

XlSheetRename

Scope: Excel remote control

Renames a sheet in the active workbook.

Declaration:

```
XlSheetRename ( TxTitleOrIndex, TxNewTitle ) -> Success
```

Parameter:

TxTitleOrIndex	Title or index of the sheet to be renamed. The first sheet's index is 1.
TxNewTitle	New title
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Examples:

An Excel-file is loaded and updated. Subsequently, the first table sheet is named according to the current date, and the file is saved under the new name.

```
XlWbOpen("z:\tmp\results.xlsx")
; ... various updates
TxDate = TimeToText( TimeSystem?(), 1)
ok = XlSheetRename(1, TxDate)
XlWbSave("z:\tmp\results_"+ TxDate + ".xlsx", 0)
```

See also:

XlSheetGetTitle, XlSheetGetCount, XlSheetAdd

XlSheetSetColumnStandardWidth

Scope: Excel remote control

Specifies the default width for table columns as a multiple of one character's width.

Declaration:

`XlSheetSetColumnStandardWidth ( SvDefaultWidth )`

Parameter:

SvDefaultWidth	Width of one table column

Examples:

A new Excel document is generated. Determines the new width for all columns.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ;specifying column width for all columns
   XlSheetSetColumnStandardWidth(25)
END
```

XlSheetSetOption

Scope: Excel remote control

Sets display and printing options for the active sheet.

Declaration:

```
XlSheetSetOption ( TxOptionName, TxNewSetting ) -> Success
```

Parameter:

TxOptionName	Designation of the option
	"GridLines.Show" : Show grid
	"GridLines.Print" : Print out grid
	"Headings.Show" : Show column-/row headings
	"Headings.Print" : Print column-/row headings
	"Page.Orientation" : Defines the page's orientation (portrait/landscape)
TxNewSetting	[TxOptionsName] determines what settings are possible:
	"GridLines.Show" : Show grid
	"on" yes; **"off"** no
	"GridLines.Print" : Print out grid
	"on" yes; **"off"** no
	"Headings.Show" : Show column-/row headings
	"on" yes; **"off"** no
	"Headings.Print" : Print column-/row headings
	"on" yes; **"off"** no
	"Page.Orientation" : Print column-/row headings
	"portrait" Portrait format; **"landscape"** Landscape format
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Examples:

A new Excel-document is generated; the first column is filled with the values of a previously calculated data set 'channel1' and printed out. In the printout, the gridlines and column/row headers are displayed.

```
channel1 = Ramp(0,1,100)
XlWbNew("")
XlSetValues("A1", 0, channel1, 0)
XlSheetSetOption("GridLines.Print", "on")
XlSheetSetOption("Headings.Print", "on")
XlWbPrint()
```

See also:

XlWbPrint

XlSheetSetPicture

Scope: Excel remote control

Inserts an image from a file.

Declaration:

```
XlSheetSetPicture ( TxFilename, SvOptions, TxRange, SvPosX, SvPosY, SvWidth, SvHeight )
```

Parameter:

TxFilename	Name of image file
SvOptions	Paste options
	0 : The cell area is valid; the image covers the specified cells and will be fitted.
	1 : Only the upper left corner of the cell area is valid; the image retains its original size.
	2 : All position specifications are valid; the image covers the spcified area and will be fitted.
	3 : Only the position is valid; the image retains its original size.
TxRange	Cell area of the image; either a cell-reference in the style 'A1', or the name of a known cell or cell area
SvPosX	Position left
SvPosY	Position top
SvWidth	Width of image
SvHeight	Height of image

Examples:

A new Excel document is generated, an image is loaded to various positions and fitted.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   ; The image covers over the cells in the specified cell area.
   XlSheetSetPicture("C:\imc\Projects\ExcelTestProjekt\Flugzeug.jpg",0,"D1:F9",0,0,0,0)
   ; The image cover the pixel region 150x100 starting at the position 25,25.
   XlSheetSetPicture("C:\imc\Projects\ExcelTestProjekt\Flugzeug.jpg",2,"",25,25,150,100)
   ; The image covers over the pixel region corresponding to the size of the image beginning at the upper left corner of the cell range.
   XlSheetSetPicture("C:\imc\Projects\ExcelTestProjekt\Flugzeug.jpg",1,"A150",0,0,0,0)
   ; The image covers over the pixel region corresponding to the size of the image beginning at the specified position.
   XlSheetSetPicture("C:\imc\Projects\ExcelTestProjekt\Flugzeug.jpg",3,"",100,200,0,0)
END
```

XlSheetSetPrintArea

Scope: Excel remote control

Defines the area comprising the cells in the active worksheet to be printed.

Declaration:

`XlSheetSetPrintArea ( TxRange )`

Parameter:

TxRange	Area to be printed. Either an area reference in the style of 'A1:B2' or the name of a known area.

Examples:

A new Excel document is generated and filled with a matrix. The printing area is defined to comprise the entire matrix and the printing area is to be fitted to 1 page.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   Matrix = MatrixInit( 20, 20, "I") ; Matrix für die Tabelle erzeugen
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ;defining the print area
   XlSheetSetPrintArea("A1:T20")
   ;specifying print area on 1 page
   XlSheetSetPrintFitTo(1,1)
END
```

XlSheetSetPrintFitTo

Scope: Excel remote control

Fits the area of cells to be printed to the specified count of either vertically or horizontally adjacent pages.

Declaration:

`XlSheetSetPrintFitTo ( SvHorizontalCount, SvVerticalCount )`

Parameter:

SvHorizontalCount	Count of horizontally adjacent pages
SvVerticalCount	Count of vertically adjacent pages

Examples:

A new Excel document is generated and filled with a matrix. The printing area is defined to comprise the entire matrix and the printing area is to be fitted to 1 page.

```
IF XlStart() ; EXCEL start
   XlVisible(1) ; EXCEL show
   XlWbNew(""); workbook create
   Matrix = MatrixInit( 20, 20, "I") ; Matrix für die Tabelle erzeugen
   LOCAL count = Leng?(Matrix) / SegLen?(Matrix)
   FOR I = 1 TO count
      ref = XlBuildA1Ref( 1, I, 1, 1)
      XlSetValues( ref, 0, Matrix[I], 0)
   END
   ;defining the print area
   XlSheetSetPrintArea("A1:T20")
   ;specifying print area on 1 page
   XlSheetSetPrintFitTo(1,1)
END
```

XlStart

Scope: Excel remote control

Starts an Excel instance

Declaration:

`XlStart ( ) -> Success`

Parameter:

Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

The function checks whether an active Excel instance already exists, which had been created by functions belonging to this Kit. If not, a new Excel instance is started which becomes the target of all subsequent calls to this Kit's functions.

A hidden instance of Excel is started. Use the function XlVisible() if necessary, in order to make the Excel main window visible.

You do not need to call this function if you wish to load a file right afterwards (XlWbOpen()) or create a new workbook (XlWbNew()). Both functions check whether an instance of Excel already exists and start Excel if needed.

At the end of the utilzation of Excel, it is absolutely necessary to call XlQuit() to close any (possibly not visible) Excel instances and thus to free up any no longer required resources.

Multithreading: Each execution thread uses its own EXCEL instance. If, for example, EXCEL is started in a parallel sequence function (BEGIN_PARALLEL) using XlWbOpen () and a document is loaded, further access to this document is only permitted within the same sequence function. If the EXCEL instance was not explicitly closed with XlQuit (), it is closed automatically at the end of the sequence function.

Examples:

Excel is started, made visible and a macro is run, which is defined in the workbook 'MyMacros.xlsm'. Subsequently, Excel is closed again.

```
IF XlStart()
   XlVisible(1)
    IF NOT( XlRunMacro("'c:\XLSTemplates\MyMacros.xlsm'!Macro2"))
      BoxMessage("Error", GetLastError(), "!1")
   END
   XlQuit()
END
```

See also:

XlQuit, XlVisible

XlVisible

Scope: Excel remote control

With this function, you can govern the visibility of the working windows which belong to the linked Excel instance.

Declaration:

`XlVisible ( Option )`

Parameter:

Option	Visibility
	0 : Excel is not visible.
	1 : Excel is visible (and operable).

Examples:

A Excel file is opened and the worksheet having the title 'Table2' is activated and displayed.

Following confirmation by the user, the data are read into the 2nd column (beginning at Row 3) and transferred to FAMOS.

```
IF XlWbOpen("c:\results\report.xlsx")
   XlVisible(1)
   XlSheetActivate("Table2")
   IF BoxMessage( "Check", "Transfer these data?", "?4") = 1
      DataColumn2 = XlGetValues("B3", 0, 0, 0)
   END
   XlQuit()
END
```

See also:

XlStart, XlQuit

XlWbActivate

Scope: Excel remote control

The workbook having the specified name is activated.

Declaration:

```
XlWbActivate ( Name ) -> Success
```

Parameter:

Name	Name of the workbook to be activated
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

This function is sometimes necessary if you have multiple workbooks open at the same time. Most of this Kit's functions affect the active workbook set here.

The name to be entered here exactly matches the name displayed in the title bar of the associated Excel window, including the filename extension (e.g. '.xls' or '.xlsx') in the case of already saved workbooks.

Examples:

At the beginning of an analysis, a new Excel file is created. The name (automatically assigned by Excel) is noted. At the end of the analysis, the newly created workbook is to be saved.

```
XlWbNew("")
XlVisible(1)
NewBook = XlWbGetActive()
;...
IF XlWbActivate(NewBook)
   XlWbSave( "c:\results\report.xlsx", 0)
END
```

See also:

XlWbOpen, XlWbGetActive

XlWbClose

Scope: Excel remote control

Closes the current workbook

Declaration:

`XlWbClose ( )`

Parameter:

Description:

Any changes to the workbook which have not already been saved are lost.

Examples:

In a specified folder, all existing Excel fles are opened, displayed and printed out (after user confirmation) in succession.

```
FileListID = FsFileListNew("c:\results", "*.xlsx", 0, 0, 1)
count = FsFileListGetCount(FileListID)
FOR I = 1 TO count
   filename = FsFileListGetName(FileListID, I)
   IF XlWbOpen( filename)
      XlVisible(1)
      IF BoxMessage( "Check", "Print this file?", "?4") = 1
         XlWbPrint()
      END
      XlWbClose()
   END
END
XlQuit()
```

See also:

XlWbOpen, XlWbNew

XlWbExist

Scope: Excel remote control

Checks whether a workbook having the specified name is currently open.

Declaration:

```
XlWbExist ( Name ) -> Result
```

Parameter:

Name	Name of the workbook to be found
Result	
Result	1, if such a workbook existiert, else 0. -1 at fault condition; the error cause can be found by using the function GetLastError().

Description:

The name to be specified here exactly matches the name displayed in the title bar of the associated Excel-window; for documents already saved, this includes the filename extension (e.g. '.xls' or '.xlxs').

Only such workbooks are taken into account which were opened by the Excel instance generated by FAMOS.

Alongside workbooks generated by remote control using XlWbOpen()/XlWbNew(), this also includes files which may have been loaded manually by the user in the user interface made visbile by means of XlVisible().

Examples:

At the beginning of a long evaluation procedure, an Excel-file is loaded and displayed, which at he end of the evaluation and after various updates is to be printed out. Since the used may already have closed the file manually in the meantime, before printing there is a check of whether the desired file is still open.

```
XlWbOpen("c:\results\report.xlsx")
XlVisible(1)
;...
ok = XlWbExist("report.xlsx")
IF ok
   XlWbActivate("report.xlsx")
   XlWbPrint()
END
XlQuit()
```

See also:

XlWbGetActive, XlWbGetName, XlWbGetCount

XlWbGetActive

Scope: Excel remote control

The name of the currently active workbook is found.

Declaration:

```
XlWbGetActive ( ) -> Name
```

Parameter:

Name	
Name	Name of the active workbook. At fault condition, an empty text; the error cause can be determined using the function GetLastError().

Description:

This funciton is occasionally necessary when you have multiple open workbooks at the same time. Most of this Kit's functions affect the active workbook. The active workbook can change if a new workbook is created using XlWbOpen() or XlWbNew(), if a new workbook is explicitly activated using XlWbActivate(), or if the user manually activated another workbook (e.g. by clicking in the corresponding Excel window).

The name found here exactly matches the name displayed in the title bar of the associated Excel window; for already saved documents, including the filename extension (e.g. '.xls' or '.xlxs').

Examples:

At the beginning of an analysis, a new Excel file is created. The name (automatically assigned by Excel) is noted for later use.

```
XlWbNew("")
NewBookName = XlWbGetActive()
```

At the beginning of a long analysis, an Excel file which is to be printed after a variety of updates is opened and displayed. Since the user may have closed the file manually, or loaded another file, the system verifies prior to printout whether the desired file is still active.

```
XlWbOpen("c:\results\report.xlsx")
XlVisible(1)
;...
tx = XlWbGetActive()
IF tx = "report.xlsx"
   XlWbPrint()
END
XlQuit()
```

See also:

XlWbOpen, XlWbNew

XlWbGetCount

Scope: Excel remote control

Gets the number of workbooks currently open.

Declaration:

`XlWbGetCount ( ) -> Amount`

Parameter:

Amount	
Amount	Count of open workbooks. -1 at fault condition; the error cause can be found using the function GetLastError().

Description:

The count determined here only pertains to workbooks opened in an instance of Excel generated by FAMOS.

Alongside workbooks generated by remote control using XlWbOpen()/XlWbNew(), this also includes files which may have been loaded manually by the user in the user interface made visbile by means of XlVisible().

Examples:

At the end of a long evaluation procedure, all open workbooks are enumerated. If the name begins with "Report_", the workbook is printed out.

```
n = XlWbGetCount()
FOR i = 1 to n
   name = XlWbGetName(i)
   IF TLike(name, "report_*", 0)
      XlWbPrint()
   END
END
XlQuit()
```

See also:

XlWbGetActive, XlWbGetName, XlWbExist

XlWbGetName

Scope: Excel remote control

Finds the name of an opened workbook.

Declaration:

```
XlWbGetName ( Index ) -> Name
```

Parameter:

Index	Index of the workbook. Lies between 1 and the count of currently open workbooks.
Name	
Name	Name of the workbook. Empty text at fault condition; the error cause can be found by using the function GetLastError().

Description:

The name found here exactly matches the name displayed in the title bar of the associated Excel window; for already saved documents, including the filename extension (e.g. '.xls' or '.xlxs').

Only such workbooks are taken into account which were opened by the Excel instance generated by FAMOS.

Alongside workbooks generated by remote control using XlWbOpen()/XlWbNew(), this also includes files which may have been loaded manually by the user in the user interface made visbile by means of XlVisible().

The count of open workbooks and this the maximum value for [Index] can be found by using the function XlWbGetCount().

Examples:

At the end of a long evaluation procedure, all open workbooks are enumerated. If the name begins with "Report_", the workbook is printed out.

```
n = XlWbGetCount()
FOR i = 1 to n
   name = XlWbGetName(i)
   IF TLike(name, "report_*", 0)
      XlWbPrint()
   END
END
XlQuit()
```

See also:

XlWbGetActive, XlWbGetCount, XlWbExist

XlWbNew

Scope: Excel remote control

Creates a new workbook.

Declaration:

```
XlWbNew ( TxTemplate ) -> Success
```

Parameter:

TxTemplate	If you specify an empty text, a new workbook with exactly one worksheet is created. You can also enter the complete pathname of an Excel file, which is then used as the template for the new document.
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Unless it has already happened by means of a previous call of XlStart()/ XlWbOpen() /XlWbNew(), a hidden instance of Excel starts, upon which all subsequent commands of this Kit take effect. If desired, use the function XlVisible() to make the main Excel window visible.

At the end of the utilzation of Excel, it is absolutely necessary to call XlQuit() to close any (possibly not visible) Excel instances and thus to free up any no longer required resources.

For the purpose of reading from and writing to Excel files, you can also use the functions FileOpenXLS() and FileOpenXLS2(). These are generally much more powerful and fast, but in return less flexible than the corresponding funcitons in this Kit.

Multithreading: Each execution thread uses its own EXCEL instance. If, for example, EXCEL is started in a parallel sequence function (BEGIN_PARALLEL) using XlWbOpen () and a document is loaded, further access to this document is only permitted within the same sequence function. If the EXCEL instance was not explicitly closed with XlQuit (), it is closed automatically at the end of the sequence function.

Examples:

A new Excel file with a worksheet is created, and a data set is transferred into the table. The second column has the header 'Data', followed by the data set's values. The first column has the header 'Time', followed by the corresponding specifications of the Date/Time. The output format for both columns is set explicitly. The file thus created is then saved.

```
XlWbNew("")
; Fill column with Date/Time
XlSetText("A1", "Time")
time = XlCreateTimeLine(channel1, 1, 0)
XlSetValues( "A2", 0, time, 1)
Count = Leng?(time)
Range = XlBuildA1Ref(2, 1, count, 1)
XlSetCellFormat(Range, "dd.mm.yy hh:mm:ss.0")
; fill data column
XlSetText("B1", "Data")
XlSetValues( "B2", 0, channel1, 0)
Range = XlBuildA1Ref(2, 2, count, 1)
XlSetCellFormat(Range, "0.00")
XlWbSave( "c:\results\report", 0)
XlQuit()
```

A new Excel file is created and filled with the data from a FAMOS data group called 'MyGroup'. The first sheet is filled with some general specifications, then a new worksheet is created for each of the group's channels and the first column is filled with the channel's values.

For cover sheets and data sheets each, a pre-made template is used which comes with fixed texts and cell formatting.

```
IF XlWbNew("c:\templates\firstpage.xlsx")
   XlVisible(1)
   XlSetText( "A3", "Name: Mike Smith")
   XlSetText( "A4", "Date: " + TimeToText( TimeSystem?(),0))
   FOR I = 1 TO GrChanNum?(MyGroup)
      XlSheetAdd( GrChanName?(MyGroup, I), 0, "c:\templates\datapage.xlsx")
      XlSetValues( "A1", 0, MyGroup:[I], 0)
   END
   XlWbSave( "c:\results\report", 0)
   XlQuit()
END
```

See also:

XlWbOpen, XlWbSave

XlWbOpen

Scope: Excel remote control

Opens the specifiedd file in EXCEL format.

Declaration:

`XlWbOpen ( Filename [, Password] ) -> Success`

Parameter:

Filename	Complete pathname of the the file to open
Password	If the workbook is protected, the required password. (optional , Default value: "")
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Description:

Unless it has already happened by means of a previous call of XlStart()/ XlWbOpen() /XlWbNew(), a hidden instance of Excel starts, upon which all subsequent commands of this Kit take effect. If desired, use the function XlVisible() to make the main Excel window visible.

At the end of the utilzation of Excel, it is absolutely necessary to call XlQuit() to close any (possibly not visible) Excel instances and thus to free up any no longer required resources.

For the purpose of reading from and writing to Excel files, you can also use the functions FileOpenXLS() and FileOpenXLS2(). These are generally much more powerful and fast, but in return less flexible than the corresponding funcitons in this Kit.

Multithreading: Each execution thread uses its own EXCEL instance. If, for example, EXCEL is started in a parallel sequence function (BEGIN_PARALLEL) using XlWbOpen () and a document is loaded, further access to this document is only permitted within the same sequence function. If the EXCEL instance was not explicitly closed with XlQuit (), it is closed automatically at the end of the sequence function.

Examples:

An Excel file is opened and data are transferred to 2 columns. The updated file is printed out and saved under a new name.

```
IF NOT (XlWbOpen("c:\Templates\Template.xlsx"))
   BoxMessage("Can't open file", GetLastError(), "!1")
ELSE
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlSetText("C1", "Channel2")
   XlSetValues("C2", 0, Channel2, 0)
   XlWbPrint()
   IF NOT(XlWbSave("c:\Results\Report.xlsx", 0))
      BoxMessage("Can't save file", GetLastError(), "!1")
   END
   XlQuit()
END
```

A Excel file is opened and the worksheet having the title 'Table2' is activated and displayed.

Following confirmation by the user, the data are read into the 2nd column (beginning at Row 3) and transferred to FAMOS.

```
IF XlWbOpen("c:\results\report.xlsx")
   XlVisible(1)
   XlSheetActivate("Table2")
IF BoxMessage( "Check", "Transfer these data?", "?4") = 1
      DataColumn2 = XlGetValues("B3", 0, 0, 0)
   END
   XlQuit()
END
```

See also:

XlWbSave, XlWbNew

XlWbPrint

Scope: Excel remote control

Prints out the current workbook

Declaration:

```
XlWbPrint ( ) -> Success
```

Parameter:

Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

An Excel file is opened and the data are transferred into 2 columns. The updated file is printed out.

```
IF NOT (XlWbOpen("c:\Templates\Template.xlsx"))
   BoxMessage("Can't open file", GetLastError(), "!1")
ELSE
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlSetText("C1", "Channel2")
   XlSetValues("C2", 0, Channel2, 0)
   XlWbPrint()
   XlQuit()
END
```

See also:

XlWbOpen, XlSheetPrint

XlWbSave

Scope: Excel remote control

Saves the current workbook.

Declaration:

```
XlWbSave ( TxFilename, Reserved ) -> Success
```

Parameter:

TxFilename	Filename under which to save the workbook. if you specify an empty text here, the file is saved under its current name.
Reserved	Reserved parameter, always set to 0
Success	
Success	Success of the function: 1, if the function could be performed successfully; 0 in case of error. In case of error, the cause of the error can be found using the function GetLastError().

Examples:

An Excel file is opened and data are transferred to 2 columns. The updated file is printed out and saved under a new name.

```
IF NOT (XlWbOpen("c:\Templates\Template.xlsx"))
   BoxMessage("Can't open file", GetLastError(), "!1")
ELSE
   XlSetText("B1", "Channel1")
   XlSetValues("B2", 0, Channel1, 0)
   XlSetText("C1", "Channel2")
   XlSetValues("C2", 0, Channel2, 0)
   XlWbPrint()
   IF NOT(XlWbSave("c:\Results\Report.xlsx", 0))
      BoxMessage("Can't save file", GetLastError(), "!1")
   END
   XlQuit()
END
```

A new Excel file is created and filled with the data from a FAMOS data group called 'MyGroup'. The first sheet is filled with some general specifications, then a new worksheet is created for each of the group's channels and the first column is filled with the channel's values.

For cover sheets and data sheets each, a pre-made template is used which comes with fixed texts and cell formatting.

```
IF XlWbNew("c:\templates\firstpage.xlsx")
   XlVisible(1)
   XlSetText( "A3", "Name: Mike Smith")
   XlSetText( "A4", "Date: " + TimeToText( TimeSystem?(),0))
   FOR I = 1 TO GrChanNum?(MyGroup)
      XlSheetAdd( GrChanName?(MyGroup, I), 0, "c:\templates\datapage.xlsx")
      XlSetValues( "A1", 0, MyGroup:[I], 0)
   END
   XlWbSave( "c:\results\report", 0)
   XlQuit()
END
```

See also:

XlWbOpen, XlWbNew

xMax

Returns the x-positions of all relative maxima which lie above a specified threshold.

Declaration:

```
xMax ( Data, SvfLimit ) -> XMaxima
```

Parameter:

Data	Data set examined. Allowed types: [ND],[XY].
SvfLimit	Threshold
XMaxima	
XMaxima	The X-coordinates of relative maxima above [SvThreshold] found.

Description:

The results are the x-coordinates of all relative maxima greater than a specified threshold.

If NyData is an XY-data set, it must have a monotonous time or x-track.

- The Value() function can be used to determine the associated y-coordinates.
- Set the threshold to -1e100 to determine all relative maxima.
- If the value of a relative maximum is held over more than one data point in a data set, only the first x-coordinate is returned.

Examples:

```
NDxmaxi = xMax(Smo5(NDdata), 10 'A')
```

The x-coordinates of all relative maxima in the data set NwData greater than 10A form the result. The data set is somewhat smoothed first to suppress insignificant relative maxima.

```
NDymaxi = Value(NDdata, xMax(NDdata, -1e100))
```

The result is the y-coordinates of all relative maxima of the NDdata data set that are greater than -1e100. In practice, all relative maxima are determined in this way.

```
xMini = xMax( -NDdata, 0)
xyMini = xyOf( xMini, Value( NDdata, xMini))
```

The result is an XY data set that contains all the relative minima of NDdata with a negative y value.

See also:

Value2, All0, Top

XOff

Specifies a data set's offset in the x-direction

Declaration:

```
XOff ( Data, SvXOffset ) -> Result
```

Parameter:

Data	Data set whose x-offset is to be specified
SvXOffset	New x-offset.
Result	
Result	Data set copy with new offset

Description:

A copy of the input data is generated and the specified x-offset is entered. All other numerical and characteristic values remain unaffected.

The x-offset is the x-coordinate of the data set's first data point. With measurement data captured plotted over time, this characteristic value is ofter referred to as the pretrigger.

- The unit of the new x-offset should match the x-unit of the data set passed.
- The size of the new Delta-X should not be too many orders of magnitude greater than the sampling interval. Otherwise, the resolution may not be sufficient to represent differences between the x-coordinates of the data set's data points.

Examples:

The x-coordinate of a histogram's value, which was imported in the form of ASCII-data without time base information , is set to -128.

```
NDcorrect = XOff(NDhisto, -128)
```

See also:

XOff?, XDel, Leng, XOFFSET

XOff?

A data set's offset in the x-direction

Declaration:

```
XOff? ( Data ) -> SvXOffset
```

Parameter:

Data	Data set whose x-offset is to be determined
SvXOffset	
SvXOffset	x-offset

Description:

The x-offset is the x-coordinate of the data set's first data point. With measurement data captured plotted over time, this characteristic value is ofter referred to as the pretrigger.

- The result has the x-unit of the data set passed.

Examples:

The x-coordinate of the data set's 2nd data point is determined:

```
xOf2ndSample = XOff?(NDdata) + XDel?(NDdata)
```

See also:

XOff, XDel?, Leng?

XOFFSET

Set X-offset

Declaration:

```
XOFFSET VariableName SvXOffset
```

Parameter:

VariableName	Name of the variable whose x-offset is to be re-apportioned
SvXOffset	Distance of the first data point from the origin, in x-units

Description

The command XOFFSET is obsolete; instead of it, the function XOff() shoud be used in newly created sequences.

The parameter's x-offset is set to a new value.

Examples:

```
XUNIT Temp s
XOFFSET Temp 10
```

First the x-unit "s" is assigned to the variable "Temp". The command following has the effect that the first data point of the variable "Temp" lies at "10 s".

See also:

XOff, XOff?, XDel

XOR

Logical "Exclusive-OR"-operator

Declaration:

```
Operand1 XOR Operand2 -> ZeroOrOne
```

Parameter:

Operand1	First single value or data set to be compared.
Operand2	Second single value or data set to be compared.
ZeroOrOne	Result, 0 or 1 (or data set consisting of 0 and 1).

Description

"Exclusive-OR" operator applied to two numbers. The result is 1, if **exactly** one of the operands equals 0. Else, the result is 0.

The operator can be applied to single values or data sets. With data sets, the operation is applied data point by data point..

If one parameter's type is XY, the other parameter must be a single value.

Both parameters may be structured (events/segments); but the respective counterpart parameter must then either have exactly the same structure (same segment length, event-count and -length) or it must be a single value.

Examples:

The exclusive disjunction operation is applied to two digital data sets. The result data set's value is 1 everywhere that the two operand data sets' values are different.

```
Result = (DigChannel1 XOR DigChannel2)
```

See also:

AND, NOT, OR

XUNIT

Set X-unit

Declaration:

```
XUNIT VariableName NewUnit
```

Parameter:

VariableName	Name of the variable to which a new x-unit is to be assigned
NewUnit	New x-unit

Description

The command XUNIT is obsolete; instead of it the function SetUnit() should be used in newly created sequences.

The unit of the x-axis of a waveform is redefined, i.e. the data set is assigned a new x-unit.

The waveform is the first parameter, the unit is the second.

The unit is specified without quotation marks. If there is no second parameter, the unit of the x-axis is deleted.

In complex data sets, only the unit of the first component is changed. In xy-data sets, the unit of the x-components is set.

Examples:

```
ASCII
FileLoad("test.dat", "", 0)
h = Histo(test)
XUNIT h V
```

The unit of the x-axis in a histogram is set to "V". This information was not available in the ASCII file.

```
XUNIT data
```

After this command is executed, the variable has no unit on the x-axis.

See also:

SetUnit, Unit?, YUNIT, XDELTA

XY

A data set given by its (X,Y)-coordinates is resampled at a fixed sampling rate.

Declaration:

```
XY ( XData, YData ) -> Result
```

Parameter:

XData	The time- or x-values for the data set to be resampled. Type: [ND]
YData	The y-values for the data set to be resampled. Type: [ND]
Result	
Result	The resulting equidistantly sampled data set

Description:

This function uses linear resampling at a constant sampling rate (delta-X) to generate an equidistantly sampled data set from an input data set's X- and Y-coordinates. The delta-X for the resampling is determined automatically from the X-component.

In general, the functions XYdt() and XYdt2() are more appropriate.. With this functions, you can specify sampling rate and interpolation type.

The function should be used only for strictly monotonous (parts of) data sets. The XY function assumes that the number of points, the sampling rate and the x-offset are the same.

- If the sampling rates or the x-offsets of the components differ, use the function RSamp or RSampEx to equalize them.
- When the components have different lengths, the shorter length is used for both components.
- If the x-component is not strictly monotonous, the XY function cannot work properly. The result in imc FAMOS must always be a definite curve.
- The x-components may not be constant, since the created data set must always have a time range.
- The resolution of the created data set corresponds to the smallest distance between adjacent xcoordinates. A maximum of 10e6 result points are generated.
- The XY-function in the Curve Window works somewhat differently (real-time and genuine superpositioning of the components).

Examples:

NwX and NwY are example data sets. NwX increases monotonously and NwY is the first harmonic of the sine function. The XY-superposition of both components yields a semicircle.)

```
NDx = -cos(Ramp(0, PI / 100, 100))
NDy = sin(Ramp(0, PI / 100, 100))
NDhalfCircle = XY(NDx, NDy)
```

See also:

XYdt2, XYdt, RSampEx, XYof

XYdt

A data set given by its (X,Y)-coordinates is resampled at a fixed sampling rate.

Declaration:

```
XYdt ( XData, YData, Svdt ) -> Result
```

Parameter:

XData	The time- or x-values for the data set to be resampled. Type: [ND]
YData	The y-values for the data set to be resampled. Type: [ND]
Svdt	Resulting sampling interval or delta-x of the calculated (equidistantly sampled) result data set
Result	
Result	The resulting equidistantly sampled data set

Description:

This function uses resampling at a constant sampling rate (or delta-x) to generate from one data set's X- and Y-coordinates a new, equidistantly sampled data set.

Waveforms to be calculated using mathematical functions generally must have the same sampling rate and be equidistant. Here equidistant means that the distance between measurement points is the same for all values in the data set. However, this is not always the case in long-term measurement or reduced data sets. The measurement values are often coordinate pairs for time and measurement values.

In order to process such data sets, they first must be converted to equidistant data sets using this function. The sampling rate of the resulting data set can be specified.

The new data set is calculated from x- and y-coordinates by linear interpolation. This is as if a set timeslot pattern with the new sampling rate is put on the curve and then all points on the curve which cut the grid are marked and transferred to the new curve.

If you wish to resample the data set with constant interpolation, you can use the function XYdt2().

- The X data set must be monotonically increasing.

Information is lost when a data set is resampled. If a high sampling rate is used for resampling, the minima and maxima of the curve may be distorted.

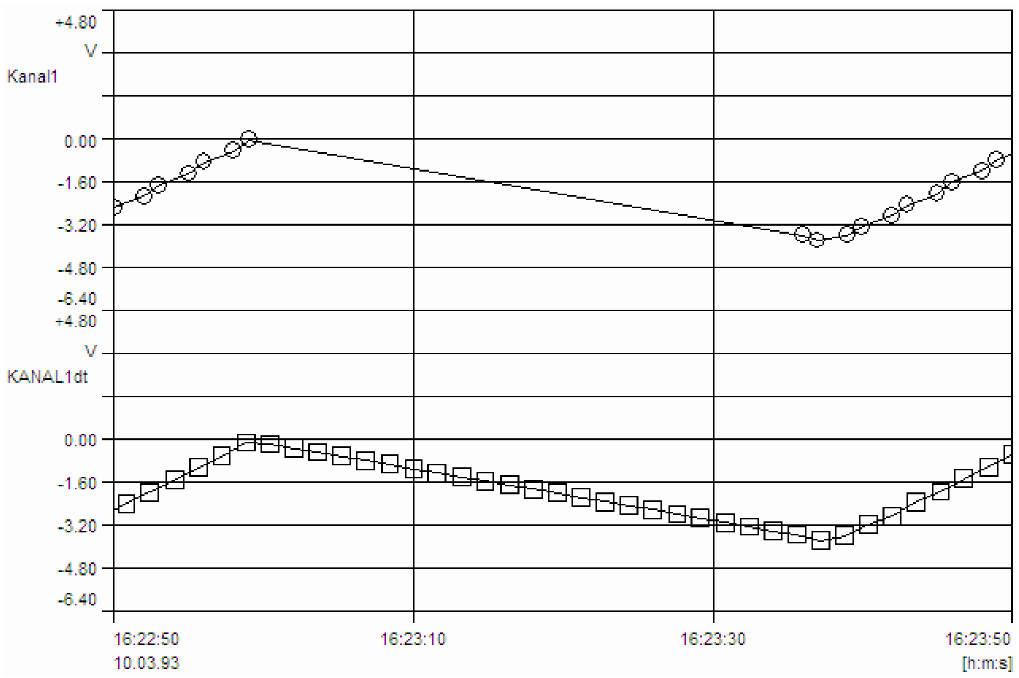
Examples:

If the data set to be sampled is an XY-data set with two components, this data set may be used with information about the component (.X, .Y) characteristics.

```
NormalSet = XYdt(XYSet.X, XYSet.Y, 0.1)
```

In this example, the equidistant data set "Channel1dt" is generated from the data sets "Channel1" and "Time" is generated with a sampling rate of 1.6 seconds.

```
Channel1dt = XYdt(Time, Channel1, 1.6)
```



The figure shows the upper curve of the original data set. Each recorded measurement value is marked by a circle. The lower curve shows the result of resampling. Here, each value is marked by a square.

See also:

XYdt2, RSampEx, Value2, XYof

XYdt2

A data set given by its (X,Y)-coordinates is resampled at a fixed sampling rate.

Declaration:

```
XYdt2 ( XData, YData, Svdt, Svnterpolation ) -> Result
```

Parameter:

XData	The time- or x-values for the data set to be resampled. Type: [ND]
YData	The y-values for the data set to be resampled. Type: [ND]
Svdt	Resulting sampling interval or delta-x of the calculated (equidistantly sampled) result data set
Svnterpolation	If an X-coordinate does not exactly coincide with one of the data set's X-coordinates, the result is interpolated as follows:
	0 : Linear. The input data set is interpolated linearly.
	1 : Constant, preceding value. The input data set is subject to level interpolation, i.e. each value is kept constant until a new sample value becomes effective. Thus, the result value is the input data set's value whose x-coordinate is immediately BEFORE the x-coordinate examined.
	2 : Constant, closest value. The result value is the input data set's value whose x-coordinate is CLOSEST to the x-coordinate examined.
Result	
Result	The resulting equidistantly sampled data set

Description:

This function uses resampling at a constant sampling rate (or delta-x) to generate from one data set's X- and Y-coordinates a new, equidistantly sampled data set.

- The X-data set must be monotonically increasing.
- The lengths of the parameter data sets for X and Y should be equal. Otherwise, 'extra' values in the longer data set will be ignored.
- **Linear** interpolation is generally used for continuous input signals. This function's behavior thus matches that of XYdt().
- The **constant** interpolation styles are most sensible to use when the input signal is comprised of predefined discrete values. This applies to digital data , or measured data which by nature can only take integer values (e.g. the current gear in a transmission system). The resulting data set consists only of values which also exist in the input data set, and has the same data format.
- Linear interpolation with digital input data is not posible; the interpolation parameter is automatically corrected to the value 1 (constant interpolation).
- With the constant interpolation styles, the result has the same data format as the Y-data provided.
- Resampling a data set causes the loss of some informationen. If you resample at a long sampling interval, some minima and maxima in the curve may be distorted.

Examples:

An XY-data set is resampled at 0.1s. If needed, linear interpolation is applied.

```
Signal01 = XYdt2(Signal.X, Signal.Y, 0.1, 0)
```

A vehicle's gear currently engaged is recorded. Each time the gear is shifted, the gear engaged (<gear>, taking the values 1 - 5) and the time <t> (since starting measurement) are saved. To simplify subsequent calculations, these two input data sets are used to generate a new data set having 0.5 s equidistant sampling.

```
gear_resampled = XYdt2(t, gear, 0.5, 1)
```

See also:

XYdt, RSampEx, Value2, XYof

XYof

From the X- and Y-components, an XY-data set is formed.

Declaration:

```
XYof ( ComponentX, ComponentY ) -> XYData
```

Parameter:

ComponentX	Data set from which the X-component is derived [ND]
ComponentY	Data set from which the Y-component is derived [ND]
XYData	
XYData	Resulting XY-data set

Description:

Joins two real data sets to form an XY-data set with two components.

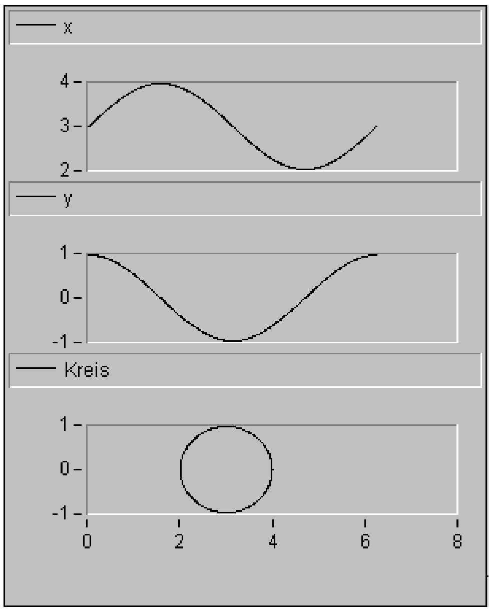
When the parameter sets have different lengths, the Y-track is truncated accordingly or filled with zeroes (for scalable integer data formats, the unscaled numerical value is set to 0), since X- and Y-components must have the same length.

Both parameters may be structured (events/ segments), however, in that case, the respective other parameter must have exactly the same structure (same segment length, event-count and -length).

Examples:

Generating an XY-data set whose value pairs form a circle:

```
x = sin(Ramp(0, 2*PI/ 360, 360)) + 3
y = cos(Ramp(0, 2*PI/ 360, 360))
circle = XYof(x, y)
```



See also:

CmpX, CmpY, XYdt, XYdt2

YUNIT

Sets Y-unit

Declaration:

```
YUNIT VariableName NewUnit
```

Parameter:

VariableName	Name of the variable to which a new y-unit is to be assigned.
NewUnit	New y-unit

Description

The command YUNIT is obsolete; instead of it, the function SetUnit() should be used in newly created sequences.

The unit of the y-axis in a data set is redefined, i.e. the data set is assigned a new y-unit.

The waveform is the first parameter, the unit is the second.

The unit is specified without quotation marks. If there is no second parameter, the unit of the x-axis is deleted.

With complex data sets, only the unit of the first component is changed. With XY-data sets, the unit of the Y-component is set.

Examples:

```
YUNIT Data A
```

The y-axis unit of the data set "Data" is set to "A".

```
YUNIT Data
```

Now the variable "Data" no longer has any units along the y-axis.

See also:

SetUnit, Unit?, XUNIT

ZDel?

The increment in the z-direction (Delta-Z) is determined

Declaration:

`ZDel? ( Data ) -> SvZDelta`

Parameter:

Data	Data set whose z-increment is to be determined
SvZDelta	
SvZDelta	Delta-Z

Description:

A data set's increment in the z-direction is set. Examples of the use of this value include for the scaling of the z-axis in 3D-displays of segmented data.

See also:

SetZDel, ZOff?, Leng?

ZOff?

The offset in the z-direction is determined.

Declaration:

```
ZOff? ( Data ) -> SvZOffset
```

Parameter:

Data	Data set whose z-offset is to be determined
SvZOffset	
SvZOffset	z-offset

Description:

The initial value in the z-direction is set. Examples of the use of this value include for the scaling of the z-axis in 3D-displays of segmented data.

See also:

SetZOff, ZDel?, SetZDel

ZoomSpectrumChirpZ

Available in: Professional Edition and above (SpectrumAnalysis-Kit)

The Chirp-z transformation is applied to the time-based signal. For this purpose, the signal's RMS-spectrum is determined in a selected frequency range. The time signal's length need not be a power of two. The spectrum can be determined with any resolution from 0 Hz up to half of the sampling frequency.

Declaration:

```
ZoomSpectrumChirpZ ( Time-based signal, FreqMin, FreqMax, FreqDelta, WindowType ) -> Result
```

Parameter:

Time-based signal	The time plot of the signal from which the spectrum is to be computed
FreqMin	Lower end of frequency range, >=0
FreqMax	Upper end of frequency range, <= half of sampling frequency
FreqDelta	Frequency line distance, >= 0
WindowType	Windowing function for the FFT used
	0 : Rectangle
	1 : Hamming
	2 : Hanning
	3 : Blackman
	4 : Blackman / Harris
	5 : Flat Top
Result	
Result	The spectrum determined is a complex data set with magnitude and phase. The magnitude of the individual frequency lines is stated as an RMS-value.

Description:

The number of frequency lines determined is:

- Count = 1 + (FreqMax - FreqMin) / FreqDelta

Toward this end, we round up. FreqMin is always adhered to as the lower limit.

For the following parameters, the time data's general DFT (discrete Fourier transformation) is determined, but with a fast algorithm. Refer also to DFTSpectrum():

- FreqMin = 0
- FreqMax = sampling frequency / 2
- FreqDelta = FreqMax / points_in_time_singal / 2

If only one frequency line is to be determined, select the following:

- FreqDelta = 0.0
- FreqMin = FreqMax

In all other cases: FreqDelta > 0.0.

Examples:

A time signal's spectrum in a narrow range around 50Hz is to be determined:

```
Spectrum = ZoomSpectrumChirpZ ( t, 48, 52, 0.01, 0 )
```

The DFT of a time signal t is to be determined:

```
fmax = 0.5 / xdel?(t) ; highest frequency
fdelta = fmax / leng?(t) / 2 ; frequency line distance
Spectrum = ZoomSpectrumChirpZ ( t, 0, fmax, fdelta, 0 )
```

See also:

FFT, DFTSpectrum

PowerPoint-Kit (Overview)

Available in: Professional Edition and above

Overview

This kit provides functions which govern Microsoft PowerPoint.

You can create a presentation.

Slides can be inserted from a different presentation into the current presentation. Slides can be duplicated, moved and deleted.

In the presentation, it is possible to replace text boxes, table contents, or pictures with content from FAMOS.

The PowerPoint-Kit requires imc FAMOS 7.3 or higher.

The prerequiste is that a supported version of PowerPoint-Version is installed on the same computer.

At this time, PowerPoint 2010, 2013 and 2016 are supported.

System requirements and installation

The Powerpoint-Kit is included in the 'Professional' and 'Enterprise' editions of imc FAMOS.

In order to be able to use the Powerpoint-Kit in FAMOS, it must be registered. During the normal installation procedure, this happens automatically and the contained functions are shown in the FAMOS function list under "Presentation / Powerpoint-Kit".

If the functions are not available, please check the registered Kits with the menu command "Extra / Options / Extensions / Kits". This list should also include an entry "PowerPoint-Kit [imcPowerPointKit.dll]". If this entry isn't there, check whether the file "imcPowerPointKit.dll" is in the same folder as the file "Famos.exe". If necessary, contact the Hotline.

Preparing a presentation for use by the imc PowerPoint Kit

Some additional preparations must be made in order to use a PowerPoint file by the imc PowerPoint Kit. The kit can replace the content of text boxes, tables and pictures. To do this, these objects must be identified in PowerPoint.

This identification is provided by means of the shape's alternative text. To each object in a slide, it is possible to assign an alternative text.

The Kit-functions search through all shapes in a slide for the alternative text. If a shape has been found, the text, the table cell content, or the picture is replaced.

In PowerPoint you can enter the alternative text in the following way:

1. On the slide, select shape object
2. Right-click the mouse within the element and select "Format shape" or "Format graphic".
3. Click on "Size and Properties", then on "Alt Text"
4. In the box "Description" enter a text as the designation, e.g. FAMOS_Text1. This designatory text is used in Kit-functions to find the shape object.
5. **Important!** Do **not** enter the designatory text in the box "Title".
6. If you wish to use the alternative text for it's original function (barrier-free PowerPoint), then first enter the designating text in the box "Description", followed by a semicolon (;). After that enter the text intended for the barrier free PowerPoint. The semicolon is not part of the designating text.

Multithreading

Each Execution thread uses its own Powerpoint instance. If, for example, Powerpoint is started in a parallel sequence function (BEGIN_PARALLEL) using PptOpenPresentation () and a document is loaded, further access to this document is only permitted within the same sequence function. If the Powerpoint instance was not explicitly closed with PptClosePresentation (), it is closed automatically at the end of the sequence function.

Copyright

Microsoft PowerPoint is a registered trademark of Microsoft Corporation, USA.

To access PowerPoint, the kit uses the PowerPointApi.dll, OfficeApi.dll, VBIDEApi.dll and NetOffice.dll assemblies from the NetOffice 1.7.3 package. The package was published under the MIT Licence copyright © 2012 Sebastian Lange.

(c) 2024 imc Meßsysteme GmbH

R-Kit (Overview)

Available in: Professional Edition and above

Overview

This Kit provides functions for connecting the R-System with FAMOS.

The functions provide a bridge to the R-system.

R is a programming system for statistical analysis. R contains a very large library of functions which can be applied in statistics analyses. R is freely available.

The R-Kit contains functions for setting and reading R variables and for executing R scripts.

Prerequisites

The R-Kit is included in the 'Professional' and 'Enterprise' editions of imc FAMOS and requires at least version 7.3.

The R-System must be installed on the PC. The minimum required version is 3.3.2. It is available for download and installation on the Internet at http://www.r-project.org. During installation, it is advisable to retain all of the suggested default settings.

In order to use the R-System in an effective way, some R-programming skills are needed.

If there is no R- System or an R version on the PC that is smaller as 3.3.2, the R version 3.3.2 is also installed. If higher versions of the R system are present on the PC, no R- System is installed.

Registration in FAMOS

In order to be able to use the R-Kit in FAMOS, it must be registered. During the normal installation procedure, this happens automatically and the contained functions are shown in the FAMOS function list under "Analysis / Statistics / R-Kit".

If the functions are not available, please check the registered Kits with the menu command "Extra / Options / Extensions / Kits". This list should also include an entry "R- Kit [imcRKit.dll]". If this entry isn't there, check whether the file "ImcRKit.dll" is in the same folder as the file "Famos.exe". If necessary, contact the Hotline.

Multithreading

All functions of the R-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are first moved internally to the FAMOS Main thread and executed from there, since the R runtime environment does not support parallel calls from different threads.

General Notes

In R, the function names and variable names are case-sensitive.

Incorrect execution of an R-Kit function causes the FAMOS-sequence to abort.

Copyright

R is is released under the GNU General Public License (GPL), Version 2.

R's source data are provided under https://cran.r-project.org/sources.html.

In order to access the R-System, the R-Kit uses these assemblies:

R.Net and RDotNet.NativeLibrary : Copyright (c) 2010, RecycleBin

DynamicInterop : Copyright (c) 2015 Jean-Michel Perraud; Copyright (c) 2014 Daniel Collins, CSIRO;Copyright (c) 2013 Kosei, evolvedmicrobe

The examples of the t-test are sourced from Friedrich Leisch: Introduction to Inductive Statistics http://groll.userweb.mwn.de/StatistikII_SS09/VL_Folien_3.pdf.

The data for the example in the Chi-square goodness-of-fit test were sourced from Dipl.-Math. Uwe Gorbracht, published under http://rechen-fuchs.de/chi-quadrat-anpassungstest-wuerfel-beispiel/.

The data for the example of sales distributions are sourced from an internet page belonging to the University of St.Gallen, University of Basel, FhbB, 2007, http://www.mri.imh.unisg.ch/Analysemethoden/Datenanalyse/Induktiv/univariat/chi2anpassungstest.html.

For the test of independence between eye and hair color, data from Annette Bieniusa's "Programmieren in Anwendungen" of the Technical University of Kaiserslautern, https://softech.informatik.uni-kl.de/homepage/teaching/PIA_SS14/7_Hypotheses.pdf were used.

For the test of independence regarding occupations, the data are sourced from http://wikis.fu-berlin.de/pages/viewpage.action?pageId=712409813.

(c) 2024 imc Meßsysteme GmbH

Python-Kit (Overview)

Available in: imc FAMOS 2022, Professional Edition and above

The Python-Kit supplies functions which provide a bridge to the programming language **Python**.

Python is an all-purpose object-oriented programming language which is very prevalent in the fields of education, science and technology due to its accessibility to beginners, its platform independence, expansion capability and open source availability. A large selection of libraries is available, for instance for numeric calculations, visualized data processing, image analysis and all the way to machine learning. Familiar expansion libraries for the technical/scientific sector include, for example: **NumPy**, **SciPy** and **TensorFlow**.

FAMOS generates an embedded instance of the Python runtime environment, which provides an interpreter for the Python programming language.

There are functions available for reading and writing Python variables, as well as for running Python functions, code lines or whole programs.

System prerequisites:

FAMOS:

- The Python-Kit requires **imc FAMOS 2022**, Professional Edition, or a higher version.

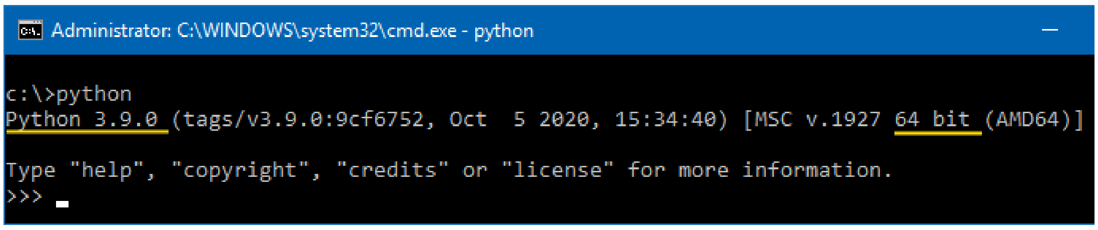
Python:

- On the PC **a supported version of Python must be installed.**
- Support is provided exclusively for the Python reference implementation of the "Python Software Foundation" (CPython) in one of the versions listed below, which can be downloaded and installed at https://www.python.org.
- Compatible **CPython-versions: 3.8 (64Bit), 3.9 (64Bit), 3.10 (64Bit), 3.11 (64Bit)**
- The path to the Python installation directory should be included in the **PATH** environment variable.

NumPy (optional):

- The FAMOS-Python bridge (optionally) offers special support for data types, which are defined in the expansion library **"NumPy"** (https://numpy.org).
- Compatible **1.19 (64Bit) ... 1.23 (64Bit)**

Tip: For a quick check of whether an appropriate Python-version is installed, you can simply enter "python" in the Windows command prompt. If Python is installed, the Interpreter starts and indicates the version.



```
c:\>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)]

Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Multithreading:

All functions of the Python-Kit can be called in any execution thread (BEGIN_PARALLEL), but have a global and cross-thread effect. All calls are initially moved internally to the FAMOS main thread and executed from there, since the Python Runtime Environment does not support parallel calls from different threads.

(c) 2024 imc Meßsysteme GmbH

ASAM-ODS-Kit (Overview)

Available in: Enterprise Edition and above

Overview

The ASAM-ODS plug-in for FAMOS consists of 2 parts: the actual Browser plug-in with its user interface for working manually with ODS-conforming data storage, and the ODS-Kit. This is a collection of functions containing basic functions for automated access to ODS servers for use in FAMOS sequences.

The description of the ODS-Kit functions is found in this section. Introductory comments on the ASAM-ODS standard, on the server types supported and on operation of the plug-in user interface are found in Chapter "Plugin".

System requirements and installation

The ASAM-ODS-Kit is included in the 'Enterprise' edition of imc FAMOS.

In order to be able to use the ASAM-ODS-Kit in FAMOS, it must be registered. During the normal installation procedure, this happens automatically and the contained functions are shown in the FAMOS function list under "Databases / ASAM-ODS-Kit".

If the functions are not available, please check the registered Kits with the menu command "Extra / Options / Extensions / Kits". This list should also include an entry "ASAM-ODS-Kit [ImcOds02.dll]". If this entry isn't there, check whether the file "imcOds02.dll" is in the same folder as the file "Famos.exe". If necessary, contact the Hotline.

Multithreading

All functions of the ODS-Kit may only be called in the standard execution thread. A call within a BEGIN_PARALLEL block (i.e. within sequence functions that are executed in a separate thread) is not permitted.

(c) 2024 Test & Measurement GmbH

VideoPlayer-Kit (Overview)

Available in: Professional Edition and above

The Video Player-Kit is a component of the Video Player Plug-in for playback of video files. It can be used both for remote control of the Plug-in and for governing Video Player elements in FAMOS-Dat Browser panel. Using the Kit's functions, it is possible to automate routines, for which all necessary functions for loading video files, controlling the playback and setting and getting relevant parameters are available.

System requirements and installation

The Video-Kit is included in the 'Professional' and 'Enterprise' editions of imc FAMOS and installs together with the Video Player Plug-in.

In order to be able to use the Video-Kit in FAMOS, it must be registered. During the normal installation procedure, this happens automatically and the contained functions are shown in the FAMOS function list under "Presentation / Video".

If the functions are not available, please check the registered Kits with the menu command "Extra / Options / Extensions / Kits". This list should also include an entry "Video-Player [ImcVpl02.dll]". If this entry isn't there, check whether the file "ImcVpl02.dll" is in the same folder as the file "Famos.exe". If necessary, contact the Hotline.

Multithreading

The functions of the Video kit can be called in every Execution thread and have a global effect.

Application example

In the following sequence, the sample file Crash.avi is opened in the Plug-in window and linked with the data set Crash.dat. Next, the video file's parameters are adjusted so that the flash of light in the picture's background coincides with the maximum value of the variable CrashTest:Light and the smashing of the windshield coincides with the maximum value of the variable CrashTest:Acceleration.

```
;Demo-sequence: VideoKit\Adjust.seq ;Shows how the Video-parameters are adjusted
;Load measurement data
LADEN "c:\Famos\VideoKit\Crash.dat"

;Open curve window
CvConfig(CrashTest:Licht,"c:\Famos4\VideoKit\Crash.ccv")

;Load video file
err=VpVideoLoad("c:\Famos\VideoKit\Crash.avi",0)
;Link curve window and video
err=VpSetLink(CrashTest:Licht,0)

;adapt parameters for synchronized display of video and curve plot
err=VpSetXOffset(0.185,0)
err=VpSetRecordRate(32,0)

;Play video back slowly
err=VpSetPlayRate(1.5,0)
err=VpSetPosFrames(15,0)
err=VpPlay(0)

;Delete unnecessary variable
DELETE err
```

It is even easier to solve a similar assignment in Data Browser panels. A Video Player element can be placed on a Panel page together with a curve window and both elements can be linked for synchronized playback already during the Design process. Configuration of the curve window can also be accomplished completely when the page is in the Design stage. This means that it would no longer be necessary to call the functions VpSetLink() and CvConfig() at runtime. In the folowing example, some playback parameters are set, but these could also be configured with the Video Player element during the page's Design stage.

```
;Load measurement data
LADEN "c:\Famos\VideoKit\Crash.dat"

;Load Panel with curve window a Video Player
DbLoadPanel("C:\VIDEO\VIDEO.PANEL", 0)

;Load video file (if not already permanently specified during the Design process)
err=VpVideoLoad("c:\Famos\VideoKit\Crash.avi",1)

;adapt parameters for synchronized display of video and curve plot
err=VpSetXOffset(0.185,1)
err=VpSetRecordRate(32,1)

;Play video back slowly
err=VpSetPlayRate(1.5,1)
err=VpSetPosFrames(15,1)
err=VpPlay(1)
```

```
;Delete unnecessary variable
DELETE err
```