Manual

TC3 Condition Monitoring

TwinCAT 3

Version: 1.4
Date: 2018-10-17
Order No.: TF3600

BECKHOFF

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with the applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® and XTS® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, DE102004044764, DE102007017835
with corresponding applications or registrations in various other countries.

The TwinCAT Technology is covered, including but not limited to the following patent applications and patents:
EP0851348, US6167425 with corresponding applications or registrations in various other countries.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

2 Overview

Beckhoff offers a toolbox consisting of hardware and software components for implementing a Condition Monitoring system that is integrated in the control system. The benefit of the Beckhoff solution is integration into the standard machine control system, thereby avoiding additional subsystems with complex cross communication. Machine control and Condition Monitoring run on the same platform and can be programmed with the same engineering tools, and they both use EtherCAT as common fieldbus system.

The TwinCAT Condition Monitoring Library forms a significant part of the software toolbox. Various mathematical algorithms are available as PLC function blocks.

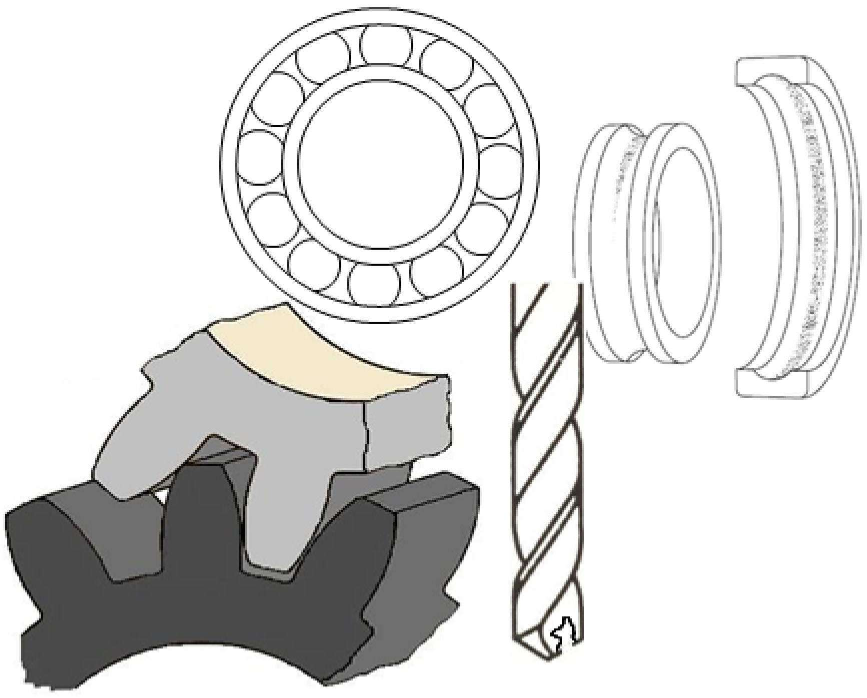
Product information

The current version of the TwinCAT 3 Condition Monitoring library is available as download. The PLC library provides different algorithms for data analysis.
Multi task applications are recommended. The data communication between different tasks and CPU cores are done by the mechanism of the library.

Product components

The TF360x Condition Monitoring product consists of the following components:

- **PLC libraries:** Tc3_CM.compiled-library, Tc3_CM_Base.compiled-library and Tc3_MultiArray.compiled-library
- **Drivers:** TcCM.sys and TcMultiArray.sys



Product features

The table shows the functionalities of the Condition Monitoring Library for the corresponding product level.

Algorithms/Features:	**TF3600 Condition Monitoring Level 1**
Signal-frame processing and inter-task communication	✓
Power Spectrum	✓
Magnitude Spectrum	✓
Signal envelope	✓
Envelope Spectrum	✓
Power Cepstrum	✓
Fast-Fourier-Transform of real signal	✓
Fast-Fourier-Transform of complex signal	✓
Instantaneous Frequency	✓
Instantaneous Phase	✓
Analytic Signal	✓
Crest Factor	✓
Moment Coefficients (mean, standard deviation, skewness, excess kurtosis)	✓
Histogram	✓
Time based RMS	✓
(Time-)Integrated RMS	✓
Multiband RMS	✓
Quantiles	✓
Discrete Classification	✓
Watch Upper Thresholds	✓
ArgSort	✓
Downsampling	✓

2.1 Introduction

For users without previous experience of Condition Monitoring and signal processing, we strongly recommend consulting additional reference material, to complement this documentation. See list of references at the end of this section.

Basic signal processing concepts, in particular Fourier analysis and statistics, are introduced below. This does not include programming details and is limited to a description of the interfaces and functions of the algorithms used in the Condition Monitoring library.

What you will learn:

- How does a frequency analysis work?

- How does a seamless analysis of a continuous data stream work?
- How do I analyze a time segment, and how do I trigger an analysis?
- How to scale a spectrum, and why is this important?
- How to obtain statistically resilient results when measuring signals are affected by noise or interference?

2.1.1 Fourier analysis

Introduction

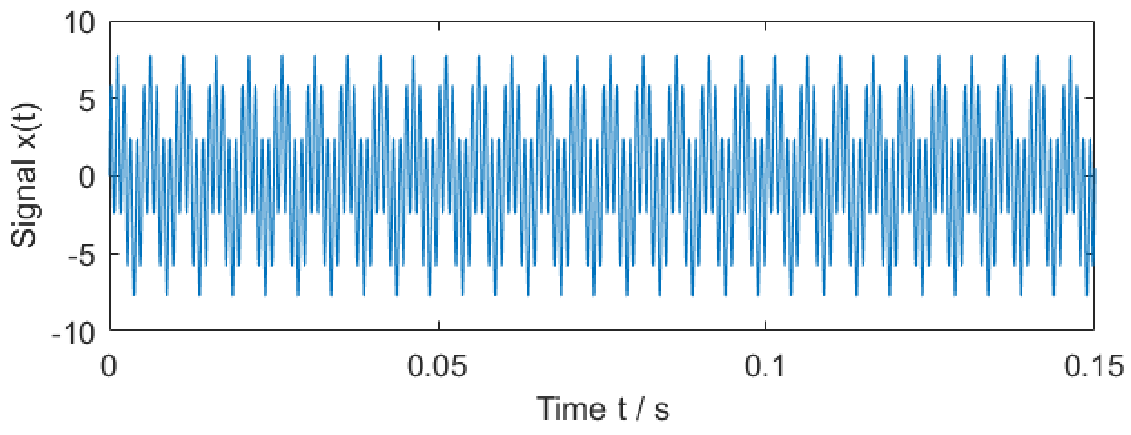
The most important frequency analysis method is the Fourier analysis. The fundamental significance of the Fourier analysis arises from the fact that it decomposes a signal *x(t)* into superimposed sine and cosine vibrations. The result of this transformation is referred to as signal spectrum or simply spectrum. Definition of the Fourier transformation:

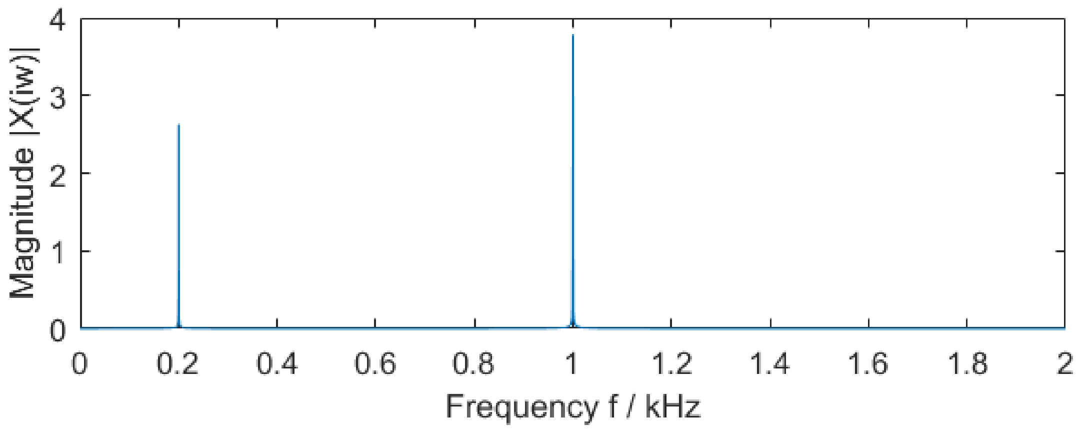
$$X(w) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}\,\mathrm{dt} = \int_{-\infty}^{\infty} x(t)\left[\cos(\omega t) - i\sin(\omega t)\right]\,\mathrm{dt}$$

In terms of information content, the signal spectrum is equivalent to the original signal. In addition, it provides information on the origin of vibrations, for example. If two machine components give rise to vibrations with different periods (frequencies) that are additively superposed, the Fourier transform makes these two components visible. The combination of sine and cosine for each frequency also enables phase angles to be mapped.

For example, superimposition of two sine waves with different frequencies and amplitudes results in the diagram shown below. From the variation over time it is difficult or impossible to glean the composition of the resulting signal. On the other hand, with appropriate scaling (see Spectrum scaling [▶ 22]) the magnitude spectrum |*X*(*w*)| and the magnitude of the Fourier transform clearly show that the signal is composed of two vibrations – one with a frequency of 0.2 kHz and an amplitude of 2.6, and one with a frequency of 1 kHz and an amplitude of 3.8. The phase information is hidden due to the absolute value calculation.

In the Magnitude spectrum: [▶ 192] sample, the magnitude spectrum is calculated and displayed for a signal of this form.





There are two processes that influence the vibration signals originating in a machine during sound transfer. Firstly transfer via machine components that attenuate the vibrations to different degrees depending on the frequency, and secondly superposition with vibrations from other machine components, with amplitudes adding up without interaction. Both factors are separated due to the properties of the Fourier transformation:

- Delays only affect the phase of the Fourier transform
- Frequency-selective attenuation and constructive superposition of vibration amplitudes show up in the magnitude of the Fourier transform.

Processing of time-discrete signals and the discrete Fourier transform

A very important aspect in the application of Fourier analysis is temporal sampling of the signal. The *Fourier transform* is mathematically defined for continuous, temporally unlimited signals.

However, in practice the *discrete Fourier transformation* (DFT) is used. It is defined for a *discrete, periodic signal* with a *finite number of discrete frequency components*. "Discrete" means that the signal is scanned at equal intervals, usually directly with an analog/digital converter, e.g. an EL3xxx or ELM3xxx.

If a time-continuous signal with a period of *T* is sampled, the resulting value string is:

$$x[n] = x(t = nT),\ n = 0..N-1$$

Using DFT, this series, which consists of *N* values, can be transformed to a discrete spectrum.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-2\pi i n k}{N}} = \sum_{n=0}^{N-1} x[n] \left[\cos(\tfrac{2\pi nk}{N}) - i\sin(\tfrac{2\pi nk}{N})\right]$$

The variable *k* represents a frequency channel, which also referred to as frequency bin. Like the variable *n*, it runs from 0 to *N* -1.

Discretization of time and quantization of values (digitization)

Two operations are required for digital processing of analog signals: Quantization from analog to digital value representation, and sampling of the temporally continuous physical signal to form a discrete sequence of quantized values.

The analog-to-digital converter digitizes the measured values in the I/O terminal. Quantization of the values generally takes the form of an integer signal with signed 16-bit representation or 24-bit representation. Processing in the TwinCAT 3 Condition Monitoring Library consistently takes place in the 64-bit IEEE double floating point format, which is hard-wired in advanced processors. The temporal sampling also takes place in the I/O terminal, through sampling of the input signal with a defined sampling frequency. The sampling frequency can be calculated from the task cycle time T_c and the oversampling factor c_s:

$$f_s = c_s \cdot \frac{1}{T_c}$$

Example: With a task cycle time of 1 ms and an oversampling factor of 10, the resulting sampling rate is f_s = 10 * 1 / (10^{-3} * 1s) = 10 kHz.

Pay particular attention to the sampling frequency

In TwinCAT the sampling frequency results from the task cycle time and the oversampling factor of the terminal used: `fs := Oversamples*1000.0/TaskCycleTime_ms`. Caution: The unit milliseconds is used for the task cycle time, as usual in TwinCAT.

Sampling theorem

The main practical limitation in the application of the DFT is the restriction of uniquely representable frequencies. According to the Nyquist theorem or sampling theorem, only signals whose highest frequency f_{max} is less than half the sampling frequency f_s can be represented unambiguously (slightly simplified description). Accordingly, the sampling frequency must be greater than the highest frequency occurring in the analog signal.

$$f_s > 2 f_{\max}$$

The presence of higher frequencies in the analog signal leads to an undesirable effect referred to as aliasing in the spectrum. The analog signal is then no longer correctly represented in the discrete signal. Before the analog-to-digital conversion, higher frequencies should therefore be removed from the analog signal using configurable analog filters.

Anti-aliasing filter

The EL3xxx and ELM3xxx EtherCAT Terminals provide various filters, depending on the terminal type. The EL3632 EtherCAT Terminal, for example, features a parameterizable analog 5th order low-pass filter, which is used to avoid aliasing. The EL3751 and ELM3xxx modules feature several filter stages, which can be used for anti-aliasing filtering and for wanted-signal filtering.

Frequency resolution

Since the frequency resolution (discrete resolution based on frequency components in the signal) enables different signal components to be allocated to certain machine elements and defects, in many cases it will be of advantage to achieve a resolution of the discrete frequency axis that is as high as possible.

Generally, the length of the Fourier transformation *N* determines the step size *Δf* of the discrete frequency axis *k* · *Δf*. A basic consideration facilitates understanding: In order to be able to represent the frequency of a sine wave in the frequency range, the measuring time must be at least one full period of this oscillation. This results in the following relationship between the resolution *Δf* and the measuring time T_m:

$$\Delta f = \frac{1}{T_m} = \frac{f_s}{N}$$

Typical PLC code syntax, e.g. in the MAIN routine of the Magnitude spectrum: [▶ 192] sample:

```
fSampleRate := cOversamples * (1000.0 / fSampleTaskCycleTime);

fResolution := fSampleRate / cFFTLength;
```

A high frequency resolution therefore requires a long measuring time. It is possible to extend the input data for the DFT through symmetric addition of zeros before and after the input signal (zero padding). This increases the length *N* of the signal sequence at constant sampling rate f_s, thereby refining the discrete resolution *Δf*. Zero padding does not add additional information to the signal. A distinction is made between two different types of resolution when zero padding is used: on the one hand the step size between one frequency bin to the next on the discrete frequency axis, i.e. the transition from $k \cdot \Delta f$ to $(k+1) \cdot \Delta f$, on the other hand the resolution for distinguishing between two adjacent frequencies of the input signal.

Although zero padding reduces the discrete resolution *Δf*, it does not change the measuring resolution. A refinement of the measuring resolution can only be realized through a correspondingly long measuring time. For practical applications, the key factor is usually the frequency resolution of the measurement, which influences the differentiability between two closely adjacent signal frequencies.

Zero padding

Zero padding does not add any information to the signal to be analyzed. For distinguishing between two adjacent signal frequencies, it is therefore not the frequency resolution that is refined, only the numeric resolution of the frequency axis.

Illustration based on an example:

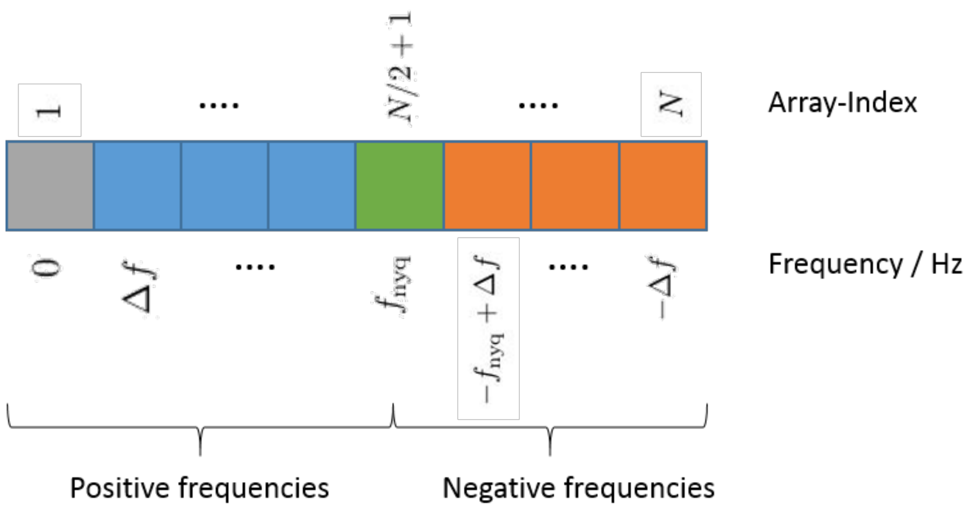
With a task cycle time T_c = 1 ms and an oversampling factor of 10 (i.e. f_s = 10 kHz), a buffer with a length of 3200 is filled. The resulting measuring time is $T_m = T_c$ * 3200 / 10 = 320 ms, with a measuring resolution of *Δf* = 1 / 320 ms = 3.125 Hz. Using FFT for further analyses/calculations, the buffer is symmetrically expanded with 2*448 zeros to reach a length *N* of 2^12 = 4096 > 3200 (*N* must be a power of 2, see next section). Zero padding therefore refines the numerical resolution to *Δf* = 10 kHz / 4096 = 2.44140625 Hz.

The discrete frequency axis is limited by the zero frequency (off-set) and the Nyquist frequency f_{nyq}, which corresponds to half the sampling frequency. According to the Nyquist theorem, it corresponds to the highest representable frequency of the recorded signal. If the discrete spectrum *X[k]* is stored in an array with index m, which runs between 1 and *N*, the resulting frequency axis *X[k]* is

```
fFrequency := (m-1) * fResolution; // m = 1..N/2+1
```

m = 1 represents the off-set, *m = N/2+1* represents the Nyquist frequency. The indices for *m* from *N/2+2* to *N* form the so-called negative frequencies, which are only relevant in practice if the input signal *x*[*n*] for the FFT has a complex value. See section Image frequencies [▶ 15].

The following diagram illustrates the configuration of the frequency axis for a DFT of length *N* (with *N* an even number).



Efficient calculation through FFT algorithms

Strictly speaking, the fast Fourier transformation (FFT) is a family of algorithms for discrete Fourier transformation (DFT) which are implemented particularly efficiently and lead to the same numerical result. While the complexity of a naïvely implemented DFT with *N* time values is O(N^2), for a FFT it is only *N*(2 * $\log_2 N$). For larger values of *N*, the difference is substantial. For *N*=1024, for example, it is already a factor of around one hundred. Generally FFT algorithms are defined for values of *N* (the length of the FFT) that represent a power of two, i.e. 256, 512, 1024 etc.

Complex valued result

The FFT (and the DFT) splits the incoming signal *x*[*n*] into a number of sine and cosine oscillations. Each frequency is associated with a coefficient for the sine and cosine components. Both factors are represented together as a complex number. The decomposition is expressed in *Euler's formula*:

$$e^{i\omega t} = \cos\omega t + i\sin\omega t$$

$$\mathrm{Re}\{e^{i\omega t}\} = \cos\omega t, \quad \mathrm{Im}\{e^{i\omega t}\} = \sin\omega t$$

The real part Re{..} of each Fourier coefficient corresponds to the cosine component, the imaginary part Im{..} to the sine component. The ratio of the two components reflects the phase angle of the frequency components.

$$\Phi(k) = -\mathrm{atan}\frac{\mathrm{Im}\{X(k)\}}{\mathrm{Re}\{X(k)\}}$$

In many cases it is not the precise temporal characteristic of the signal that is of interest, but the magnitude spectrum. This can be determined from the Fourier transform by calculating the absolute value of the complex number:

$$|X(k)| = \sqrt{\mathrm{Re}\{X(k)\}^2 + \mathrm{Im}\{X(k)\}^2}$$

Complex data type

The result of the FFT of a real-valued or complex-valued input signal is complex-valued. The data types `LREAL` and `LCOMPLEX` are used for the signal representation. If a function block is used for calculating the magnitude spectrum [▶ 127] or power spectrum [▶ 142], the result is directly real-valued.

Image frequencies

In the Fourier transform of a real signal the coefficients for negative frequencies are equal to the complex conjugate coefficients for positive frequencies. If *X[k]* is the Fourier-transform of *x*[*n*] and *X**[*k*] the complex conjugated, the following applies for a Fourier transformation with *N* points:

$$X[k] = X^*[N-k]$$

For real-valued signals a time reversal of the input signal corresponds to complex conjugation of the Fourier transform. It follows that the spectral value for frequencies below the Nyquist frequency occur mirrored in the values above the Nyquist frequency. Since the values with *k* > *N/2 +1* are therefore redundant for real input frequencies, the Fourier transform for real sequences is usually limited to the first *N/2 +1* values.

Function blocks in the Condition Monitoring Library

The Condition Monitoring Library offers various function blocks that facilitate a Fourier analysis.

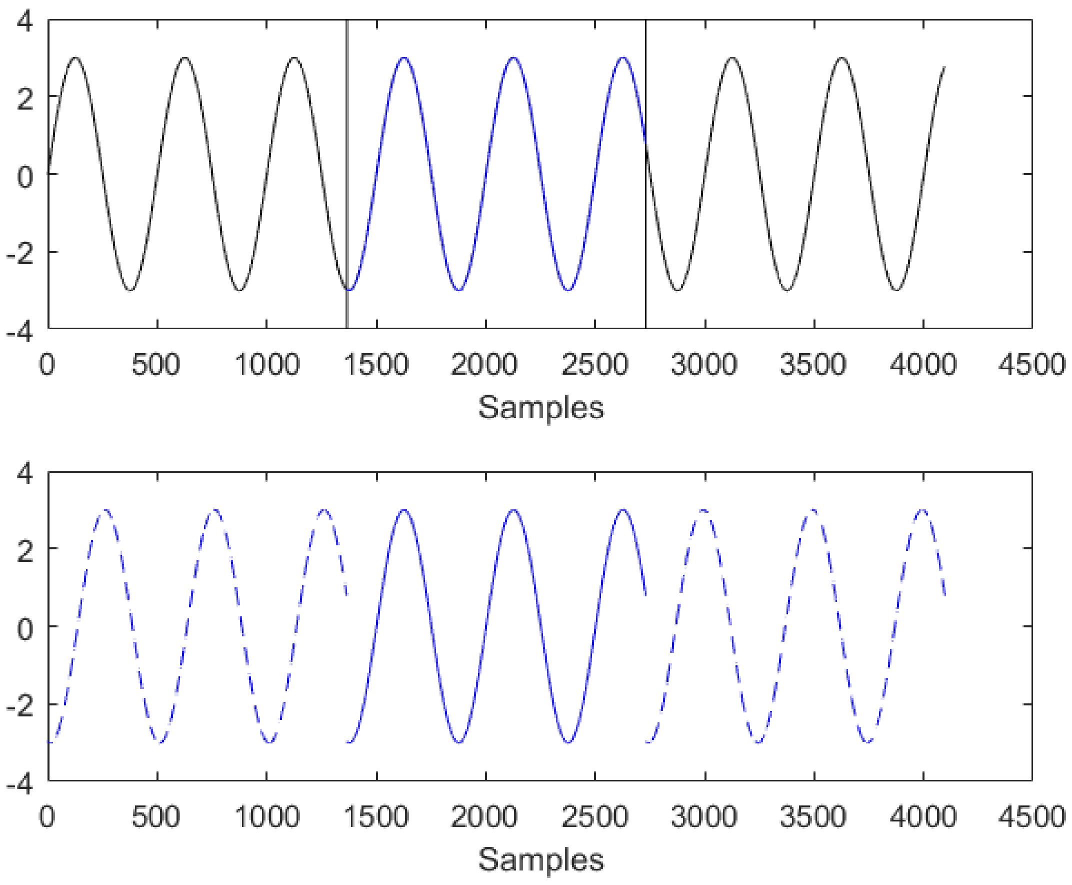
- FB_CMA_RealFFT [▶ 145]: Calculating the FFT of a real-valued input signal.
- FB_CMA_ComplexFFT [▶ 86]: Calculating the FFT of a complex-valued input signal.

- FB_CMA_MagnitudeSpectrum [▶ 127]: Calculating the magnitude spectrum of a real-valued input signal, including windowing of the input signal with overlapping windows [▶ 16] and different scaling options [▶ 22].
- FB_CMA_PowerSpectrum [▶ 142]: Calculating the power spectrum (periodogram) of a real-valued input signal, including windowing of the input signal with overlapping windows [▶ 16] and different scaling options [▶ 22].

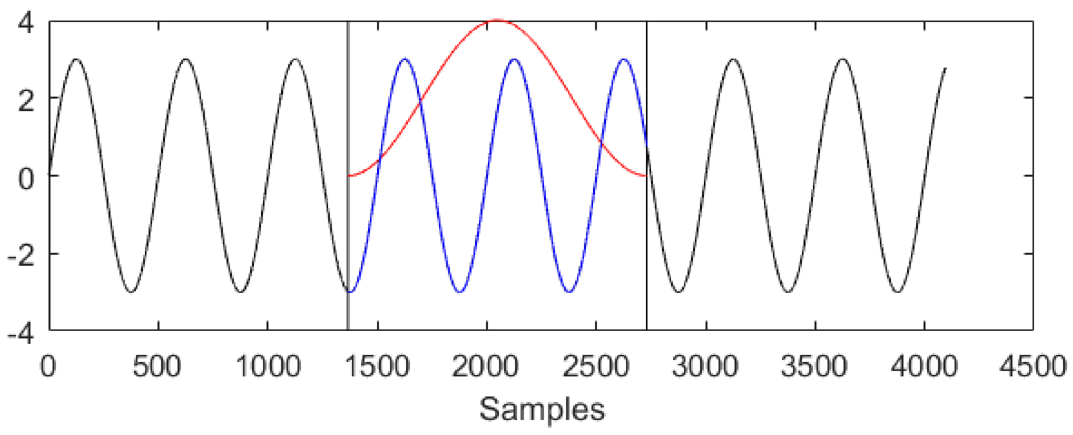
2.1.2 Analysis of data streams

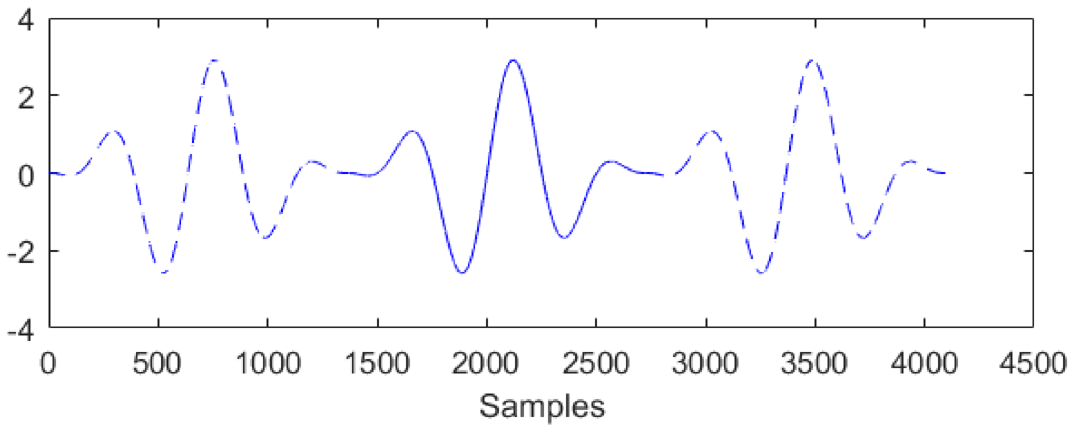
Block-by-block FFT analysis from a data stream

The DFT/FFT is defined on a continued cyclic, periodic signal. This leads to an initially surprising conclusion: If an FFT analysis for a long signal is required, the input signal cannot simply be subdivided into sections and transformed with DFT. Because if the last value in such a section does not match the first, the FFT interprets this as a discontinuity in the cyclic sequence, which clearly shows up in the spectrum (*spectral leakage*). The following diagram illustrates the principle. A partial signal (blue) is cut from the total signal (black). The FFT implies a cyclic continuation of the partial signal (lower diagram) and assumes step changes in the signal to be transformed, as clearly indicated in the spectrum.



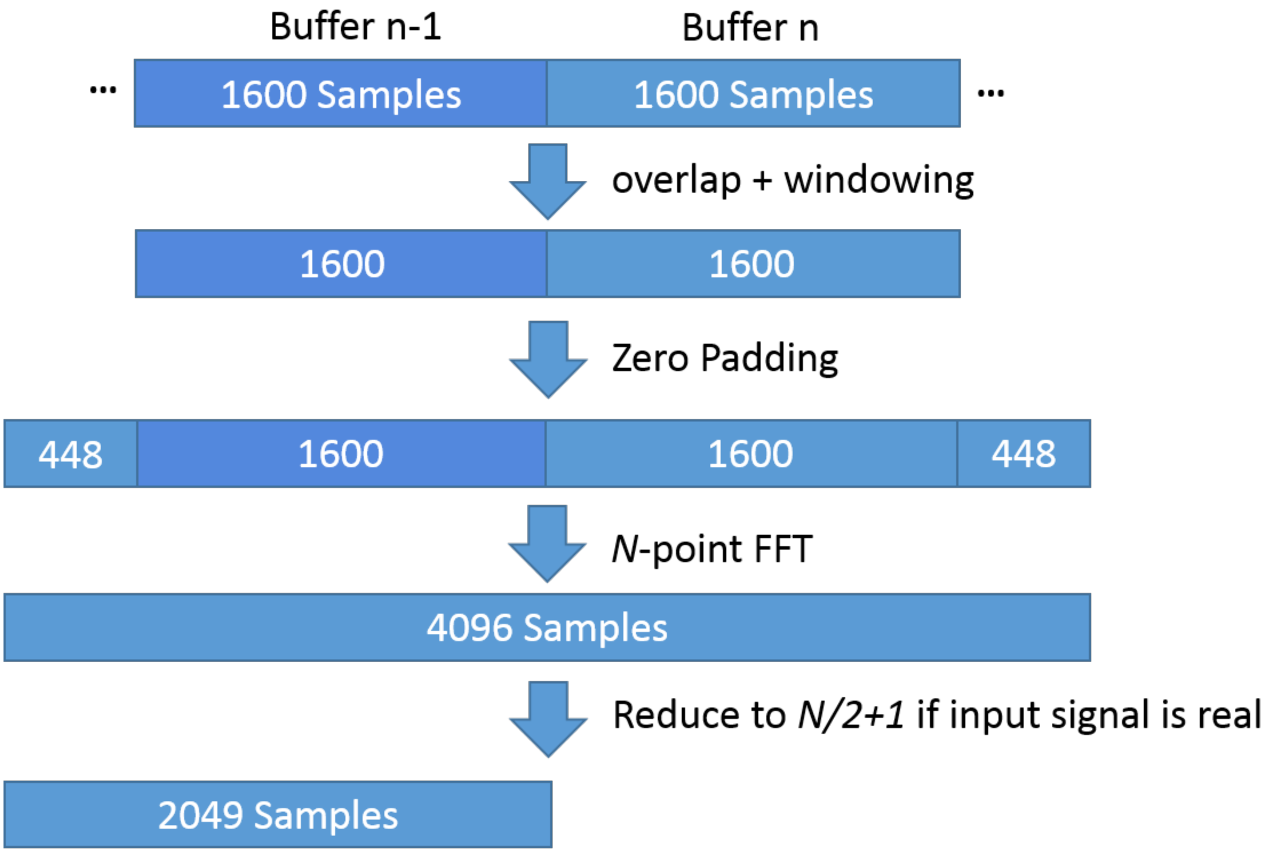
The situation can be rectified by weighting the signal sections before the transformation with a suitable window function (for details see next section). In a suitable window, time values near the start and the end are multiplied with a factor zero or closed to zero. The following diagram shows the same scenario, but now with a window function (red). Windowing removes the step changes in the cyclic continuation, although please note that the properties of the window show up in the spectrum of the windowed partial signal. However, the window property generally affects the spectrum to a significantly lower degree.





A problematic aspect of windowing is that values at the edge of the window are hardly taken into account in the spectrum. In situations where this region contains signal characteristics that indicate possible damage, key information may be lost. In order to prevent the loss of information, the TwinCAT Condition Monitoring Library uses overlapping signal sections for the windowing procedure. For example, the standard Hann window is based on 50% overlapping. As a result, samples that are at the edge of one window section are in the center of the next window section.

The following diagram illustrates the process for a FFT analysis from a data stream. Initially, buffers with a defined length of 1600 values are filled from the data stream. The previous buffer is included in the evaluation of the data from buffer n, so that the data packet that is windowed now contains 3200 values. The maximum of the window function is precisely in the middle between the two buffers and falls to zero towards the edges of the two buffers. Zero padding extends the data packet to a length of 4096 values, so that the length is a power of 2 and can therefore be efficiently calculated with an FFT algorithm. The result of the FFT is a data packet with 4096 values, which can be reduced to 2049 values if required, provided the input data are all real-valued (see Fourier analysis [▶ 15]).



During the evaluation of buffer n, buffer n+1 is filled, and buffer n is included in its evaluation. This approach always results in a 50% overlap of the windowed time ranges.

Analysis of a data stream in TwinCAT 3

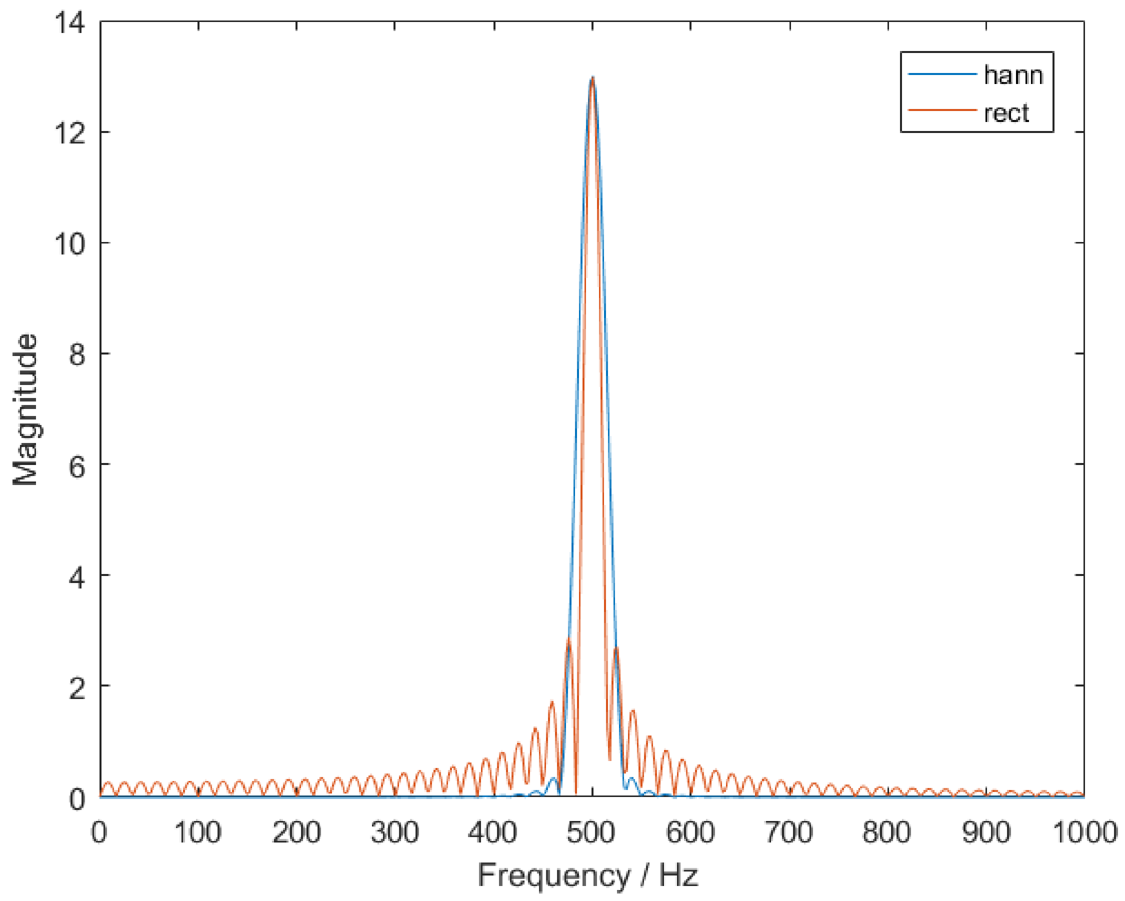
The signal analysis scheme shown in the diagram above is implemented in the Condition Monitoring Library through FB_CMA_MagnitudeSpectrum and FB_CMA_PowerSpectrum. All that is required is a configuration of the parameters (length of the buffers, length of the FFT, …) and provision of the data buffers.

Window functions

The properties of the window functions used are shown in the result of the transformation. It is not the signal *x*[*n*] that is Fourier-transformed, but the signal *x*[*n*]·*w*[*n*], with *w*[*n*] as time values of the window function. Note the basic characteristics of window functions.

If "no" window function is used, i.e. if a signal section is taken from a longer overall signal, this corresponds to the application of a rectangular window. An example is used to compare the properties of window functions with a rectangular window: A harmonic sine with an amplitude of 13 and a frequency of 500 Hz is sampled with a rate of 10 kHz and windowed with a window function with a length of 1600 samples, followed by calculation of the magnitude spectrum (with the scaling option [▶ 227] `eCM_PeakAmplitude`). The following diagram shows the magnitude spectrum based on a Hann window (hann) and a rectangular window (rect).

A sample for the reconstruction of the following graphic can be found here: Window functions [▶ 198]



It illustrates two key features of window functions:

- The width of the *main lobe*, in this case around 500 Hz.
- The attenuation of the *side lobes*, relative to the maximum of the *main lobe*.

The width of the *main lobe* affects the achievable frequency resolution. The height of the *side lobes* indicates the *spectral leakage*, since it is caused solely by the window and not by the signal to be analyzed. Note that the rectangular window enables very good frequency resolution but results in strong spectral leakage, which becomes problematic if a frequency component with an amplitude of 0.5 occurs at 550 Hz, in addition to the peak at 500 Hz, for example. The Hann window reduces the *side lobes* significantly, although it also reduces the achievable frequency resolution. Nevertheless, it provides a good compromise.

An important parameter of the frequency resolution, if a window function is used, is the equivalent noise bandwidth (ENBW).

$$ENBW = N\frac{\sum w^2[n]}{(\sum w[n])^2}\Delta f = \frac{\sum w^2[n]}{(\sum w[n])^2} f_s$$

The value Δf is derived from the FFT-length N and the sampling rate f_s (see Fourier analysis [▶ 11]). The expression in the equation before Δf is defined via the window properties. For a rectangular window it is 1, for the Hann window it is 1.5, for example. This means, for example, that each frequency bin also contains components from the neighboring frequency bins. When selecting the measuring time, the reduction of the frequency resolution due to the window used must be taken into account.

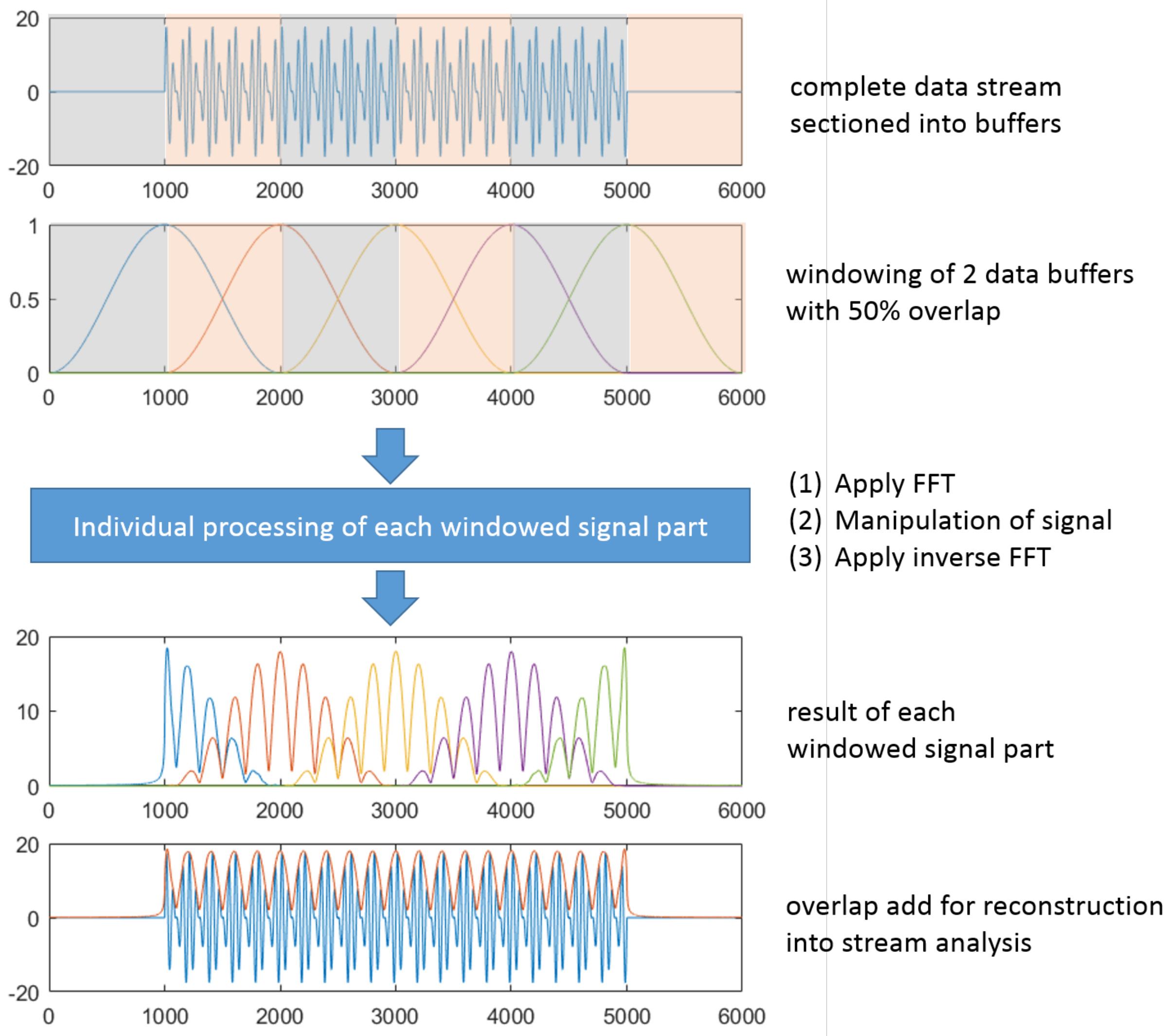
Sample for determining the frequency resolution

The following approach can be used, if a frequency resolution of 1 Hz is required from the application. The measuring time for achieving a resolution of 1 Hz corresponds to 1 second. If a Hann window is used, the resolution deteriorates by the factor 1.5, i.e. the measuring time has to be longer by this factor, in order to bring the effective frequency resolution back to 1 Hz: In other words, the measuring time should be 1.5 s. The number of sampling values to be buffered results from the selected sampling rate. The resulting FFT-length is the next higher numerical value that is also a power of 2.

The choice of the used window is realized with initialization parameters for the respective function blocks via the block-specific structure, e.g. ST_CM_MagnitudeSpectrum_InitPars [▶ 179].

Overlap-Add Method

Some function blocks of the Condition Monitoring Library work by manipulating the spectrum of the input signal, i.e. the input signal is first split into overlapping partial signals and Fourier-transformed, as described above. The spectrum is then manipulated, and an inverse Fourier transformation is calculated. Depending on the window function used, a correction function may be required to compensate the influence of the window. The individual overlapping results are then added up at the function block output, so that once again a data stream is created. This method is referred to as overlay-add and is illustrated in the diagram below by means of the calculation of the signal envelope (FB_CMA_Envelope [▶ 109]).


complete data stream sectioned into buffers
windowing of 2 data buffers with 50% overlap
Individual processing of each windowed signal part
(1) Apply FFT
(2) Manipulation of signal
(3) Apply inverse FFT
result of each windowed signal part
overlap add for reconstruction into stream analysis
20
0
-20
10
1
0.5
0
1000
2000
3000
4000
5000
6000

Overlap-add in TwinCAT 3

The method is used within of some function blocks of the library and does not have to be implemented by the user. All that is required is a configuration of the parameters (length of the buffers, length of the FFT, …) and provision of the data buffers.

2.1.3 Triggered analysis of a time period

Motivation

In addition to the continuous time analysis of a process, e.g. the vibration behavior of a continuously rotating shaft, another frequent application is the analysis of a defined timeframe. Application examples include analysis of vibration signals on a drill head, a milling unit or a shaft which only rotates during certain periods of time.

The advantage of an analysis integrated into the controller is particularly apparent in this case. The control usually initiates a certain process step, e.g. drilling. Accordingly, the sequential machine control can be used to trigger not only the process step but also the corresponding analysis step.

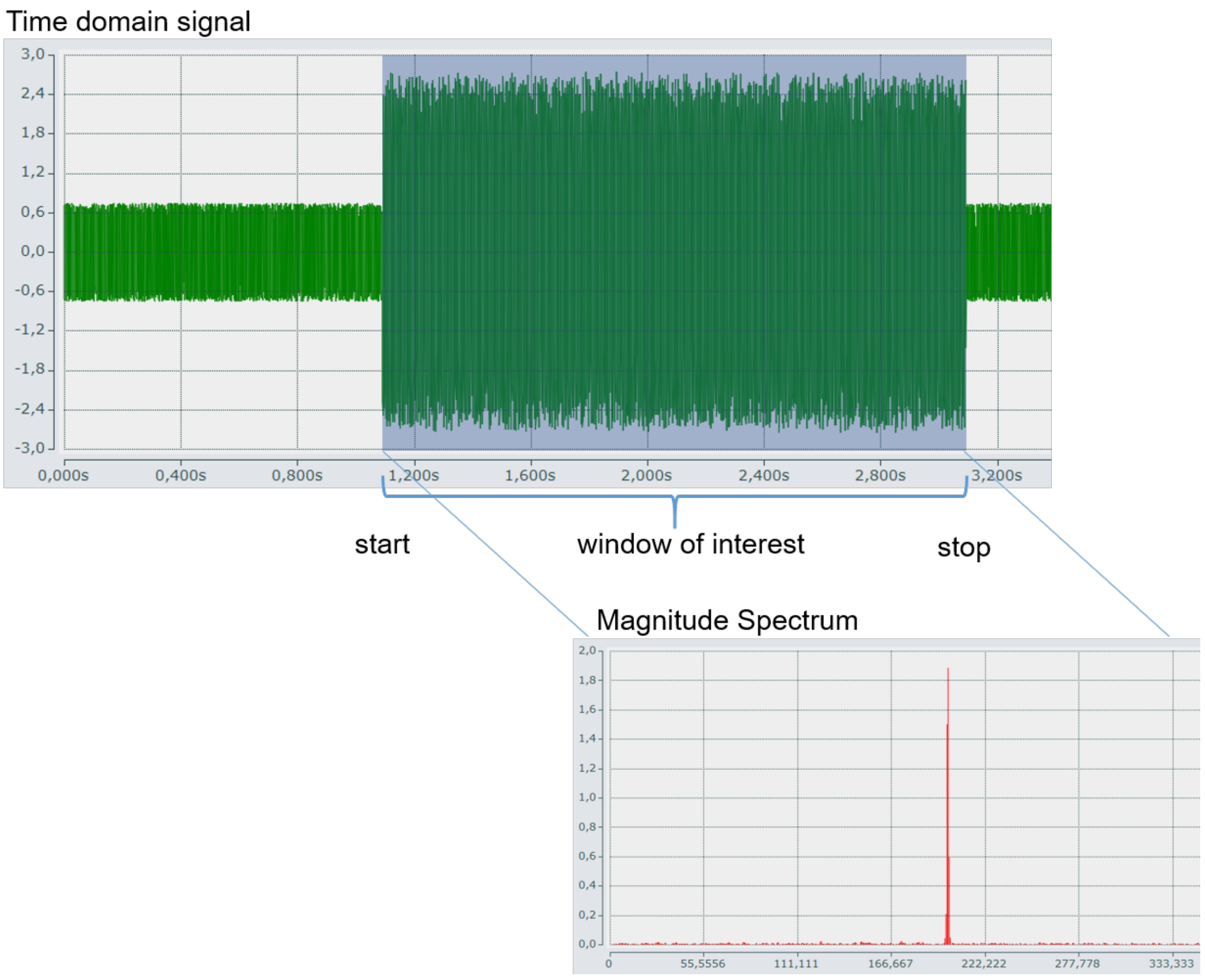
Implementation in the Condition Monitoring Library

With regard to the analysis functions, there is virtually no difference between the evaluation of a defined timeframe and the continuous analysis of a data stream. The only difference is that each triggered analysis is independent, i.e. not in a continuous context. Accordingly, all analysis function blocks of the TC3 Condition Monitoring Library can be used for continuous and triggered time window analyses. In order to clearly separate the individual analyses from each other, it is only necessary to ensure that all analysis function blocks with memory properties (see the respective documentation for the individual algorithms; section Memory Properties) are reset once an analysis has been completed. For all these function blocks the `ResetData()` method is available for this purpose.

Sample implementation

A sample based on a synthetic signal is described below. The synthetic signal consists of background noise and an additively superimposed sine signal with a frequency of 200 Hz and amplitude 2. The sine signal is switched on and off alternately every two seconds.

If continuous evaluation is selected for such a signal, it is not possible to determine in which time intervals the signal segments used for the evaluation lie. Accordingly, it is advisable to always start an evaluation window for a defined measuring time when the sine signal is switched on. The schematic diagram below shows the described synthetic signal and the amplitude spectrum based on the indicated evaluation window.



The source code and a more detailed description of the sample can be found here: Event-based frequency analysis [▶ 221].

Documents about this

🗎 Event_based_FrequencyAnalysis.zip (Resources/zip/5261425419.zip)

2.1.4 Scaling of spectra

Magnitude and power spectrum

There are several common ways of evaluating the spectrum:

- The magnitude spectrum [▶ 127], which uses linearly scaled magnitude values of the complex-valued spectral values $|X[k]|$. It is also called the magnitude spectrum or amplitude spectrum.
- The power spectrum [▶ 142], whose values represent the squares of the magnitude values $|X[k]|^2$.

Using the power spectrum makes sense if power values are added up or consolidated, since the squared spectral values $|X[k]|^2$ relate exactly to the RMS value of the time signal via Parseval's theorem.

According to Parseval’s theorem, the power of signal $x[n]$ in the time representation equals the power of the signal in the Fourier transform:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

If one now calculates the RMS value of the signal *x*[*n*], this can be realized in the time range or in the frequency range, since both representations are identical with regard to the power:

$$\mathrm{RMS}\{x[n]\} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}|x[n]|^2} = \sqrt{\frac{1}{N^2}\sum_{k=0}^{N-1}|X[k]|^2}$$

In practice, this enables calculation of RMS values for limited frequency ranges of a signal, which is used internally in function block FB_CMA_IntegratedRMS [▶ 124] or FB_CMA_MultiBandRMS [▶ 135], for example. Practical scaling options [▶ 227] of the Condition Monitoring Library, which relate to the properties referred to in this section, include `eCM_ROOT_POWER_SUM` and `eCM_RMS`.

The power spectral density

Another important concept for spectral analysis is the *Power Spectral Density* (PSD). It refers to the output value based on the effective frequency resolution, as indicated by the *Equivalent Noise Bandwidth* (ENBW)

$$\mathrm{PSD} = \frac{1}{ENBW}|X[k]|^2$$

A look at the physical units for the signal, magnitude spectrum and PSD illustrates the relationships. If a signal *x*[*n*] is measured in volt (V), the discrete magnitude spectrum |*X*[*k*]| is also stated in V. Squaring means that the power spectrum is stated in V^2. By definition, the power density spectrum is a power value (V^2) based on the frequency in Hz. Relating the power spectrum to the effective frequency resolution in hertz (Hz) results in the unit V^2/Hz.

This representation can also be used for magnitude values. Correspondingly, the linear spectral density (LSD) is

$$\mathrm{LSD} = \sqrt{\frac{1}{ENBW}}|X[k]|$$

Decibel scale

In vibration analysis and machine acoustics, it is common practice to convert values from the linear scale to the logarithmic decibel scale. The decibel scale facilitates interpretation in cases where very large and very small values occur in the same spectrum, and the analysis should cover both large and small values. The magnitude spectrum can be converted to the decibel scale via:

$$|X[k]|_{\mathrm{dB}} = 10\log_{10}|X[k]|^2 = 20\log_{10}|X[k]|$$

The decibel scale can be expressed as 10 times the logarithm of the power spectrum and 20 times the logarithm of the magnitude spectrum. The result of a calculation from FB_CMA_MagnitudeSpectrum [▶ 127] and FB_CMA_PowerSpectrum [▶ 142] is therefore identical in the decibel scale.

The conversion of results to the decibel scale can conveniently activated in the Condition Monitoring Library via a Boolean variable in the function block initialization parameters, see ST_CM_PowerSpectrum_InitPars [▶ 182], for example.

Scaling options based on signal type

By selecting a suitable scaling option [▶ 227], the spectral values calculated through the power spectrum [▶ 142] or the magnitude spectrum [▶ 127] function block can automatically be adapted to a reference parameter, as required. The correct interpretation of the reference variable is of particular importance here.

In terms of the scaling options, in practice and assuming a steady-state signals, it is important to distinguish between deterministic and stochastic signals.

Deterministic signals consist of periodic vibrations with a defined frequency. The key is that the frequency resolution (ENBW) is wider than a harmonic frequency. The total power of this frequency component of the signal is consolidated in this frequency channel. The spectral values are therefore directly scalable to an amplitude (scaling option [▸ 227]eCM_PeakAmplitude) or an RMS value of an equivalent sine signal. If the signal does not fall into the center of the FFT frequency channel, then so-called scalloping losses occur, which decrease the observed maximum amplitude. This can be compensated by analyzing the power values from adjacent frequency channels in added-up form, see scaling option [▸ 227]eCM_ROOT_POWER_SUM and eCM_RMS.

Stochastic or broadband signals require analysis of power spectral densities (PSD) or linear spectral densities (LSD), since all frequencies contain signal power over a defined frequency range. In this case the determined power values depend on the effective width of the frequency channels of the FFT. It makes sense to use this bandwidth as a basis, in order to obtain results that are independent of the analysis parameters. When window functions are used, the effective width of the frequency channels depends on the length and form of the window function. In this case, the equivalent noise bandwidth (ENBW) referred to above should therefore be used, see scaling option [▸ 227]eCM_PowerSpectralDensity or eCM_UnitaryScaling.

Scaling based on PSD does not enable consistent scaling of the "direct current component". If required this should be determined by low-pass filtering or averaging.

If a signal contains both deterministic portions and wide-band portions, both scalings must be used independently of each other in order to obtain values that are independent of the processing parameters. An example would be the analysis of signals that is composed of a harmonic sine and band-limited noise. If the amplitude of the harmonic sine is to be determined, scaling for deterministic signals is required. If the stochastic background noise is to be analyzed, scaling as PSD or LSD should be applied.

Scaling of spectra with the Condition Monitoring Library

Various scaling options are already implemented in the Condition Monitoring Library and can be parameterized with initialization parameters via the function-block-specific structure. See E_CM_ScalingType [▸ 165] and Spectrum Scaling Options [▸ 227]. A tutorial can be found here: Scaling of spectra [▸ 199]

Referencing

Classification of the scaling

While comparison of absolute measured values is very important for measurement technology, for vibration assessment [▸ 31] according to ISO 10816-3 and for machine protection, absolute calibration is not required for trend-based or comparative condition monitoring.

In many cases, generic limit values that are not tailored to a specific machine, are less suitable for early diagnostic detection of damage. Since the choice of measuring point (location of the measurement, coupling of the sensor etc.) has significant influence on the attenuation factors of the transmission link, for **trend monitoring** it is much more important to consistently maintain the selected test point and the coupling conditions. In many cases signal components with initially low signal level can be important. If they are periodic, they appear particularly clearly and early when using high-resolution FFT spectra with the narrowest possible bandwidth and suitable statistical functions. In condition monitoring trend observations over long periods and relative comparisons at the decibel scale usually play a much more important role than individual absolute values. For the sensors this means that expensive, high-precision absolute calibration and smooth frequency response are generally less important than high long-term stability and sufficiently low temperature dependence, although this does not mean that a calibration can be neglected completely.

Scaling on the basis of reference signals

In many cases, mathematical referencing (scaling by means of a reference) of measured values be much more complex than would appear at first glance. As soon as the processing involves several steps that are non-linearly dependent on diverse parameters, it is in many cases simpler and above all less prone to error to carry out the scaling with the aid of a calibration device. Here we make use of the fact that the magnitude values of the calculated spectra are always linear to the input values. In order to scale the signal correctly, therefore, we only need to determine the associated linear factor on the basis of a well-known reference input value. Professionally this is done by generating a physical signal with a defined amplitude (or a defined RMS value) using a calibration device, measuring the output value and determining the required correction factor as the quotient of input and output. The big advantage of scaling on the basis of a reference signals is that physical defects such as damage to an accelerometer as well as incorrect configurations of the measuring system can be reliably discovered. This method has its limits if a large number of parameter combinations are to be tested when evaluating.

2.1.5 Statistical analysis

Condition monitoring is used for monitoring of limit values. Value transgressions cause messages and warnings. In practice the individual values of the FFT often fluctuate strongly, so that averaging or other statistical analysis is required. An analysis of individual values would result in a high value leading to a transgression of the limits.

Basic concepts

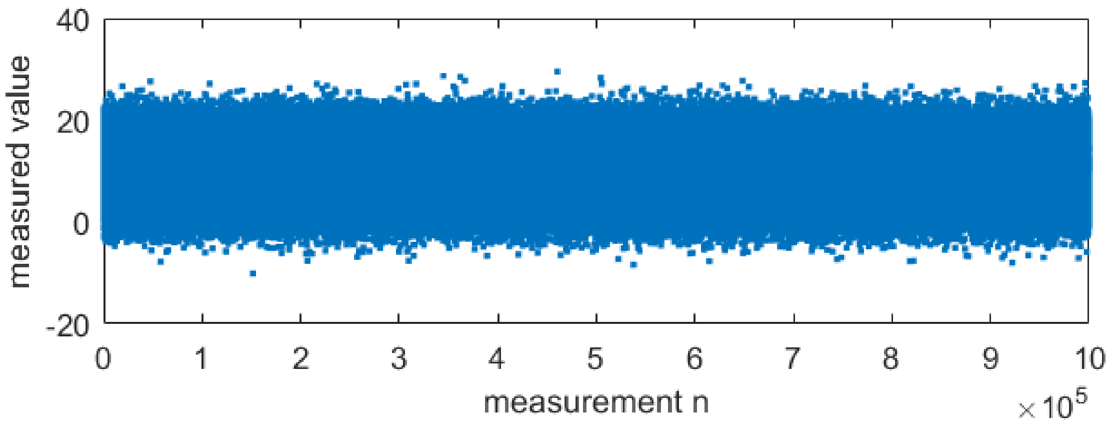
If a quantity (e.g. temperature, pressure, voltage etc.) is measured in an actual process, for a repeated measurement it is very likely that the previous measured value does not match the value determined in the repeat measurement. Since the sequence of randomly fluctuating quantities cannot be determined deterministic (i.e. via a concrete equation), statistical parameters are used for describing such signals. The fact that in practice deterministic and stochastic signals are often superimposed (e.g. a direct voltage superimposed by measurement noise) is irrelevant. The summary result is random and therefore a stochastic signal.

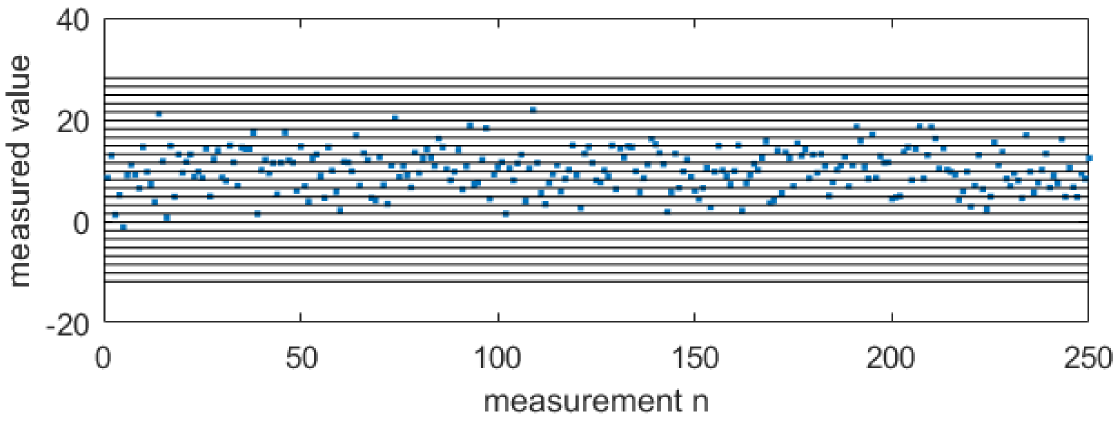
An individual measurement of a randomly fluctuating quantity is a *random event*. Each individual measurement is referred to as *realization* of a *random experiment*. If *N* random samples are taken from the random experiment, this number of realizations describes the *sample size*.

Histograms

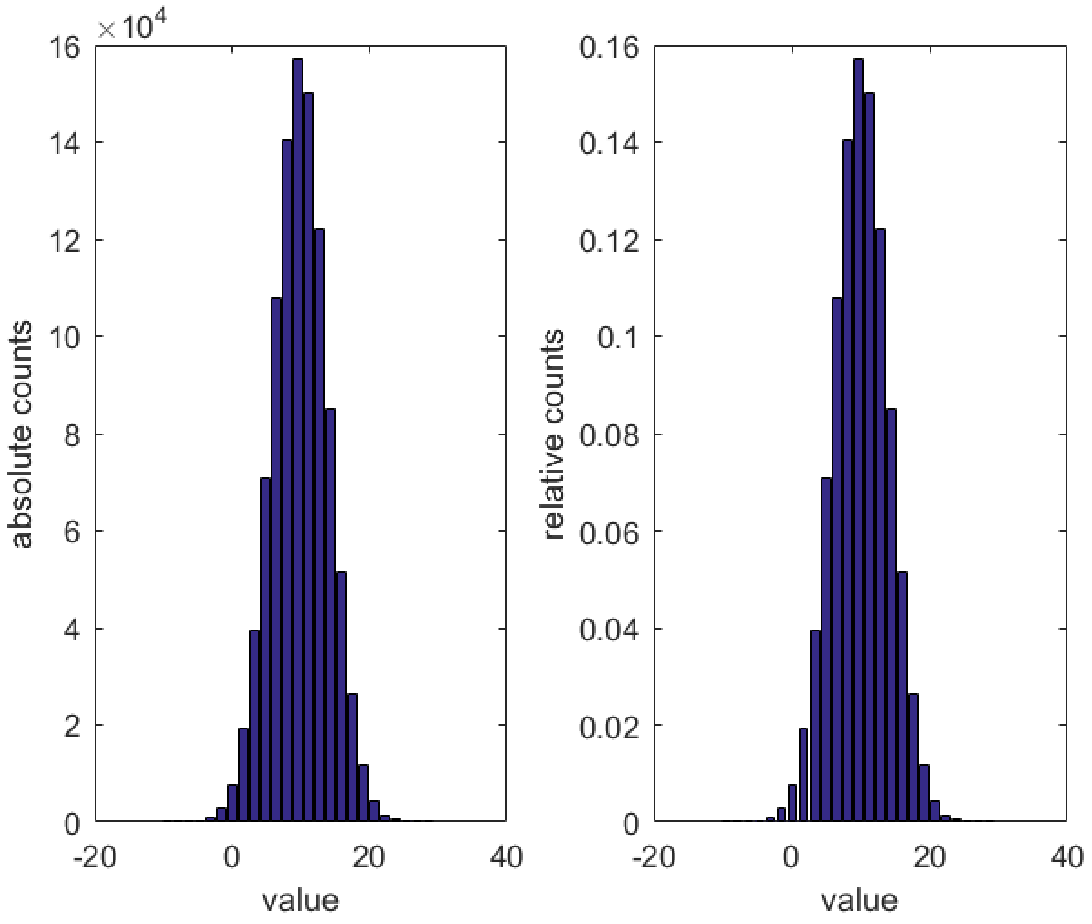
A central property of random events is the probability that the measured parameter assumes a certain value. This is described via the absolute or relative *frequency distribution*, which is represented in a *histogram*.

Simple example: Suppose a measured variable of 10 V is superimposed with normal distribution noise (average value 0 V, standard deviation 4 V). Repeating the measurement for this parameter 1 million times results in the diagram below (upper part). The 1 million realization of the random experiment can be shown in a histogram for a better overview. The absolute frequency distribution can be generated such that the range of the measured variable is subdivided into classes (bins). The upper part of the diagram shows the measured variable over each individual measurement, the lower part only shows the first 250 measurements and the class limits for the histogram.





The absolute frequency distribution is then simply results from the number of measured values that lie within a class (bin), see diagram below, left. The distribution is parameterized based on the number of considered classes – the more classes, the finer the distribution. The relative frequency distribution can be calculated from the absolute frequency distribution through referencing of the sample size; see diagram below, right. This is then independent of the number of measurements and shows the probability with which a value was measured, e.g. values in the class around 10 V were measured with a probability of 0.157=15.7%.



The frequency distribution can be used for simple initial visual examination of an experimentally examined process. Three questions can be explored:

- How strong is the scattering of the measured value?
- Is the measured value scattered around a single value (as above around 10 V), or around further values?
- How are the values distributed? - normal distribution, Student's *t*-distribution, chi-square distribution?

Calculation of absolute frequency distribution in TwinCAT 3

The Condition Monitoring Library can be used to calculate the absolute frequency distribution conveniently via the function block FB_CMA_HistArray [▸ 115]. Only the range under consideration and the number of classes are required for parameterizing the function block. A graphic display is possible with the array bar chart in TwinCAT Scope View. A sample is available for download from here [▸ 204].

The Statistical methods [▸ 206] sample illustrates further Condition Monitoring Library options for statistical data evaluation.

Central moments

A value that is as close as possible to the actual value can be estimated based on multiple observations of a random process. It is referred to as *best estimate*. Different *estimators* (e.g. the arithmetic mean) with different properties can be used for this purpose. In addition to the calculation of the best estimate, in many cases it is important to also express the *uncertainty* of the estimate, which is usually calculated via the *experimental standard deviation* (also referred to as empirical standard deviation).

The central moments: average value, variance, skew, kurtosis etc. are particularly suitable for calculating statistical parameters from a given sample. While the average value provides a suitable estimated value for the sample, the other central moments provide insight into the distribution of the values around this estimated value.

Illustration based on a sample:

The sample described above under histogram has a "true value" of 10 V and was retrospectively subjected to noise. From the given sample of 1 million realization the average value can be calculated as 9.9977 V. This is the best estimate of the true value. The variance around this average value is 16.01 V^2. The square root of the variance corresponds to the standard deviation and is 4.0013 V. If the measured values are distributed normally, as in this case, the distribution of the measured values is fully described with these two central moments, i.e. the skew and kurtosis are (theoretically) zero. The skew describes the symmetry of the distribution around the average value, the kurtosis describes the steepness (peakiness) of a distribution function.

Assessing the uncertainty of an estimated result:

In 1995 the Joint Committee for Guides in Metrology (JCGM) published a guide on stating measurement uncertainty. The JCGM is composed of central umbrella organizations such as BIPM, IEC; IFC, ISO etc., who developed this guide as a joint effort. The basic paper "Guide to the Expression of Uncertainty in Measurement" (GUM) is available for download free of charge from the BIPM website. A brief introduction into the central idea is provided below.

As described above, a best estimate can be calculated from a given set of *N* observations (average value = sample mean). The variance of the best estimate is calculated and used as uncertainty value, rather than the variance of the set of observations (standard deviation). This makes sense, because the aim is to assess the uncertainty of the estimated value. The variance of the best estimate can simply be calculated from the standard deviation of the set of observations by dividing this value by the root of *N*. If the sample size is sufficiently large, the uncertainty value can be multiplied by 2 (otherwise a larger factor), in order to calculated the extended uncertainty. The average value plus/minus this extended uncertainty will then contain the true measured value with a probability of 95%.

Accordingly, the algorithms of the Condition Monitoring Library can be used to make GUM-compliant statements on the measurement uncertainty.

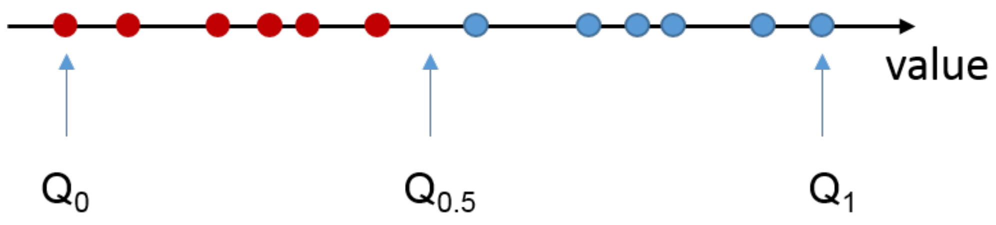
Calculating the central moments in TwinCAT 3

The function block FB_CMA_MomentCoefficients [▶ 130] contained in the Condition Monitoring Library can be used to calculate the first four central moments of a sample. The function block only has to be parametrized in terms of the sample size used.

Quantile

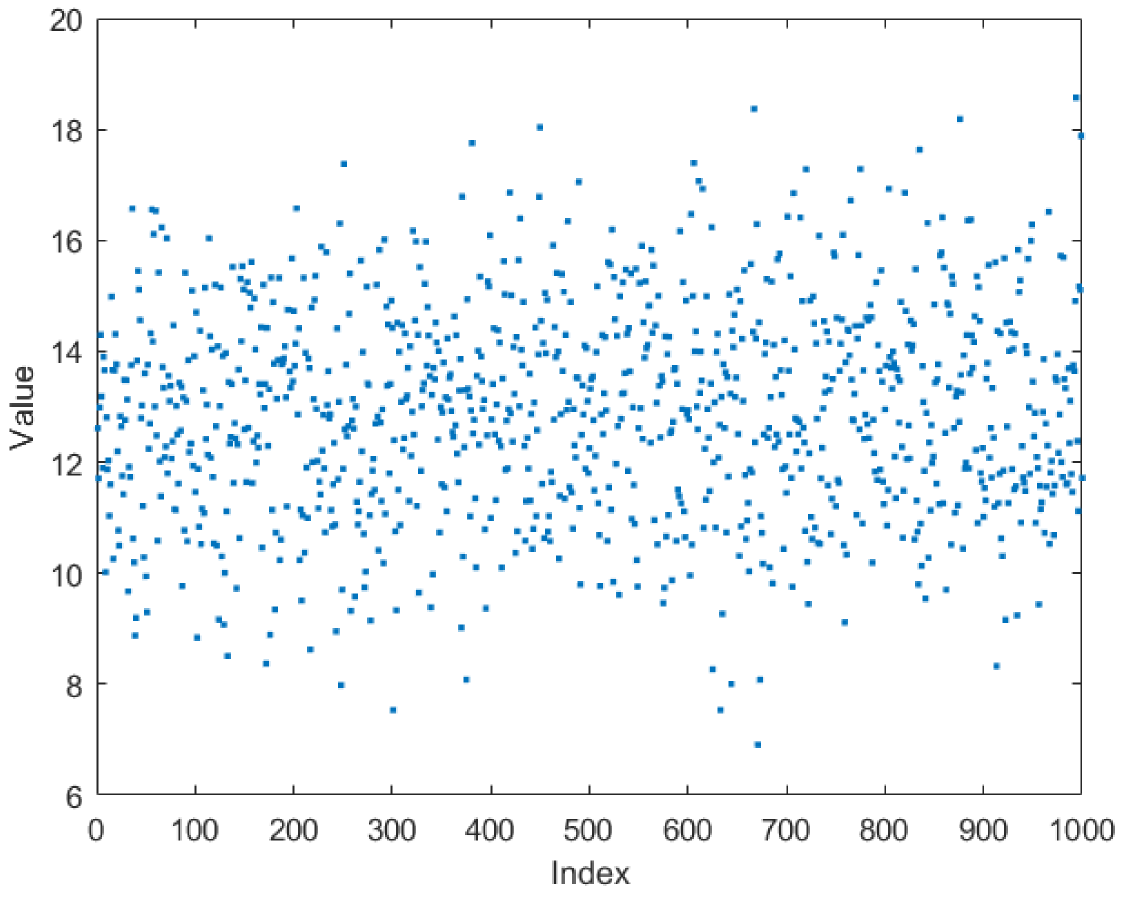
The p-quantile Q_p of a random variable *x* is the value for which $Q_p > x$ applies for the component *p* of all realizations of *x*. In other words: If a finite number of values is given, the p-quantile divides the data into two areas. The 50%-quantile (median), for example, marks the value below which at least 50% of all the values lie. This value should not be confused with the average value of the sample.

The value of *p* can be between zero and one. If *p* is specified in percent, the values are percentiles. $Q_{0.5}$ precisely corresponds to the median, while $Q_{0.9}$ represents the 90-percent-percentile and Q_1 the maximum of an observed value series.

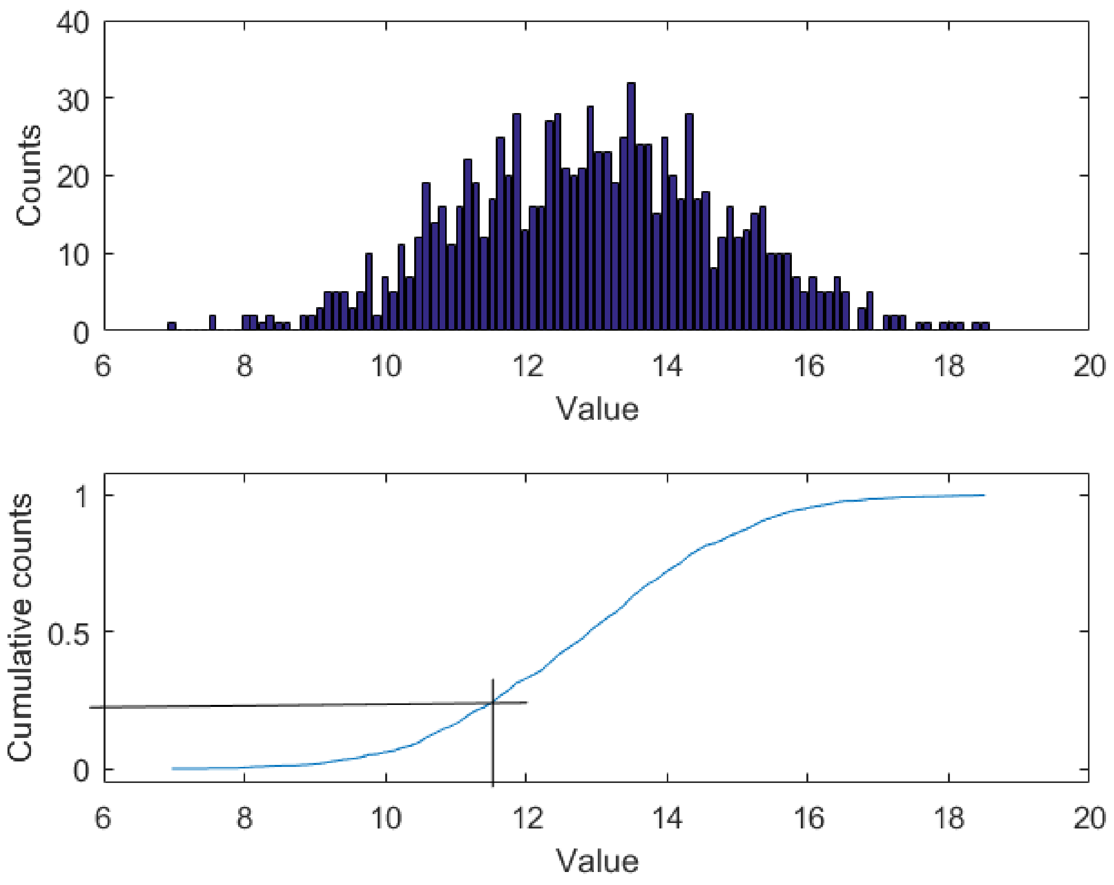


The closer *p* comes to one, the stronger Q_p is determined by outliers and extreme individual values. The closer *p* comes to 0.5, the closer Q_p comes to the median, which is very robust against outliers. The value of *p*, which can be configured in TwinCAT at runtime, can be used to dynamically change the sensitivity of a sample evaluation in relation to individual values.

To illustrate the basic idea of quantiles, the following diagram shows a series of 1000 values, which are spread around an average value of 13.



The value sequence can be used to calculate a histogram, which indicates how often a value occurs in the series (sample) under consideration. The empirical cumulative frequency distribution can be calculated through integration of the absolute frequency shown in the histogram and referencing with the total number of values in series under consideration (here 1000), from which the quantiles become apparent. In this case the 25% quantile is 11.8, for example, i.e. at least 25% of the individual values of the sample of 1000 values are below this value.



The library function blocks for calculating quantiles [▶ 148] operate in two substeps, which can be called up together or individually. In the first step values are added to an internal histogram, whose parameters can be configured in advance. This step requires very little computational effort. In the second step the previously selected quantiles are calculated from the stored histogram. Depending on the configuration, this second operation is significantly more computationally intensive, since it is defined through more complex operations, although it has to be executed much less frequently.

Calculation of quantiles in TwinCAT 3

The function block FB_CMA_Quantiles [▶ 148] can be used for calculating quantiles. Several quantiles can be calculated with a single function block call. The function block is parameterized like the histogram function block, plus the quantile to be calculated and sample size to be used.

2.2 Application concepts

This part of the introduction provides an overview of basic application patterns and solutions for Condition Monitoring tasks. The overview focuses on some underlying strategies and solutions, without providing programming and interface details. At the end of each concept an implementation scheme based on the Condition Monitoring Library is provided, thereby building up an overview of the library options.

You will learn the following:

- How does vibration monitoring according to ISO 10816-3 work?
- How does threshold value monitoring in the frequency range work?
- How is Condition Monitoring for a roller bearing configured?
- How is Condition Monitoring for a gear unit configured?

2.2.1 Vibration assessment

Introductory disambiguation

Vibration assessment aims to ensure reliable and safe operation of a machine, based on evaluation of the machine operating state by means of vibration measurements. Local diagnostics/analysis of machine components is outside the scope of this documentation. Solutions for diagnostic condition monitoring of components such as roller bearings and gear units are described separately below.

References to common standards

A number of standards exist for assessing machine vibrations, including the following:

- ISO 5348, Mechanical vibration and shock - Mechanical mounting of accelerometers
- ISO 10816, Mechanical vibration - Evaluation of machine vibration by measurements on non-rotating parts (previously VDI Guideline 2056). This standard consists of several parts.
  - ISO 10816-3 refers to industrial machines with a rated capacity of more than 15 kW and rated speeds between 120 rpm and 15000 rpm, measured on site.
  - ISO 10816-7 refers to centrifugal pumps for industrial application
  - ISO 10816-21 refers to wind turbines with horizontal axis and gearbox
- ISO 7919, Mechanical vibration - Evaluation of machine vibration by measurements on rotating shafts. This standard consists of several parts.
  - ISO 7919-3 refers to coupled industrial machines
  - ISO 7919-2 refers to stationary steam turbines and generators with a capacity of more than 50 MW an nominal operating speeds of 1500 min^{-1}, 1800 min^{-1}, 3000 min^{-1} and 3600 min^{-1}
- ISO 20816-1, Mechanical vibration - Measurement and evaluation of machine vibration. Consolidation of ISO 7919-1 and ISO 10816-1.

Evaluation of machine vibrations based on DIN ISO 10816-3

The scope of this standard includes steam turbines up to 50 MW, electric motors and fans. Because the scope is quite wide, the standard is explained in more detail below. The standard aims to classify the machine state in four different classes by means of vibration data for acceptance measurements and operational monitoring.

Assessment criteria according to the standard are the RMS value of the vibration velocity and the RMS value of the vibration displacement. Usually it is sufficient to measure the vibration velocity. The additional evaluation of the vibration displacement is recommended if low frequency components are encountered. If both vibration parameters are logged and analyzed, the poorer of the two determined classes is applied.

The frequency range of the vibrations to be captured depends on the machine speed:

- 10 Hz to 1000 Hz for speeds of more than 600 rpm
- 2 Hz to 1000 Hz for speeds of less than 600 rpm

Suitable measuring points are characterized by the fact that they reflect the dynamic forces of the machine as purely as possible. For example, locations where local resonances occur are not suitable. Suitable locations tend to be bearing stands and bearing covers; measurements are usually carried out in two orthogonal directions.

The classification also takes into account the machine substructure, subdivided into rigid and elastic substructures. If the lowest natural frequency of the whole system consisting of machine and substructure is at least 25% higher than the main exciting frequency (generally the rotational frequency), the substructure can be regarded as rigid, otherwise as elastic. This evaluation should be carried out separately for each measuring direction (two orthogonal directions, see above).

DIN ISO 10816-3:2009 describes four evaluation zones (A, B, C, D), with limit values as listed in the following table.

Machine group	1		2	
Installation	rigid	elastic	rigid	elastic

RMS value of the vibration velocity in mm/s	11.00 .. ∞	D	D	D	D
	7.10 .. 11.00	D	C	D	D
	4.50 .. 7.10	C	B	D	C
	3.50 .. 4.50	B	B	C	B
	2.80 .. 3.50	B	A	C	B
	2.30 .. 2.80	B	A	B	B
	1.40 .. 2.30	A	A	B	A
	0.00 .. 1.40	A	A	A	A

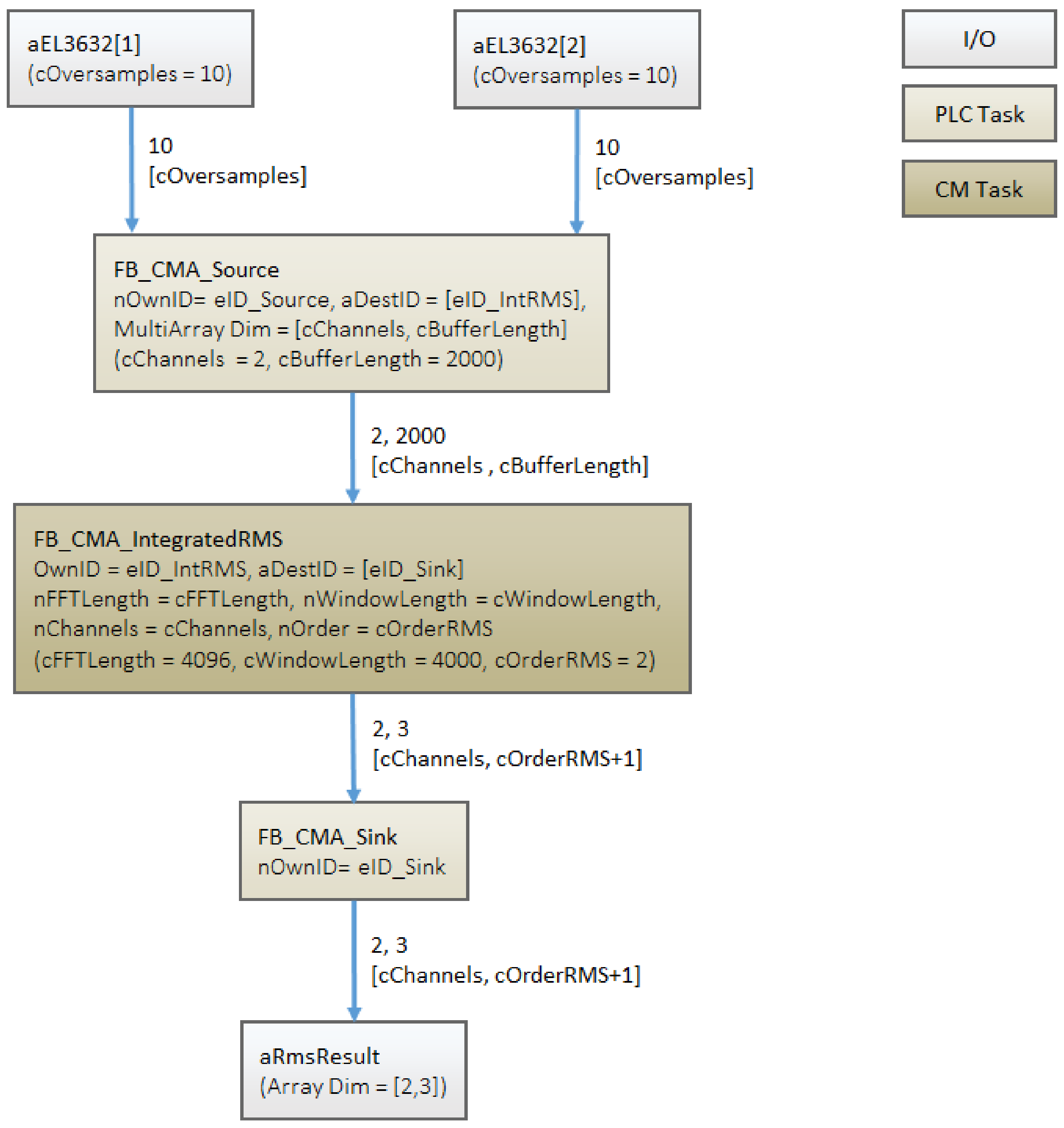
Machine group		1		2	
Installation		rigid	elastic	rigid	elastic
RMS value of the vibration displacement in µm	140 .. ∞	D	D	D	D
	113 .. 140	D	C	D	D
	90 .. 113	D	C	D	C
	71 .. 90	C	B	D	C
	57 .. 71	C	B	C	B
	45 .. 57	B	B	C	B
	37 .. 45	B	A	B	B
	29 .. 37	B	A	B	A
	22 .. 29	A	A	B	A
	0 .. 22	A	A	A	A

Zone A	The vibrations of recently commissioned machines tend to be in this zone.
Zone B	Machines with vibrations in this zone are usually regarded as suitable for continuous operation without restrictions.
Zone C	Machines with vibrations in this zone are usually regarded as unsuitable for continuous operation. The machine may generally be operated in this state for a limited period, until a suitable opportunity for remedial measures arises.
Zone D	Vibration values in this zone are usually regarded as dangerous in the sense that damage to the machine may occur.
Machine group 1	Large machines with a rated output of 300 kW to 50 MW and electrical machines with a shaft height of more than 315 mm
Machine group 2	Medium-sized machines with a rated output of 15 kW to 300 kW and electrical machines with a shaft height between 160 mm and 315 mm

Processing concept

The classification defined in DIN 10816-3 can easily be implemented in TwinCAT 3 with the Condition Monitoring Library. A proposal for implementation is described below.

The diagram below provides an abstract description of the processing structure for the data collected via the fieldbus. The numbers at the arrows represent the dimension of the (multi-)array transferred from one function block to the next. The color of the function block indicates the task in whose context the block runs. The corresponding sample can be downloaded from here [▸ 207].


aEL3632[1]
(cOversamples = 10)
10
[cOversamples]
aEL3632[2]
(cOversamples = 10)
10
[cOversamples]
I/O
PLC Task
CM Task
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_IntRMS],
MultiArray Dim = [cChannels, cBufferLength]
(cChannels = 2, cBufferLength = 2000)
2, 2000
[cChannels , cBufferLength]
FB_CMA_IntegratedRMS
OwnID = eID_IntRMS, aDestID = [eID_Sink]
nFFTLength = cFFTLength, nWindowLength = cWindowLength,
nChannels = cChannels, nOrder = cOrderRMS
(cFFTLength = 4096, cWindowLength = 4000, cOrderRMS = 2)
2, 3
[cChannels, cOrderRMS+1]
FB_CMA_Sink
nOwnID= eID_Sink
2, 3
[cChannels, cOrderRMS+1]
aRmsResult
(Array Dim = [2,3])

Data input

In the sample, oversampling is set to 10, and the PLC task linked to the I/Os is set to 1 ms. This results in a sampling rate of 10 kHz for the data input. According to the sampling theorem, signals in the spectrum up to 5 kHz can now be analyzed correctly, provided the anti-aliasing filter is set correctly in the I/O terminal (see Fourier analysis [▶ 13]).

Buffering of the data stream

The input data of the two channels are buffered in the MAIN routine with a source function block. Accordingly, a two-dimensional array with the size [cChannels, cBufferLength] is established. According to DIN ISO 10816-3, frequency range of 10 Hz to 1000 Hz should be evaluated for a rotational speed of more than 600 min^{-1}. The frequency resolution of the frequency analysis (calculated internally in the

IntegratedRMS function block) should therefore be well below 10 Hz. If a buffer of 4000 samples is used at a sampling rate of 10 kHz, the resulting frequency resolution is 2.5 Hz. If the Hann window is used, this is formally reduced to 2.5 Hz * 1.5 = 3.75 Hz. In addition, the FFT-length must be a power of 2 and greater than the WindowLength. The BufferLength results from a 50% overlap of the windows. The parameterization in terms of the internal FFT is defined accordingly in the GVL_Constant as follows:

```
cFFTLength     : UDINT  := 4096;            // length of FFT
cWindowLength  : UDINT  := 4000;            // 96 samples Zero padding
cBufferLength  : UDINT  := cWindowLength/2; // buffer due to 50% overlap
```

The result is an array of size [2.2000], as shown in the diagram above, for transfer to the FB_CMA_IntegratedRMS function block.

Frequency-selective RMS value calculation

In the function block FB_CMA_IntegratedRMS an FFT is now calculated, and the RMS value for the transferred frequency range (here 10 Hz to 1000 Hz) is calculated (formally several ranges may be specified). In addition to the RMS values of the direct input signal (when an accelerometer is connected, this is usually an acceleration signal), the function block also calculates the respective integrated parameters, i.e. the RMS value of the vibration velocity and the RMS value of the vibration displacement. Accordingly, the function block output is a two-dimensional [2,3] array (2 channels, 3 RMS values per channel).

```
// define frequency interval according to ISO 10816-3
// e.g. 10 .. 1000 Hz for rotating speed over 600r/min
cfLowerFrequencyLimit : UDINT := 10;
cfUpperFrequencyLimit : UDINT := 1000;

// Parameters for RMS calculation
cOrderRMS    : UDINT := 2; // acceleration, velocity, and displacement
cChannels    : UDINT := 2; // ISO 10816-3 says 2 orthogonal sensors
cResult_Length : UDINT := cOrderRMS+1; // nOrder+1 (see InfoSys)
```

In the settings referred to above the source function block requires 2000/10 = 200 cycles with 1 ms for filling the buffer. The cycle time of the PlcTask_CM should be less than 0.5 * 200 ms, see Task Setting [▶ 63]. Since the function block only requires little computing time, the cycle time of the PlcTask_CM is set to 10 ms. The transfer of the data from the source function block to FB_CMA_IntegratedRMS across the task boundaries is handled internally by the Condition Monitoring Library.

Analyzing the result

The results of the RMS value calculation are transferred back to the fast PLC task with 1 ms via a sink function block. All that is required for this purpose is specification of an array in the MAIN routine, which matches the size of the array at the output of FB_CMA_IntegratedRMS, see variable `aRMSResult`.

The sink function block sets a flag to TRUE when a valid result was calculated, see variable `bCalcuate`.

```
(* Push results to sink *)
fbSink.Output2D(    pDataOut         := ADR(aRMSResult),
                    nDataOutSize     := SIZEOF(aRMSResult),
                    eElementType     := eMA_TypeCode_LREAL,
                    nWorkDim0         := 0,
                    nWorkDim1         := 1,
                    nElementsDim0    := 0,
                    nElementsDim1    := 0,
                    pStartIndex     := 0,
                    nOptionPars        := 0,
                    bNewResult        => bCalculate );
```

Based on this flag, the result of the RMS value calculation can then be used in the MAIN routine. In this case the RMS values of the vibration velocity and the vibration displacement are checked for the limit values defined in the ISO standard. Simple IF queries are used in order to keep the sample simple.

The class according to ISO 10816-3 is determined for each two channels and stored in the variables `ISOClassIs_Vel` (for the classification with regard to the vibration velocity) and `ISOClassIs_Displ` (for the classification with regard to the vibration displacement). This sample results in four classifications. According to ISO 10816-3, the larger of the two values should be used, if orthogonally arranged sensors are used. In addition, the stricter evaluation should be used if both the vibration displacement and the vibration velocity are used. Accordingly, the worst case of the four evaluations is sought in the source code and defined as output variable `ISO_10816_Classification`.

Interaction and comments on the sample

In the sample two harmonic vibrations with identical amplitude (4 m/s^2) but different frequency (400 Hz and 35 Hz) are defined as input variables. While this acceleration amplitude means classification in class A for a frequency of 400 Hz for an evaluation based on vibration displacement and vibration velocity, for a vibration with 35 Hz an evaluation based on vibration displacement results in class C, for vibration velocity even down to class D. The output variable `ISO_10816_Classification` therefore corresponds to class D.

If the amplitude of the oscillation with 35 Hz is changed to 1 m/s^2, the classification changes to B (for vibration velocity) or A (for vibration displacement). Accordingly, the variable `ISO_10816_Classification` is set to B.

Alternatively, you can leave the amplitude at 4 m/s^2 and increase the frequency, e.g. to 800 Hz. This results in Class A classification throughout, and the variable `ISO_10816_Classification` is set to A.

2.2.2 Frequency analysis

Motivation

One of the main techniques for diagnostic/analytical machine monitoring is logging of vibrations with accelerometers and corresponding frequency analysis. This is based on the fact that machines are made of metal and therefore elastically resilient structures that are nearly always subjected to periodic forces. This leads to vibrations in which the excitation frequencies and forces and the characteristic frequencies of the respective structures are reflected. Vibration measurements therefore enable conclusions to be drawn regarding the structures and forces in the machine. Damage and structural changes of machine elements, such as bearings, result in changes to the vibration pattern.

The vibrations spread in the form of sound waves (structure-borne noise) in the machine components. Since machines consist of a large number of parts, which on the one hand elastically transfer vibrations originating from other parts and on the other hand oscillate themselves, the vibration patterns are characterized by filtering and superimpositions of the individual vibration components. Accordingly, a vibration signal consists of several components, which add up to the total signal based on different delays and attenuation that depend on the travelled path. Individual vibration components may therefore no longer be recognizable in the total signal curve. The power of frequency analysis is that it can split the linearly superimposed vibrations into frequency components. These frequency components can then be more readily allocated to a particular machine state, component or process.

The concept of frequency-selective monitoring of components is split into:

- Calculation of the spectrum
- Statistical evaluation of the result
- Threshold value monitoring.

Practical elements of frequency analysis

The key aspects of Fourier analysis were are already discussed in section Fourier analysis [▸ 11]. The main practical aspects are repeated here.

The following questions are of central importance for the configuration of the parameters for the function block for Fourier analysis (e.g. FB_CMA_MagnitudeSpectrum [▸ 127] or FB_CMA_PowerSpectrum [▸ 142]).

- What is the highest frequency to be analyzed?
  The sampling frequency should be set accordingly via the oversampling factor for the terminal and the corresponding task cycle time. An anti-aliasing filter should also be set. See section on Fourier analysis [▸ 13].

- What are the requirements for frequency resolution?
  The measuring time (length of the input array) should be as long as required, see Fourier analysis [▸ 13]. The deterioration of the frequency resolution when applying a window function should also be taken into account, see Window functions [▸ 18].
- The FFT length must be larger than the length of the input array, and it must be a power of two. The remaining elements are filled with zeros, see zero padding or Fourier analysis [▸ 13].
- Select a suitable scaling for the spectrum, see Scaling of spectra [▸ 22].

Statistical evaluation

The Fourier spectrum is very sensitive to noise and interference in the signal. Normally, the Fourier transform of noisy signals is therefore not particularly well suited for direct analysis or evaluation. In order to compensate this, the magnitude spectrum is usually averaged, or a quantile evaluation method is used, see Statistical analysis [▸ 25]. This approach requires temporal stability or cyclic repetition of the signals to be analyzed. The parameters determined in this way are significantly more robust against interference and easier to assess visually. An evaluation based on the average value of several spectra is illustrated below as an example.

Statistical analysis of the magnitude spectrum

It makes sense to form several magnitude spectra and analyze them statistically, e.g. via averaging or quantile calculation. This reduces the uncertainty of the determined values and makes a threshold value analysis more reliable.

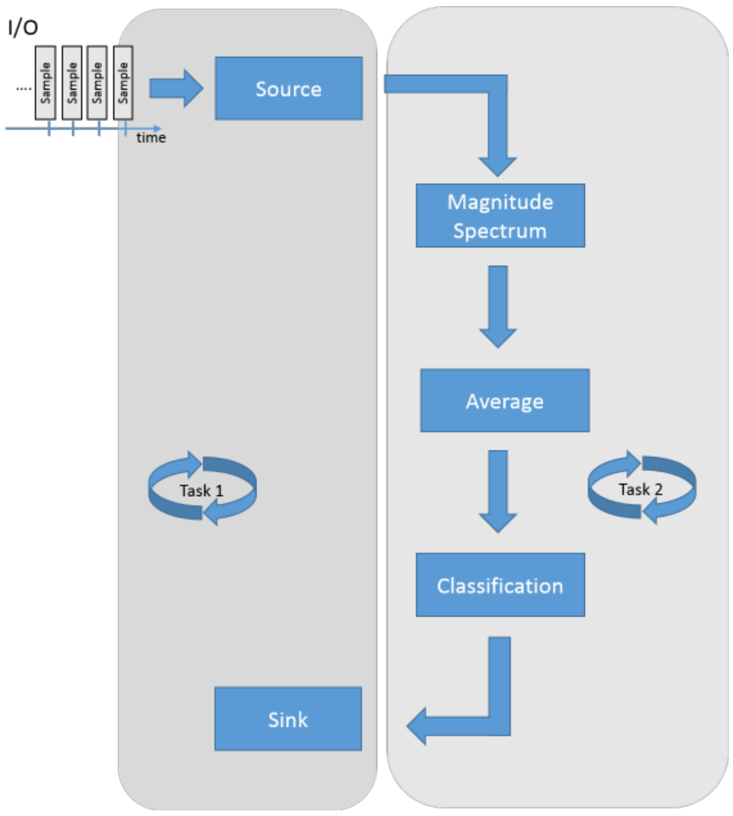
An alternative method is averaging of the calculated Fourier coefficients via the frequency. In this case several adjacent frequency values are combined through averaging. On the one hand this method is largely equivalent, although somewhat more computationally intensive, than calculating the FFT with lower frequency resolution and temporal averaging of the spectral values. The equivalence is a result of the fact that the averaging of several short FFT spectrums equivalent in terms of time with the calculation of a long spectrum and subsequent averaging via the frequency. The higher computational effort is a result of the fact that the complexity of the FFT calculation increases slightly above-proportional with increasing length. Excessive high frequency resolution should therefore generally be avoided.

Threshold value monitoring

The last step of the concept explained here consists of automatic threshold value monitoring. For each frequency channel threshold values are defined that are allocated to several categories of different priority, e.g. "normal operation", "warning" and "alarm". These threshold values can be set based on experiences and adjusted during operation.

Processing concept

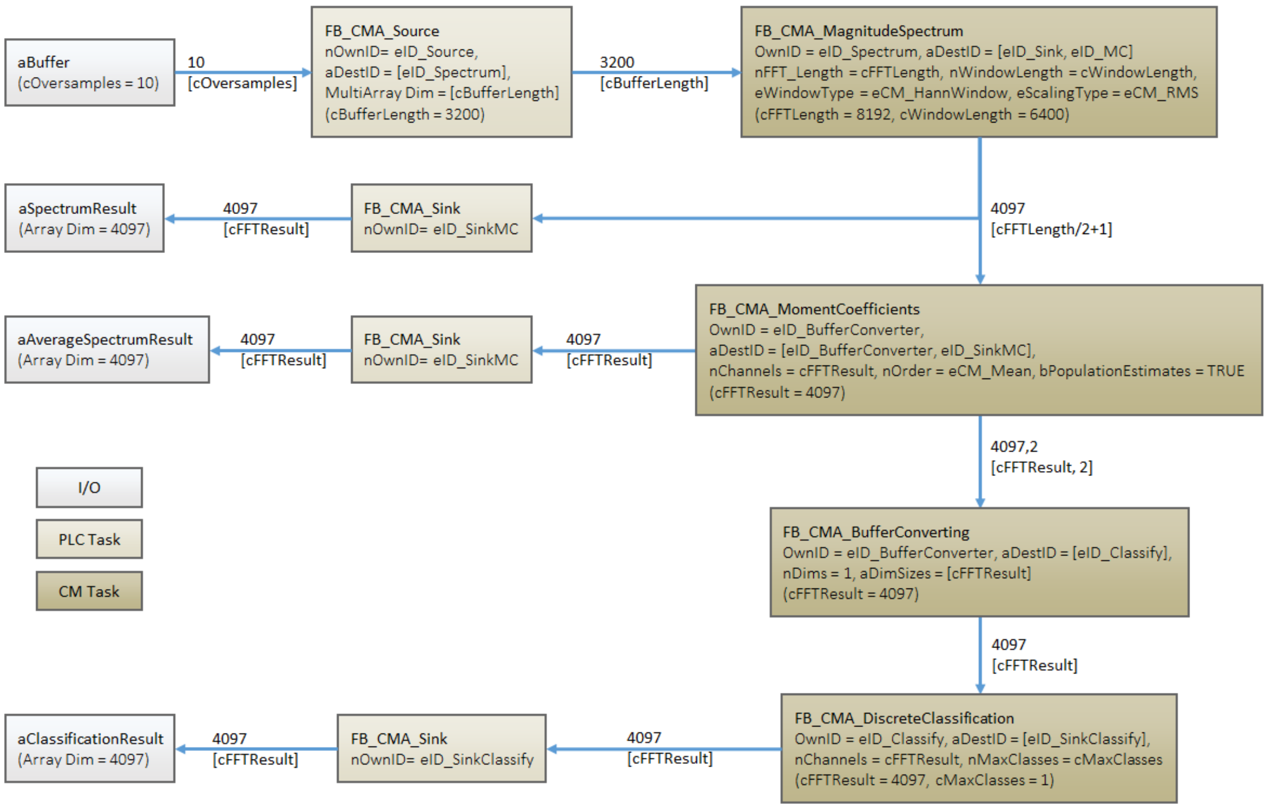
The concept described above can be implemented conveniently with the TwinCAT Condition Monitoring Library through parameterization of the function blocks provided. A sample configuration is described below.



Threshold value monitoring of averaged magnitude spectra is to be implemented. The following components of the Condition Monitoring Library with the described functions are used:

- FB_CMA_Source
  - Buffers of the input data
- FB_CMA_MagnitudeSpectrum
  - Arrangement of the input buffers into overlapping sections
  - Windowing of the calculation sections
  - Calculation of the Fourier transformation
  - Calculation of the absolute value of the Fourier coefficients
  - Scaling of the result (RMS)
- FB_CMA_MomentCoefficients
  - Formation of the arithmetic mean value
- FB_CMA_BufferConverting
  - Adjustment of the buffer dimensions for transfer between FB_CMA_MomentCoefficients and FB_CMA_DiscreteClassification
- FB_CMA_DiscreteClassification
  - Monitoring of each calculated frequency for threshold value violation

In a representation of the above diagram that is closer to the source code, a possible implementation is as follows:


aBuffer
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source,
aDestID = [eID_Spectrum],
MultiArray Dim = [cBufferLength]
(cBufferLength = 3200)
3200
[cBufferLength]
FB_CMA_MagnitudeSpectrum
OwnID = eID_Spectrum, aDestID = [eID_Sink, eID_MC]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
eWindowType = eCM_HannWindow, eScalingType = eCM_RMS
(cFFTLength = 8192, cWindowLength = 6400)
aSpectrumResult
(Array Dim = 4097)
4097
[cFFTResult]
FB_CMA_Sink
nOwnID= eID_SinkMC
4097
[cFFTLength/2+1]
aAverageSpectrumResult
(Array Dim = 4097)
4097
[cFFTResult]
FB_CMA_Sink
nOwnID= eID_SinkMC
4097
[cFFTResult]
FB_CMA_MomentCoefficients
OwnID = eID_BufferConverter,
aDestID = [eID_BufferConverter, eID_SinkMC],
nChannels = cFFTResult, nOrder = eCM_Mean, bPopulationEstimates = TRUE
(cFFTResult = 4097)
4097,2
[cFFTResult, 2]
I/O
PLC Task
CM Task
FB_CMA_BufferConverting
OwnID = eID_BufferConverter, aDestID = [eID_Classify],
nDims = 1, aDimSizes = [cFFTResult]
(cFFTResult = 4097)
4097
[cFFTResult]
aClassificationResult
(Array Dim = 4097)
4097
[cFFTResult]
FB_CMA_Sink
nOwnID= eID_SinkClassify
4097
[cFFTResult]
FB_CMA_DiscreteClassification
OwnID = eID_Classify, aDestID = [eID_SinkClassify],
nChannels = cFFTResult, nMaxClasses = cMaxClasses
(cFFTResult = 4097, cMaxClasses = 1)

The sample project for the concept shown here can be downloaded from here: download [▸ 214].

Parameterization of the calculation of a magnitude spectrum

The cycle time and the oversampling factor are set such that the resulting sampling rate is 10 kHz. The following settings are used in the sample for parameterization of the MagnitudeSpectrum function block and the source function block

```
// constant for input
cOversamples  : UDINT     := 10;    // number of oversamples
cFSample      : UDINT     := 10000; // 1ms task with 10 oversamples = 10kHz

// constants for FFT (Magnitude Spectrum)
cBufferLength : UDINT   := 3200;              // buffer size
cWindowLength : UDINT   := 2*cBufferLength;   // 50% overlap
cFFTLength    : UDINT:= 8192;// length of FFT for mag. spectrum, power of 2
cFFTResult    : UDINT   := cFFTLength/2+1;     // result of mag. spectrum
```

The numerical frequency resolution is 10 kHz / 8192 = 1.22 Hz. As described in the context of zero padding and Fourier analysis [▸ 13], this does not correspond to the frequency resolution, which enables two adjacent frequencies to be distinguished. In this case this is 10 kHz / (2*3200) * 1.5 = 2.34 Hz; 2*3200 corresponds to the length of the signal section used for calculating the FFT (measuring time in sampling values). The expansion factor of 1.5 is defined through the choice of the Hann window (windowing in the MagnitudeSpectrum function block). The FFT-length of 8192 is the smallest number greater than 2*3200 representing a power of two. The length of the result array from the calculation of magnitude spectrum is 4097. It is defined through the symmetry property of the FFT.

Averaging of the magnitude spectra

The result of the MagnitudeSpectrum function block is transferred to FB_CMA_MomentCoefficients, which is configured such that it returns the average value (first central moment) as result. By default, the function block also provides the sample size that was used for calculating the central moment. For this reason the result array becomes two-dimensional. The present sample uses the CallEx() method of the function block to average 25 magnitude spectra and then reset the function block. Since in this case the sample size is always 25, this information is no longer required. The corresponding sink function block is therefore parameterized such that only the mean values are copied from the function block result to the context of the PLC task. In addition, a buffer conversion (FB_CMA_BufferConverting) is applied between the classification function block and the MomentCoefficients function block, which also omits the column containing the sample size information.

Classification

In the present case, the function block FB_CMA_DiscreteClassification is used as simple threshold value classifier. Accordingly, only two classes are defined (nMaxClasses = 1). The configuration of the threshold values, which is applied individually for each discrete frequency, takes place at runtime. In the sample the threshold value for array indices 30 to 50 (corresponding to approx. 36 Hz to 61 Hz) is set to 6 V_{RMS}, for the remaining frequencies it is set to 2 V_{RMS}. If the value falls below the threshold value, -1 is returned as result for the corresponding frequency. If the threshold value is exceeded, 0 is output.

Further information on the sample code

The project includes an measurement project, which contains a scope array project with three axes. The upper diagram shows the result of FB_CMA_MagnitudeSpectrum, i.e. the magnitude spectrum of the input signal. The input signal is generated by a function generator and represents a noisy sine with 50 Hz and an amplitude of 25 V. Accordingly, the result of the non-averaged magnitude spectrum is time-variable (uncertain). Averaging stabilizes the result noticeably. The averaged magnitude spectrum is shown in the center of the Scope Array Project. The lower diagram shows the classification result; for each frequency -1 is shown if the value is below the threshold, 0 is shown if the defined threshold value is exceeded.

Further example for Condition Monitoring with frequency analysis

The Examples section contains several code examples. Section Condition Monitoring with frequency analysis [▸ 209] contains an example that is similar to the one described in this section. It is intended to illustrate the flexibility your individual solution, which you can create with the Condition Monitoring Library.

Overview: Various function blocks for frequency analysis

The TwinCAT 3 Condition Monitoring library offers various function blocks for frequency analysis. The following table provides an overview of the differences in the algorithms.

Function block	Input data type	Output data type	Window-ing	Comment
FB_CMA_RealFFT [▸ 145]	LREAL	LCOMPLEX	No	Pure FFT formation for real input signals
FB_CMA_ComplexFFT [▸ 86]	LCOMPLEX	LCOMPLEX	No	Pure FFT formation for complex input signals
FB_CMA_MagnitudeSpectrum [▸ 127]	LREAL	LREAL	Yes	FFT analysis with overlapping buffering and windowing, and formation of the magnitude spectrum.
FB_CMA_PowerSpectrum [▸ 142]	LREAL	LREAL	Yes	FFT analysis with overlapping buffering and windowing, and formation of the power spectrum.

2.2.3 Bearing monitoring

Motivation

Bearings are among the commonest and most highly stressed machine elements. In many cases they can be of critical importance for the operation of a plant. While the downtime alone can cause high costs for the procurement of spare parts for large bearings, the failure of small bearings can also cause costs that far exceed the costs of the spare part.

Causes of damage

There are many different possible causes of the failure of roller bearings:

- The ‘natural’ cause of the failure of roller bearings is material fatigue due to the high stresses that occur on the contact surfaces of the rolling elements during operation. After a certain time these lead to cracks in the material and to break-outs on the running surface. Small defects result that initially grow very slowly and become larger with increasing speed towards the end of the service life. The mechanisms of material fatigue are understood well in theory and can be statistically described; they are a component of normal wear. In designing a normal bearing the dimensions are selected such that the probability of serious damage within the service life of the machine is low. Under normal circumstances, therefore, it can be expected that correctly dimensioned and maintained bearings will have a very long service life. The service life actually attained is often considerably shorter, but not accurately predictable and can vary considerably due to the following causes.
- The stress on rolling elements and running surfaces is significantly increased by incorrect lubrication, since the lubricant distributes part of the stress and also prevents the bearing from overheating.
- A further cause of damage is contamination, for example due to faulty seals or metal swarf. The penetration of water can also lead to the failure of the lubrication, since even small amounts of water render lubricants unusable.
- Further, not unimportant sources of error are inaccuracies in the alignment or damaging stresses during the installation.
- Excessive stresses lead to plastic deformations of the running surface (brinelling). A similar situation can be caused by vibrations when the bearing is at a standstill, which are not mitigated by a lubricant film (false brinelling).
- In electrical machines the flow of current can destroy the running surfaces.
- Corrosion can be the cause of the initial surface damage.

The common factor in all these causes is that damage to the contact surfaces of the roller bearing can be detected at an early stage. From the fact that, in the overwhelming majority of cases, bearing failures are not caused by material fatigue, it follows that the early recognition of damage and the analysis and tracing of the primary causes (Root Cause Failure Analysis (RCFA)) make it possible in the mid-term to preventively avoid many types of damage and to significantly prolong the service lives of bearings, in addition to reducing downtime costs.

Consequences of damage

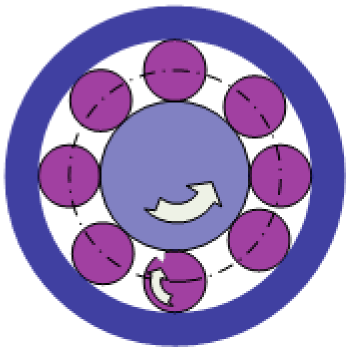
Following the initial damage to the running surfaces, the increasing stresses result in the spreading of defects. Apart from the running surface other components can also be affected, such as the cage of the rolling elements. Vibrations do not necessarily indicate the first stages of the damage process, since they usually represent a symptom rather than a cause of the damage. Nevertheless, all damage processes lead sooner or later to defects at the points of contact, which express themselves in the form of increasing vibrations.

Monitoring strategies

Since the direct recognition of the first causes may be difficult, the focus is placed on the early recognition of the consequential damage to the running surface of the bearing. The earlier this is noticed and investigated, the greater the chances are of finding the initial damage and rectifying it on the basis of the causes. This strategy often leads to sustainable savings in the long run. Furthermore, early recognition facilitates the planning of maintenance, which is an advantage above all for plant operators. Another strategy is to identify the elements concerned by analyzing the vibration signals.

To aid understanding of the following signal analysis options, a short phenomenological introduction into the formation of vibrations in defective roller bearings is provided.

Schematic cross-section of a roller bearing:



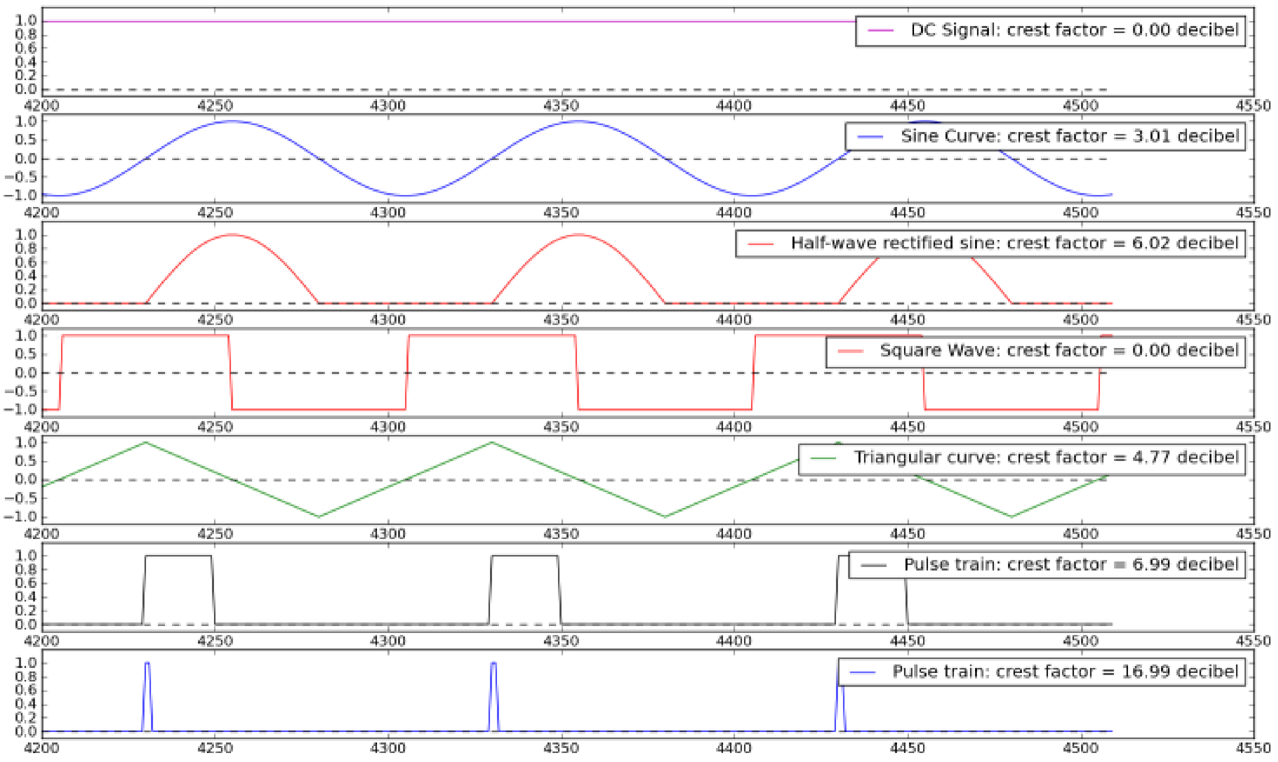
The critical parts of a roller bearing are moving surfaces in contact with each other. These are the rolling element surface, the contact surface of the inner race and the contact surface of the outer race. Rolling over local damage in a contact surface results in a shock pulse, which can be picked up by an accelerometer. The more severe the damage, the stronger the shock pulse.

Evaluation of the vibration in the time range

A simple method for evaluating the state of a roller bearing is to analyze the pulse content of the vibration signal. Common methods are the calculation of the crest factor of the kurtosis value.

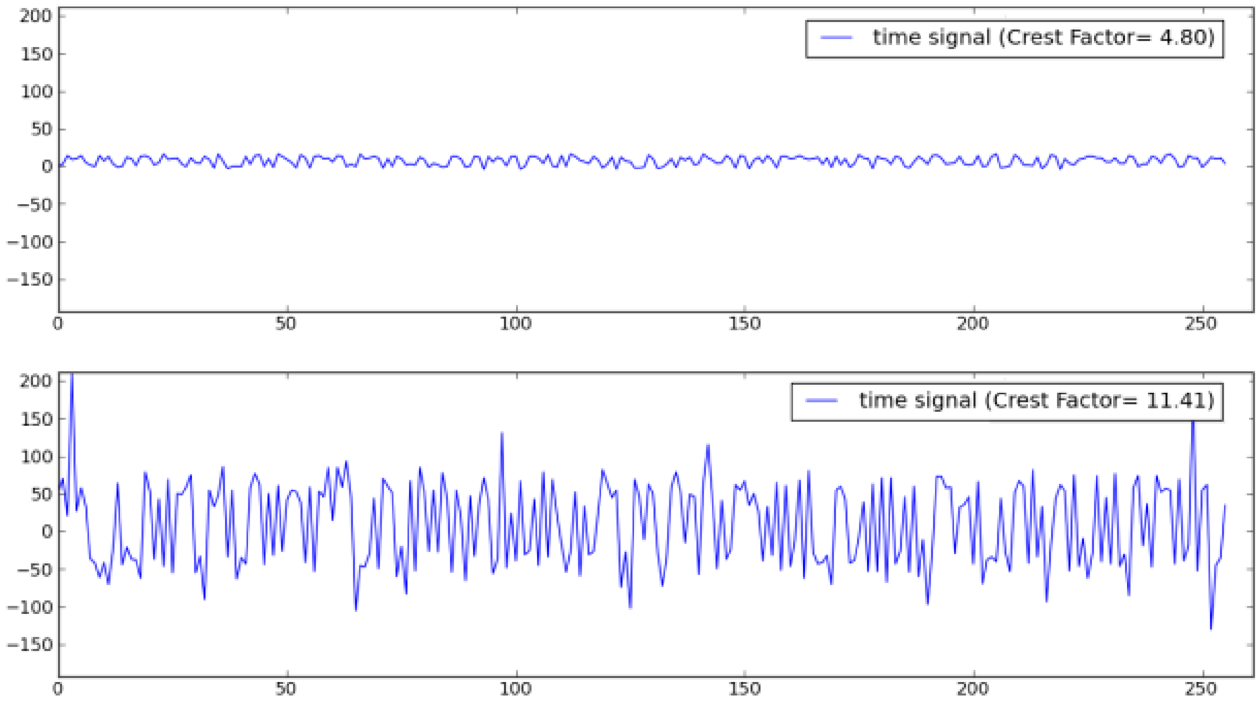
The crest factor

The crest factor is defined as the ratio between the maximum amplitude and the RMS value of the signal. It is specified in decibel and is a number greater than or equal to zero. The crest factor thus determines the relationship between maximum amplitude and the effective mean measured oscillation amplitude. Shock pulses resulting from incipient damage lead to an increase in the crest factor. The following diagram shows the increase in crest factor with increasing pulse content of the signals.



The bottom diagram shows the typical strong increase in the crest factor when acute shock pulses are encountered. An increase in the crest factor is usually a good indicator for damage. That makes this variable a suitable tool for the early recognition of damage and for trend analyses.

Signals from a roller bearing can be interpreted as follows.



The diagram above shows two vibration signals from bearings with different wear, in each case with the corresponding crest factor. The signal sequence at the bottom clearly shows peaks resulting from damage. While the undamaged bearing has a crest factor of 4.8 dB, the damaged component has a value of 11.4 dB, clearly indicating the presence of damage.

The crest factor has the advantage that it is very efficient to calculate and easy to interpret. In addition, it can easily be displayed in a diagram over the time. In order to be able to use it correctly, it is important to understand the fundamental limits of this type of evaluation:

- The crest factor is strongly influenced by the signal maximum and is therefore not a robust parameter in a statistical sense.
- The crest factor increases with increasing local damage. However, above a certain degree of damage, the peak values of the shock pulses will no longer increase significantly, although the number of local defects will increase. As a result, the signal maxima will not increase, while the RMS value of the signal continues to increase. Consequently, for heavily damaged bearings the crest factor will fall again.

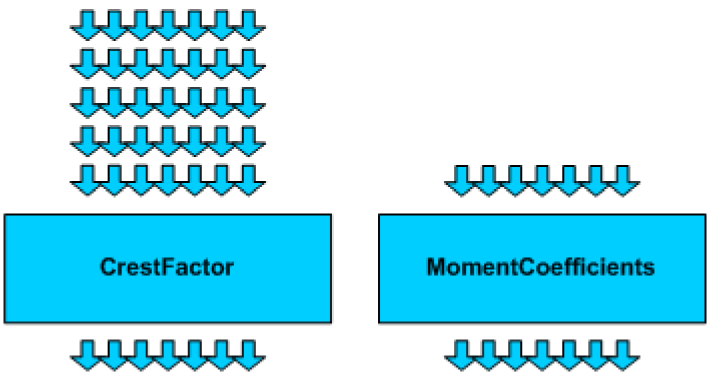
For this reason it is advisable to measure the crest factor continuously and analyze the results in terms of the trend.

The kurtosis value

In some cases the limited statistical robustness of the crest factor can be problematic. A more robust, yet somewhat more computationally demanding parameter is the kurtosis value (also referred to as curvature, fourth central moment). Like the average value and the variance, the kurtosis is a so-called moment coefficient, with can be used to describe parameters statistically. The kurtosis describes the ratio between the extreme values (far away from the mean value) of a distribution and the mean variation. Since occasional outliers in a measurement series have no significant effect on the result, the kurtosis is statistically much more robust than the crest factor.

In practice, the kurtosis tends to be used similar to the crest factor. The kurtosis (or the excess) and other common statistical moment coefficients are calculated in the TwinCAT 3 Condition Monitoring Library using the MomentCoefficients [▶ 130] function block.

Processing concepts



The above diagram shows the function blocks available for calculating the crest factor and kurtosis. The CrestFactor [▶ 83] function block can evaluate data from several sensors at the same time, provided that the number of individual values is the same for each channel. The return value consists of an individual value for each channel. The individual values are returned in a vector. In the above diagram this is indicated by the arrows in horizontal and vertical direction. The crest factor function block in each case contains 5 individual values (vertical) for 7 channels (horizontal) and returns the crest factor for each of the 7 channels.

The kurtosis can be evaluated with the MomentCoefficients [▶ 130] function block. Here the values are transferred alternatively for all channels and individual time steps, or block-by-block for several time steps, which is more efficient for single-channel signals due to the smaller overhead.

Envelope spectrum

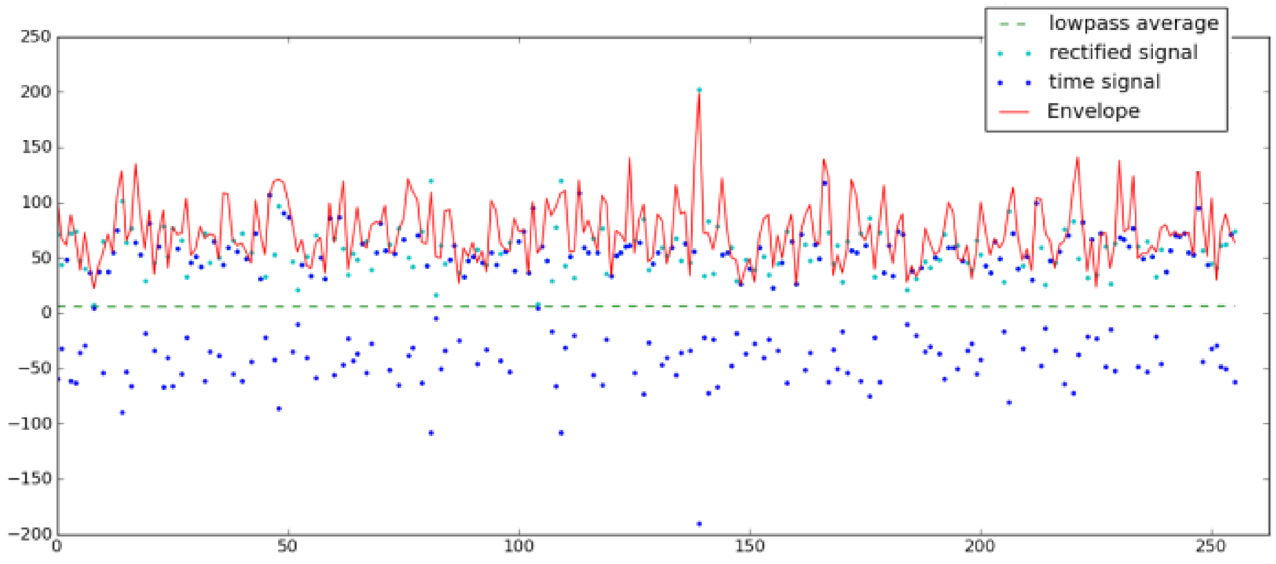
Theory

The determination of the crest factor or kurtosis provides early pointers to the presence of damage with very little expenditure. Since the dismantling and inspection of components always entails expense – in some cases considerable – and in view of the fact that there may be a large number of bearings, additional diagnostic possibilities are of interest with which damaged bearings or even individual components can be more accurately identified. The identification of defects is based on the evaluation of shocks which can be traced back to damage to the contact surfaces. In case of damage to a rotating part, the shock pulses occur periodically, wherein the length of the period depends on the frequency with which a defect touches the contact surface. This shock pulse period depends on the speed of rotation of the bearing on the one hand and on the geometry of the element on the other. Hence, the period of the shock pulses can identify the defective component.

The shock pulses contain a high-frequency signal component, which is due to the vibrations of the activated machine component, and a superimposed (folded) and possibly also a modulated low-frequency component, which contains information about the periodic repetitions of the shocks. These low-frequency portions of the signal can be determined by the calculation of the envelope. The envelope can be calculated efficiently by applying the Hilbert transformation in the frequency range. Prior filtering by a high-pass filter, such as that provided by the TwinCAT 3 Controller Toolbox, may be useful, but is not absolutely necessary. Following the calculation of the envelope the power spectrum of the envelope signal is determined. The distinctive frequencies of this envelope spectrum identify the shock periods.

Application

The envelope spectrum is helpful in particular for diagnosing which units or which components of a bearing may be defective. In addition to that, the possibility of evaluating specific important portions of the signal and excluding interfering parts is of interest for the early recognition of damage. If they are to be used for early recognition, then the damage frequencies in question must be determined from the bearing geometry and monitored.



The above diagram shows the envelope of the signal of a damaged roller bearing already used before. The time signal is marked by the blue points, the envelope by the red line. For better understanding a green dotted line plots the sliding mean value of the time signal, which is not exactly zero, and the light blue line plots the amounts of the negative values in the time signal. It can be seen that the envelope always lies very close to the maximum values of the time signal or the amount of the time values. Peaks or negative deflections in the time signal lead to peaks in the envelope, whilst ‘background noise’ in the time signal is changed very little by the envelope formation.

Analysis of the envelope spectrum

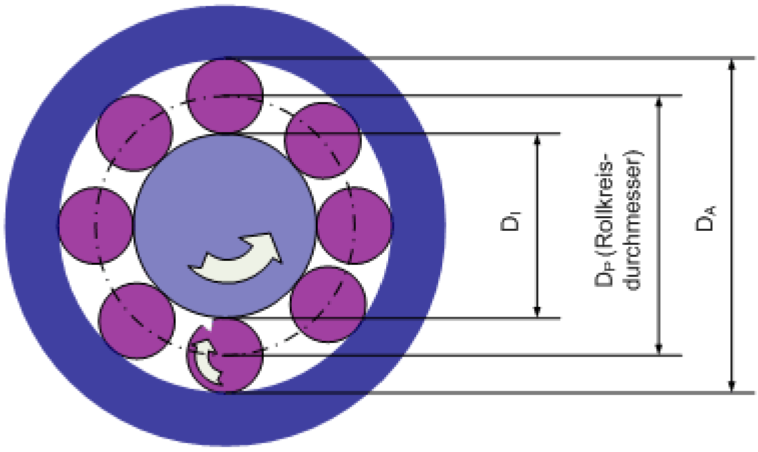
Since a sequence of periodic shocks (pulse train) corresponds to a signal with many harmonics, the envelope spectrum contains the base frequency on the one hand and the integer multiples of the base frequency on the other. Just like for a power or magnitude spectrum, the frequency associated with the spectral values is derived from the index of the result array multiplied with the frequency resolution of the FFT; see Fourier analysis [▸ 13]. with the length of the FFT *N* and the sampling rate f_s it follows: $\Delta f = f_s / N$ and therefore for the frequency of the frequency bin with index *m* : $f_m = (m-1)\ \Delta f$ (assuming the array index *m* starts with 1).

For diagnosis the base frequency of the pulse train must be identified. The harmonics are recognizable by a comb-like sequence of sharp maxima with an even spacing. The base frequency is the distance between these maxima, i.e. usually the frequency of the first maximum of this series. Their inverse value results in the period of the shocks; the unit of the inverse value is thus a time difference. Together with the rotational speed of the axis, which has to be measured, and the speed ratios of the damage frequencies, which can be determined from the bearing geometry, components from which the damage may have originated can be determined.

Characteristic damage frequencies in roller bearings

The illustration below shows by way of example the speed ratios that can occur in a simple roller bearing. In principle, shock pulses occur at the frequency with which the point of contact between two bearing elements passes a point with a damaged surface (on the upper side of the rolling element at the very bottom in the picture). This point of contact also moves due to the movement of the elements relative to each other. The rotary speed or angular speed of the point of contact can be determined based on the rule that there is hardly any slip in a correctly functioning bearing, which means that the elements roll off one another almost completely.

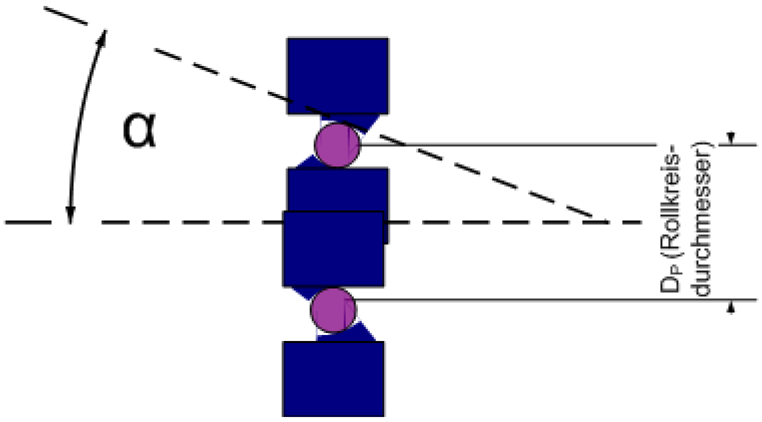
Roller bearing geometry parameter



Assuming the speed f_{rot} of an axis that is connected to the inner race is measured and the diameters of the bearing parts behave as follows: Diameter of the inner race D_I, diameter of the balls D_B, diameter of the outer race D_A. Suppose the number of balls is Z. D_I and D_A can be used to determine the **pitch diameter**: $D_P = (D_I + D_A)/2$ If the inner race rotates with a speed of f_{rot}, this can be used to determine the pulse frequency. The following acronyms are common for the designation of the frequencies:

- **BPFO** (Rolling element pass frequency outer race): Frequency with which the roller elements pass the outer race.
- **BPFI** (Rolling element pass frequency inner race): Frequency with which the roller elements pass the inner race.
- **BSF** (Bearing spin frequency rolling elements): Frequency with which the balls/rolling elements roll relative to a running surface.
- **BPF** (Ball pass frequency): Rolling element frequency, the frequency with which a defect on a ball passes a running surface.
- **FTF** (Fundamental Train frequency): Speed of rotation of the cage or the bearing element modulation frequency

Angle of contact:



For an accurate calculation in the case of bearings that bear axial loads, the diameter of the balls is to be corrected with the angle of contact *α* with which the balls touch the running surface: $D_b = \cos(\alpha)\ D_B$. For radial bearings this angle is 0°.This results in the following formulas used in practice:

$BPFO = Z * f_{rot} /2 * (1 - D_b / D_P)$
$BPFI = Z * f_{rot} /2 * (1 + D_b / D_P)$
$BSF = f_{rot} /2 * D_P/D_B * (1 - (D_b/D_P)^2)$
$BPF = 2 * BSF$
Rotating inner race:
$FTF = f_{rot} /2 * (1 - D_b/D_P)$
Rotating outer race:
$FTF = f_{rot} /2 * (1 + D_b/D_P)$ $(BPFI + BPFO) / f_{rot}$ always equals the number of roller elements Z. Slight deviations result from these formulas in practice because, for example, the angle of contact *α* can vary under load. As a simple rule of thumb, the value

$f_{BPFI} = 0.6 * f_{rot} * Z$

is often used as indicator frequency for a defective inner race, while

$f_{BPFO} = 0.4 * f_{rot} * Z$

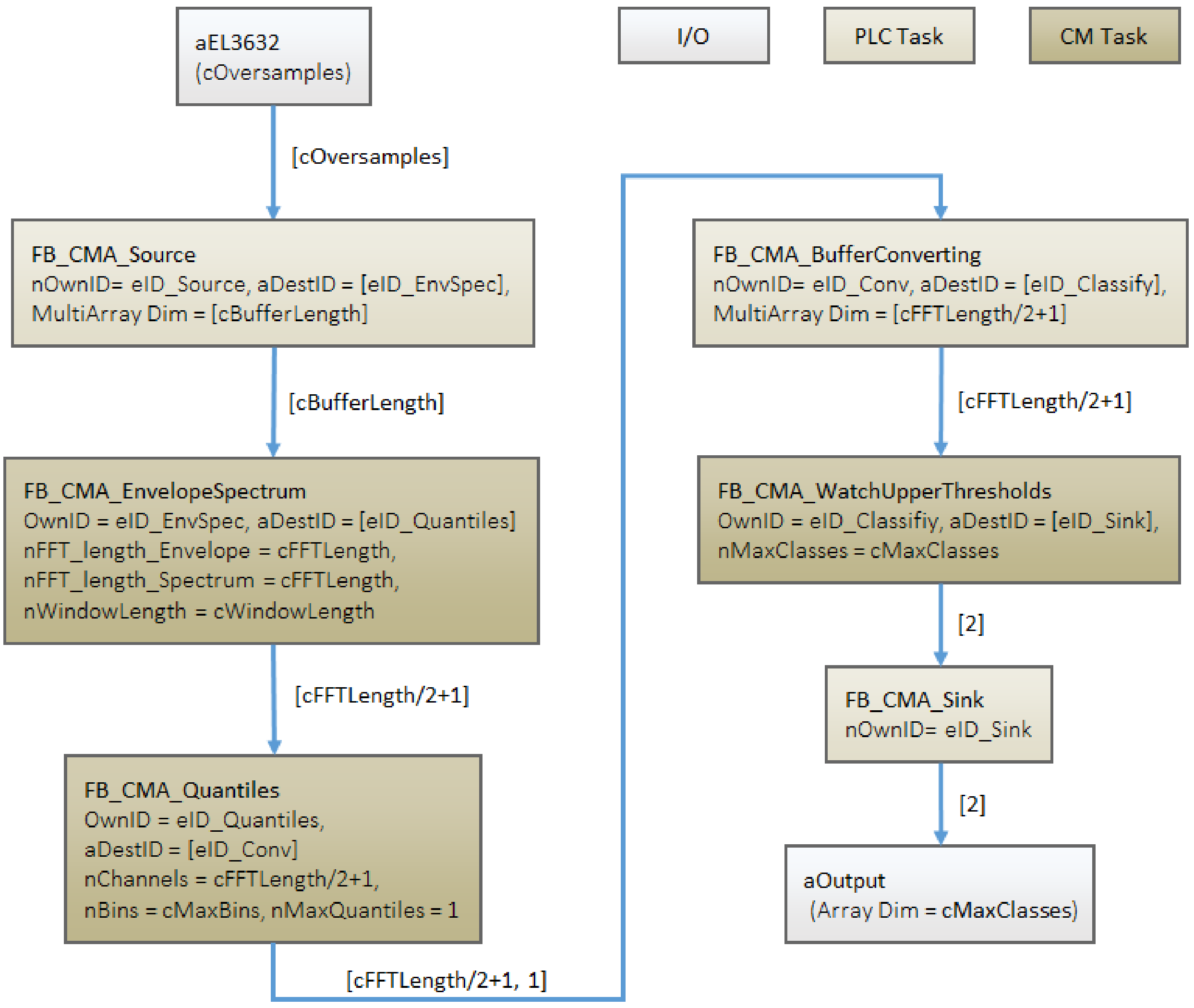
is used as indicator for a defective outer race. For the determination of the bearing geometry it is useful to refer to the bearing manufacturer’s data. It may be helpful to use calculation programs made available for download by some manufacturer.

Praktischer Hinweis: The type number of a roller bearing does not allow any clear conclusion to be drawn with regard to the bearing geometry; parameters such as the number of rolling elements can by all means change.

Processing concept

Frequency analysis processing steps

Analysis steps:


aEL3632
(cOversamples)
I/O
PLC Task
CM Task
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_EnvSpec],
MultiArray Dim = [cBufferLength]
[cBufferLength]
FB_CMA_EnvelopeSpectrum
OwnID = eID_EnvSpec, aDestID = [eID_Quantiles]
nFFT_length_Envelope = cFFTLength,
nFFT_length_Spectrum = cFFTLength,
nWindowLength = cWindowLength
[cFFTLength/2+1]
FB_CMA_Quantiles
OwnID = eID_Quantiles,
aDestID = [eID_Conv]
nChannels = cFFTLength/2+1,
nBins = cMaxBins, nMaxQuantiles = 1
[cFFTLength/2+1, 1]
FB_CMA_BufferConverting
nOwnID= eID_Conv, aDestID = [eID_Classify],
MultiArray Dim = [cFFTLength/2+1]
[cFFTLength/2+1]
FB_CMA_WatchUpperThresholds
OwnID = eID_Classifiy, aDestID = [eID_Sink],
nMaxClasses = cMaxClasses
[2]
FB_CMA_Sink
nOwnID= eID_Sink
[2]
aOutput
(Array Dim = cMaxClasses)

The above diagram shows the processing steps for the envelope spectrum as well as the function blocks that can be used here. First of all the envelope is calculated using the Envelope [▶ 109] function block. Subsequently the power spectrum is calculated (PowerSpectrum [▶ 142]) function block, in the same way as the spectrum for any time signal. Since the envelope spectrum obtained fluctuates relatively strongly with non-stationary signals, it is recommended to evaluate it statistically using the quantile calculation method (Quantiles) as described above in the section Frequency analysis [▶ 35]. The values obtained can be automatically checked for adherence to certain threshold values by means of limit value monitoring using the WatchUpperThreshold [▶ 162] function block.

2.2.4 Gearbox monitoring

Motivation

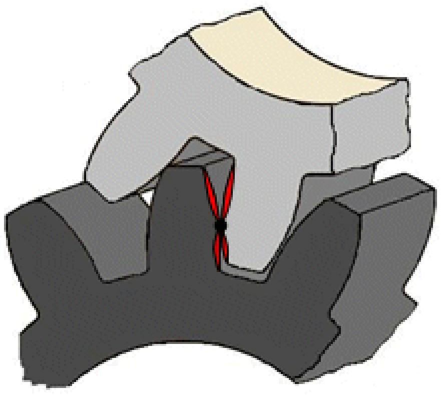
This section describes the concept of the monitoring of gearboxes. Like roller bearings, gearboxes are among the commonest machine elements. Since they are used in a wide range of drives, they usually play a key role for the reliable function of a system. Typical gearbox damage differs from damage in roller bearings. This is due on the one hand to the fact that in gearboxes highly stressed parts slide directly on top of one another, which places particular demands both on the lubrication and on the quality of the surface. Due to the forces resulting from normal operation that have to be absorbed, gearboxes are relatively large and thus expensive and a replacement may be necessary during the service life of the machine even if maintenance has been performed correctly. Adequate lubrication and correct assembly are also important here. However, the damage patterns that occur are by no means exclusively attributable to errors in these points. Excessive voltages at the contact points or interaction between corrosion and overheating can lead to incipient surface damage (pitting, micropitting, spalling, wear) right up to chipping and deformation of the tooth surfaces. Mechanical shocks and overload can cause the direct breakage of gear wheels. Compared with roller bearings, gear unit defects tend to result in abrupt failure and significant consequential costs. This is due to the fact that in gearwheels the greatest tension is at the tooth base – see diagram below (red surfaces). Consequently, fatigue symptoms occur at an early stage there, which lead in the course of the time to deep cracks and ultimately to the breaking off of teeth. The latter leads in extreme cases to the whole gearbox blocking apparently without preliminary warning and causing extensive consequential damage, for example due to the breakage of axles. The causes just mentioned and the consequential behavior give rise to two objectives for the monitoring of gearboxes:

- Firstly, it is of interest to monitor symptoms of wear on a long-term basis and to recognize problems at an early stage through trend observations and to rectify them promptly, before damage occurs.
- Secondly, acute damage can be recognized immediately by monitoring, whereby repair measures can be initiated earlier and failures and downtime can be reduced.

Theory

The theoretical background of the early detection for gearbox damage is briefly outlined below.

Meshing oscillations

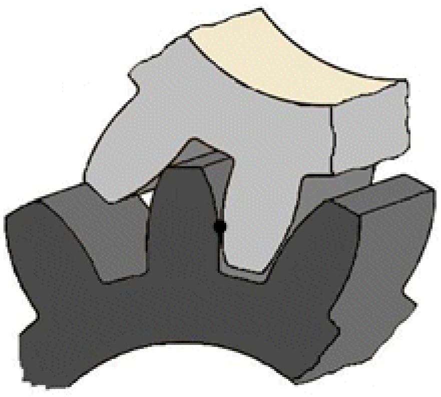


In a gearbox the gear wheels roll off one another, in the course of which the individual teeth periodically come into contact, transfer force and then separate from each other again. While it is possible for this to take place with a precisely constant transmission ratio and largely constant force in a new, well-designed gearbox (involute toothing), it is not feasible for this roll-off to take place without a portion of sliding movement. As the above picture shows, a predominantly rotary motion takes place in the center of the tooth surface, with a growing portion of sliding movement as the distance from the center increases. In addition, the speed ratio is largely constant with such toothing, but the transmitted torque varies. Since the teeth are made of hard, elastic material and therefore deform slightly, they are excited to oscillate with the period of the meshing – the so-called meshing frequency.

Harmonics of the meshing frequency

Since the meshing oscillation is a forced oscillation that does not have a sinusoidal appearance, but is based on the comparatively sudden occurrence and abatement of the forces, it consists in the spectrum of numerous harmonics whose frequencies are integer multiples of the meshing frequency. The oscillations depend on the load on the gear wheel, since the torque deforms the teeth elastically. Gear wheel oscillations are thus load-dependent.

Consequences of wear



With increasing wear the tooth profiles deviate more and more from the ideal shape, since material is removed by the sliding of the surfaces over one another. This happens more and more intensively the further away the surface is from the center of the tooth flank, as the diagram above shows. The sliding motion itself therefore increases and the torque varies more strongly, whereby the meshing oscillations and in particular the harmonics they contain are amplified. *The analysis of the harmonics is thus the key to the evaluation of the condition of the gearbox.* Note that the sudden reduction in the harmonics in an already clearly damaged gearbox must be taken as an alarm signal: The breakage of a tooth flank may be so advanced that the elasticity of the toothing has increased. In this case the total failure of the gearbox can be expected soon.

The cepstrum

The cepstrum is the most important tool for the analysis of gearbox oscillations as well as harmonics and modulations. This is an operation that highlights periodicities in the signal spectrum.

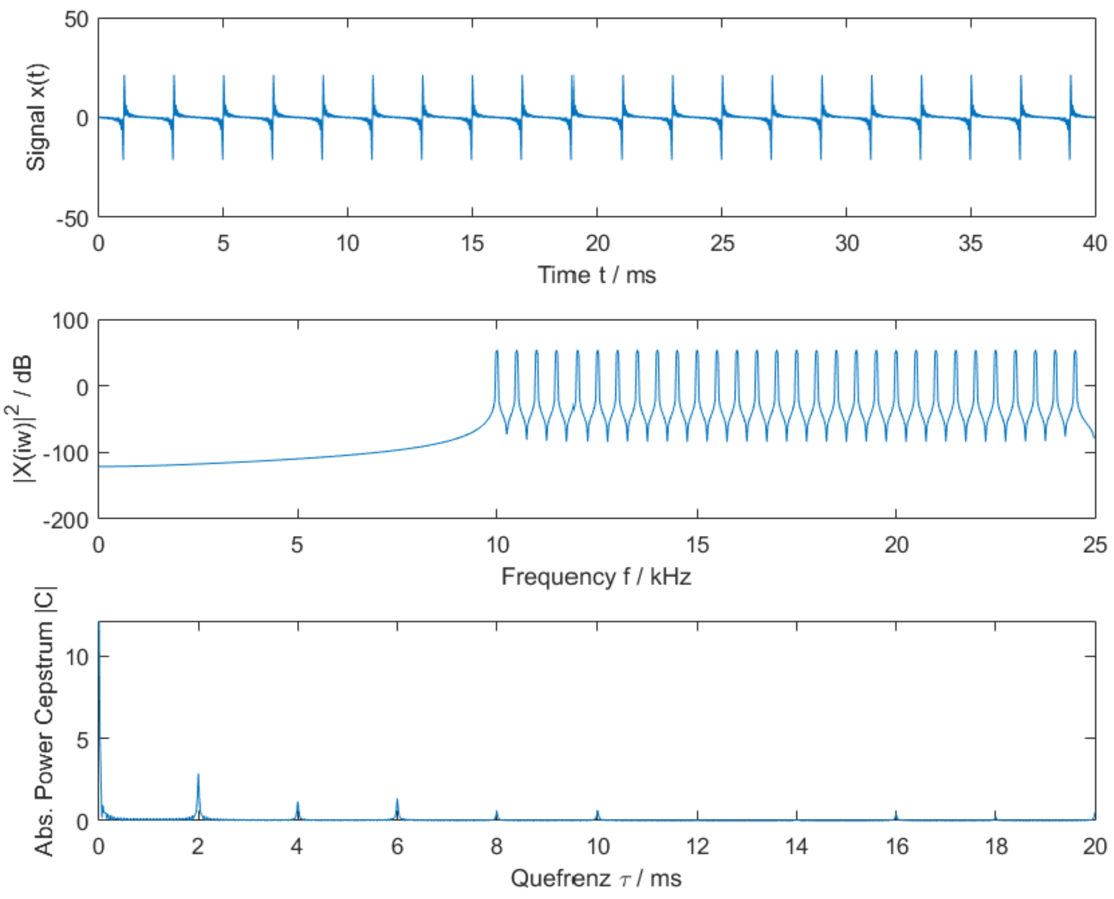
The power cepstrum for a signal *x*(*t*) is defined as:

$$C_\mathrm{p}(\tau) = X^{-1}\left\{\log(|X(\omega)|^2)\right\} = X^{-1}\left\{2\log|X(\omega)|\right\}, \qquad \text{Quefrency } \tau$$

Interpretation

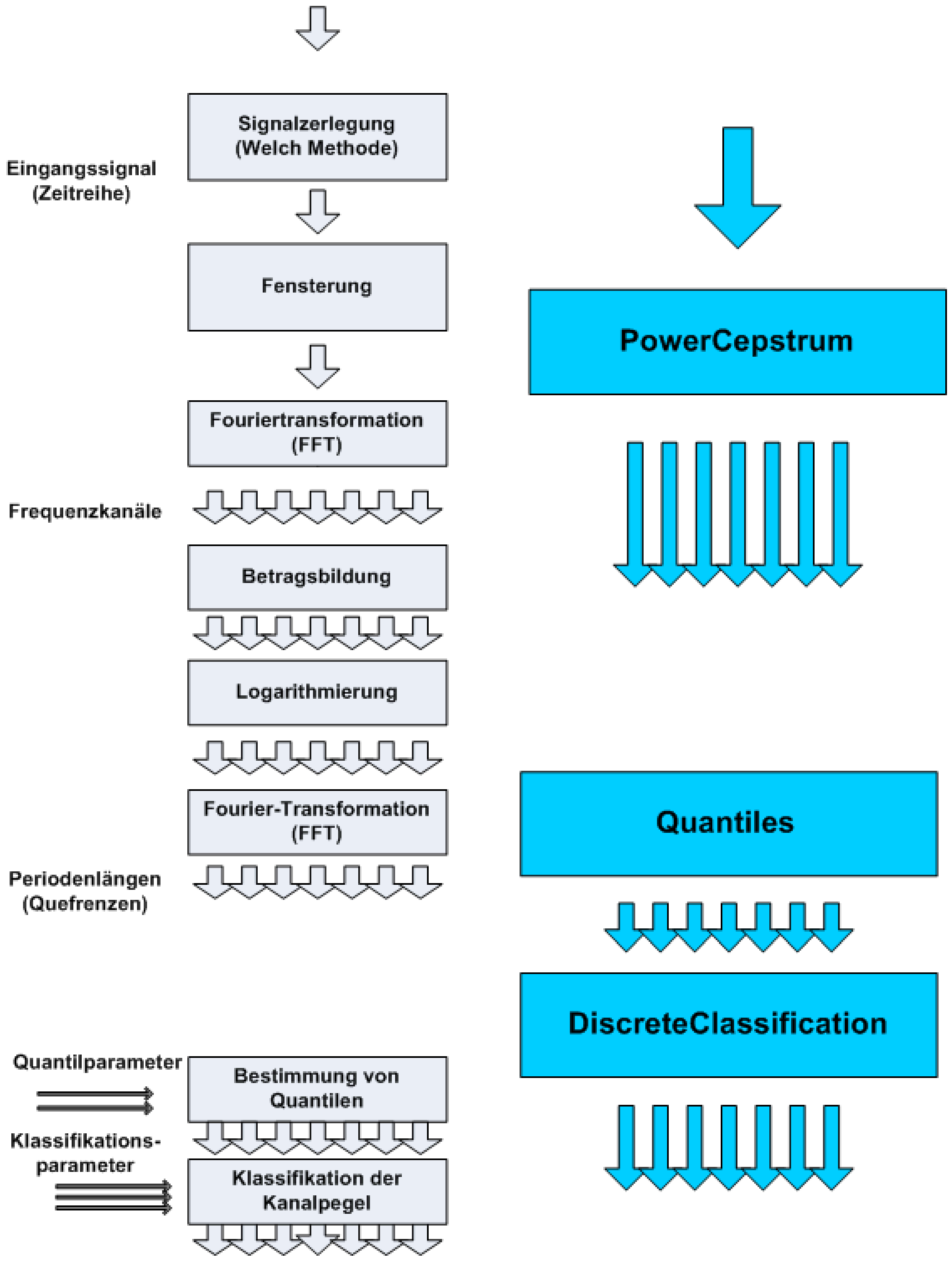
While a Fourier analysis indicates periodicities in the time range of a signal, the cepstrum indicates periodicities in the frequency range. An inverse Fourier transformation maps the result back into the time domain. However, the associated value index does not represent the original time axis relating to *t*, but the spectrum periods that have occurred. The parameter has the unit of time and is referred to quefrency, to indicate that it is a combination of inversion and inverse transformation. There are similar differentiating designations, for example, for entities and operations such as harmonic, filtering and phase analysis. The longer the length *N* of the two Fourier transforms employed is, the more input values are referred to for the calculation of the cepstrum, which reduces the influence of noise and (non-systematic) fluctuations. The time resolution can only be enlarged if the sampling rate is increased.

As an example, the following diagram shows the power spectrum and power cepstrum of a so-called harmonic sound complex. The time domain of the signal shows a repeated pulse every 2 ms. Each individual pulse is made up of superimposed harmonics, which means the situation is similar (coarse model) to the case of the gear unit damage described above. The diagram in the center shows the power spectrum. The periodicity of the power spectrum is clearly visible; the maxima are 0.5 kHz apart. The bottom diagram shows the magnitude the power cepstrum. The largest (global) maximum is at a quefrency of 0 ms, what has no relevance in practice (it merely shows the average value of the power spectrum). Apart from this maximum, the largest maximum can be seen at 2 ms, which precisely corresponds to the temporal repetition of the time signal or the reciprocal value of the distance of the local maxima in the power spectrum 1/0.5 kHz = 2 ms.



Signal x(t)
50
0
-50
0
5
10
15
20
25
30
35
40
Time t / ms
$|X(iw)|^2$ / dB
100
0
-100
-200
0
5
10
15
20
25
Frequency f / kHz
Abs. Power Cepstrum |C|
10
5
0
0
2
4
6
8
10
12
14
16
18
20
Quefrenz τ / ms

Processing concept (calculation steps)


Eingangssignal (Zeitreihe)
Signalzerlegung (Welch Methode)
Fensterung
Fouriertransformation (FFT)
Frequenzkanäle
Betragsbildung
Logarithmierung
Fourier-Transformation (FFT)
Periodenlängen (Quefrenzen)
Quantilparameter
Bestimmung von Quantilen
Klassifikations-parameter
Klassifikation der Kanalpegel
PowerCepstrum
Quantiles
DiscreteClassification

Calculating the power cepstrum

The calculation of the cepstrum is based, as follows from the definition, on the "normal" frequency analysis. Accordingly, as described in section Analysis of data streams [▶ 16], initially the signal has to be split into sections, followed by multiplication with a window function, also referred to as "windowing". The power cepstrum is then calculated based on the calculation steps described about, i.e. Fourier transformation, absolute value calculation, logarithmic calculation and further Fourier transformation. It is important here to avoid exceeding value ranges because, similar to division by zero, the logarithm of zero is not defined.

The initial calculation result has a complex value. Typically, the magnitude or the square of the magnitude is used for the further analysis.

A sample is available for download from here: Power cepstrum [▶ 220]

Calculation of quantiles

The short-term values of the cepstrum usually fluctuate quite strongly like those of the FFT from which they are derived. Therefore the next recommended processing step is the calculation of quantiles for each period obtained, i.e. each quefrency. For monitoring tasks, for example, the 95% quantile will often be determined. This is the value that will not be exceeded by the measured values in 95% of all cases. This calculation takes place as with the frequency analysis using the Quantiles [▶ 148] function block.

Threshold value monitoring

Further processing depends on the specific objective:

- For trend analysis, it is useful to save the values obtained and to display their development over long periods.
- For automatic machine monitoring, a classification with configurable thresholds or limit values is useful. This is done by the DiscreteClassification function block [▶ 89] sketched in here.
- For tasks such as machine protection with limited scope for individual analysis, the WatchUpperThreshold function block [▶ 162] can be used, which automatically calculates the number of the highest limit category. If, for example, the state ‘Everything OK’ is assigned to category 0, the state ‘Warning’ to category 1 and the state ‘Alarm’ to category 2, then a warning can be sent by a text message when Level 1 is the output and the plant can be switched off automatically if Level 2 is the output.

2.3 Literature notes

Information - not recommendations - on secondary literature is provided below. The list is not all-embracing, and only provides a small subset of the relevant literature.

Digitale Signalverarbeitung, Fourier-Analyse, Fensterung (Deutsch)

- A.V. Oppenheim, R.W. Schafer, J.R. Buck: Zeitdiskrete Signalverarbeitung. Pearson Studium, 2004. ISBN 3-8273-7077-9
- K.-D. Kammeyer, K. Kroschel: Digitale Signalverarbeitung – Filterung und Spektralanalyse mit MATLAB-Übungen. Teubner, 2002. ISBN 3-519-46122-6

Discrete-Time signal processing, Fourier-analysis, windowing (English)

- A.V. Oppenheim, R.W. Schafer, J.R. Buck: Discrete-Time Signal Processing. Pearson Education, 2009. ISBN 987-0131988422
- J.G. Proakis, D.K. Manolakis: Digital Signal Processing. Pearson Education, 2013. ISBN 978-0131988422

Zustandsüberwachung (Deutsch)

- J. Kolerus, J. Wassermann: Zustandsüberwachung von Maschinen. Expert Verlag, 2008. ISBN: 978-3-8169-2597-2
- DIN ISO 10816, Mechanische Schwingungen – Bewertung der Schwingungen von Maschinen durch Messung an nicht-rotierenden Teilen (vorher VDI-Richtlinie 2056). Die Norm besteht aus mehreren Bestandteilen
  - DIN ISO 10816-3 bezieht sich auf industrielle Maschinen mit einer Nennleistung über 15 kW und Nenndrehzahlen zwischen 120 U/min und 15000 U/min bei Messung am Aufstellungsort.
  - DIN ISO 10816-7 bezieht sich auf Kreiselpumpen für den industriellen Einsatz
  - DIN ISO 10816-21 Windenergieanlagen mit horizontaler Drehachse und Getriebe beziehen
- DIN ISO 7919, Mechanische Schwingungen - Bewertung der Schwingungen von Maschinen durch Messungen an rotierenden Wellen. Die Norm besteht aus mehreren Teilen
  - DIN ISO 7919-3 bezieht sich auf Gekuppelte industrielle Maschinen

    - DIN ISO 7919-2 bezieht sich auf Stationäre Dampfturbinen und Generatoren über 50 MW mit Nenn-Betriebsdrehzahlen von 1500 min^{-1}, 1800 min^{-1}, 3000 min^{-1} und 3600 min^{-1}
- DIN ISO 20816-1, Mechanische Schwingungen – Messung und Bewertung der Schwingungen von Maschinen. Zusammenfassung von DIN ISO 7919-1 und DIN ISO 10816-1.
- DIN ISO 13373-1, Zustandsüberwachung und -diagnostik von Maschinen - Schwingungs-Zustandsüberwachung - Teil 1: Allgemeine Anleitungen
- DIN ISO 13373-2, Zustandsüberwachung und -diagnostik von Maschinen - Schwingungs-Zustandsüberwachung - Teil 2: Verarbeitung, Analyse und Darstellung von Schwingungsmesswerten
- DIN ISO 17359, Zustandsüberwachung und -diagnostik von Maschinen - Allgemeine Anleitungen

Condition Monitoring (English)

- R.B. Randall: Vibration-based Condition Monitoring. Wiley, 2011. ISBN: 978-0-470-7485-8
- ISO 10816, Mechanical vibration -- Evaluation of machine vibration by measurements on non-rotating parts.
    - ISO 10816-3 Industrial machines with nominal power above 15 kW and nominal speeds between 120 U/min and 15000 U/min when measured in situ.
    - ISO 10816-7 Rotodynamic pumps for industrial applications, including measurements on rotating shafts
    - DIN ISO 10816-21 Horizontal axis wind turbines with gearbox
- ISO 7919, Mechanical vibration -- Evaluation of machine vibration by measurements on rotating shafts.
    - ISO 7919-3 Coupled industrial machines
    - ISO 7919-2 Land-based steam turbines and generators in excess of 50 MW with normal operating speeds of 1 500 r/min, 1 800 r/min, 3 000 r/min and 3 600 r/min
- ISO 13373-1, Condition monitoring and diagnostics of machines - Vibration condition monitoring -Part 1: General procedures
- ISO 13373-2, Condition monitoring and diagnostics of machines - Vibration condition monitoring - Part 2: Processing, analysis and presentation of vibration data
- ISO 17359:2011, Condition monitoring and diagnostics of machines - General guidelines

3 Installation

3.1 System requirements

The following article describes the minimum requirements needed for engineering and/or runtime systems. The Condition Monitoring setup is to install on engineering and runtime system.

Engineering enviroment

An engineering environment describes a computer which used to develop but NOT run PLC or other application code. On an engineering computer, the following requirements are needed:

- TwinCAT3 XAE (engineering installation) build 4018 or higher
- Please note: For engineering purposes, a 7-Day trial license may be (repeatedly) used, as described in our licensing article

Runtime environment

A runtime environment describes a computer which runs PLC programs. On a runtime computer, the following requirements are needed:

- TwinCAT3 XAR (runtime installation) build 4018 or higher
- 32 bit and 64 bit systems are supported
- Licenses for TC1200 PLC and for TF360X Condition Monitoring
- Please note: For testing purposes, a 7-Day trial license may be used, as described in our licensing article

Engineering and runtime on the same computer

In case you would like to run both the engineering and runtime environments on the same computer (for example to test the PLC program before downloading it to the target runtime), the following requirements are needed:

- TwinCAT3 XAE (engineering installation) build 4018 or higher
- Licenses for TC1200 PLC and for TF360X Condition Monitoring
- Please note: For testing purposes, a 7-Day trial license may be used, as described in our licensing article

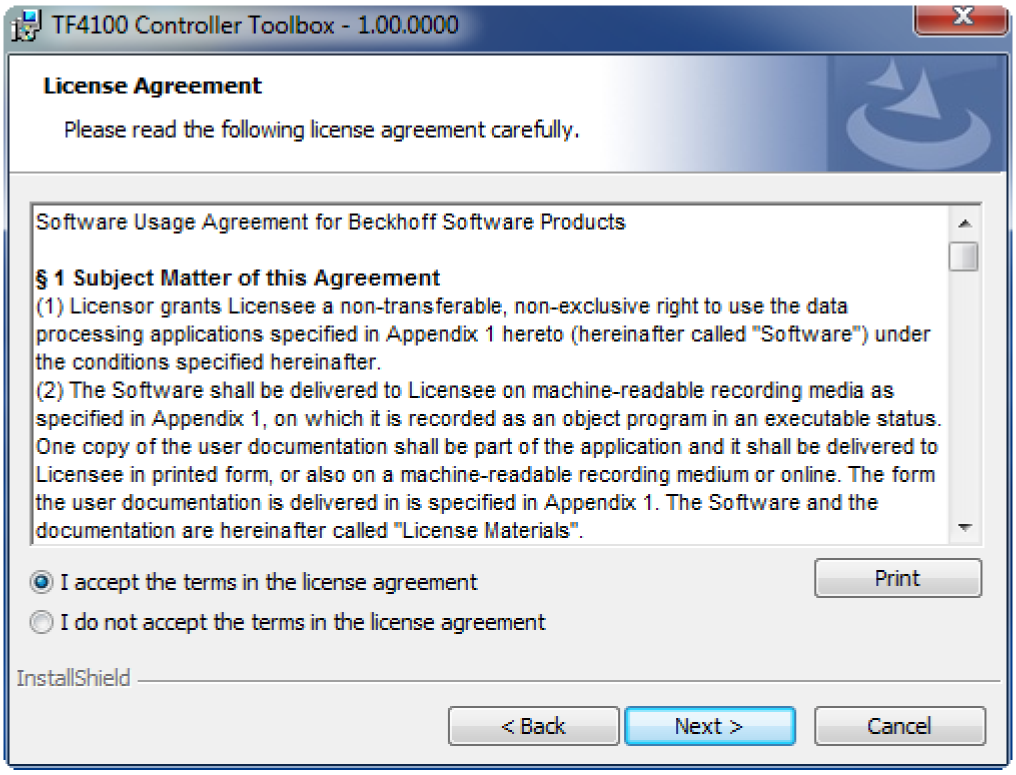
3.2 Installation

The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

✓ The TwinCAT 3 Function setup file was downloaded from the Beckhoff website.

1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.
   ⇨ The installation dialog opens.

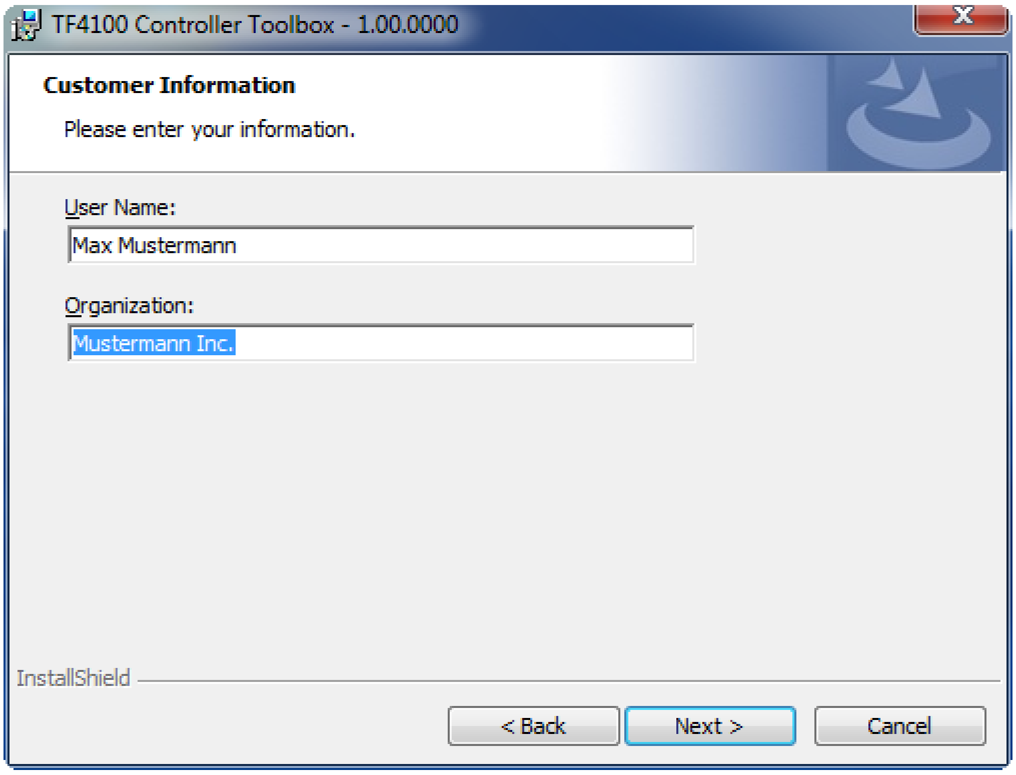
2. Accept the end user licensing agreement and click **Next**.


TF4100 Controller Toolbox - 1.00.0000
License Agreement
Please read the following license agreement carefully.
I accept the terms in the license agreement
I do not accept the terms in the license agreement
Print
InstallShield
< Back
Next >
Cancel

Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.
(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified in Appendix 1. The Software and the documentation are hereinafter called "License Materials".

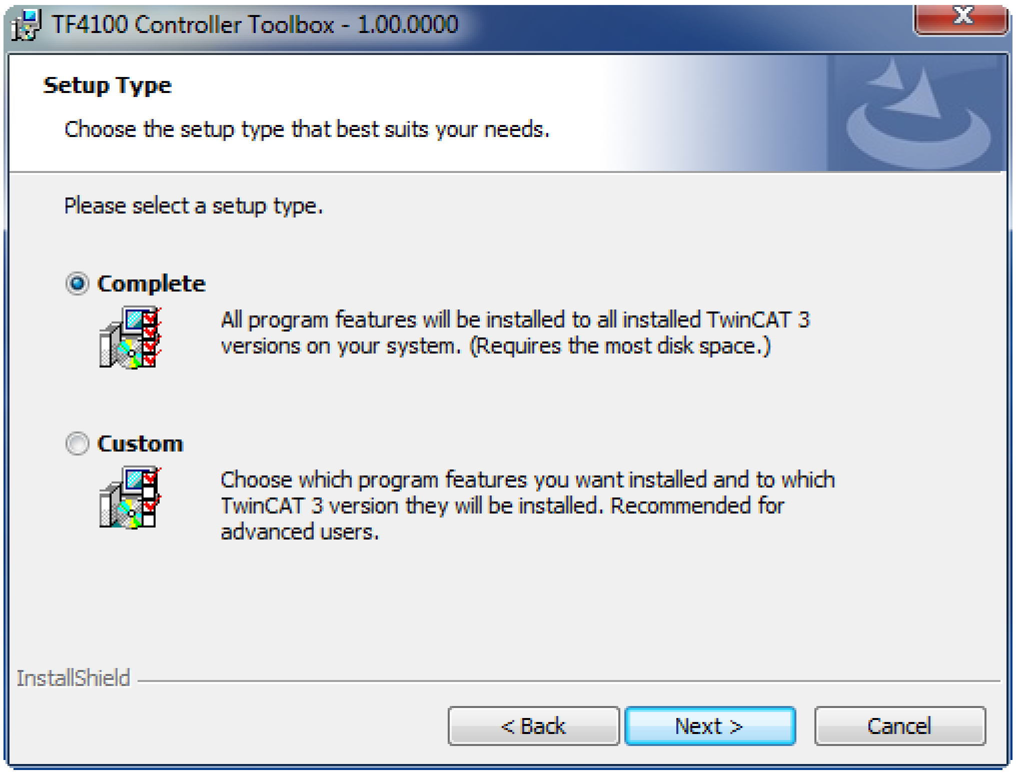
3. Enter your user data.


TF4100 Controller Toolbox - 1.00.0000
Customer Information
Please enter your information.
InstallShield
< Back
Next >
Cancel

User Name: Max Mustermann

Organization: Mustermann Inc.

4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.



TF4100 Controller Toolbox - 1.00.0000

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

[✓ Complete]

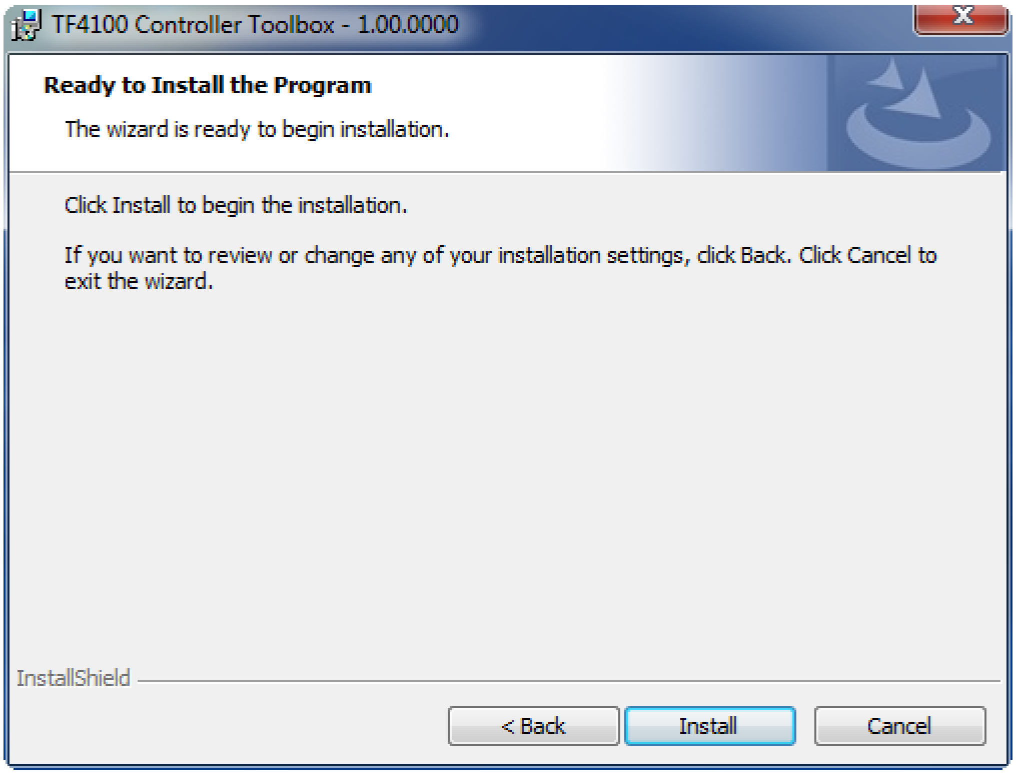
All program features will be installed to all installed TwinCAT 3 versions on your system. (Requires the most disk space.)

Custom

Choose which program features you want installed and to which TwinCAT 3 version they will be installed. Recommended for advanced users.

InstallShield

5. Select **Next**, then **Install** to start the installation.



TF4100 Controller Toolbox - 1.00.0000

Ready to Install the Program

The wizard is ready to begin installation.

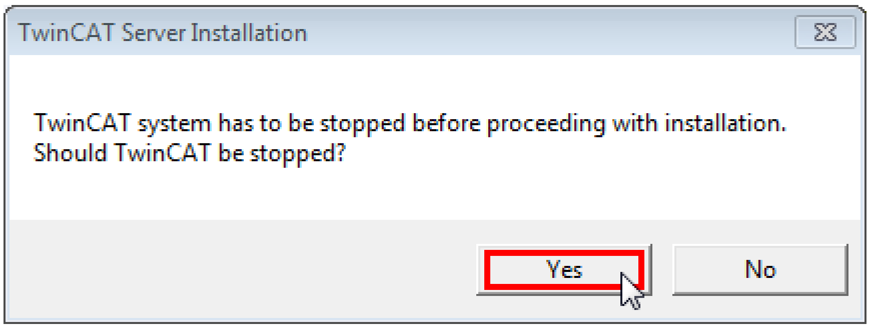
Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

InstallShield

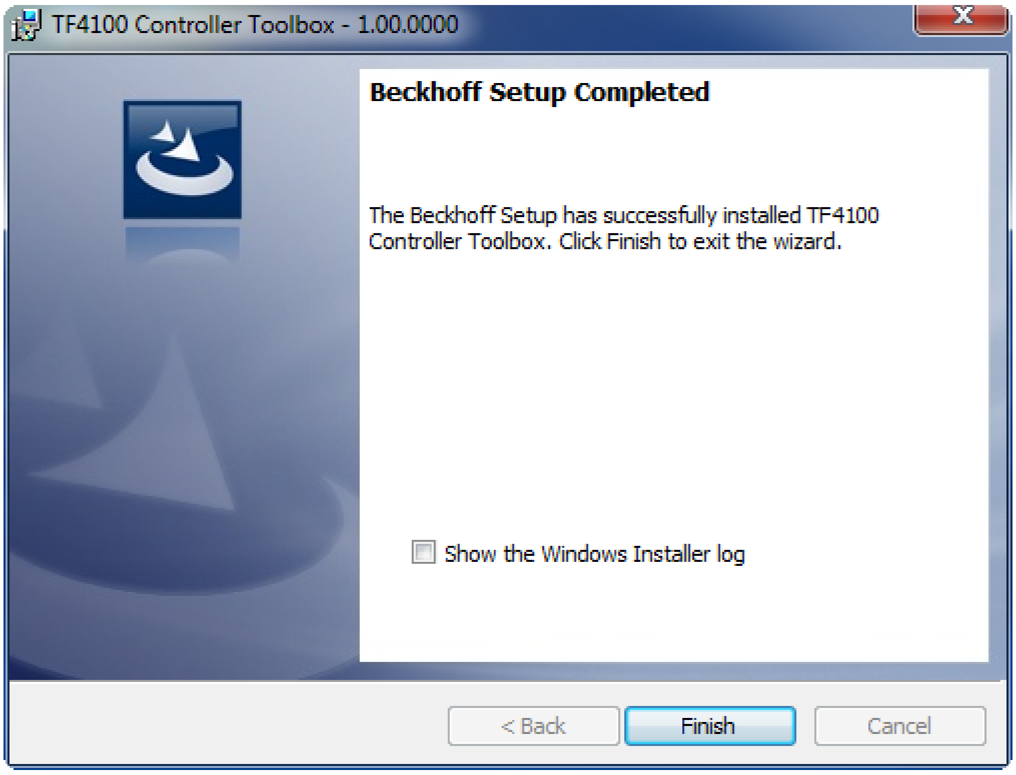
⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

6. Confirm the dialog with **Yes**.


TwinCAT Server Installation

TwinCAT system has to be stopped before proceeding with installation.
Should TwinCAT be stopped?

7. Select **Finish** to exit the setup.


TF4100 Controller Toolbox - 1.00.0000

Beckhoff Setup Completed

The Beckhoff Setup has successfully installed TF4100 Controller Toolbox. Click Finish to exit the wizard.

Show the Windows Installer log

⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▸ 56]).

3.3 Licensing

The TwinCAT 3 Function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

The licensing of a TwinCAT 3 Function is described below. The description is divided into the following sections:

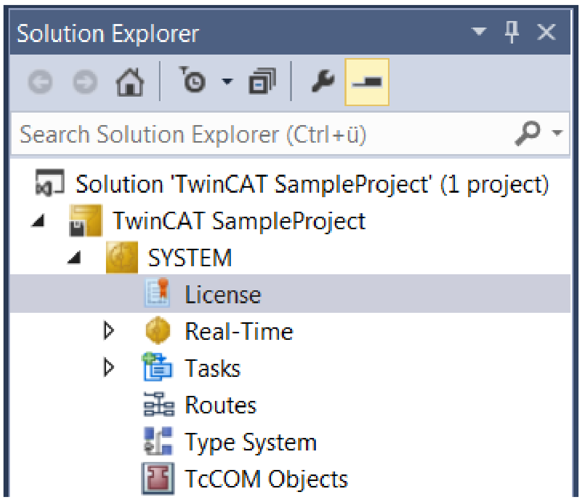
- Licensing a 7-day test version [▸ 56]
- Licensing a full version [▸ 58]

Further information on TwinCAT 3 licensing can be found in the “Licensing” documentation in the Beckhoff Information System (TwinCAT 3 > Licensing).

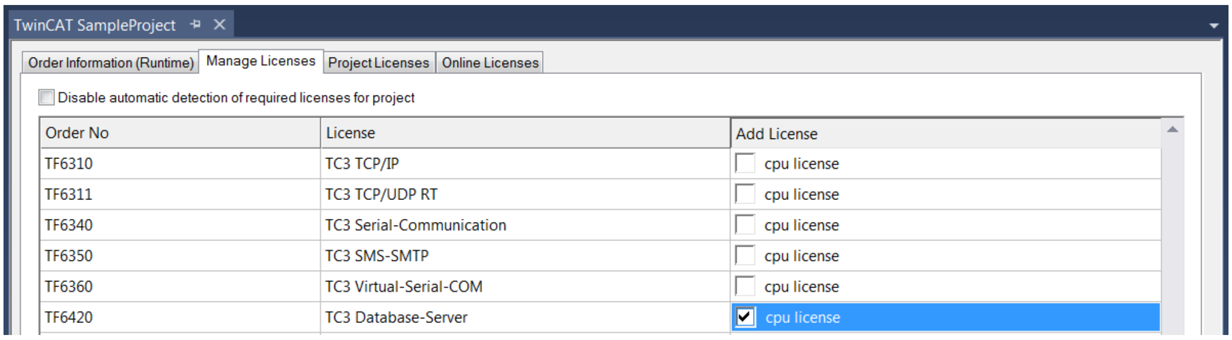
Licensing a 7-day test version

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.

3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.
4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



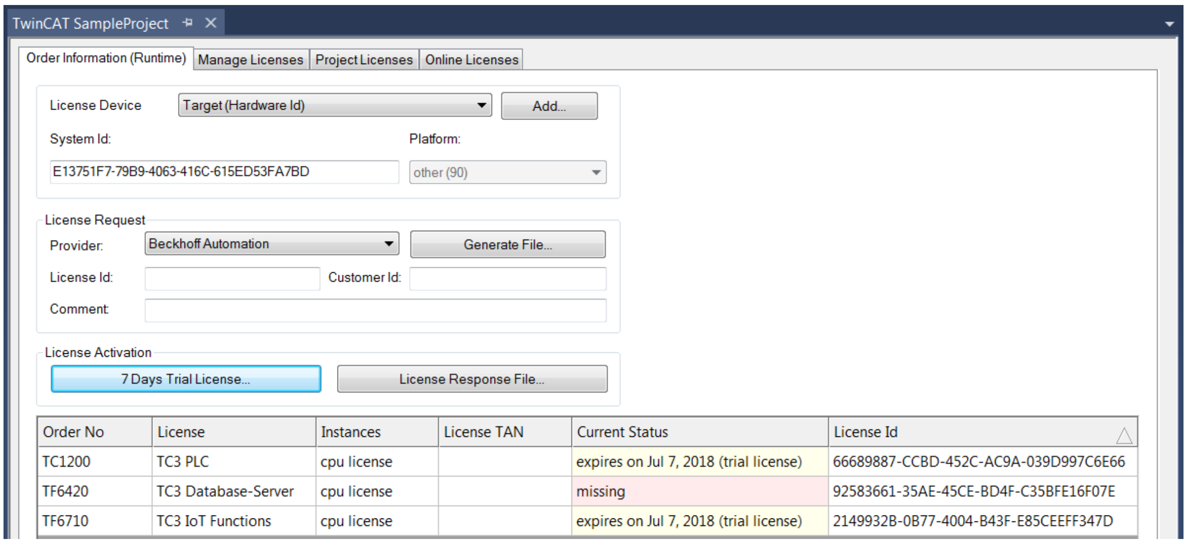
   ⇨ The TwinCAT 3 license manager opens.
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. “TF6420: TC3 Database Server“).



Disable automatic detection of required licenses for project

Order No	License	Add License
TF6310	TC3 TCP/IP	☐ cpu license
TF6311	TC3 TCP/UDP RT	☐ cpu license
TF6340	TC3 Serial-Communication	☐ cpu license
TF6350	TC3 SMS-SMTP	☐ cpu license
TF6360	TC3 Virtual-Serial-COM	☐ cpu license
TF6420	TC3 Database-Server	☑ cpu license

6. Open the **Order Information (Runtime)** tab.
   ⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”.



License Device: Target (Hardware Id)

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

License Request

Provider: Beckhoff Automation

License Id:

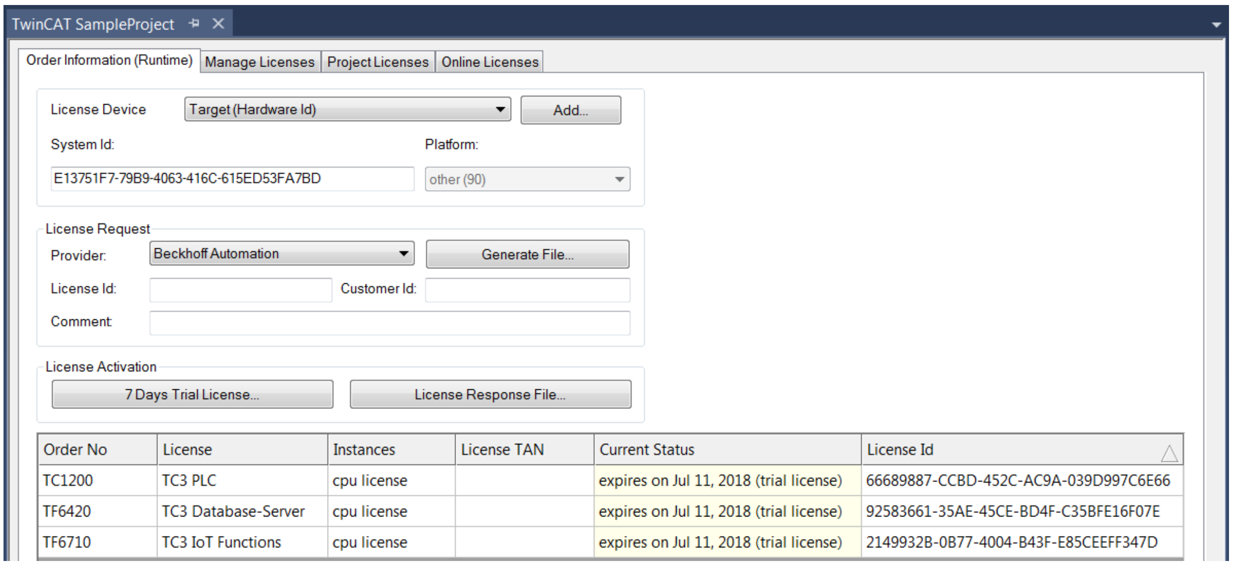
Customer Id:

Comment:

License Activation

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 7, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TF6420	TC3 Database-Server	cpu license		missing	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 7, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

7. Click **7-Day Trial License...** to activate the 7-day trial license.



License Device: Target (Hardware Id)
System Id: E13751F7-79B9-4063-416C-615ED53FA7BD
Platform: other (90)
Provider: Beckhoff Automation
License Id:
Customer Id:
Comment:

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 11, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TF6420	TC3 Database-Server	cpu license		expires on Jul 11, 2018 (trial license)	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 11, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.

8. Enter the code exactly as it appears, confirm it and acknowledge the subsequent dialog indicating successful activation.

   ⇨ In the tabular overview of licenses, the license status now indicates the expiration date of the license.

9. Restart the TwinCAT system.

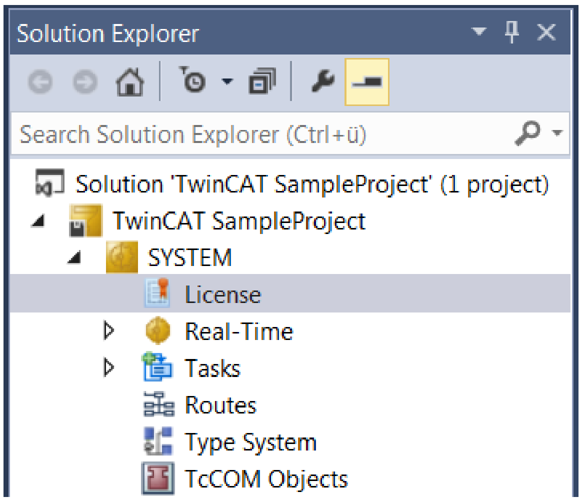
⇨ The 7-day trial version is enabled.

Licensing a full version

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.

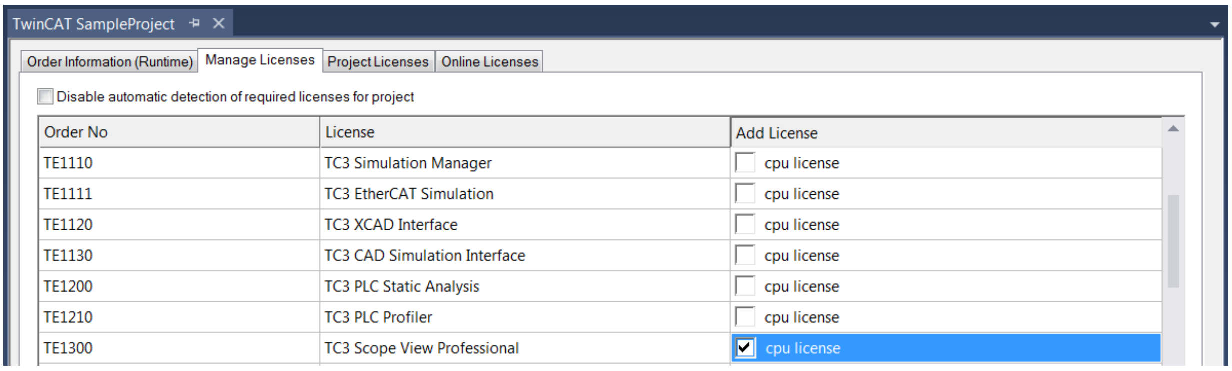
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. “TE1300: TC3 Scope View Professional”).



TwinCAT SampleProject

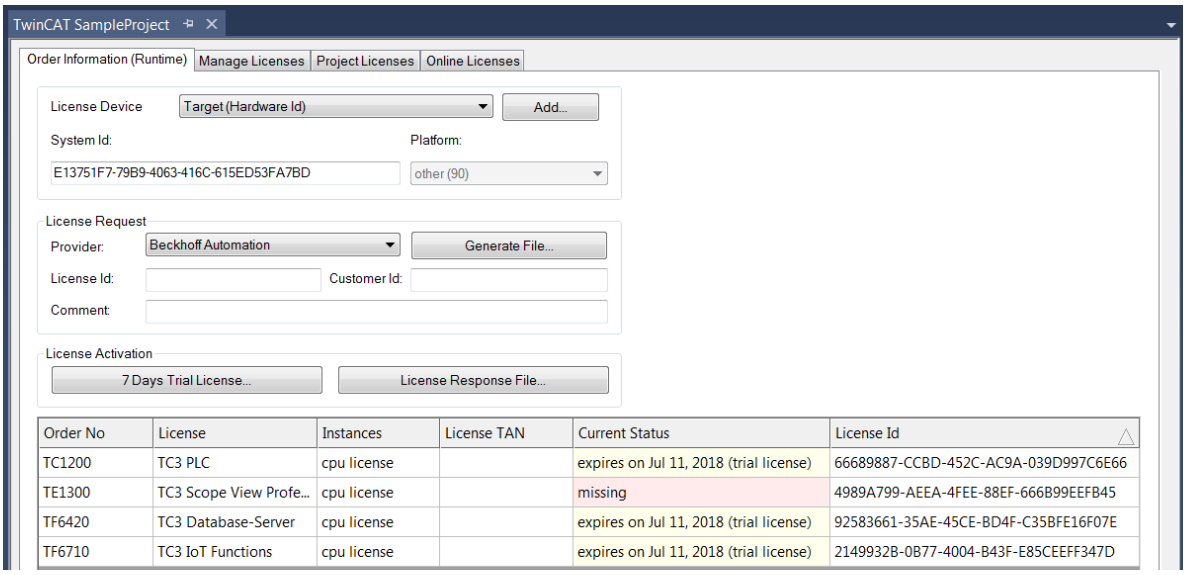
Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses

☐ Disable automatic detection of required licenses for project

Order No	License	Add License
TE1110	TC3 Simulation Manager	☐ cpu license
TE1111	TC3 EtherCAT Simulation	☐ cpu license
TE1120	TC3 XCAD Interface	☐ cpu license
TE1130	TC3 CAD Simulation Interface	☐ cpu license
TE1200	TC3 PLC Static Analysis	☐ cpu license
TE1210	TC3 PLC Profiler	☐ cpu license
TE1300	TC3 Scope View Professional	☑ cpu license

6. Open the **Order Information** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**



TwinCAT SampleProject

Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses

License Device: Target (Hardware Id) — Add...

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

License Request

Provider: Beckhoff Automation — Generate File...

License Id:

Customer Id:

Comment:

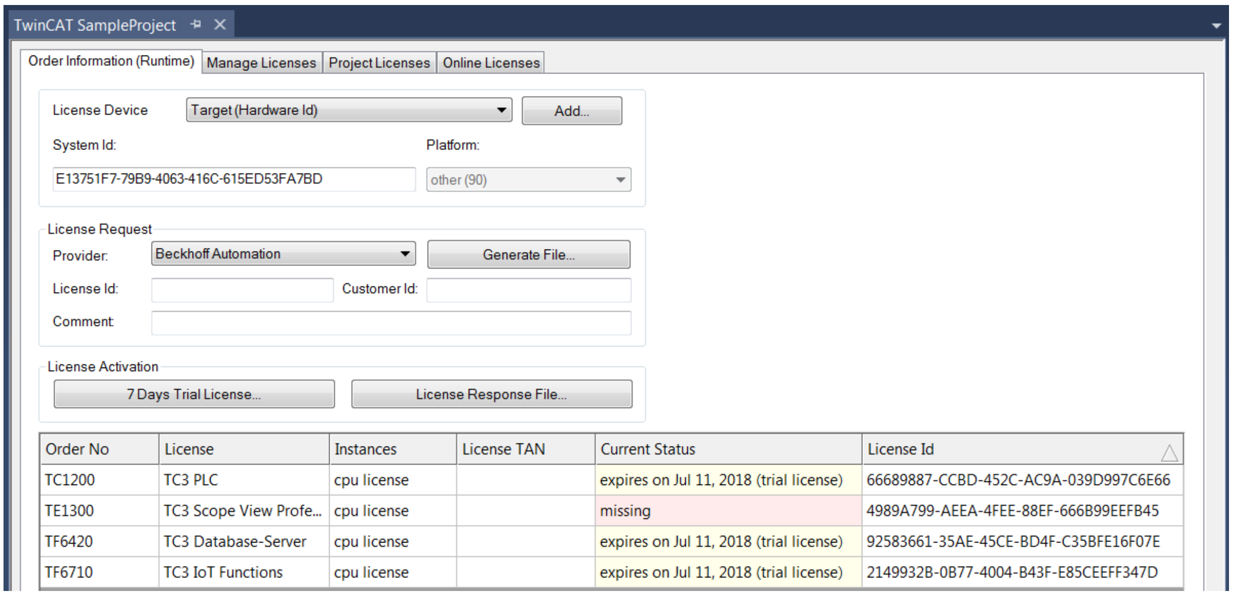
License Activation

7 Days Trial License... — License Response File...

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 11, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TE1300	TC3 Scope View Profe...	cpu license		missing	4989A799-AEEA-4FEE-88EF-666B99EEFB45
TF6420	TC3 Database-Server	cpu license		expires on Jul 11, 2018 (trial license)	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 11, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

A TwinCAT 3 license is generally linked to two indices describing the platform to be licensed:
System ID: Uniquely identifies the device
Platform level: Defines the performance of the device
The corresponding **System Id** and **Platform** fields cannot be changed.

7. Enter the order number (**License Id**) for the license to be activated and optionally a separate order number (**Customer Id**), plus an optional comment for your own purposes (**Comment**). If you do not know your Beckhoff order number, please contact your Beckhoff sales contact.



License Device: Target (Hardware Id)

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

License Request

Provider: Beckhoff Automation

License Id:

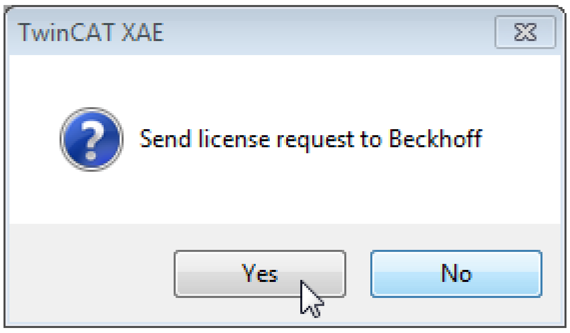
Customer Id:

Comment:

License Activation

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 11, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TE1300	TC3 Scope View Profe...	cpu license		missing	4989A799-AEEA-4FEE-88EF-666B99EEFB45
TF6420	TC3 Database-Server	cpu license		expires on Jul 11, 2018 (trial license)	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 11, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

8. Click the **Generate File**... button to create a License Request File for the listed missing license.
   ⇨ A window opens, in which you can specify where the License Request File is to be stored. (We recommend accepting the default settings.)
9. Select a location and click **Save**.
   ⇨ A prompt appears asking whether you want to send the License Request File to the Beckhoff license server for verification:

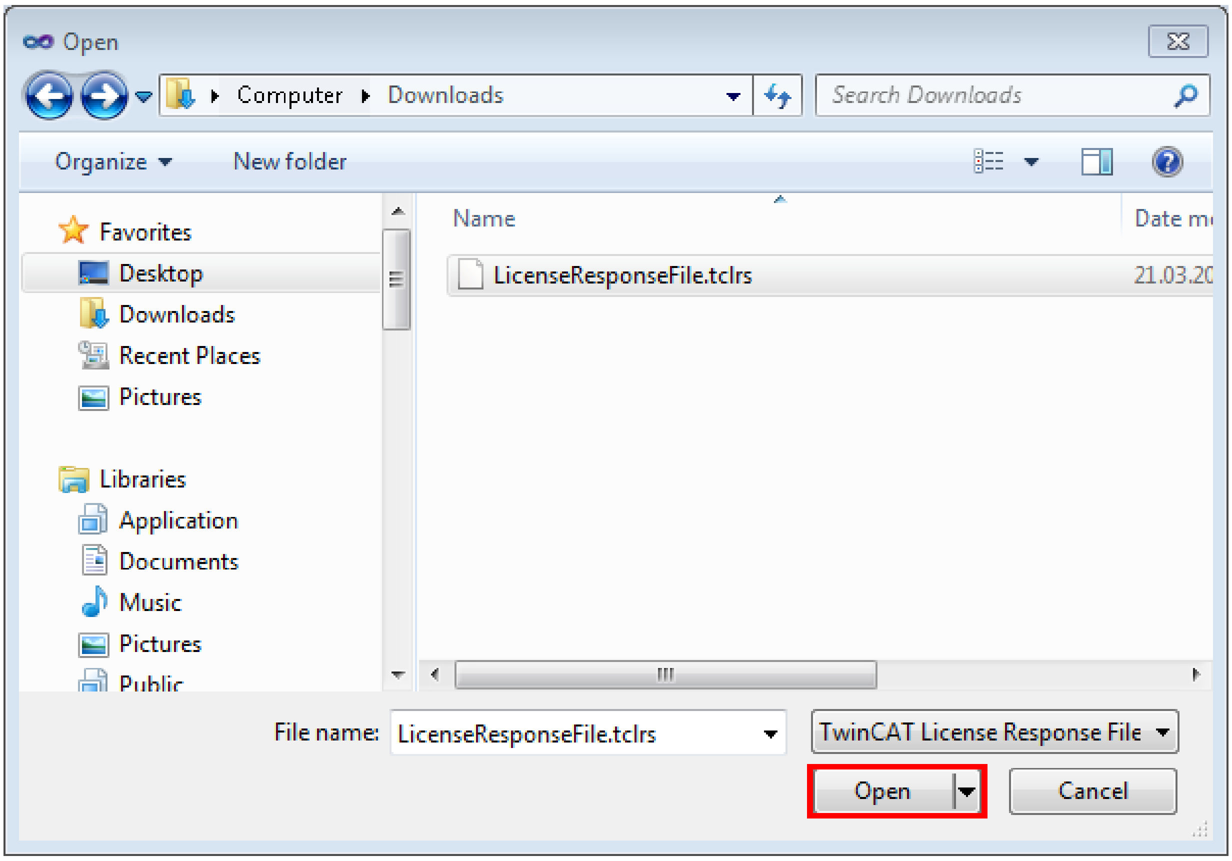


Send license request to Beckhoff

- Click **Yes** to send the License Request File. A prerequisite is that an email program is installed on your computer and that your computer is connected to the internet. When you click **Yes**, the system automatically generates a draft email containing the License Request File with all the necessary information.
- Click **No** if your computer does not have an email program installed on it or is not connected to the internet. Copy the License Request File onto a data storage device (e.g. a USB stick) and send the file from a computer with internet access and an email program to the Beckhoff license server (tclicense@beckhoff.com) by email.

10. Send the License Request File.
   ⇨ The License Request File is sent to the Beckhoff license server. After receiving the email, the server compares your license request with the specified order number and returns a License Response File by email. The Beckhoff license server returns the License Response File to the same email address from which the License Request File was sent. The License Response File differs from the License Request File only by a signature that documents the validity of the license file content. You can view the contents of the License Response File with an editor suitable for XML files (e.g. “XML Notepad”). The contents of the License Response File must not be changed, otherwise the license file becomes invalid.
11. Save the License Response File.

12. To import the license file and activate the license, click **License Response File...** in the **Order Information** tab.
13. Select the License Response File in your file directory and confirm the dialog.



⇨ The License Response File is imported and the license it contains is activated. Existing demo licenses will be removed.

14. Restart the TwinCAT system.

⇨ The license becomes active when TwinCAT is restarted. The product can be used as a full version. During the TwinCAT restart the license file is automatically copied to the directory ...*\TwinCAT\3.1\Target\License* on the respective target system.

4 Technical introduction

4.1 Memory Management

The Condition Monitoring library internally uses TcCOM objects provided by the installed drivers. These are created dynamically using the TwinCAT AMS router memory.

Necessity for dynamic memory management

All memory requests and initializations are accomplished during the initialization phase. Since the number of elements of the input data and the internal structures depend on the configuration of the respective function blocks, the memory space for them is allocated dynamically as a matter of principle. This is done automatically by using the PLC Condition Monitoring Library.

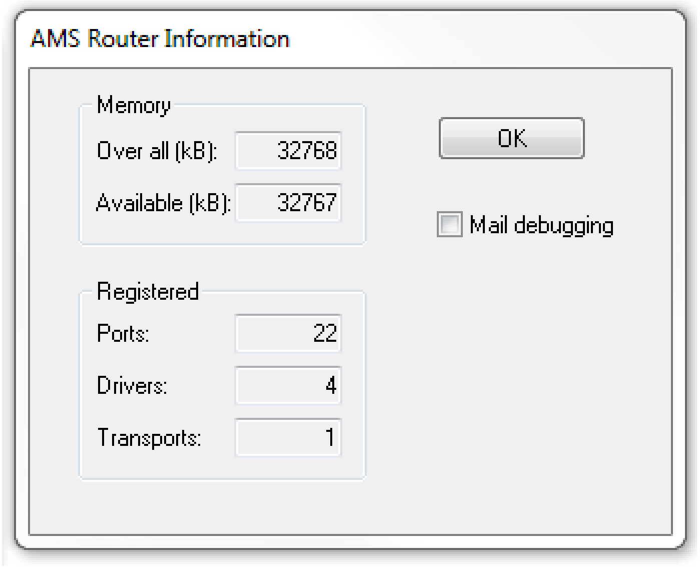
Since all memory assignments take place during the initialization and the initialization of function blocks may therefore take up a relatively large quantity of memory, it can also fail at this point – but not later – due to a lack of memory space.
The allocated memory is released again once the object is deleted.

TwinCAT router memory for dynamically created objects

The buffers reserved by the TwinCAT 3 Condition Monitoring Library are created during the initialization of function blocks in the TwinCAT AMS router memory, so that they are available for execution under real-time conditions. Certain functions, such as high-resolution histograms and quantiles as well as the calculation of spectra with very high resolutions, require considerably more router memory than conventional control programs. Therefore it may be necessary to increase the size of the router memory.

Adapting the router memory

The standard size is 32 MB (2 MB up to TwinCAT 3.1.4016). The current setting can be displayed with the AMS Router Information dialog box.

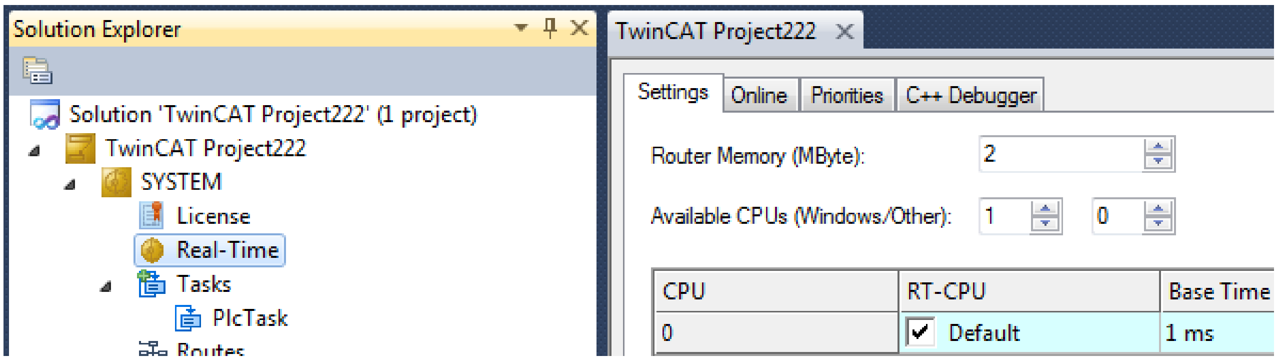

AMS Router Information

Memory
Over all (kB): 32768
Available (kB): 32767

Registered
Ports: 22
Drivers: 4
Transports: 1

To increase the router memory capacity, a value in MB is entered in the TwinCAT configuration under System\ Real-Time\ Settings and the configuration is activated.

Up to TwinCAT 3.1.4022.4, a reboot of the target device was required for adaptation of the router memory.



Router Memory (MByte): 2

Available CPUs (Windows/Other): 1 0

CPU	RT-CPU	Base Time
0	✓ Default	1 ms

4.2 Task Setting

Applications with several real-time tasks

A Condition Monitoring analysis chain is made up of the data collection, usually several algorithms and the provision of the results. The further processing of the results as well as the reactions of the program to these depend on the application.

Since the scope of the input data, e.g. the length of input vectors, strongly depends on the respective application, signal processing software requires arrays with different lengths and different element types. Therefore the TwinCAT 3 Condition Monitoring Library uses a flexible data structure throughout for numerical arrays. This allows numerical data to be saved, transferred and evaluated block by block. It can represent both multi-dimensional and one-dimensional data.

The Condition Monitoring algorithms are very CPU-intensive depending on the configuration. The algorithms are therefore preferentially outsourced to a separate task. In this case the analysis chain extends over several tasks. The associated difficulties of synchronous data exchange and thread security are internally encapsulated by the library function blocks in order to enable flexibly manipulable analysis chains.

Further information on data exchange can be found in section "Parallel processing" [▶ 66].

Tip: Of course, the program can also be implemented as an application of a single task. This is recommended if the required algorithms can be processed fast enough, depending on the CPU and the task cycle time.

Task cycle times

The analysis steps and the corresponding buffer sizes represent a condition for the task cycle time. The calculation must be performed often enough to be able to process all input data.

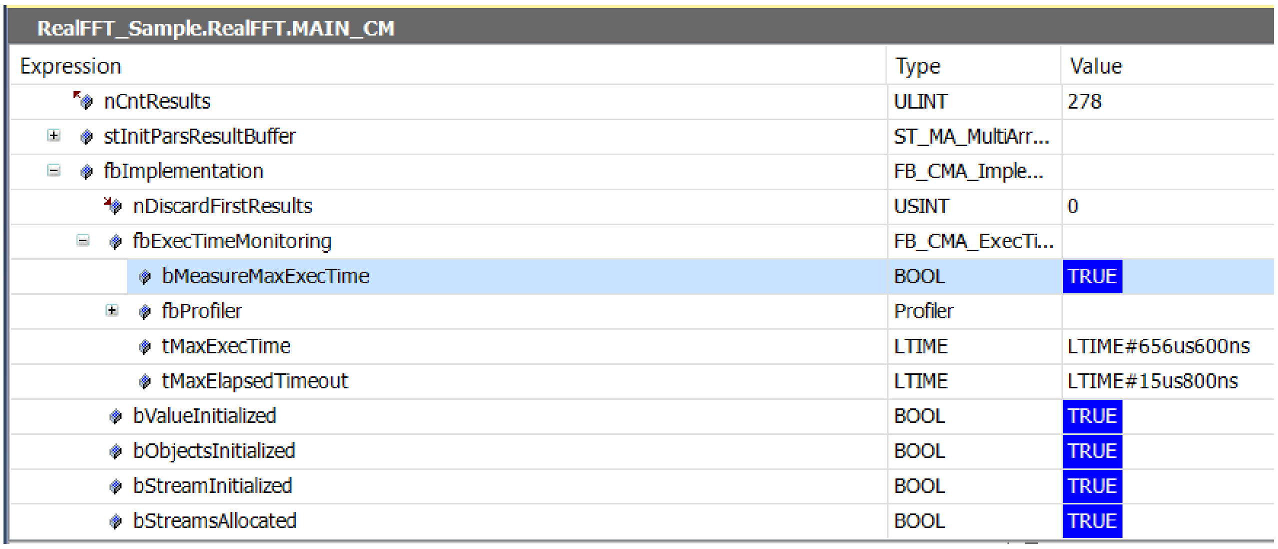
Example: The data collection is stored in buffers, the size of which was declared as 1,600 elements. With an oversampling rate of 10x, a buffer takes 160 cycles to fill. If the signal collection is triggered by a 1 ms task, the task calculation must be triggered with a cycle time of less than 160 ms.

It is recommended to set the calculation cycle time to a lower value, in order to realize a faster response (at least a factor of 0.5). On the other hand, the smallest possible calculation cycle time depends on the complexity of the algorithms to be calculated and the performance of the CPU used.

Determination of the task cycle times

Calculation cycle time < 0.5 * signal collection cycle time * buffer size / oversampling rate

Most algorithms (spectrum, cepstrum,...) contain computationally intensive mathematical operations. They should be called in a task context with sufficient cycle time. The required execution point also depends on the hardware platform. The above equation represents an upper guide value for the calculation cycle time. For example, a profiler is provided for each function block for estimating a lower guide value, which can be activated during online monitoring. You can find this profiler in the instance of the function block under `fbImplementation → fbExecutionTimeMonitoring`. By **manually** setting `bMeasureMeasureMaxExecTime` you activate the profiler. As usual, you do not want to access internal variables of a function block programmatically.



RealFFT_Sample.RealFFT.MAIN_CM

Expression	Type	Value
nCntResults	ULINT	278
stInitParsResultBuffer	ST_MA_MultiArr...	
fbImplementation	FB_CMA_Imple...	
nDiscardFirstResults	USINT	0
fbExecTimeMonitoring	FB_CMA_ExecTi...	
bMeasureMaxExecTime	BOOL	TRUE
fbProfiler	Profiler	
tMaxExecTime	LTIME	LTIME#656us600ns
tMaxElapsedTimeout	LTIME	LTIME#15us800ns
bValueInitialized	BOOL	TRUE
bObjectsInitialized	BOOL	TRUE
bStreamInitialized	BOOL	TRUE
bStreamsAllocated	BOOL	TRUE

The displayed values are maximum execution times. The task settings should provide a small reserve for possible combinations of parameters and input values that could lead to longer execution points.

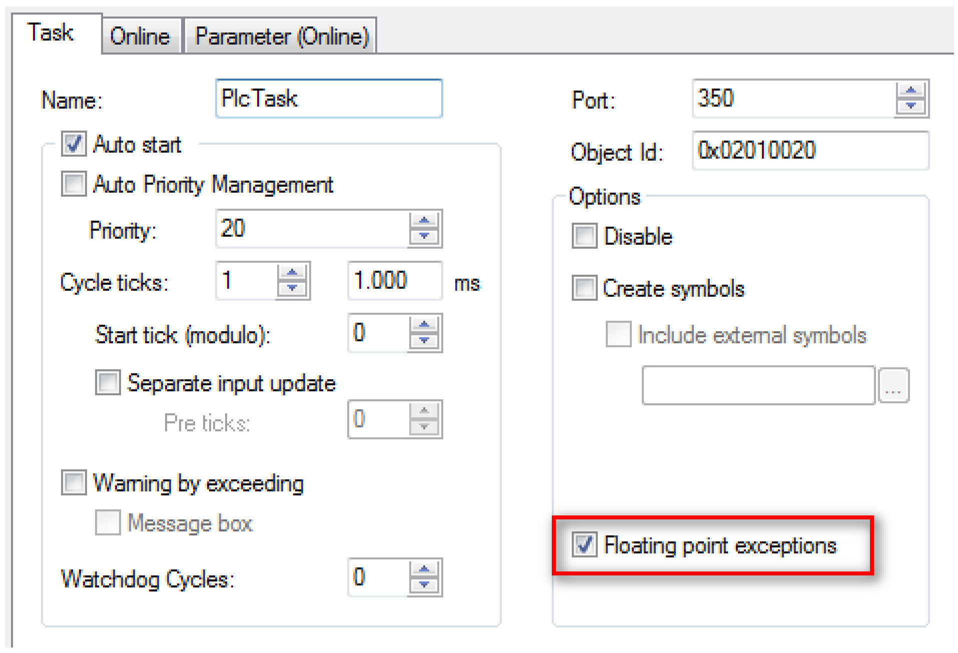
Exceptions to the above considerations are some statistical building blocks (quantiles, histograms,...). As a rule, these function blocks initially only add data for several task cycles to the internal memory. Only the subsequent calculation (collecting data after *N* cycles) takes time. The corresponding task cycle time can be adapted to the simple call without calculation. While this leads to exceeding of the cycle time in the event of calls with calculation, it ensures fast response times. This is a special case for PLC programming. Normally, a task cycle time should never be exceeded.

Note the cycle time

The cycle time of tasks, which only call Condition Monitoring algorithms, can be adjusted in such a way that the cycle time is rarely exceeded. Program blocks, which are called by this task, should not contain other program code! And the priority of these slower tasks should, of course, be lower than that of other tasks.

Floating point exceptions

These exceptions can be disabled separately for each task. They are enabled by default.


Task | Online | Parameter (Online)

Name: PlcTask
[✓] Auto start
[ ] Auto Priority Management
Priority: 20
Cycle ticks: 1 1.000 ms
Start tick (modulo): 0
[ ] Separate input update
Pre ticks: 0
[ ] Warning by exceeding
[ ] Message box
Watchdog Cycles: 0

Port: 350
Object Id: 0x02010020
Options
[ ] Disable
[ ] Create symbols
[ ] Include external symbols
[✓] Floating point exceptions

Some algorithm calls can lead to a NaN (not a number) result. If NaNs are to be processed in the application, the FP exceptions have to be disabled for this task. Then, you must verify that the whole program code and all functions can handle NaNs.
Further information regarding the handling of NaN values can be found in the separate section "NaN values" [▶ 65].

NOTE

Execution stop

Floating point exceptions are active by default. Comparisons with NaN (Not a Number) can cause such an exception that leads to an execution stop and may possibly cause machine damage. It is urgently recommended to check the result for NaN before it is processed. (see section "NaN values")

4.3 NaN values

In some cases error handling by error codes [▶ 224] is not the best choice, in particular if operations return undefined values on account of unusual, but in principle possible input data, or if values are to be excluded from the processing.

The IEC 745 standard defines symbolic values of the category NaN (Not a Number) for these purposes. In the following situations these are generated or taken into account in the TwinCAT 3 Condition Monitoring library:

- Sufficient values are not yet present for a statistical evaluation.
- Certain values are to be excluded from an evaluation in statistical function blocks.
- Interruptions occur in the frequency analysis of a time series, so that gaps in the values have to be accounted for.

The following points rank among the main features of NaN values:

- All arithmetic operations that use NaN as input data return NaN as the result.
- All relational operators =, !=, > < >= <= always return the value False if at least one of the operands is NaN.
- The standard function `isnan()` or `_isnan()` or the PLC function LrealIsNaN() (Tc2_Utilities library) returns the value True if the argument has the value NaN.
- The expression `isnan(a)` is equivalent to the expression `!(a == a)` or `NOT(a = a)`.

The fact that NaN values reproduce themselves when used in further calculations is advantageous in that invalid values cannot be overlooked. The ability of a function block to create NaN values is noted in its description.

NOTE

Malfunctions of software

NaN values may only be used in other PLC libraries, in particular as control values in functions for Motion Control and for drive control, if they are expressly approved! Otherwise NaN values can lead to potentially dangerous malfunctions of the software concerned!

NOTE

Floating point exceptions

If NaNs are to be used and processed in the application, the FP exceptions must be switched off. Otherwise, comparisons with NaN can lead to an exception, which will cause a stop of the runtime and possible machine damage.

Further explanations on the option to switch the FP exceptions off and on can be found in chapter Task settings [▶ 63].

4.4 Parallel processing with Transfer Tray

The following section deals with **thread-safe** and **multi-core capable data transmission**, which is provided by the TwinCAT 3 Condition Monitoring Library.

Asynchronous communication and parallel execution of computationally intensive steps

Condition Monitoring applications often require data sets of several megabytes in size, which increase the demands on computing time and power. The maximum permissible computing time is based on the cycle time, which must never be exceeded for drive controllers, for example. For this reason, multi-task software architectures for TwinCAT 3 Condition Monitoring applications are recommended in the case of computationally intensive algorithms. See Chapter "Task settings [▶ 63]".

Idea of the transfer tray

This requires thread-safe implementations of the algorithms. The TwinCAT 3 Condition Monitoring Library offers a very efficient and easy-to-use communication mechanism that eliminates typical problems with locking and unlocking data as far as possible. The library offers a very efficient mechanism for parallel processing of data, e.g. with different data rates. This allows for error-free transfer of array data between multiple tasks for exclusive synchronized access - using *queues* based on the transfer tray. This also allows the use of multi-core CPUs without synchronization problems and prevents hard to diagnose errors such as blockages and inconsistencies caused by not synchronized overrides of numerical data.

The library function blocks may not be declared as global instances in the list of global variables because parallel write access to *MultiArray buffers* (see section MultiArray Handling [▶ 68]) and parallel execution of the same function blocks are expressly prohibited.

Example of the necessity of cycle time transitions

In some circumstances, a sequential concept is not sufficient. This is always the case when the processing of a data set takes more time than the cycle time of a control task allows.

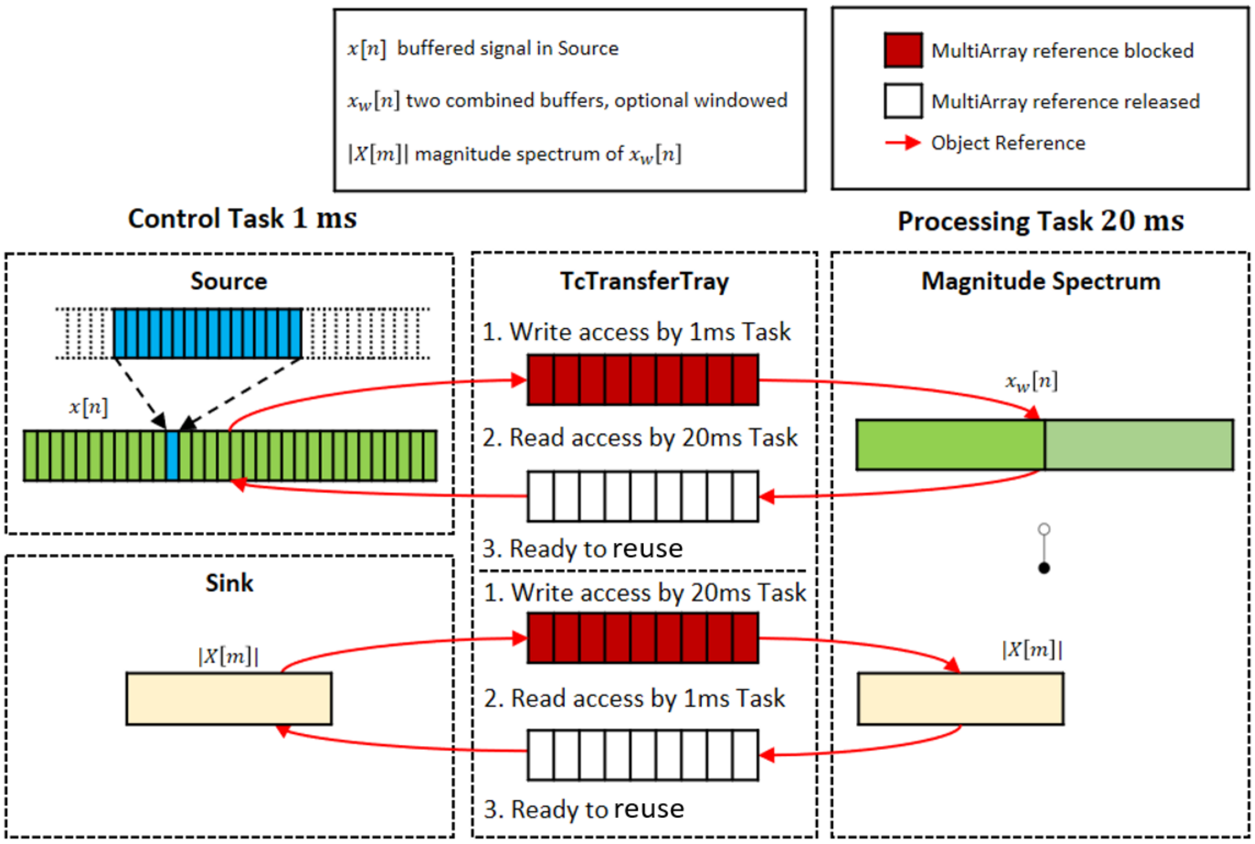
For example, the control task has a cycle time of 1 millisecond and data oversampling of 20 samples per cycle (equivalent to a sampling rate of 20 kHz). For signal processing, a frequency resolution of 0.16 Hz is required, which may be necessary for the analysis of large roller bearings, for example, in order to distinguish between deficiencies in the inner and outer raceway, which run at only slightly different speeds.

The relationship between FFT-length N, frequency resolution Δf and sampling rate f_s is: $N = f_s / \Delta f$ (for simplification, a rectangular window is assumed here). This results in an FFT length of $N = 125000$. In addition, the FFT length N' must be a power of two, resulting in $\log_2 (125000) = 16.93$, which means that the signal of length N is filled with zeros to $N' = 2^{17} = 131072$.

The required computing time depends on the performance of the CPU, but the calculation in the control task is definitely not possible. The required amount of input data corresponds to a signal segment of several seconds, so that the calculation is therefore rarely necessary.

Solution concept with the transfer tray

The high-performance solution provided by the Condition Monitoring Library is shown in the diagram below. The control task collects data in "packets" of 20 samples via the oversampling terminal (shown in blue in the diagram). These are stored in a buffer whose size corresponds to the length of the input buffer of the amplitude spectrum function block (125000 / 2 = 62500, shown in green in the diagram). Once the buffer is full, i.e. after 3125 cycles of the control task, its object reference is transferred to a second task (processing task) with the aid of an asynchronous communication mechanism (FIFO principle), which has a much longer cycle time of 20 milliseconds. According to the rule of thumb described in Task Setting [▶ 63], a maximum cycle time of 1562.5 ms is allowed for the calculating task. This requirement is clearly met with the value of 20 ms.


$x[n]$ buffered signal in Source
$x_w[n]$ two combined buffers, optional windowed
$|X[m]|$ magnitude spectrum of $x_w[n]$
MultiArray reference blocked
MultiArray reference released
Object Reference
Control Task 1 ms
Processing Task 20 ms
Source
$x[n]$
TcTransferTray
1. Write access by 1ms Task
2. Read access by 20ms Task
3. Ready to reuse
1. Write access by 20ms Task
2. Read access by 1ms Task
3. Ready to reuse
Magnitude Spectrum
$x_w[n]$
$|X[m]|$
Sink
$|X[m]|$

This communication mechanism uses hardware-secured, so-called *atomic operations* to guarantee that only one of the tasks has access to the corresponding buffer (hereinafter also referred to as *MultiArray*) at the same time. This is similar to a *transfer tray* at a bank counter, which ensures that either the customer or the cashier (but not both simultaneously) can access its contents.

Response latency

The FIFO principle applies to queues. Therefore, and because of asynchronous communication, the result is not immediately available. Responses with variable latency are possible.

The calculation result (the magnitude spectrum) is returned to the control task via a further queue with the same communication mechanism, which can then further evaluate it. Of course, communication to another, third task and the provision of the result in the computing task itself is also possible.

In general, compared to motion applications the computing task is not subject to hard real-time conditions and can therefore be executed with a lower priority than the control task. The task management of the TwinCAT 3 system ensures that the task with the highest priority is always executed first, so that these real-time conditions can be fulfilled even with complex calculations.

The presented concept can be used on both single-core and multi-core CPUs. Distribution over many cores is possible without the central locks causing bottlenecks.

Timeout

The internal communication commands for the transfer tray may fail in rare cases, e.g. depending on the properties of the hardware. If, for example, there is an empty buffer in the queue that cannot be removed, because another task is currently accessing it. A synchronous timeout is specified and may occur as a result of a timeout error. The program must therefore always be prepared for the possible error state to the effect that a buffer required for the continuity of the signal data is not available. Consequential errors such as data overflow and discontinuities of analyzed time series must be processed in a consistent manner. As long as the input signal data of an analysis chain can be collected without errors, discontinuities do not occur. If a single timeout occurred in a down-stream algorithm function block, or if no result MultiArray buffer was available for the downstream algorithm function block, neither input data nor result data are lost. They are transferred during the next call.

How the transfer tray works

The transfer tray itself is displayed using an internal function block provided by the Tc3_CM library. This function block is initialized with initial parameters that are defined in the global structure instance.

The typical use of queues is that buffers from exactly one task are added to the queue with a fixed data stream identifier, and these buffers are removed from a specific other task for processing. These buffers are then sent back via another queue with a different binding identifier and reused. However, it is also no problem if several tasks have read or write access to the same queues, e.g. when analyzing statistical data.

The MultiArray buffers

So-called MultiArray buffers are used to communicate data via the transfer tray from one task to the next. These are explained in the chapter "Using the MultiArray feature [▶ 68]".

4.5 MultiArray Handling

A MultiArray is a **multidimensional data buffer** that is used in the Condition Monitoring Library in combination with the transfer tray. It enables an application to easily exchange multidimensional data between several PLC tasks. During communication between the tasks, no memory is copied, only references to the data buffers are transferred, making communication extremely efficient. Communication requires only a very low overhead with execution times in the microsecond range.

The MultiArray communication ring

The filling (writing of content) and sending (transfer of access rights) of MultiArrays for input or result data streams have the consequence that "free" MultiArrays are constantly required. For this reason, the evaluated MultiArrays are returned as "empty" data containers to the task that filled them. This creates a continuous cycle of MultiArrays, see the diagram in section Parallel processing with Transfer Tray [▶ 66].

Normally, at least three MultiArrays are required per circuit: The first MultiArray "belongs" to the control task and is about to be filled with new data. The process task accesses the second MultiArray and processes it. A third MultiArray must be kept in reserve, so that it is available if the control task has filled the current MultiArray, but remaining oversampling data has to be written into a next MultiArray in exactly this cycle. Therefore, the minimum number is three.

Number of MultiArrays

For safety, four MultiArrays per circuit are recommended as a worst-case requirement. If more than one algorithm accesses the data of a MultiArray, it is recommended to provide an additional MultiAr-ray for each further accessing algorithm.

The number of MultiArrays provided is set via the input parameters `nResultBuffers` of the function blocks of the Condition Monitoring Library. The default value is 4.

Number of MultiArrays in the communication ring

More than four MultiArrays are only required if the result buffers (= MultiArrays) are to be processed directly by several algorithms. In other words, if more than two analysis modules in the communication ring participate for these results. It is recommended to increase the number of result buffers by one with each additional analysis module. The number of MultiArray buffers used in an asynchronous communication ring can be configured in each analysis function block.

These additional buffers are created and managed internally. They require a certain amount of additional memory in the AMS router.

Basically, the dimension of a MultiArray can be configured separately in terms of length, size and even data type. The parameters together define the shape of the MultiArray for its entire lifecycle.

Note that the internal structure of the MultiArray is automatically managed and does not require any programming. The service life of the MultiArray is the same as that of the application, i.e. from PLC start to PLC stop; the MultiArrays are transferred from one task to another using the so-called transfer tray.

The concept is very flexible. Changing and redistributing the calculation to other tasks and/or CPUs is simple and uncomplicated.

Configuration of MultiArrays

MultiArrays are configured with the ST_MA_MultiArray_InitPars [▸ 184] structure. This is part of the Tc3_MultiArray library, which is installed with the Condition Monitoring Setup.

Example configuration of a MultiArray:

```
cInitSource : ST_MA_MultiArray_InitPars:= ( eTypeCode := eMA_TypeCode_LREAL,
                              nDims := 2,
                              aDimSizes := [cChannels, cBufferLength]);
```

If the MultiArray is used with the `FB_CMA_Source` function block, then a configured MultiArray instance (or several) is required by the source instance `fbSource`. The MultiArray described above has 2 dimensions (`nDims = 2, nDims = 1` is also allowed); the size of the dimensions is described with `aDimSizes`. Accordingly, the described MultiArray is of dimension `cChannels` x `cBufferLength` with data type LREAL for each element.
Example of using MultiArrays with `FB_CMA_Source`:

```
fbSource : FB_CMA_Source := ( stInitPars := cInitSource,
                  nOwnId := eID_Source,
                  aDestIDs := [eID_Rms],
                  nResultBuffers := 4);
```

MultiArrays are flexible in terms of data storage management. For example, in the above case, the rows and columns are completely interchangeable. If the dimensions are correctly assigned/identified (as shown in the example below), this has no effect on the results.

Advanced configuration options

As you can see in the example below, FB_CMA_Source [▸ 158] (or FB_CMA_Sink [▸ 155], FB_CMA_BufferConverting [▸ 81]) provides parameters such as `nWorkDim`, `pStartIndex` or `nElementsDim`. These parameters can be used to:

- Describe/read out a certain segment of the MultiArray
- Write/read/copy from a specific location
- Copy a certain number of elements from a specific point onwards

A combination of these parameters not only guarantees memory optimization, but also guarantees selectivity in multi-channel, multi-task applications. See the example below.

Application scenario

This application scenario is only valid within the TwinCAT Condition Monitoring application area. As mentioned above, the MultiArrays are managed automatically, but they must first be initialized. This is done in the PLC declaration with the help of `ST_MA_MultiArray_InitPars` and is passed to the `FB_CMA_Source` instance.

Each algorithm function block transfers its results using the MultiArrays configured with stInitPars. Their shapes are defined with the initialization parameters (see respective explanations of the function blocks), with the exception of `FB_CMA_Sink`. It is also possible to copy only a part of the MultiArray into a PLC array for further processing or evaluation. This is done with `FB_CMA_BufferConverting`.

The function blocks have methods with which PLC variables can be written or read in MultiArrays. For more information on the methods and their parameters, see the descriptions of the function blocks.

Note:

- The `FB_CMA_Sink` function block does not require any initialization of a MultiArray. The shape of the MultiArrays used by `FB_CMA_Sink` is specified internally.
- Each dimension of a MultiArray, called WorkDim, has an index beginning with 0.
- In the case of two-dimensional MultiArrays, the working dimension 0 is normally linked to the number of channels in the Condition Monitoring Library (see "Example configuration of a MultiArray" in the text above)

Examples for handling MultiArrays

For a better understanding of how to use a MultiArray in a Condition Monitoring application, we consider the following case study.

Three signals from an acceleration sensor with an oversampling factor of 10 are recorded, e.g. with two EL3632s. The input data is collected in a MultiArray with the length 1000 and transferred to a function block. In this case it is the function block for calculating the moment coefficients [▶ 130]. `FB_CMA_MomentCoefficients` calculates different statistical parameters of the input data for each channel, depending on the configuration. Our goal is now to configure the MultiArray at the output of the `FB_CMA_MomentCoefficient` so that only a certain part of the result, for example the mean value and the standard deviation, is output.

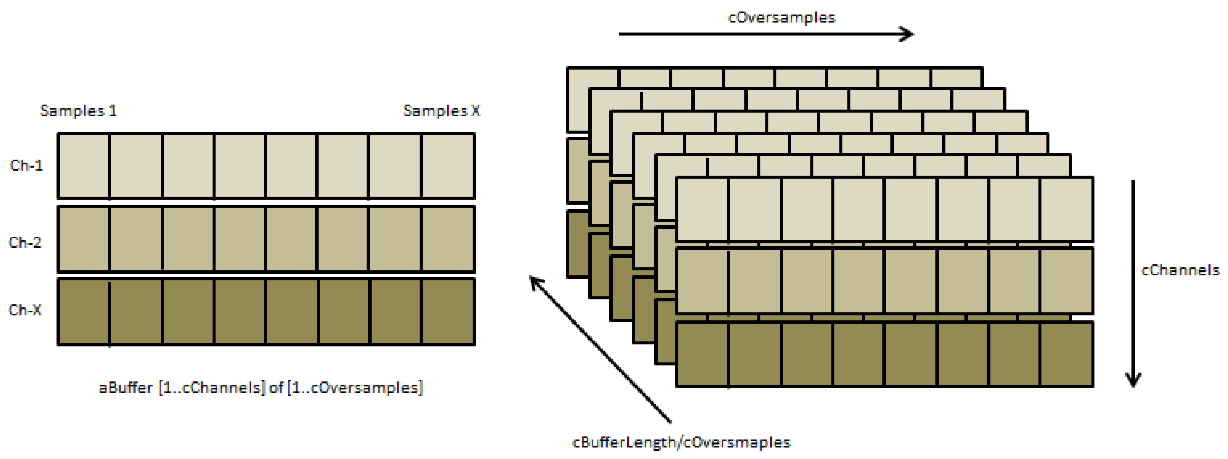
The input and output variables are declared and initialized as follows:

```
cInitSource : ST_CM_MultiArray_InitPars := (eTypeCode := eMA_TypeCode_LREAL,
                          nDims := 2,
                          aDimSizes := [3,1000]);
```

```

aBuffer   : ARRAY [1..3] OF ARRAY [1..cOverSamples] OF LREAL;
fbSource : FB_CMA_Source := (stInitPars := cInitSource,
                 nOwnID := eID_Source,
                 aDestIDs := [eID_MomentCoeffs]);

// MultiArray indices begin with 0, not 1!
// aStartIndex := [0,0],[0,1],[0,2],[1,0],[1,1],[1,2],[2,0],...
aStartIndex : ARRAY [1..2] OF UDINT := [0, 1];

// Select channels := 1: one, 2: one and two, 3: one, two and three and so on
// Select moments := 0: count, 1: mean, 2: standard deviation, 3: skew, 4: kurtosis
aMomentCoef : ARRAY [1..3, 1..2] OF LREAL;
```

As shown above, the `fbSource` gets a MultiArray with 2 dimensions and should pass the data from `aBuffer` to the `FB_CMA_MomentCoefficients` after appropriate buffering. As a function of the initialization parameters, you can either save the data:

- by saving the channels via the rows and the samplings via the columns,
- or by saving the samples via the rows and the channels via the columns.


Samples 1
Samples X
Ch-1
Ch-2
Ch-X
aBuffer [1..cChannels] of [1..cOversamples]
cOversamples
cChannels
cBufferLength/cOversmaples

Because the **MultiArray** is **two-dimensional**, this is done by calling the **Input2D() method**.

```
fbSource.Input2D(pDataIn := ADR(aBuffer),
         nDataInSize := SIZEOF(aBuffer),
         eElementType := eMA_TypeCode_LREAL,
         nWorkDim0 := 0, (* aBuffer stores channels across first dim*)
         nWorkDim1 := 1, (* aBuffer stores samples across second dim*)
         pStartIndex := 0,
         nOptionPars := 0 );
```

Let's go through this method call step by step:

- The local PLC variable `aBuffer` is passed as reference.
- The data type to be transferred is specified.
- The method assigns the first working dimension of the MultiArray to the first dimension of `aBuffer` (`cChannels`) and the second working dimension to the sampled values (`cOversamples`). Alternatively, the variable `aBuffer : ARRAY [1.. cOversamples] OF ARRAY [1.. 3] OF LREAL` could be declared and the necessary transposition could be realized by `nWorkDim0 =1` and `nWorkDim1 =0`.
- `pStartIndex=0` copies the entire `aBuffer` to the MultiArray, which is the default setting. How to copy only parts of an array is shown below using `FB_CMA_Sink` .

All the above settings completely configure the MultiArray to store the channels along its first dimension (rows) and the sampled values along its second dimension (columns) up to the length `cBufferLength`.

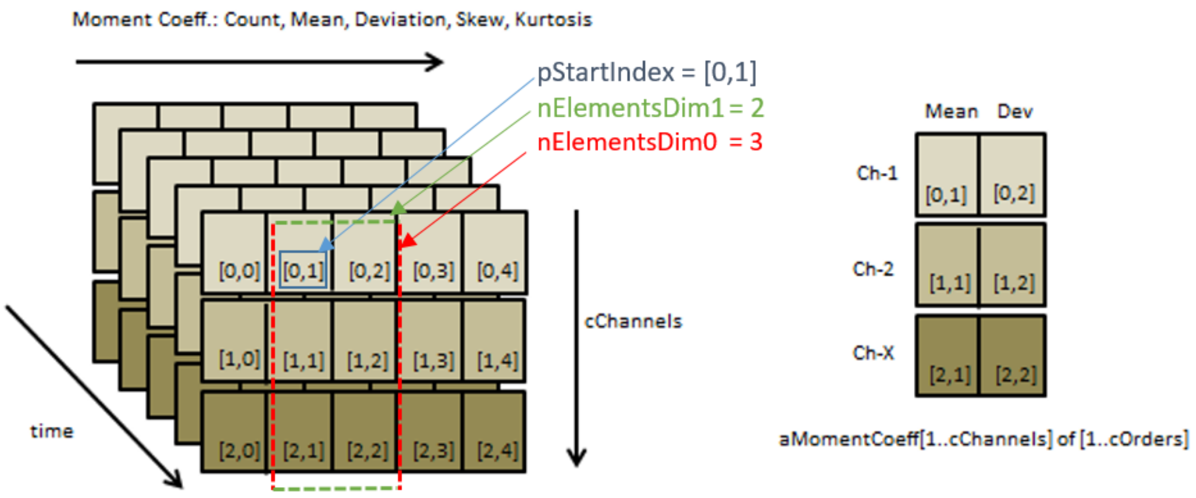
Similarly, a `FB_CMA_Sink` instance can write the contents of the MultiArray to the local PLC variable `aMomentCoef`.

```
fbSink.Output2D(pDataOut := ADR(aMomentCoef),
           nDataOutSize := SIZEOF(aMomentCoef),
           eElementType := E_MA_ElementTypeCode.eMA_TypeCode_LREAL,
           nWorkDim0 := 0,         (* aMomentCoef stores channels across first dim *)
           nWorkDim1 := 1,         (* aMomentCoef stores moments across second dim *)
           nElementsDim0 := 3,     (* aMomentCoef stores all 3 channels *)
           nElementsDim1 := 2,     (* aMomentCoef stores mean and deviation*)
           pStartIndex := ADR(aStartIndex),
           nOptionPars := 0);
```

Again, let's go through this method call step by step:

- The local PLC variable `aMomentCoef` (to which write access is now required) is passed as reference.
- The data type is specified.
- The first working dimension of the MultiArray is assigned to the first working dimension of the variable `aMomentCoef`, i.e. to the channels. The second dimension is transferred analogously and corresponds to the statistical parameters *count, mean, deviation, skew, kurtosis*.
- The parameters `nElementsDim0` and `nElementsDim1` specify how many elements of the MultiArray are to be copied in WorkDim0 direction and WorkDim1 direction. In this case, 3 elements in WorkDim0 direction (all three channels) and 2 elements in WorkDim1 direction.

- The parameter `pStartIndex` defines the first element in the 2x3 rectangle to be copied. The parameter is a pointer to a 2D array (here `aStartIndex`).



In the configuration shown, the Output2D() method will only copy one segment of the MultiArray into the PLC variable `aMomentCoef`. The segment to be copied is configured with the parameters `nWorkDim0`, `nWorkDim1`, `nElementsDim0`, `nElementsDim1` and `pStartIndex` as explained above.

5 PLC API

The TwinCAT3 Condition Monitoring Library provides analysis options in a TwinCAT PLC application. Please refer to our product description [▶ 9] and the technical introductions for an overview and important background information on the product.

The PLC API sets consist of three **PLC libraries**. These libraries have to be integrated in a Condition Monitoring PLC project:

- Tc3_CM
- Tc3_CM_Base
- Tc3_MultiArray

Condition Monitoring analysis

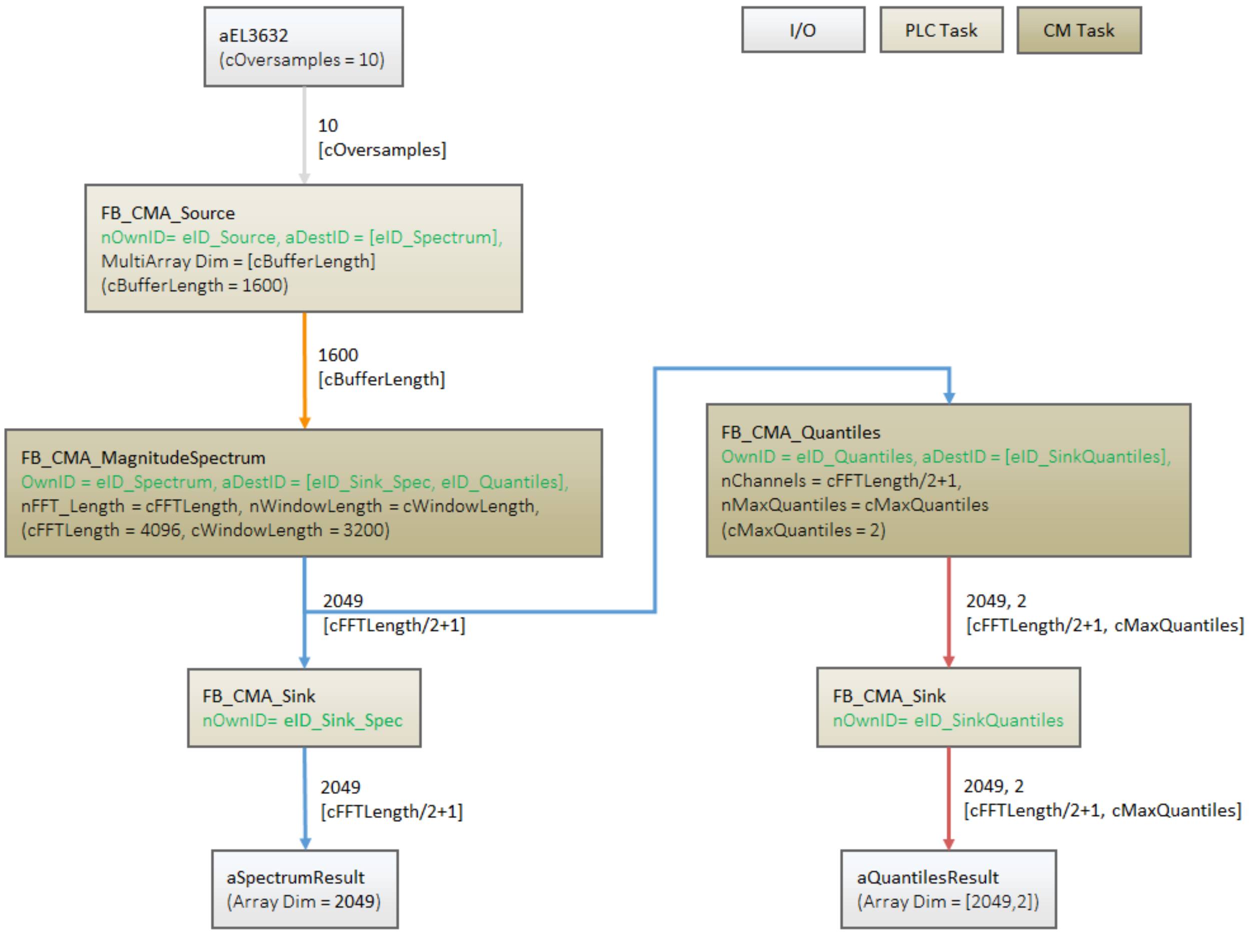
In addition to programming, which includes logging of the measured data, processing based on different algorithms and evaluation of the results, each signal processing relies on an appropriate analysis chain. For that reason the TwinCAT 3 Condition Monitoring Library supports you with function blocks that turn the implementation of the planned analysis chain into virtually pure parameterization work.

Analysis chain as diagram

It makes sense to create a diagram (example see below) regarding the analysis steps before programming the Condition Monitoring application!
It includes a representation of each PLC function block. Usually at least two tasks are used, one task for the regular control program and another (slower and lower priority) task for the computationally intensive operations of Condition Monitoring.

Each analysis function block uses a special way of communicating with each other. This internal implementation also enables cross-communication across multiple tasks. Internally, one TransferTray object and several MultiArrays are used (see chapter Parallel processing [▶ 66]). However, a function block or its methods may only be called from a task context in the application!
The analysis function blocks can be placed in different task contexts. The sequence of the analysis steps is assigned using transfer IDs (green values in the figure below). Each function block receives its own arbitrary ID and the target ID(s) to which the results are to be sent. The transfer IDs are best defined as enumeration.

The diagram below shows four different data buffers: gray, orange, blue and red. The shape of all corresponding buffers (PLC arrays, MultiArrays) and the algorithm parameters must match these buffer sizes.


I/O
PLC Task
CM Task
aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_Spectrum],
MultiArray Dim = [cBufferLength]
(cBufferLength = 1600)
1600
[cBufferLength]
FB_CMA_MagnitudeSpectrum
OwnID = eID_Spectrum, aDestID = [eID_Sink_Spec, eID_Quantiles],
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
(cFFTLength = 4096, cWindowLength = 3200)
FB_CMA_Quantiles
OwnID = eID_Quantiles, aDestID = [eID_SinkQuantiles],
nChannels = cFFTLength/2+1,
nMaxQuantiles = cMaxQuantiles
(cMaxQuantiles = 2)
2049
[cFFTLength/2+1]
2049, 2
[cFFTLength/2+1, cMaxQuantiles]
FB_CMA_Sink
nOwnID= eID_Sink_Spec
FB_CMA_Sink
nOwnID= eID_SinkQuantiles
2049
[cFFTLength/2+1]
2049, 2
[cFFTLength/2+1, cMaxQuantiles]
aSpectrumResult
(Array Dim = 2049)
aQuantilesResult
(Array Dim = [2049,2])

Cyclic call

As long as the functionality of FB_CMA_Source is called and signal data is transferred to a target function block, all other modules of the analysis chain must be called cyclically. See description of the internal communication principle in chapter Parallel processing. If not all target blocks are to be processed during a particular phase, their call is still necessary, but the PassInputs() method can be used to pass only the input buffers without producing results.

Note consequential errors

A cyclically recurring error in an analysis function block can cause consequential errors in the analysis chain.

5.1 Function blocks

In the list below, the available function blocks are sorted based on different criteria, to make them easier to find.

Entire Condition Monitoring library

Signal processing	Statistics	Classification	Buffer handling
FB_CMA_AnalyticSignal [▶ 76]	FB_CMA_HistArray [▶ 115]	FB_CMA_DiscreteClassification [▶ 89]	FB_CMA_BufferConverting [▶ 81]
FB_CMA_ArgSort [▶ 78]	FB_CMA_MomentCoefficients [▶ 130]	FB_CMA_WatchUpperThresholds [▶ 162]	FB_CMA_Sink [▶ 155]
FB_CMA_ComplexFFT [▶ 86]	FB_CMA_Quantiles [▶ 148]		FB_CMA_Source [▶ 158]

FB_CMA_CrestFactor [▶ 83]	FB_CMA_EmpiricalMean [▶ 97]
FB_CMA_Downsampling [▶ 92]	FB_CMA_EmpiricalStandardDeviation [▶ 105]
FB_CMA_Envelope [▶ 109]	FB_CMA_EmpiricalSkew [▶ 101]
FB_CMA_EnvelopeSpectrum [▶ 112]	FB_CMA_EmpiricalExcess [▶ 93]
FB_CMA_InstantaneousFrequency [▶ 119]	
FB_CMA_InstantaneousPhase [▶ 121]	
FB_CMA_IntegratedRMS [▶ 124]	
FB_CMA_MagnitudeSpectrum [▶ 127]	
FB_CMA_MultiBandRMS [▶ 135]	
FB_CMA_PowerCepstrum [▶ 138]	
FB_CMA_PowerSpectrum [▶ 142]	
FB_CMA_RealFFT [▶ 145]	
FB_CMA_RMS [▶ 152]	

Further thematic structuring for signal processing

Algorithms for signal analysis in the:

time range	Frequency range	time/frequency range	Further
FB_CMA_AnalyticSignal [▶ 76]	FB_CMA_ComplexFFT [▶ 86]	FB_CMA_InstantaneousFrequency [▶ 119]	FB_CMA_ArgSort [▶ 78]
FB_CMA_CrestFactor [▶ 83]	FB_CMA_EnvelopeSpectrum [▶ 112]		FB_CMA_Downsampling [▶ 92]
FB_CMA_Envelope [▶ 109]	FB_CMA_IntegratedRMS [▶ 124]		
FB_CMA_InstantaneousPhase [▶ 121]	FB_CMA_MagnitudeSpectrum [▶ 127]		
FB_CMA_RMS [▶ 152]	FB_CMA_MultiBandRMS [▶ 135]		
FB_CMA_PowerCepstrum [▶ 138]	FB_CMA_PowerSpectrum [▶ 142]		
	FB_CMA_RealFFT [▶ 145]		

5.1.1 FB_CMA_AnalyticSignal

Calculation of the analytical signal of a time series.

The analytical signal is the complex-valued complement of the incoming real signal, whereby the imaginary part is phase-shifted by 90 degrees relative to the unchanged real part. The imaginary part is formed via the Hilbert transform of the incoming real signal. In a time-continuous representation, the analytical signal x_{analytic}(t) of the real signal x(t) is described by

$$x_{\text{analytic}}(t) = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

The function block calculates the analytical signal via a discrete Hilbert transformation in the frequency range. The result is a complex-valued vector of length nWindowLength/2.

The input vector is combined with a 50% overlapping preceding input vector based on the Welsch method. It is then multiplied with a window function. Subsequently an FFT for real input values is applied. In the frequency range the Hilbert transform is applied to the signal. It delivers a complex-valued result. The result is then transformed back into the time range via an FFT. The time signal is added up overlapping using the Overlap-Add method. By selecting suitable window functions a continuous output signal without step changes can be achieved.

Memory properties

Since the Overlap-Add method is used, in each case the current input buffer together with the two last transferred buffers is used for the calculation.

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▶ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength`/2
output stream	LCOMPLEX	1	`nWindowLength`/2

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_AnalyticSignal_InitPars;       // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_AnalyticSignal_InitPars [▶ 172]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.

- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode    : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults    : ULINT;       // counts outgoing results (MultiArrays were calculated and sent to 
transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the analytical signal from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent 
to transfer tray.
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_AnalyticSignal_InitPars;       // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_AnalyticSignal_InitPars [▶ 172]. The parameters must correlate to the above definition of the input and output buffers.

- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_Envelope [▶ 109] calculates the envelope of a time series.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base

5.1.2 FB_CMA_ArgSort

Sorts the incoming arguments

The incoming arguments are sorted optionally in ascending or descending order. A one-dimensional array such as the output from a power spectrum is supplied as the input data stream. A two-dimensional array is obtained as the output data: in the first dimension the amplitude and in second the index where this amplitude is to be found in the input array. A scaling factor can be used instead of the index to display the frequency directly. See the corresponding initialization parameters of type ST_CM_ArgSort_InitPars [▶ 173].

The function block calculates internally with “0”-based arrays. This must be taken into account in the evaluation.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nInLength`
output stream	LREAL	2	`nInLength` x 2

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_ArgSort_InitPars;  // init parameter
    nOwnID           : UDINT;                   // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;              // number of MultiArrays which should be initialized f
or results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;  // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars**: Function-block-specific structure with initialization parameters of the type ST_CM_ArgSort_InitPars [▶ 173]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID**: Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs**: Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers**: The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout**: Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;        // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;       // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call():

The method is called in each cycle in order to generate sorted values from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;          // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError      : BOOL;          // TRUE if an error occurs.
    hrErrorCode : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError**: The output is TRUE if an error occurs.

- **hrErrorCode**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init():

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_ArgSort_InitPars;  // init parameter
    nOwnID         : UDINT;                // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;  // number of MultiArrays which should be initialized for results (
0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars**: Function-block-specific structure with initialization parameters of the type ST_CM_ArgSort_InitPars [▸ 173]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID**: Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs**: Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers**: The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base

5.1.3 FB_CMA_BufferConverting

Copies data from one multi-array to another multi-array.

If the defined input buffer of an algorithm function block does not match the output buffer of the preceding function block of the analysis chain, the transfer can be achieved with this functionality. A different number of dimensions can be converted accordingly.

Another option is to use only a subset of the data for further processing, for example in order to take into account only relevant frequency ranges of a spectrum.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
output stream	eTypeCode	`nDims`	`aDimSizes`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_MA_MultiArray_InitPars;       // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_MA_MultiArray_InitPars [▸ 184]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError         : BOOL;        // TRUE if an error occurs. Reset by next method call.
    hrErrorCode    : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults    : ULINT;       // counts outgoing results (MultiArrays were calculated and sent to
 transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Copy1D() :

Copies one-dimensional data from one multi-array to another multi-array.

```
METHOD Copy1D :   HRESULT
VAR_INPUT
    nWorkDimIn      : UDINT;                // It designates the dimension in the input MultiArray be
ing processed.
    nWorkDimOut     : UDINT;                // It designates the dimension in the output MultiArray b
eing processed.
    nElements       : UDINT;                // optional: default:0-
>full copy; It designates the number of elements to be copied out of the MultiArray.
    pStartIndexIn   : POINTER TO UDINT;    (* optional: default:0-
>internally handled as [0,0,..]; It designates the index of the first element to be copied out of th
e MultiArray.
                         If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray. *)
    pStartIndexOut : POINTER TO UDINT;    (* optional: default:0-
>internally handled as [0,0,..]; It designates the index of the first MultiArray element to be copie
d.
                         If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray. *)
    nOptionPars     : DWORD;               // option mask
END_VAR
VAR_OUTPUT
    bNewResult      : BOOL;                // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError          : BOOL;                // TRUE if an error occurs.
    hrErrorCode    : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nWorkDimIn :** If the input MultiArray is multi-dimensional, you can select the dimension whose data you want to copy. The first dimension would be 0 (0-based).
- **nWorkDimOut :** If the output MultiArray is multi-dimensional, you can select the dimension to which you want to copy data. The first dimension would be 0 (0-based).
- **nElements :** To copy the complete data of a MultiArray dimension, this parameter can be set to 0. The total number is determined internally in this case. Alternatively, you can specify the number of elements to be copied.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_MA_MultiArray_InitPars;        // init parameter
    nOwnID          : UDINT;                            // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                        // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 184]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.

- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.4 FB_CMA_CrestFactor

Calculates the crest factor for each channel for multi-channel time series.

This is defined as the ratio between the peak value of a signal and the RMS value.

$$C_{\mathrm{dB}} = 20\log_{10}\left(\frac{\max\{|x(t)|\}}{\mathrm{rms}\{x(t)\}}\right)$$

The crest factor is calculated in the logarithmic unit decibel. A sine wave, for example, has a crest factor of 3.01 dB (=1.414).

The crest factor allows conclusions regarding the condition of roller bearings. In general the crest factor increases at the start of damage to a roller bearing and can decrease again as the damage progresses.

The function block can process several independent input signal channels. The result is a one-dimensional array, with the index corresponding to the channel number. To avoid value range errors, each value is compared with a minimum value before the logarithm is applied. If the value is smaller, it is replaced with the minimum value.

Since the crest factor is defined by the ratio between peak value and RMS value, this means that the result is strongly influenced by the individual maxima, which can lead to unexpected results.

Memory properties

The function block stores a number of time values corresponding to `nBufferLength` (ST_CM_CrestFactor_InitPars [▶ 173]). In a call with smaller input buffer size, fewer values can be transferred. In this case the buffer content is shifted, and the signal length is filled with the corresponding number of newly transferred values.

Sample implementation

A sample implementation is available under the following link: Crest factor [▸ 215].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	1	`nChannels`

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_CrestFactor_InitPars;      // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_CrestFactor_InitPars [▸ 173]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError       : BOOL;        // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;      // counts outgoing results (MultiArrays were calculated and sent to
transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the crest factor from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError       : BOOL;        // TRUE if an error occurs.
    hrErrorCode  : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_CrestFactor_InitPars;      // init parameter
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_CrestFactor_InitPars [▶ 173]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The FB_CMA_HistArray [▸ 115] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▸ 148] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

The FB_CMA_MomentCoefficients [▸ 130] block provides the kurtosis as an alternative measure for the peakiness of a signal that is less sensitive to outliers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.5 FB_CMA_ComplexFFT

Calculation of the fast Fourier transformation (FFT) for complex-valued input signals.

The FB_CMA_ComplexFFT function block calculates the Fourier transform of the complex-valued input signal *x*[n] at the function block. A high-performance FFT algorithm is used for this purpose. Both the original transformation and the inverse transformation can be calculated. The input `stInitPars` is used for the setting (see inputs and outputs).

Definition of the Fourier forward transformation in DFT notation:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-i2\pi nk/N}$$

Definition of the Fourier inverse transformation in DFT notation:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, e^{i2\pi nk/N}$$

The highest frequency of an input signal component should not exceed half the sampling rate of the input signal, in order to avoid aliasing effects.

The FFT is defined as transform of a cyclically continuous signal. This can result in step changes, if the cyclically continuous signal is not exactly continuous, i.e. not the same at the start and finish. The function blocks FB_CMA_PowerSpectrum [▸ 142] and FB_CMA_MagnitudeSpectrum [▸ 127] can be used to avoid these issues by using overlapping sections, which are multiplied with a window function, as the basis for the analysis.

Scaling

For a quantitative evaluation of the spectrum the calculated spectrum should be weighted with `1/nFFT_Length` for the off-set, i.e. the first array element of the outputs, and with `2/nFFT_Length` for all other outputs elements.

During the forward transformation and the inverse transformation the function block scales such that during consecutively transformations and inverse transformations the original input signal is calculated again directly at the output.

Memory properties

The function block result is only determined by the current input values, i.e. no past values are taken into account.

NaN occurrence

If one or several elements at the input are NaN (not a number), the total output signal for the real and the imaginary part is NaN.

Sample implementation

A sample implementation is available under the following link: FFT with complex-value input signal [▸ 190].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LCOMPLEX	1	`nFFT_Length`
output stream	LCOMPLEX	1	`nFFT_Length`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_ComplexFFT_InitPars;       // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of type ST_CM_ComplexFFT_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;          // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;         // counts outgoing results (MultiArrays were calculated and sent t
o transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the FFT from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent to
 transfer tray.
    bError      : BOOL;       // TRUE if an error occurs.
    hrErrorCode : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_ComplexFFT_InitPars;       // init parameter
    nOwnID    : UDINT;                                // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of type ST_CM_ComplexFFT_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_RealFFT [▶ 145] calculates the Fourier transformation of a real-valued signal.

The function block FB_CMA_PowerSpectrum [▶ 142] calculates the power spectrum of a continuous time signal.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.6 FB_CMA_DiscreteClassification

Classification of multi-channel data based on configurable threshold values

The function block assigns the individual channels of a multi-channel signal to a fixed number of discrete categories based on configurable threshold values. The number of channels and the number of categories are specified during instantiation. The function block can be configured at runtime by specifying the threshold value for each channel and each threshold value category.

During the operation phase an input vector is adopted for each time step, and the number of applicable category is calculated for each channel. The return value is a one-dimensional array, which for each input channel contains a signed integer value, i.e. the index of the allocated category.

If the input value is less than the threshold value for the first category, the value -1 is returned for this channel. If an input value is greater than or equal the threshold value for a category, the zero-based index for this category is returned. If several threshold values are configured in the same way, the value with the largest index is used.

Configuration

The function block must be configured based on parameters such as the number of classification classes. The classification threshold values for each channel can be assigned individually. These threshold values must be monotonically increasing (but not strictly monotonically). Accordingly, no threshold value must be smaller than the previous value.

Memory properties

The function block only takes into account the values stored during configuration and training. The values transferred during classification have no influence on later calls.

NaN occurrence

If the input value is NaN, the result is -2. No NaN values are expected at the output.

Sample implementation

A sample implementation is available under the following link: Threshold value consideration for averaged magnitude spectra [▶ 214] and Condition Monitoring with frequency analysis [▶ 209].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	DINT (32bit)	1	`nChannels`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_DiscreteClassification_InitPars;  // init parameter
    nOwnID           : UDINT;                                  // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;        // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                             // number of MultiArrays which should
 be initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                   // timeout checking off during access
 to inter-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_DiscreteClassification_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;       // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;      // counts outgoing results (MultiArrays were calculated and sent to
 transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the classification result from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
 to transfer tray.
    bError       : BOOL;         // TRUE if an error occurs.
    hrErrorCode  : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Configure() :

The classification arguments must be configured at the beginning with the call of this method. The corresponding PLC array must be defined as follows. The Configure() method can also be used for a new configuration with a different set of arguments.

	Element type	Dimensions	Dimensional variables
Argument	LREAL	2	`nChannels` x `nMaxClasses`

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg      : POINTER TO LREAL; // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize  : UDINT;            // size of arguments buffer in bytes
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_DiscreteClassification_InitPars;   // init parameter
    nOwnID        : UDINT;                                   // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;        // IDs of destinations for output
    nResultBuffers : UDINT := 4;                             // number of MultiArrays which should b
e initialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_DiscreteClassification_InitPars [▶ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.7 FB_CMA_Downsampling

Downsampling of signal data through copying of the signal data from one PLC buffer to another PLC buffer (array).

A signal that is present as a buffer (e.g. an oversampling array), can be scanned with a rate that is reduced by an individual factor. Downsampling is a way of analyzing lower frequencies without having to increase the FFT length to maintain a high resolution.

Usually, a downsampling block is inserted in the Condition Monitoring analysis chain before an FB_CMA_Source [▶ 158].

Inputs and outputs

Input parameters

```
VAR_INPUT
    nDivider    : UDINT := 1;  // given input signal is sampled down by the specified divider. (1 = n
o downsampling)
    nChannels   : UDINT;       // number of channels in data buffer (=1:onedimensional dataset, >1:two
dimensional dataset )
END_VAR
```

- **nDivider** : Specifies the downsampling factor as an integer divisor. For example, a sample rate of 10 kHz can be converted to a sample rate of 2 kHz with nDivider=5.
- **nChannels** : For downsampling a multi-channel data set, the number of channels is specified at input nChannels.

Output parameters

```
VAR_OUTPUT
    bError       : BOOL;       // TRUE if an error occurs. Reset by next method call.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults  : ULINT;      // counts outgoing results
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

Writes data from the input data buffer into the output data buffer. The output data buffer is full when bNewResult is set.

```
METHOD Call : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;  // address of data buffer (e.g. oversampling data)
    nDataInSize   : UDINT;            // size of data buffer in bytes
    pDataOut      : POINTER TO BYTE;  // address of result buffer
    nDataOutSize  : UDINT;            // size of data buffer in bytes
    nElementSize  : UDINT;            // Size of element type in bytes
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;             // TRUE every time when outgoing data buffer was calculated.
    bError        : BOOL;             // TRUE if an error occurs.
    hrErrorCode   : HRESULT;          // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nDataInSize** : Specifies the size of the PLC input buffer and must meet the following condition: nChannels * nElementSize * number of elements per channel
- **pDataOut** : The assigned output buffer must remain unchanged until the output bNewResult is set. Usually, input and output buffers are always maintained.
- **nDataOutSize** : Specifies the size of the PLC output buffer and must meet the following condition: nDataOutSize = n * nDataInSize
  If the quotient is divisible by the parameter nDivider without remainder, the following condition can be used as an alternative: nDataOutSize = n * (nDataInSize/nDivider)
- **nElementSize** : Specifies the size of an element in bytes. For an array of LREAL elements the size is 8, for example.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.

Notice: If an errors occur, no copy action was performed.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM

5.1.8 FB_CMA_EmpiricalExcess

Calculates the excess for single- and multi-channel real-valued time series.

The function block treats the input signal as a time series, if necessary with several independent channels. For each channel, the empirical excess is calculated according to equation,

$$E = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^4 - 3$$

where *s* is the empirical standard deviation. The excess is the value of the empirical kurtosis reduced by the value 3, where 3 corresponds to the kurtosis of a normal distribution. This results in the interpretation of the excess:

excess > 0: Distribution more acute than normal distribution; indicates frequent peaks in the sample

excess < 0: Flattened distribution compared to normal distribution

The excess offers, for example, the possibility of assessing shocks in the vibration signal as they occur when rolling over local damage in the roller bearing.

A single sample per channel (see Inputs and Outputs, first table) can be added in each cycle, and several samples per channel can be added to the sample quantity in one cycle (see Inputs and Outputs, second table).

Memory properties

The sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data record, i.e. the function block uses all input values since its instantiation. Resetting of the sample quantity to zero (deleting the internal memory of the FB) is provided by a ResetData() method or, if the CallEx() method is used, by the variable `bResetData`.

Further comments

Four values must be available for calculating an initial result. Furthermore, the standard deviation must not be zero. Results may become inaccurate if the input values are many orders of magnitude apart.

NaN occurrence

If the number of input values is insufficient for calculating a result for a particular channel or the variance is zero, the value NaN (not a number) according to IEC 754 is returned for this channel. The presence of this error value can be checked with the function LrealIsNaN().
The reason may be that so far not enough input data were transferred or that only NaNs were transferred as input values for individual channels.
A variance of zero may occur if the time series of the values is constant, for example if no sensor data were transferred due to a broken wire or switching errors.

If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Error values

If the situations described above, which lead to NaN values, cannot be ruled out or safely neglected, the application program must be able to handle these error values.

Sample implementation

A sample implementation is available under the following link: Statistical methods [▸ 206]

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	LREAL	1	`nChannels`

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	1	`nChannels`

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars; // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;            // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.

- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;          // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;         // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle to calculate and output the current value of the excess from the input signal and the current internal memory of the FB. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;          // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError      : BOOL;          // TRUE if an error occurs.
    hrErrorCode : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle to update the internal memory from the input signal. A result is output only every `nAppendData` cycles. An alternative method is Call().

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;       // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;        // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.
- **bResetData** : If set, the internal data buffer is completely deleted after calculation.

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

ResetData() :

This method deletes all the data sets already added. Alternatively, the automatic reset can be used via the variable `bResetData` in the method CallEx().

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID        : UDINT;                             // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT;   // IDs of destinations for output
    nResultBuffers: UDINT := 4;                        // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_EmpiricalMean [▶ 97] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalStandardDeviation [▶ 105] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalSkew [▶ 101] calculates the empirical skew of input values.

The function block FB_CMA_MomentCoefficients [▶ 130] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▶ 115] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 148] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

As an alternative to the kurtosis the FB_CMA_CrestFactor [▶ 83] block calculates a different measure for peakiness (Crest Factor) of a signal, although this is more sensitive to outliers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022	PC or CX (x86, x64)	Tc3_CM (>= 1.0.22), Tc3_CM_Base (>= 1.1.10)

5.1.9 FB_CMA_EmpiricalMean

Calculates the mean value for single- and multi-channel real-valued time series.

The function block treats the input signal as a time series, if necessary with several independent channels. For each channel, the empirical (arithmetic) mean value according to equation

$$\overline{x} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]$$

is calculated. A single sample per channel (see Inputs and Outputs, first table) can be added in each cycle, and several samples per channel can be added to the sample quantity in one cycle (see Inputs and Outputs, second table).

Memory properties

The sample quantity *N*, which is used to calculate the current mean value, automatically increases with each new incoming data record, i.e. the function block uses all input values since its instantiation. Resetting of the sample quantity to zero (deleting the internal memory of the FB) is provided by a ResetData() method or, if the CallEx() method is used, by the variable `bResetData`.

NaN occurrence

If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Error values

If the situations described above, which lead to NaN values, cannot be ruled out or safely neglected, the application program must be able to handle these error values.

Sample implementation

A sample implementation is available under the following link: Statistical methods [▶ 206]

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	LREAL	1	`nChannels`

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	1	`nChannels`

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars; // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle to calculate and output the current mean value from the input signal and the current internal memory of the FB. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle to update the internal memory from the input signal. A result is output only every `nAppendData` cycles. An alternative method is Call().

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.
- **bResetData** : If set, the internal data buffer is completely deleted after calculation.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

ResetData() :

This method deletes all the data sets already added. Alternatively, the automatic reset can be used via the variable `bResetData` in the method CallEx().

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID        : UDINT;                            // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                         // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The function block FB_CMA_EmpiricalStandardDeviation [▸ 105] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalSkew [▸ 101] calculates the empirical skew of input values.

The function block FB_CMA_EmpiricalExcess [▸ 93] calculates the empirical excess of input values.

The function block FB_CMA_MomentCoefficients [▸ 130] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▸ 115] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▸ 148] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

As an alternative to the kurtosis the FB_CMA_CrestFactor [▸ 83] block calculates a different measure for peakiness (Crest Factor) of a signal, although this is more sensitive to outliers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022	PC or CX (x86, x64)	Tc3_CM (>= 1.0.22), Tc3_CM_Base (>= 1.1.10)

5.1.10 FB_CMA_EmpiricalSkew

Calculates the empirical skew for single- and multi-channel real-valued time series.

The function block treats the input signal as a time series, if necessary with several independent channels. For each channel, the empirical skew according to equation

$$v = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^3$$

where *s* is the empirical standard deviation. The skew quantifies the asymmetry of a sample. It offers a possibility to assess impacts (e.g. by rolling over local damage in the roller bearing) in a vibration signal. The calculated skew is a more robust criterion than the kurtosis/excess, although local damage does not necessarily lead to asymmetrical signal distributions.

A single sample per channel (see Inputs and Outputs, first table) can be added in each cycle, and several samples per channel can be added to the sample quantity in one cycle (see Inputs and Outputs, second table).

Memory properties

The sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data record, i.e. the function block uses all input values since its instantiation. Resetting of the sample quantity to zero (deleting the internal memory of the FB) is provided by a ResetData() method or, if the CallEx() method is used, by the variable `bResetData`.

Further comments

Three values must be available for calculating an initial result. Furthermore, the standard deviation must not be zero. Results may become inaccurate if the input values are many orders of magnitude apart.

NaN occurrence

If the number of input values is insufficient for calculating a result for a particular channel or the variance is zero, the value NaN (not a number) according to IEC 754 is returned for this channel. The presence of this error value can be checked with the function LrealIsNaN().
The reason may be that so far not enough input data were transferred or that only NaNs were transferred as input values for individual channels.
A variance of zero may occur if the time series of the values is constant, for example if no sensor data were transferred due to a broken wire or switching errors.

If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Error values

If the situations described above, which lead to NaN values, cannot be ruled out or safely neglected, the application program must be able to handle these error values.

Sample implementation

A sample implementation is available under the following link: Statistical methods [▶ 206]

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	LREAL	1	`nChannels`

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	1	`nChannels`

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars; // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;            // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle to calculate and output the current value of the skew from the input signal and the current internal memory of the FB. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle to update the internal memory from the input signal. A result is output only every `nAppendData` cycles. An alternative method is Call().

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData  : UDINT;       // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData   : BOOL;        // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult   : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError       : BOOL;         // TRUE if an error occurs.
    hrErrorCode  : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.
- **bResetData** : If set, the internal data buffer is completely deleted after calculation.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

ResetData() :

This method deletes all the data sets already added. Alternatively, the automatic reset can be used via the variable `bResetData` in the method CallEx().

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID        : UDINT;                              // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                         // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▸ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The function block FB_CMA_EmpiricalMean [▸ 97] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalStandardDeviation [▸ 105] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalExcess [▸ 93] calculates the empirical excess of input values.

The function block FB_CMA_MomentCoefficients [▸ 130] calculates the empirical mean, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▸ 115] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▸ 148] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

As an alternative to the kurtosis the FB_CMA_CrestFactor [▸ 83] block calculates a different measure for peakiness (Crest Factor) of a signal, although this is more sensitive to outliers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022	PC or CX (x86, x64)	Tc3_CM (>= 1.0.22), Tc3_CM_Base (>= 1.1.10)

5.1.11 FB_CMA_EmpiricalStandardDeviation

Calculates the empirical standard deviation for single- and multi-channel real-valued time series.

The function block treats the input signal as a time series, if necessary with several independent channels. For each channel, the empirical standard deviation according to equation

$$s = \sqrt{\frac{1}{N-1}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

is calculated. The Bessel's correction is always applied, in contrast to FB_CMA_MomentCoefficients [▸ 130]. A single sample per channel (see Inputs and Outputs, first table) can be added in each cycle, and several samples per channel can be added to the sample quantity in one cycle (see Inputs and Outputs, second table).

Memory properties

The sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data record, i.e. the function block uses all input values since its instantiation. Resetting of the sample quantity to zero (deleting the internal memory of the FB) is provided by a ResetData() method or, if the CallEx() method is used, by the variable `bResetData`.

Further comments

Two values must be available for calculating an initial result. Results may become inaccurate if the input values are many orders of magnitude apart.

NaN occurrence

If the number of input values is insufficient for calculating a result for a particular channel or the variance is zero, the value NaN (not a number) according to IEC 754 is returned for this channel. The presence of this error value can be checked with the function LrealIsNaN().
The reason may be that so far not enough input data were transferred or that only NaNs were transferred as input values for individual channels.
A variance of zero may occur if the time series of the values is constant, for example if no sensor data were transferred due to a broken wire or switching errors.

If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Error values

If the situations described above, which lead to NaN values, cannot be ruled out or safely neglected, the application program must be able to handle these error values

Sample implementation

A sample implementation is available under the following link: Statistical methods [▶ 206]

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	LREAL	1	`nChannels`

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	1	`nChannels`

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars; // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;            // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;          // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle to calculate and output the current value of the standard deviation from the input signal and the current internal memory of the FB. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle to update the internal memory from the input signal. A result is output only every `nAppendData` cycles. An alternative method is Call().

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.
- **bResetData** : If set, the internal data buffer is completely deleted after calculation.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

ResetData() :

This method deletes all the data sets already added. Alternatively, the automatic reset can be used via the variable `bResetData` in the method CallEx().

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID        : UDINT;                              // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                         // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 174]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_EmpiricalMean [▶ 97] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalSkew [▶ 101] calculates the empirical skew of input values.

The function block FB_CMA_EmpiricalExcess [▶ 93] calculates the empirical excess of input values.

The function block FB_CMA_MomentCoefficients [▸ 130] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▸ 115] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▸ 148] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

As an alternative to the kurtosis the FB_CMA_CrestFactor [▸ 83] block calculates a different measure for peakiness (Crest Factor) of a signal, although this is more sensitive to outliers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022	PC or CX (x86, x64)	Tc3_CM (>= 1.0.22), Tc3_CM_Base (>= 1.1.10)

5.1.12 FB_CMA_Envelope

Calculates the envelope of a time signal.

The envelope is defined mathematically as the absolute value of the analytical signal, see FB_CMA_AnalyticSignal [▸ 76]. In the time-continuous display, the envelope $x_{\mathrm{env}}(t)$ of the signal $x(t)$ is defined as:

$$x_{\mathrm{env}}(t) = |x_{\mathrm{analytic}}(t)| \quad , \quad x_{\mathrm{analytic}}(t) = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

The envelope can be interpreted as amplitude-modulated component of the signal x(t), for example

$$x(t) = x_{\mathrm{env}}(t) \cos \varphi(t)$$

The phase-modulated component $\varphi(t)$ can also be calculated, see FB_CMA_InstantaneousPhase [▸ 121]. The envelope can be used for efficient evaluation of rise or decay processes.

The discrete calculation of the envelope with the function block takes place efficiently in the frequency range.

The input vector is first overlapped with the immediately preceding buffer and multiplied with a window function. Subsequently an FFT for real input values is applied. Within the frequency range the Hilbert transform is applied to the signal, and the result is transformed back to the time range. The absolute value of the resulting complex values is calculated. The time signal is added up overlapping using the Overlap-Add method. By selecting suitable window functions a continuous output signal without step changes can be achieved.

The envelope only provides valid results for mean-free signals. If a signal with a mean value is to be analyzed, the signal average must be subtracted beforehand and added back to the result of the function block with the previously subtracted value.

Memory properties

Since the Overlap-Add method is used, in each case the current input buffer together with the two last transferred buffers is used for the calculation.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	nWindowLength/2
output stream	LREAL	1	nWindowLength/2

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method) It can only be assigned once. It cannot be changed at runtime.

```
VAR_INPUT
    stInitPars    : ST_CM_Envelope_InitPars;         // init parameter
    nOwnID        : UDINT;                           // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers: UDINT := 4;                      // number of MultiArrays which should be initia
lized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;         // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_Envelope_InitPars [▸ 175]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError         : BOOL;       // TRUE if an error occurs. Reset by next method call.
    hrErrorCode    : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults    : ULINT;      // counts outgoing results (MultiArrays were calculated and sent to
transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the envelope from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.

- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_Envelope_InitPars;          // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_Envelope_InitPars [▸ 175]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The FB_CMA_AnalyticSignal [▸ 76] block calculates the analytical signal of a time series.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.13 FB_CMA_EnvelopeSpectrum

Calculates the envelope spectrum of a time signal.

The envelope spectrum is a combined function block of FB_CMA_Envelope [▸ 109] and FB_CMA_PowerSpectrum [▸ 142].
Accordingly, the signal envelope of a time signal is calculated first, followed by the power spectrum.
The function block is very important for frequency-resolved analysis of roller bearing damage, see Bearing monitoring [▸ 39].

Memory properties

Since the Overlap-Add and the Welch methods are used, in each case the current input buffer together with the three last transferred buffers is used for the calculation.

Sample implementation

A sample implementation is available under the following link: Envelope spectrum [▸ 218].

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▸ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength`/2
output stream	LREAL	1	`nFFT_length_Spectrum`/2 +1

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars        : ST_CM_EnvelopeSpectrum_InitPars;    // init parameter
    nOwnID            : UDINT;                              // ID for this FB instance
    aDestIDs          : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers    : UDINT := 4;                         // number of MultiArrays which should be
initialized for results (0 for no initialization)
```

```
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to 
inter-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_EnvelopeSpectrum_InitPars [▶ 175]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError          : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode     : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults     : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to
 transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the envelope spectrum from the input signal. The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult      : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError          : BOOL;        // TRUE if an error occurs.
    hrErrorCode     : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars       : ST_CM_EnvelopeSpectrum_InitPars;  // init parameter
    nOwnID     : UDINT;                                  // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_EnvelopeSpectrum_InitPars [▸ 175]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The function block FB_CMA_Envelope [▸ 109] calculates the envelope of a time series.

The FB_CMA_PowerSpectrum [▸ 142] block calculates the power spectrum by means of squaring of the values in the last step.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.14 FB_CMA_HistArray

Calculates the frequency distribution for single- and multi-channel real-valued time series.

The function block FB_CMA_HistArray calculates the frequency distribution (in the graphical representation referred to as histogram) of single- and multi-channel real-valued input data. Each channel is considered independently. For each individual channel the frequency distribution of the cyclic incoming data buffer is calculated; both individual values and arrays are permitted as input buffer.

The lower and upper limit values and the number of classes (also referred to as bins) are transferred for parameterization. The individual class limits are then distributed in identical intervals across the defined range, cf. Histograms [▶ 25]. Values below the lower limit and values above the upper limit are counted in two additional bins.

The return value is a two-dimensional array with unsigned 64-bit integer values. The first index is the channel number, the second index is the number of the respective histogram bin. The counts for elements with values below the lower limit value or above the upper limit value are contained in the first and last bin respectively.

If a histogram counter exceeds a value of 2 to the power of 64, approx. 18E19, in the current implementation the counter overruns without generating an error message. With a counting step of 100 microseconds, this would happen after 59 million years at the earliest.

Configuration

The initialization parameters specify the limits for counting samples in the regular histogram bins. They can be individually adjusted for each channel with the `Configure()` method.

NaN occurrence

If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Memory properties

The function block takes into account all input values since the instantiation or since the last call of the ResetData() method, if it was called since the start.
Sample implementation

A sample implementation is available under the following link: Histogram [▶ 204].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	ULINT	2	`nChannels x (nBins` +2)

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels x` *not specified**
output stream	ULINT	2	`nChannels x (nBins` +2)

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_HistArray_InitPars;         // init parameter
    nOwnID           : UDINT;                  // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;             // number of MultiArrays which should be initialized f
or results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;   // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_HistArray_InitPars [▸ 176]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;        // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;       // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the histogram from the input signal. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle in order to calculate the histogram from the input signal. An alternative method is Call().
The histogram evaluation is generally significantly more computationally demanding than the registration of new input values. Therefore a use of the method Callex() can considerably shorten the runtime, depending on the configured parameters, by only calculating statistic results when they are required.

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.
- **bResetData** : If set, the internal data buffer is completely deleted after calculation.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Configure() :

By calling this method, the histogram arguments can be reconfigured. This allows fine adjustment of the fMinBinned and fMaxBinned parameters during runtime. The corresponding PLC array must be defined as follows.

	Element type	Dimensions	Dimensional variables
Argument	LREAL	2	`nChannels` x 2

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg       : POINTER TO LREAL;  // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize   : UDINT;             // size of arguments buffer in bytes
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

This method deletes all the data sets already added. Alternatively the automatic reset in the method CallEx() can be used.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_HistArray_InitPars;         // init parameter
    nOwnID         : UDINT;       // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;  // number of MultiArrays which should be initialized for results (
0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_HistArray_InitPars [▶ 176]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The FB_CMA_Quantiles [▶ 148] function block calculates the quantiles of input value distributions.
The FB_CMA_MomentCoefficients [▶ 130] function block calculates the statistical moment coefficients: average value, standard deviation, skew and kurtosis.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base

5.1.15 FB_CMA_InstantaneousFrequency

Calculation of the instantaneous frequency of a time signal

The instantaneous frequency in the mathematical sense is defined as temporal derivative of the instantaneous phase, see FB_CMA_InstantaneousPhase [▶ 121]. In the time-continuous display, the instantaneous frequency *ω*(t) of the signal *x*(t) is defined as:

$$\omega(t) = \frac{\mathrm{d}}{\mathrm{d}t}\varphi(t) = \frac{\mathrm{d}}{\mathrm{d}t}\arctan\frac{\mathcal{H}[x(t)]}{x(t)} \quad , \quad x_{\text{analytic}} = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

The instantaneous frequency can be interpreted as frequency-modulated component of the signal x(t), for example

$$x(t) = x_{\text{env}}(t)\cos\varphi(t) = x_{\text{env}}(t)\cos\int\omega(t)\mathrm{d}t$$

In this way the signal *x*(t) can be transformed into the amplitude- and frequency-modulated representation through calculation of the instantaneous frequency and the envelope [▶ 109] .

The function blocks instantaneous phase and instantaneous frequency only provide valid results for signals without mean values. If a signal with a mean value is to be analyzed, the signal average must be subtracted beforehand.

The instantaneous frequency is well suited for analyzing torsional vibrations of a crankshaft. Torsional vibrations can be caused by a fluctuating torque, for example, and result in a frequency modulation on an otherwise uniform speed.

Memory properties

Since the Overlap-Add method is used, in each case the current input buffer together with the two last transferred buffers is used for the calculation.

NaN occurrence

If one or several elements at the input are NaN (Not a Number), the total output signal is NaN.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength/2`
output stream	LREAL	1	`nWindowLength/2`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_InstantaneousFrequency_InitPars;   // init parameter
    nOwnID           : UDINT;                               // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_InstantaneousFrequency_InitPars [▸ 177]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;        // counts outgoing results (MultiArrays were calculated and sent t
o transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the instantaneous frequency from the input signal. The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent to
 transfer tray.
    bError      : BOOL;       // TRUE if an error occurs.
    hrErrorCode : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute ‘call_after_init’ (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_InstantaneousFrequency_InitPars;       // init parameter
    nOwnID    : UDINT;                                      // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                            // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_InstantaneousFrequency_InitPars [▶ 177]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_InstantaneousPhase [▶ 121].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.16 FB_CMA_InstantaneousPhase

Calculation of the instantaneous phase of a time signal

The instantaneous phase $\varphi(t)$ of a signal $x(t)$ is defined via the phase of the analytical signal, see FB_CMA_AnalyticSignal [▶ 76]:

$$\varphi(t) = \arctan \frac{\mathcal{H}[x(t)]}{x(t)} \quad , \quad x_{\text{analytic}} = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

The instantaneous phase can be interpreted as phase-modulated component of the signal x(t):

$$x(t) = x_{\text{env}}(t) \cos \varphi(t)$$

The amplitude-modulated component (envelope) of the signal can also be determined, see FB_CMA_Envelope [▶ 109].

The function blocks instantaneous phase and instantaneous frequency only provide valid results for signals without mean values. If a signal with a mean value is to be analyzed, the signal average must be subtracted beforehand.

Memory properties

Since the Overlap-Add method is used, in each case the current input buffer together with the two last transferred buffers is used for the calculation.

NaN occurrence

If one or several elements at the input are NaN (Not a Number), the total output signal is NaN.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength/2`
output stream	LREAL	1	`nWindowLength/2`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_InstantaneousPhase_InitPars;   // init parameter
    nOwnID           : UDINT;                              // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                         // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_InstantaneousPhase_InitPars [▶ 177]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;        // counts outgoing results (MultiArrays were calculated and sent t
o transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the phase angle from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent to
 transfer tray.
    bError      : BOOL;       // TRUE if an error occurs.
    hrErrorCode : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_InstantaneousPhase_InitPars;       // init parameter
    nOwnID    : UDINT;                                // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function block-specific structure with initialization parameters of type ST_CM_InstantaneousPhase_InitPars [▶ 177]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The function block FB_CMA_InstantaneousFrequency [▸ 119].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.17 FB_CMA_IntegratedRMS

Calculates (optionally integrated) RMS values for single- and multi-channel real-valued time series.

Calculates the RMS value for single- and multi-channel time series; both the frequency range used and the integration order of the time series can be defined. For an acceleration signal this results in effective values for the vibration acceleration, vibration velocity and the vibration amplitude, each in a defined frequency range.

The block treats the input signal as a signal with several independent channels. For each channel the values for three different integration orders within the frequency range are integrated over a defined frequency interval, and the effective values are then calculated. The block is suitable for vibration assessment according to DIN ISO 10816 and DIN ISO 7919 or DIN ISO 20816, see Vibration assessment [▸ 31].

The sampling rate and the limits of the integrated intervals can be parameterized. In order to obtain reproducible scaling, the input signals and the frequencies must be transferred scaled in SI units, i.e. 1 m/(sec)^2 for acceleration values and 1/sec = 1 Hz for frequencies. The maximum order of the integration can be configured between zero and two.

The number of integrated RMS values to be calculated is to be specified by means of (nOrder + 1). The result is forwarded as an array of these values with the corresponding indices.

In many cases the underlying short-term power spectrum is not a good statistical estimator for the spectrum of a signal, so that the values can fluctuate despite averaging over frequencies. It is therefore advisable to use a sufficiently large window length. In many cases it may additionally be advisable to reduce the fluctuation by calculating the geometric mean over several consecutive values.

Memory characteristics

Since the Welch method is used, in each case the current input buffer together with the last transferred buffer is used for the calculation.
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, the last two input buffers should therefore be consecutive without step changes.

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▸ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x `nWindowLength`/2
output stream	LREAL	2	`nChannels` x (`nOrder` +1)

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars        : ST_CM_IntegratedRMS_InitPars;    // init parameter
    nOwnID        : UDINT;                               // ID for this FB instance
    aDestIDs          : ARRAY[1..cCMA_MaxDest] OF UDINT;// IDs of destinations for output
    nResultBuffers    : UDINT := 4;                      // number of MultiArrays which should be init
ialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to inte
r-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_IntegratedRMS_InitPars [▸ 178]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;        // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;       // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.

- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the integrated RMS values from the input signal. The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;      // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;      // TRUE if an error occurs.
    hrErrorCode   : HRESULT;   // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_IntegratedRMS_InitPars;     // init parameter
    nOwnID          : UDINT;                            // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_IntegratedRMS_InitPars [▶ 178]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called

cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The FB_CMA_MagnitudeSpectrum [▶ 127] function block calculates the magnitude spectrum without squaring of the values in the last step.

The FB_CMA_PowerCepstrum [▶ 138] function block calculates a transformation that emphasizes harmonics.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.18 FB_CMA_MagnitudeSpectrum

Calculates the magnitude spectrum (also referred to as amplitude spectrum) of a real-valued input signal.

The function block FB_CMA_MagnitudeSpectrum calculates the magnitude spectrum from a real-valued input signal. The function block performs several functions, see Analysis of data streams [▶ 16] and Frequency analysis [▶ 35]:

The input data buffer is first overlapped with the immediately preceding buffer and multiplied with a window function. If the value of parameter `nFFT_Length` is greater than the parameter `nWindowLength`, the windowed time signal is filled with the same number of zeros at the beginning and the end, in order to reach the required FFT input length (zero padding). Subsequently a FFT for real values is applied, and the absolute value of the resulting complex values is calculated. If the parameter `bTransformToDecibel` is `TRUE`, the values are transformed to decibel values. These decibel values are the same for magnitude and power spectra, i.e. the influence of squaring is taken into account in the calculation of the decibel value by a factor of two for the magnitude spectrum. In addition, the magnitude spectrum can be scaled via the parameter `eScalingType`, see Scaling of spectra [▶ 22].

The block FB_CMA_MagnitudeSpectrum behaves similar to FB_CMA_PowerSpectrum [▶ 142]. The difference is squaring of the results in FB_CMA_PowerSpectrum [▶ 142].

In many cases the short-term spectrum is not a good statistical estimator for the spectrum of a signal. In many cases it is advisable to reduce the fluctuation of the estimated values through averaging over several frequencies or over consecutive spectra.

Scale

The scaling of the result values, e.g. the acceleration spectral densities matches the definition of the FFT by default. This means that the influence of the window length and the window function can be eliminated.
For mathematical scaling of absolute measurements tabulated parameters can be used, which are described in section "Scaling factors".

Memory characteristics

Since the Welch method is used, in each case the current input buffer together with the last transferred buffer is used for the calculation.
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, the last two input buffers should therefore be consecutive without step changes.

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▸ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Sample implementation

A sample implementation is available under the following link: Magnitude spectrum: [▸ 192].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength`/2
output stream	LREAL	1	`nFFT_Length`/2 +1

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_MagnitudeSpectrum_InitPars;  // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;            // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to i
nter-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_MagnitudeSpectrum_InitPars [▸ 179]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError       : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode  : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults  : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to
 transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the magnitude spectrum from the input signal. The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
 to transfer tray.
    bError       : BOOL;         // TRUE if an error occurs.
    hrErrorCode  : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MagnitudeSpectrum_InitPars; // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_MagnitudeSpectrum_InitPars [▶ 179]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The FB_CM_PowerSpectrum [▶ 142] block calculates the power spectrum by means of squaring of the values in the last step.

The FB_CMA_PowerCepstrum [▶ 138] function block calculates a transformation that emphasizes harmonics.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.19 FB_CMA_MomentCoefficients

Calculates the average value, the empirical standard deviation, the skew and the excess for single- and multi-channel real-valued time series.

The function block treats the input signal as a time series, if necessary with several independent channels. For each channel the moment coefficients are calculated, optionally up to the fourth order. The maximum order of the moments to be calculated can be configured via the parameter `nOrder`. A specific enumeration for application of the moment coefficients is also available: E_CM_MCoefOrder [▶ 165]. The result is forwarded as an array of these coefficients with corresponding indices.

By default no Bessel's correction is applied for the calculation of the empirical standard deviation, the skew and the excess. In the initialization parameters the correction can optionally be switched on, see `bPopulationEstimates`. The parameter should be set to `TRUE`, in order to obtain results that meet expectations. The influence of Bessel's correction becomes smaller with increasing sample size. The relative deviation between the corrected and the non-corrected empirical standard deviation can be determined unambiguously. The following table provides clues:

Sample size *N*	**Relative deviation / %**
10	-5.13
100	-0.501
1000	-0.05001

10000	-0.0050001

Output from the function block: The sample size *N* (for all `nOrder`), the arithmetic mean value (`nOrder = 1`), the empirical standard deviation (`nOrder = 2`), the skew (`nOrder = 3`), the excess (`nOrder = 4`).

Definition of empirically calculated moments

The arithmetic mean value

$$\overline{x} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]$$

The empirical standard deviation, without Bessel's correction

$$s' = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

The empirical standard deviation, with Bessel's correction

$$s = \sqrt{\frac{1}{N-1}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

The empirical skew (without Bessel's correction *v‘* and with correction *v*))

$$v' = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s'}\right)^3 \qquad v = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^3$$

The empirical excess (without Bessel's correction *E‘* and with correction *E*)

$$K' = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s'}\right)^4 \qquad K = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^4$$

$$E' = K' - 3 \qquad E = K - 3$$

The excess *E* is therefore the difference between the kurtosis *K* and the value 3; this corresponds to the kurtosis of the normal distribution. It describes the evaluation of the calculated kurtosis in terms of a normal distribution.

A single sample per channel (see Inputs and Outputs, first table) can be added in each cycle, and several samples per channel can be added to the sample quantity in one cycle (see Inputs and Outputs, second table).

Further comments

The calculation of the standard deviation and higher moments requires a minimum number of sample values. If Bessel's correction is inactive, the mean value and the standard deviation are calculated for a sample size of 1. Two values are required for calculating the skew and the excess. If Bessel's correction is active, the minimum sample size required corresponds to the order (mean value - 1, standard deviation - 2, skew - 3, excess - 4). In addition, for calculating skew and excess the variance cannot be null.

Results for higher moments may become imprecise, if the input values differ by many orders of magnitude.

Memory properties

The sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data record, i.e. the function block uses all input values since its instantiation. Resetting of the sample quantity to zero (deleting the internal memory of the FB) is provided by a ResetData() method or, if the CallEx() method is used, by the variable `bResetData`.

NaN occurrence

If the number of input values is insufficient for calculating a result for a particular channel or the variance is zero, the value NaN (not a number) according to IEC 754 is returned for this channel. The presence of this error value can be checked with the function LrealIsNaN().
The reason may be that so far not enough input data were transferred or that only NaNs were transferred as input values for individual channels.
A variance of zero may occur if the time series of the values is constant, for example if no sensor data were transferred due to a broken wire or switching errors.

If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Error values

If the situations described above, which lead to NaN values, cannot be ruled out or safely neglected, the application program must be able to handle these error values

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	LREAL	2	`nChannels` x (`nOrder` +1)

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	2	`nChannels` x (`nOrder` +1)

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_MomentCoefficients_InitPars; // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;            // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_MomentCoefficients_InitPars [▸ 180]. The parameters must correlate to the above definition of the input and output buffers.

- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;          // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;         // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle to calculate and output the current results from the input signal and the current internal memory of the FB. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle to update the internal memory from the input signal. A result is output only every `nAppendData` cycles. An alternative method is Call().

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;       // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;        // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.

- **bResetData** : If set, the internal data buffer is completely deleted after calculation.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

ResetData() :

This method deletes all the data sets already added. Alternatively, the automatic reset can be used via the variable `bResetData` in the method CallEx().

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_MomentCoefficients_InitPars;  // init parameter
    nOwnID        : UDINT;                              // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                         // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_MomentCoefficients_InitPars [▸ 180]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_EmpiricalMean [▶ 97] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalStandardDeviation [▶ 105] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalSkew [▶ 101] calculates the empirical skew of input values.

The function block FB_CMA_EmpiricalExcess [▶ 93] calculates the empirical excess of input values.

The FB_CMA_HistArray [▶ 115] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 148] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

As an alternative to the kurtosis the FB_CMA_CrestFactor [▶ 83] block calculates a different measure for peakiness (Crest Factor) of a signal, although this is more sensitive to outliers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.20 FB_CMA_MultiBandRMS

Calculated RMS value for single- and multi-channel real-valued time series for configurable frequency bands

The block calculates the RMS values of the signals for single- and multi-channel time series, based on individually configurable frequency bands.

The number of channels is described via the input stream. The maximum number of frequency bands configured for a channel and the parameters of the internal Fourier transformation are transferred via ST_CM_MultiBandRMS_InitPars [▶ 180]. The frequency bands are configured by calling the Configure() method.

The block is well suited for monitoring of bearing damage frequencies, for example.

Delimitation to FB_CMA_IntegratedRMS [▶ 124]:

The IntegratedRMS block has additional functionality in that the input time series can be temporally integrated before the frequency band-limited RMS calculation, optionally up to second order. In this way the IntegratedRMS block can directly calculate the RMS value for vibration acceleration, vibration velocity and vibration displacement for a defined frequency band, for example. On the other hand, the IntegratedRMS block can only be used to define a single frequency band.

Configuration

As configuration parameters, a three-dimensional array with values is transferred to the Configure() method of the function block (or optionally two-dimensional in case of a single input channel). Each value specifies the lower and upper limit of a frequency band. The function block then calculates the RMS values for these frequency bands of each channel on the basis of the input data.

Memory characteristics

Since the Welch method is used, in each case the current input buffer together with the last transferred buffer is used for the calculation.
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, the last two input buffers should therefore be consecutive without step changes.

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▶ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Sample implementation

A sample implementation is available under the following link: http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649246859.zip.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	nChannels x nWindowLength/2
output stream	LREAL	2	nChannels x nMaxBands

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_MultiBandRMS_InitPars;     // init parameter
    nOwnID           : UDINT;                 // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;            // number of MultiArrays which should be initialized f
or results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;  // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_MultiBandRMS_InitPars [▶ 180]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;        // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;       // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.

- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate multi-band RMS values from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Configure() :

The RMS bands must be configured at the beginning with the call of this method. The corresponding PLC array must be defined as follows. The Configure() method can also be used for a new configuration with a different set of arguments.

	Element type	Dimensions	Dimensional variables
Argument	LREAL	3	nChannels x nMaxBands x 2

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg        : POINTER TO LREAL;  // pointer to 3-dimensional array (LREAL) of arguments (or 2-
dimensional in case of single input channel)
    nArgSize    : UDINT;             // size of arguments buffer in bytes
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute ‘call_after_init’ (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MultiBandRMS_InitPars;      // init parameter
    nOwnID         : UDINT;         // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;    // number of MultiArrays which should be initialized for results
(0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_MultiBandRMS_InitPars [▸ 180]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.21 FB_CMA_PowerCepstrum

The function block calculates the power cepstrum for a real-valued input signal.

The power cepstrum $C_p(\tau)$ is defined as follows, in this case in time-continuous representation:

$$C_p(\tau) = X^{-1}\left\{\log(|X(\omega)|^2)\right\} = X^{-1}\left\{2\log|X(\omega)|\right\}, \qquad \text{Quefrency } \tau$$

Accordingly, it is defined as inverse Fourier transformation of the logarithmized power spectrum (see FB_CMA_PowerSpectrum [▸ 142]). Transformation and inverse transformation bring the result back into the time range.

The function block is helpful for monitoring of gear units, see Gearbox monitoring [▶ 47].

In the numerical implementation the PowerSpectrum is calculated first. The input data buffer is this overlapped with the immediately preceding buffer and multiplied with a window function. If the value of parameter `nFFT_Length` is greater than the parameter `nWindowLength`, the windowed time signal is filled with the same number of zeros at the beginning and the end, in order to reach the required FFT input length (zero padding). Subsequently a FFT for real values is applied, and the absolute value of the resulting complex values and the square of the values is calculated. The values are then logarithmized. Before the logarithmization the argument is compared with the value of the parameter `fLogThreshold`. If they are smaller they are set to this value in order to avoid value range errors or the attempt to calculate the logarithm of zero. This is followed by another inverse Fourier transformation. The result is an array with complex values.

Evaluation of the complex-valued result

In practice the absolute value, the squared absolute value or only the real part of the complex-valued power cepstrum is evaluated, depending on the application. This has to be implemented by the user as required.

Differentiation to the complex cepstrum:

The power cepstrum differs from the complex cepstrum, which is defined as logarithmized Fourier back transformation of a complex signal spectrum. Due to the absolute value calculation the power cepstrum is less sensitive to the properties of the phase angle of the signal, compared with the complex cepstrum. In addition, the complex cepstrum directly provides a real-valued result.

Definition of the power cepstrum:

A number of slightly different definitions exist for the power cepstrum. The definition used here is based on common definitions by J. Korelus and Robert B. Randall, see Literature notes [▶ 51].

Memory characteristics

Since the Welch method is used, in each case the current input buffer together with the last transferred buffer is used for the calculation.
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, the last two input buffers should therefore be consecutive without step changes.

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▶ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Sample implementation

A sample implementation is available under the following link: Power cepstrum [▶ 220]

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength`/2
output stream	LCOMPLEX	1	`nFFT_Length`/2 +1

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_PowerCepstrum_InitPars;    // init parameter
    nOwnID           : UDINT;                 // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;            // number of MultiArrays which should be initialized f
or results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;  // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_PowerCepstrum_InitPars [▶ 181]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;      // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;   // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;     // counts outgoing results (MultiArrays were calculated and sent to tra
nsfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the power cepstrum from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError      : BOOL;        // TRUE if an error occurs.
    hrErrorCode : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_PowerCepstrum_InitPars;     // init parameter
    nOwnID         : UDINT;         // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;    // number of MultiArrays which should be initialized for results
 (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_PowerCepstrum_InitPars [▶ 181]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The function block FB_CMA_Envelope [▶ 109] is also suitable for the analysis of pulse-like excitations with linear and non-linear system components.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base

5.1.22 FB_CMA_PowerSpectrum

Calculation of the power spectrum of a real-valued input signal, and optional decibel scaling.

The block calculates the power spectrum (also referred to as correlogram or periodogram) of a real-valued input signal. The function block performs several functions, see Analysis of data streams [▸ 16] and Frequency analysis [▸ 35]:

The input data buffer is first overlapped with the immediately preceding buffer and multiplied with a window function. If the value of parameter `nFFT_Length` is greater than the parameter `nWindowLength`, the windowed time signal is filled with the same number of zeros at the beginning and the end, in order to reach the required FFT input length (zero padding). Subsequently a FFT for real values is applied, and the absolute value of the resulting complex values and the square of the values is calculated. If the parameter `bTransformToDecibel` is `TRUE`, the values are transformed to decibel values. These decibel values are the same for magnitude and power spectra, i.e. the influence of squaring is taken into account in the calculation of the decibel value by a factor of two for the magnitude spectrum. In addition, the magnitude spectrum can be scaled via the parameter `eScalingType`, see Scaling of spectra [▸ 22].

The block FB_CMA_PowerSpectrum behaves similar to FB_CMA_MagnitudeSpectrum [▸ 127]. The difference is squaring of the results in FB_CMA_PowerSpectrum [▸ 142].

In many cases the short-term power spectrum is not a good statistical estimator for the spectrum of a signal. In many cases the fluctuation of the estimated values should be reduced through averaging over several frequencies or over consecutive spectra.

Scale

The scaling of the result values, e.g. the acceleration spectral densities matches the definition of the FFT by default. This means that the influence of the window length and the window function can be eliminated. For mathematical scaling of absolute measurements tabulated parameters can be used, which are described in section "Scaling factors".
The scaling of the power densities may become imprecise, if very small values are specified for the window lengths.

Memory characteristics

Since the Welch method is used, in each case the current input buffer together with the last transferred buffer is used for the calculation.
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, the last two input buffers should therefore be consecutive without step changes.

NaN occurrence

If the input vector contains one or more NaN values, the entire spectrum result is filled with NaN.
This property can be used to mark results as undefined in case a gap in the input signal leads to jumps in the time series. Refer here to the description of the input methods at the FB_CMA_Source [▸ 159].
If incoming NaN values cannot be excluded, the user program must handle these error values.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nWindowLength`/2
output stream	LREAL	1	`nFFT_Length`/2 +1

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_PowerSpectrum_InitPars;    // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_PowerSpectrum_InitPars [▸ 182]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;        // counts outgoing results (MultiArrays were calculated and sent t
o transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the power spectrum from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent to
 transfer tray.
    bError      : BOOL;       // TRUE if an error occurs.
    hrErrorCode : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.

- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_PowerSpectrum_InitPars;     // init parameter
    nOwnID    : UDINT;                                 // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_PowerSpectrum_InitPars [▶ 182]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

The method deletes all data records that have already been added, see Memory property of the function block. If the Call() method is called again after a ResetData(), the internal memory must be replenished in order to calculate a valid result.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The FB_CMA_MagnitudeSpectrum [▸ 127] function block calculates the magnitude spectrum without squaring of the values in the last step.

The FB_CMA_PowerCepstrum [▸ 138] function block calculates a transformation that emphasizes harmonics.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.23 FB_CMA_RealFFT

Calculation of the fast Fourier transformation (FFT) for real-valued input signals.

The FB_CMA_RealFFT function block calculates the Fourier transform of the real-valued input signal *x*[n] at the function block. A high-performance FFT algorithm is used for this purpose. The transformation and the inverse transformation can be calculated. The input `stInitPars` is used for the setting (see inputs and outputs).

Definition of the Fourier forward transformation in DFT notation:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-i2\pi nk/N}$$

Definition of the Fourier inverse transformation in DFT notation:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, e^{i2\pi nk/N}$$

The highest frequency of an input signal component should not exceed half the sampling rate of the input signal, in order to avoid aliasing effects.

The FFT is defined as transform of a cyclically continuous signal. This can result in step changes, if the cyclically continuous signal is not exactly continuous, i.e. not the same at the start and finish. The function blocks FB_CMA_PowerSpectrum [▸ 142] and FB_CMA_MagnitudeSpectrum [▸ 127] can be used to avoid these issues by using overlapping sections, which are multiplied with a window function, as the basis for the analysis.

Scaling

For a quantitative evaluation of the spectrum the calculated spectrum should be weighted with `1/nFFT_Length` for the off-set, i.e. the first array element of the outputs, and with `2/nFFT_Length` for all other outputs elements.

During the forward transformation and the inverse transformation the function block scales such that during consecutively transformations and inverse transformations the original input signal is calculated again directly at the output.

Memory properties

The function block result is only determined by the current input values, i.e. no past values are taken into account.

NaN occurrence

If one or several elements at the input are NaN (not a number), the total output signal for the real and the imaginary part is NaN.

Sample implementation

A sample implementation is available under the following link: FFT with real-value input signal [▸ 188].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nFFT_Length`
output stream	LCOMPLEX	1	`nFFT_Length`/2 +1

The output buffer can be made to output the full spectrum by negating the parameter `bHalfSpec` (:=FALSE). This enables the following alternative use option:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nFFT_Length`
output stream	LCOMPLEX	1	`nFFT_Length`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_RealFFT_InitPars;          // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of type ST_CM_RealFFT_InitPars [▸ 183]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError         : BOOL;          // TRUE if an error occurs. Reset by next method call.
    hrErrorCode    : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults    : ULINT;         // counts outgoing results (MultiArrays were calculated and sent t
o transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Call() :

The method is called in each cycle in order to calculate the FFT from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent to
 transfer tray.
    bError      : BOOL;       // TRUE if an error occurs.
    hrErrorCode : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_RealFFT_InitPars;    // init parameter
    nOwnID     : UDINT;                               // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of type ST_CM_RealFFT_InitPars [▸ 183]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The function block FB_CMA_ComplexFFT [▸ 86] calculates the Fourier transformation of a complex-valued input signal.

The function block FB_CMA_PowerSpectrum [▸ 142] calculates the power spectrum of a real-valued input signal.

The function block FB_CMA_MagnitudeSpectrum [▸ 127] calculates the magnitude spectrum of a real-valued input signal.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.24 FB_CMA_Quantiles

Calculates the quantile of the input value distribution for single- and multi-channel real-valued signals.

The block FB_CMA_Quantiles calculates p-quantiles of single- or multi-channel real-valued input data. Each channel is considered independently.

The block is initially based on the calculation of a frequency distribution, see FB_CMA_HistArray [▸ 115]. The lower and upper limit values and the number of classes (also referred to as bins) of the frequency distribution are transferred for parameterization. The individual class limits are then distributed in identical intervals across the so defined total range, see Histograms [▸ 25]. The cumulative frequency distribution is then calculated, and from this the configured quantile, see Statistical analysis [▸ 25]. A further configuration parameter is the number of quantiles to be calculated for each channel.

The result is a two-dimensional array with real values. The first index is the channel number, the second index is the number of the respective quantile.

Values that are below the lower limit and values above the upper limit with regard to the classification are ignored for the quantile calculation. Within an interval the quantile values are interpolated. If the empirical cumulative frequency distribution is constant section by section, the smallest suitable value is used.

Memory properties

The block takes into account all input values since the instantiation or since the last call of the ResetData() method, if it was called since the start.

Configuration

A two-dimensional array with values is transferred to the Configure() method of the block as configuration parameter. Each value represents the relative frequency for a channel and quantile to be calculated. The block then calculates the quantiles for these frequencies for each channel, based on the input data. The set frequency is 0.5, which corresponds to the median.

NaN occurrence

If results are not yet available for a channel, the value NaN (not a number) is returned for this channel. Reasons may be that no input data have been transferred yet, all data transferred so far are outside the interval between `fMinBinned` and `fMaxBinned`, or only NaNs were transferred as input values for

individual channels.
If a set of input values contains the special constant NaN, no value is added to the statistics for this channel for this time step, i.e. it is treated as indicator for missing values.

Error values

If the situations described above, which lead to NaN values, cannot be ruled out or safely neglected, the application program must be able to handle these error values.

Router memory

The quantile calculation is a statistical calculation based on histograms, which require a lot of memory. The memory usage depends on the parameters nChannels, nBins and nMaxQuantiles. It is recommended to keep these parameters as small as possible! Otherwise an out-of-memory error occurs, and the function block is not initialized.
Currently the maximum possible calculation comprises approx. 4200 channels, 100 bins and 4 quantile arguments.

Sample implementation

A sample implementation is available under the following link: Condition Monitoring with frequency analysis [▸ 209].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	LREAL	2	`nChannels` x `nMaxQuantiles`

If several data sets are to be added with each call, the following alternative usage is available with this function block:

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	2	`nChannels` x `nMaxQuantiles`

*: The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (Alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_Quantiles_InitPars; // init parameter
    nOwnID           : UDINT;                    // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;   // IDs of destinations for output
    nResultBuffers   : UDINT := 4;               // number of MultiArrays which should be initialize
d for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;     // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_Quantiles_InitPars [▸ 182]. The parameters must correlate to the above definition of the input and output buffers.

- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default value is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to
transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate quantiles from the input signal. An alternative method is CallEx().
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent to
 transfer tray.
    bError      : BOOL;       // TRUE if an error occurs.
    hrErrorCode : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

CallEx() :

The method is called in each cycle in order to calculate quantiles from the input signal. An alternative method is Call().
The quantile evaluation is generally significantly more computationally demanding than the registration of new input values. Therefore a use of the method Callex() can considerably shorten the runtime, depending on the configured parameters, by only calculating statistic results when they are required.

The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData  : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData   : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult   : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
```

```
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **nAppendData** : Defines how many input data buffers are to be collected before a calculation is carried out, because several data blocks are preferably added in order to achieve a precise result.
- **bResetData** : If set, the internal data buffer is completely deleted after calculation.
- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Configure() :

The quantile arguments must be configured at the beginning with the call of this method. The corresponding PLC array must be defined as follows. The Configure() method can also be used for a new configuration with a different set of arguments.

	Element type	Dimensions	Dimensional variables
Argument	LREAL	2	nChannels x nMaxQuantiles

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg          : POINTER TO LREAL;    // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize      : UDINT;               // size of arguments buffer in bytes
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

This method deletes all the data sets already added. Alternatively the automatic reset in the method CallEx() can be used.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_Quantiles_InitPars;  // init parameter
    nOwnID         : UDINT;                     // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_Quantiles_InitPars [▸ 182]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▸ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▸ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

The FB_CMA_HistArray [▸ 115] function block calculates the histograms of input value distributions.

The FB_CMA_MomentCoefficients [▸ 130] block calculates the statistical moment coefficients: average value, standard deviation, skew and kurtosis.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.25 FB_CMA_RMS

Calculates the temporal RMS value for single- and multi-channel real-valued signals.

This function block calculates the temporal RMS of one or more input channels. The RMS is calculated block-by-block over an internal buffer length *M*.

$$x_{\mathrm{RMS}} = \sqrt{\frac{1}{M}\sum_{m=0}^{M-1} x^2[m]}$$

If this internal buffer is full, the oldest values are replaced by the current ones and a new result is output. The number of input values to be replaced depends on the multi-array size set at the source function block (FB_CMA_Source [▸ 158]).

Memory properties

Input values `nBufferLength` are stored in the internal buffer (refer to the initialization parameters of the type ST_CM_RMS_InitPars [▶ 183]). These are successively replaced by new input values.

NaN occurrence

If a set of input values contains the special constant NaN, then the result is also NaN. This remains the case until the internal buffer is completely filled with new valid values.

Sample implementation

A sample implementation is available under the following link: Time-based RMS [▶ 200].

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	2	`nChannels` x *not specified**
output stream	LREAL	1	`nChannels`

*The length of the second dimension can be selected as desired and thus adapted to the application or the output buffer of the preceding algorithm.

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars       : ST_CM_RMS_InitPars;    // init parameter
    nOwnID           : UDINT;                 // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;            // number of MultiArrays which should be initialized f
or results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;  // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars**: Function-block-specific structure with initialization parameters of the type ST_CM_RMS_InitPars [▶ 183]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID**: Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs**: Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers**: The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout**: Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError      : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults : ULINT;        // counts outgoing results (MultiArrays were calculated and sent to t
ransfer tray)
END_VAR
```

- **bError**: The output is TRUE if an error occurs.
- **hrErrorCode**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call():

The method is called in each cycle in order to calculate RMS values from the input signal.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent 
to transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError**: The output is TRUE if an error occurs.
- **hrErrorCode**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Init():

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute ‘call_after_init’ (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_RMS_InitPars;  // init parameter
    nOwnID         : UDINT;               // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;  // number of MultiArrays which should be initialized for results (
0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars**: Function-block-specific structure with initialization parameters of the type ST_CM_RMS_InitPars [▶ 183]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID**: Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs**: Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers**: The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Similar function blocks

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base

Documents about this

🗎 RMS_Sample.zip (Resources/zip/9007202649250187.zip)

5.1.26 FB_CMA_Sink

This function block writes data from a multi-array buffer into an external PLC data buffer.

It contains all the multi-arrays that are transferred to the specified analysis ID.
Depending on the analysis chain the output results can contain NaN values.

NOTE

Exception

Comparisons with NaN (Not a Number) can cause an exception that leads to an execution stop and may possibly cause machine damage. It is urgently recommended to check the result for NaN before it is processed. Or if NaNs are to be processed in the application, the floating point exception must be deactivated for this task.

Inputs and outputs

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    nOwnID           : UDINT;                  // ID for this FB instance
    tTransferTimeout : LTIME := LTIME#40US;    // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **tTransferTimeout** : Synchronous timeout setting for internal multi-array transfer commands. See section Parallel processing [▸ 66].

Output parameters

```
VAR_OUTPUT
    bError       : BOOL;              // TRUE if an error occurs. Reset by next method call.
    hrErrorCode  : HRESULT;           // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults  : ULINT;             // counts outgoing results
END_VAR
```

- **bError** : This output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▸ 224].

Methods

Output1D() :
Writes data from a multi-array into an external one-dimensional data buffer.
The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Output1D : HRESULT
VAR_INPUT
    pDataOut      : POINTER TO BYTE;      // address of data buffer
    nDataOutSize  : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDim      : UDINT:=0;             // It designates the dimension in the MultiArray being pro
cessed.
    nElements     : UDINT:=0;             // optional: default:0->full
 copy; It designates the number of elements to be copied out of the MultiArray.
    pStartIndex   : POINTER TO UDINT;    (* optional: default:0-
>internally handled as [0,0,..]; It designates the index of the first element to be copied out of th
e MultiArray.
                         If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray. *)
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when data was written from MultiArray t
o data buffer.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **eElementType** : This input is of the type E_MA_ElementTypeCode [▶ 166]. The element type of the specified multi-array buffer must correlate to the element type of the specified external data buffer.
- **nWorkDim** : The dimension of the multi-array to be processed. These data are copied into the specified external data buffer. In general the multi-array is also one-dimensional and nWorkDim:=0, but the multi-array can also have additional dimensions, which may not then be copied, however.
- **nElements** : Specifies the number of elements to be copied from the multi-array. Set nElements=0 to copy everything. If you are only interested in a certain bandwidth of your result, however, then it is not necessary to copy the entire data quantity. This also reduces the necessary size of your specified external data buffer.
- **pStartIndex** : This is an optional parameter that is useful if the multi-array has more than one dimension or if not all elements are to be copied. Specifies the index of the first element that is to be copied from the multi-array. If assigned, it must point to a one-dimensional array of UDINT that has as many elements as the multi-array has dimensions.
- **bNewResult** : This output is TRUE each time a new result has been successfully written into the data buffer.
- **bError** : This output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, then a descriptive error code of the type HRESULT is displayed. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Tip: If a timeout occurs, the input data are not lost. They are processed on the next call.

Output2D() :

Writes data from a multi-array into an external two-dimensional data buffer.
If the method is returned with neither an indication of a new result nor with an error, then the object waits for input data. This is a regular behavior in the analysis chain.

```
METHOD Output2D : HRESULT
VAR_INPUT
    pDataOut      : POINTER TO BYTE;  // address of data buffer
    nDataOutSize  : UDINT;            // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDim0     : UDINT:=0;         // It designates the first dimension in the MultiArray being p
rocessed.
    nWorkDim1     : UDINT:=1;         // It designates the second dimension in the MultiArray being
processed.
    nElementsDim0 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim0 of the MultiArray.
    nElementsDim1 : UDINT:=0;         // optional: default:0-
```

```
>full copy; It designates the number of elements to be copied out of WorkDim1 of the MultiArray.
    pStartIndex   : POINTER TO UDINT; (* optional: default:0->internally handled as [0,0,..];
                                        It designates the index of the first element to be copied out
of the MultiArray.
                                        If allocated it must point to a onedimensional array of UDINT
with so many elements as dimensions of the MultiArray. *)
    nOptionPars   : DWORD;            // option mask END_VARVAR_OUTPUT
    bNewResult    : BOOL;             // TRUE every time when data was written from MultiArray to dat
a buffer.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;          // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **pDataOut** : Address of the external two-dimensional data buffer.
- **eElementType** : This input is of the type E_MA_ElementTypeCode [▸ 166]. The element type of the specified multi-array buffer must correlate to the element type of the specified external data buffer.
- **nWorkDim0** : Specifies the first dimension of the multi-array to be processed. These data are copied into the first dimension of the specified external data buffer. In general the multi-array is also two-dimensional and nWorkDim0:=0, but the multi-array can also have additional dimensions, which may not then be copied, however.
  E.g.: If the second dimension is to be copied into the first index of the target data buffer, then set nWorkDim0:=1.
- **nWorkDim1** : Specifies the second dimension of the multi-array to be processed. These data are copied into the second dimension of the specified external data buffer. In general the multi-array is also two-dimensional and nWorkDim1:=1, but the multi-array can also have additional dimensions, which may not then be copied, however.
  E.g.: If the first dimension is to be copied into the second index of the target data buffer, then set nWorkDim1:=0.
- **nElementsDim0** : Specifies the number of elements to be copied from nWorkDim0 of the multi-array. Set nElementsDim0=0 to copy everything. If you are only interested in a certain bandwidth of your result, however, then it is not necessary to copy the entire data quantity. This also reduces the necessary size of your specified external data buffer.
- **nElementsDim1** : Specifies the number of elements to be copied from nWorkDim0 of the multi-array. Set nElementsDim1=0 to copy everything. If you are only interested in a certain bandwidth of your result, however, then it is not necessary to copy the entire data quantity. This also reduces the necessary size of your specified external data buffer.
- **pStartIndex** : This is an optional parameter that is useful if the multi-array has more than two dimensions or if not all elements are to be copied. Specifies the index of the first element that is to be copied from the multi-array. If assigned, it must point to a one-dimensional array of UDINT that has as many elements as the multi-array has dimensions.
- **bNewResult** : This output is TRUE each time a new result has been successfully written into the data buffer.
- **bError** : This output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, then a descriptive error code of the type HRESULT is displayed. Possible values are described in the List of error codes [▸ 224]. This output is identical to the return value of the method.
  Tip: If a timeout occurs, the input data are not lost. They are processed on the next call.

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    nOwnID: UDINT; // ID for this FB instance
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.27 FB_CMA_Source

This function block writes data from an external PLC data buffer into a multi-array buffer.

It accumulates input data continuously, until the maximum size of the multi-array is reached. Once the multi-array is full, it is transferred to the target analysis ID.
An instance of FB_CMA_Source must not be used as target for any other analysis block. It offers only source functionality.

A time series collection can be interrupted in the event of an error. Lost signal data can lead to an unexpected result of the analysis chain, depending on the configuration of the algorithms.

Inputs and outputs

The output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
output stream	eTypeCode	`nDims`	`aDimSizes`

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars        : ST_MA_MultiArray_InitPars;  // init parameter
    nOwnID            : UDINT;                      // ID for this FB instance
    aDestIDs          : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers    : UDINT := 4;                 // number of MultiArrays which should be initiali
zed for results (0 for no initialization)
```

```
    tTransferTimeout : LTIME := LTIME#40US;        // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 184]. Multi-array buffers are specified for the result buffers. These parameters must correlate to the above definition of the output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Synchronous timeout setting for internal multi-array transfer commands. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError        : BOOL;    // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT; // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;   // counts outgoing results (MultiArrays were calculated and sent to tra
nsfer tray)
END_VAR
```

- **bError** : This output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Input1D() :

Writes data from an external one-dimensional data buffer into a multi-array.
This method must be called whenever new input samples are available, usually cyclically.

```
METHOD Input1D : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDim      : UDINT;                // It designates the dimension in the multi array being pr
ocessed.
    pStartIndex    : POINTER TO UDINT;    (* optional: default:0-
>internally handled; It designates the index of the first MultiArray element to be copied.
                         If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray.
                         Upon successful completion of the copy, corresponding StartIndex is increme
nted by the number of copied elements. *)
    nOptionPars    : DWORD;               // option mask (cCMA_Option_MarkInterruption, ...)
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;                // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError         : BOOL;                // TRUE if an error occurs.
    hrErrorCode    : HRESULT;             // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **eElementType** : This input is of the type E_MA_ElementTypeCode [▶ 166]. The element type of the specified multi-array buffer (initialization parameters) must match the element type of the specified external data buffer.
- **nWorkDim** : Defines the dimension in which the data are accumulated. In general the multi-array is also one-dimensional and nWorkDim:=0, but the multi-array can also have additional dimensions, which may not then be processed, however.

- **pStartIndex** : This is an optional parameter, which can be useful if the multi-array has more than one dimension. Specifies the index of the first multi-array element to be copied. If assigned, it must point to a one-dimensional array of UDINT that has as many elements as the multi-array has dimensions. After a successful copy process, the corresponding StartIndex is incremented by the number of copied elements.
- **nOptionMask:** Available options:
    - cCMA_Option_MarkInterruption : Several errors can occur and cause interruptions of the time series collection. If the flag is set and the element type is LREAL, the first data buffer element is marked as invalid (NaN). This can be used to detect an interruption in the result data sets, because it is not possible to calculate correct spectra based on fragmented time series. See separate section for further information on NaN values [▶ 65].
- **bError** : This output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, then a descriptive error code of the type HRESULT is displayed. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.

Input2D() :

Writes data from an external two-dimensional data buffer into a multi-array.
This method must be called whenever new input samples are available, usually cyclically.

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data)
 as twodimensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDim0     : UDINT:=0;             // It designates the first dimension in the MultiA
rray being processed. (e.g. 1..channels)
    nWorkDim1     : UDINT:=1;             // It designates the second dimension in the Multi
Array being processed.
    pStartIndex   : POINTER TO UDINT;     (* optional: default:0->
internally handled; It designates the index of the first MultiArray element to be copied.
                         If allocated it must point to a onedimensional array of UDINT with
so many elements as dimensions of the MultiArray.
                         Upon successful completion of the copy, corresponding StartIndex is
 incremented by the number of copied elements. *)
    nOptionPars   : DWORD;                 // option mask (cCMA_Option_MarkInterruption, ...
)
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                  // TRUE every time when outgoing MultiArray was c
alculated and sent to transfer tray.
    bError     : BOOL;                     // TRUE if an error occurs.
    hrErrorCode   : HRESULT;               // '< 0' = error; '> 0' = info; '0' = no error/
info
END_VAR
```

- **pDataIn** : The data buffer must contain the data from all channels.
- **eElementType** : This input is of the type E_MA_ElementTypeCode [▶ 166]. The element type of the specified multi-array buffer (initialization parameters) must match the element type of the specified external data buffer.
- **nWorkDim0** : Defines the dimension that matches the number of channels. In general the multi-array is also two-dimensional and nWorkDim0:=0, but the multi-array can also have additional dimensions, which may not then be processed, however.
  E.g.: If the first index of the specified data buffer stands for the channels, while the second dimension of the multi-array counts the channels, then set nWorkDim0:=1.
- **nWorkDim1** : Defines the dimension in which the data are accumulated. In general the multi-array is also two-dimensional and nWorkDim1:=1, but the multi-array can also have additional dimensions, which may not then be processed, however.
  E.g.: If the second index of the specified data buffer stands for the accumulated data, while the first dimension of the multi-array collects the data, then set nWorkDim1:=0.

- **pStartIndex** : This is an optional parameter, which can be useful if the multi-array has more than two dimensions. Specifies the index of the first multi-array element to be copied. If assigned, it must point to a one-dimensional array of UDINT that has as many elements as the multi-array has dimensions. After a successful copy process, the corresponding StartIndex is incremented by the number of copied elements.
- **nOptionMask:** Available options:
  - cCMA_Option_MarkInterruption : Several errors can occur and cause interruptions of the time series collection. If the flag is set and the element type is LREAL, the first data buffer element (of each channel) is marked as invalid (NaN). This can be used to detect an interruption in the result data sets, because it is not possible to calculate correct spectra based on fragmented time series. See separate section for further information on NaN values [▶ 65].
- **bError** : This output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, then a descriptive error code of the type HRESULT is displayed. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.

ResetData() :

The method deletes all records that were already added, in order to make space for the current output buffer (multi-array).
To use external indices (`pStartIndex` parameter) for the filling procedure, these have to be reset explicitly.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute ‘call_after_init’ (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_MA_MultiArray_InitPars;       // init parameter
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 184]. Multi-array buffers are specified for the result buffers. These parameters must correlate to the above definition of the output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.28 FB_CMA_WatchUpperThresholds

Configurable threshold value monitoring of multi-channel data

Similar to the FB_CMA_DiscreteClassification [▶ 89] block, this block allocates the individual channels of a multi-channel signal to a number of configurable discrete categories, based on configurable threshold values. After the configuration the block calculates a one-dimensional array with precisely two values for each input vector. The type of both elements is a signed 32-bit Integer number. The first value of the result identifies the number of highest determined category, the second value the channel number with the highest category. In both cases numbering starts with zero. If no input value of a channel matches the respective threshold value for the lowest category, the resulting value is -1. If an input value equals the threshold value of a category, it is counted under this category. If several channels are allocated the highest category, the channel number with the lower number is returned.

Configuration

The block can be configured at runtime by specifying the threshold value for each channel and each threshold value category.

Memory properties

Depending on the block configuration, the number of the highest threshold value category and the number of the triggering channel are saved until the method `ResetData()` is called, or the values are recalculated after each step.

Inputs and outputs

The input and output buffers correspond to the following definition (Shape). The variable parameters are part of the function block input `stInitPars`.

Multi-array in the	Element type	Dimensions	Dimensional variables
input stream	LREAL	1	`nChannels`
output stream	INT (32bit)	1	2

Input parameters

The input parameters of this function block represent initialization parameters and must already be assigned in the declaration of the FB instance! (alternatively: Init() method)
They may only be assigned once. A change at runtime is not possible.

```
VAR_INPUT
    stInitPars      : ST_CM_WatchUpperThresholds_InitPars; // init parameter
    nOwnID          : UDINT;                               // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;     // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                           // number of MultiArrays which should be
 initialized for results (0 for no initialization)
    tTransferTimeout: LTIME := LTIME#500US;                // timeout checking off during access to
inter-task FIFOs
END_VAR
```

- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_WatchUpperThresholds_InitPars [▶ 184]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.

- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers. The default is four.
- **tTransferTimeout** : Setting of the synchronous timeout for internal multi-array forwardings. See section Parallel processing [▶ 66].

Output parameters

```
VAR_OUTPUT
    bError          : BOOL;         // TRUE if an error occurs. Reset by next method call.
    hrErrorCode   : HRESULT;       // '< 0' = error; '> 0' = info; '0' = no error/info
    nCntResults   : ULINT;         // counts outgoing results (MultiArrays were calculated and sent to
 transfer tray)
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Methods

Call() :

The method is called in each cycle in order to calculate threshold value results from the input signal. The function block waits for input data if the method indicates neither new results nor an error. This is a regular behavior in the process of the analysis chain.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult  : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError      : BOOL;         // TRUE if an error occurs.
    hrErrorCode : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- **bError** : The output is TRUE if an error occurs.
- **hrErrorCode** : If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224]. This output is identical to the return value of the method.
  Note: If a timeout occurs or no multi-array buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

Configure() :

The classification arguments must be configured at the beginning with the call of this method. The corresponding PLC array must be defined as follows. The Configure() method can also be used for a new configuration with a different set of arguments.

	Element type	Dimensions	Dimensional variables
Argument	LREAL	2	`nChannels` x `nMaxClasses`

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL; // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize : UDINT;            // size of arguments buffer in bytes
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

ResetData() :

This method deletes all the data sets already added. Alternatively bMemorize=FALSE can be set in the initialization structure for an automatic reset.
Resets the reached maximum category of the function block to -1.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Init() :

This method is not usually necessary in a Condition Monitoring application. It offers an alternative to the function block initialization. The Init() method may only be called during the initialization phase of the PLC. It cannot be used at runtime. You are referred to the use of an FB_init method or the attribute 'call_after_init' (see TwinCAT PLC reference). In addition, this facilitates the function block encapsulation.

The input parameters of the function block instance may not be assigned in the declaration if the initialization is to take place using the Init() method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_WatchUpperThresholds_InitPars; // init parameter
    nOwnID         : UDINT;                             // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].
- **stInitPars** : Function-block-specific structure with initialization parameters of the type ST_CM_WatchUpperThresholds_InitPars [▶ 184]. The parameters must correlate to the above definition of the input and output buffers.
- **nOwnID** : Identifies the function block instance with a unique ID. This must always be greater than zero. A proven approach is to define an enumeration for this purpose.
- **aDestIDs** : Defines the destinations to which the results are to be forwarded by specifying the IDs of the destinations. The definition of the output buffer (as described above) must correlate to the definition of the input buffer of each selected destination.
- **nResultBuffers** : The function block initializes a Transfer Tray Stream with the specified number of multi-array buffers.

PassInputs() :

As long as an FB_CMA_Source instance is called and thus signal data are transferred to a target block, all further function blocks of the analysis chain have to be called cyclically, as explained in the API PLC reference [▶ 73].
Sometimes it is useful not to be execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. Although the function block still has to be called cyclically, it is sufficient that the data arriving at the function block are relayed in the communication ring [▶ 66]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called, and accordingly no result is calculated and no output buffer is generated.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the List of error codes [▶ 224].

Similar function blocks

The block FB_CMA_DiscreteClassification [▶ 89] classifies multi-channel input data.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.2 Functions

5.2.1 F_MA_IsNAN

This function tests for presence of a NaN (Not-a-Number) value, returning TRUE when the value is a NaN.

```
FUNCTION F_MA_IsNAN : BOOL
VAR_INPUT
     fValue : LREAL;
END_VAR
```

For further information see chapter Handling with NaN values [▶ 65].

The function is obsolete.
Please use the LrealIsNaN() function instead (Tc2_Utilities library).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.3 Data types

5.3.1 E_CM_MCoefOrder

```
TYPE E_CM_MCoefOrder :
(
    eCM_N        := 0,        (* count of included cases *)
    eCM_Mean     := 1,        (* mean value *)
    eCM_StDev    := 2,        (* standard deviation *)
    eCM_Skew     := 3,        (* skew value (third moment) *)
    eCM_Kurtosis := 4         (* Excess Kurtosis value *)
) UDINT;
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.2 E_CM_ScalingType

For deeper details to the scaling options see "Spectrum Scaling Options [▶ 227]".

```
TYPE E_CM_ScalingType :
(
     eCM_NoScaling             := 0,
     eCM_DiracScaling          := 1,
     eCM_PeakAmplitude         := 2,
     eCM_ROOT_POWER_SUM        := 3,
     eCM_RMS                   := 4,
     eCM_GainCorrection        := 5,
     eCM_PowerSpectralDensity := 6,
     eCM_UnitaryScaling       := 7
) UDINT;
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.3 E_CM_WindowType

For deeper details to the different window types see "".

```
TYPE E_CM_WindowType :
(
     eCM_HannWindow               := 16#05300901,
     eCM_RectangularWindow        := 16#05300902
) UDINT;
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.4 E_MA_ElementTypeCode

```
TYPE E_MA_ElementTypeCode :
(
     eMA_TypeCode_NONE        := 16#0000,    (*   [ internally used ] *)
     eMA_TypeCode_STRING      := 16#0001,    (*   [ internally used: length byte followed by ANSI C st
ring ] *)
     eMA_TypeCode_BYTE        := 16#0002,
     eMA_TypeCode_CHAR        := 16#0003,
     eMA_TypeCode_WCHAR       := 16#0004,
     eMA_TypeCode_BOOL        := 16#0005,    (* boolean type *)
     eMA_TypeCode_INT16       := 16#0006,
     eMA_TypeCode_UINT16      := 16#0007,
     eMA_TypeCode_INT32       := 16#0008,    (* used e.g. for classification results *)
     eMA_TypeCode_UINT32      := 16#0009,
     eMA_TypeCode_INT64       := 16#000A,
     eMA_TypeCode_UINT64      := 16#000B,    (* 64-bit long unsigned. use
this for statistical counters *)
     eMA_TypeCode_REAL        := 16#000C,    (*   [Unsupported: 32-bit  floating point type. ]*)
     eMA_TypeCode_LREAL       := 16#000D,    (* Standard floating-point type. *)
     eMA_TypeCode_COMPLEX     := 16#000E,    (*   [ Unsupported: 64-bit
complex type. Use LCOMPLEX. ] *)
     eMA_TypeCode_LCOMPLEX    := 16#000F,    (* Standard 128-bit complex  type (real part first) *)
     eMA_TypeCode_LQUATERNION := 16#0010,    (* 256-bit quaternion
 values, for representation of spatial rotations. *)
     eMA_TypeCode_PUNKNOWN     := 16#0011,    (*   [ internally used: pointer to object implementing I
TcUnknown. ] *)
     eMA_TypeCode_MFPOINT      := 16#0012,    (*   [ internally used: type for motion control. ] *)
     eMA_TypeCode_OTHER        := 16#0013
) UDINT;
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.3.5 Error codes

5.3.5.1 E_CM_ErrorCode

code (HRESULT)	symbol	description / solution
0	eCM_OK	No Error, everything is OK
Top category		
16#9851_FFFF	eCM_LibraryError_MASK	CM Library Bitmask
logic errors		
16#9851_0100	eCM_ErrLogic	general logic error
16#9851_01FF	eCM_ErrLogic_MASK	logic error category bitmask
16#9851_0101	eCM_ErrLogic_AssertionFailed	internal assumptions are violated
16#9851_0102	eCM_ErrLogic_NotImplemented	function is not yet implemented
16#9851_0110	eCM_ErrLogic_LackOfInitialization	algorithm was not initialized correctly
16#9851_0111	eCM_ErrLogic_LackOfInitialization_WindowFunction	window function was not initialized correctly
16#9851_0121	eCM_ErrLogic_MissingLicense	no valid license key was found
16#9851_0122	eCM_ErrLogic_InvalidHandle	an invalid object ID was passed
16#9851_0123	eCM_ErrLogic_NullHandle	a null handle was passed
16#9851_0124	eCM_ErrLogic_InvalidHandleType	an invalid type of handle was passed
16#9851_0125	eCM_ErrLogic_InvalidObjectState	the operation is invalid for object state
16#9851_0126	eCM_ErrLogic_NoHandlesLeft	it was not possible to assign a new object ID
16#9851_0127	eCM_ErrRTime_InstanceExists	there is already an allocated instance
configuration errors		
16#9851_1000	eCM_ErrConfig	general configuration error
16#9851_1FFF	eCM_ErrConfig_MASK	general configuration error category bitmask
maps to ADSERR_DEVICE_NOMEMORY		
16#9851_1100	eCM_ErrConfig_OutOfMemory	memory allocation failed => increase router memory (see chapter Memory management [▶ 62])
all errors below result in a HRESULT of ADS_E_INVALIDPARM		
16#9851_1800	eCM_ErrConfig_IllegalParameter	configuration parameter is not valid
16#9851_1900	eCM_ErrConfig_ParameterOutOfRange	configuration parameter is out of range
16#9851_1901	eCM_ErrConfig_ParameterOutOfRange_NoPowerOfTwo	parameter is not power of two as required
16#9851_1902	eCM_ErrConfig_ParameterOutOfRange_FFT_length_Zero	the FFT length is zero or smaller and needs to be positive
16#9851_1903	eCM_ErrConfig_ParameterOutOfRange_DecibelThreshold_too_small	fDecibelThreshold is too small), which would cause underflow
16#9851_1904	eCM_ErrConfig_ParameterOutOfRange_LogThreshold_too_small	fLogThreshold is too small), which would cause underflow
16#9851_1905	eCM_ErrConfig_ParameterOutOfRange_nInLength_Minimum_two	nInLength value must be at least two
16#9851_1906	eCM_ErrConfig_ParameterOutOfRange_nInLength_NotEven	nInLength is not an even number
16#9851_1907	eCM_ErrConfig_ParameterOutOfRange_nFrameShift_not_positive	nFrameShift is not at least one
16#9851_1908	eCM_ErrConfig_ParameterOutOfRange_nOutWindowLength_not_even	nOutWindowLength is not even
16#9851_1909	eCM_ErrConfig_ParameterOutOfRange_Unsuitable_WindowFunction	window function not suitable for reconstruction

code (HRESULT)	symbol	description / solution
16#9851_190A	eCM_ErrConfig_ParameterOutOfRange_FFT_length_less_two	FFT length is less than two which is needed here
16#9851_190B	eCM_ErrConfig_ParameterOutOfRange_WindowLength_odd	Window length is not an even number
16#9851_190C	eCM_ErrConfig_ParameterOutOfRange_FFT_length_odd	FFT length is not an even number
16#9851_190D	eCM_ErrConfig_ParameterOutOfRange_nChannels_smaller_one	nChannels is smaller than one
16#9851_190E	eCM_ErrConfig_ParameterOutOfRange_nBins_smaller_one	nBins is smaller than one
16#9851_190F	eCM_ErrConfig_ParameterOutOfRange_invalid_limit_interval	lower limit is not smaller than upper limit
16#9851_1910	eCM_ErrConfig_ParameterOutOfRange_unknown_scaling_type	scaling type is not known
16#9851_1911	eCM_ErrConfig_ParameterOutOfRange_illegal_quantile_argument	quantile argument outside [0 .. 1]
16#9851_1912	eCM_ErrConfig_ParameterOutOfRange_illegal_threshold_order	thresholds must be in ascending order
16#9851_1913	eCM_ErrConfig_ParameterOutOfRange_threshold_number_toolarge	more threshold values than configured
16#9851_1914	eCM_ErrConfig_ParameterOutOfRange_Integration_limit_too_low	integration limit is too low
16#9851_1915	eCM_ErrConfig_ParameterOutOfRange_Integration_limit_too_high	integration limit is too high
16#9851_1916	eCM_ErrConfig_ParameterOutOfRange_Integration_limits_inconsistent	integration limits are inconsistent
16#9851_1917	eCM_ErrConfig_ParameterOutOfRange_Samplerate_not_positive	sample rate is zero or negative
16#9851_192B	eCM_ErrConfig_ParameterOutOfRange_Nonascending_Sequence	sequence is nonascending
16#9851_1A10	eCM_ErrConfig_IllegalParamType	configuration parameter has wrong type
16#9851_1A20	eCM_ErrConfig_ParameterNameNotFound	parameter id was not found
16#9851_1B00	eCM_ErrConfig_ParameterMismatch	parameter dependency not met
16#9851_1B01	eCM_ErrConfig_ParameterMismatch_WindowLength_larger_FFT_length	window length larger than FFT length
16#9851_1B02	eCM_ErrConfig_ParameterMismatch_LengthDifference_odd	difference between window length and FFT length not even
16#9851_1B03	eCM_ErrConfig_ParameterMismatch_nFrameShift_larger_nInLength	nFrameShift is larger than nInLength
16#9851_1B04	eCM_ErrConfig_ParameterMismatch_nOutWindowLength_larger_nInLength	nOutWindowLength is larger than nInLength
16#9851_1B05	eCM_ErrConfig_ParameterMismatch_LogThreshold_too_small	fLogThreshold is too small), which would cause underflow
runtime errors (while data processing)		
these errors result in a HRESULT of ADS_E_INVALIDPARM		
16#9851_2000	eCM_ErrRTime	general runtime error
16#9851_2FFF	eCM_ErrRTime_MASK	general runtime error category bitmask
16#9851_2020	eCM_ErrRTime_WrongFunctionSignature	algorithm called with wrong signature
16#9851_203F	eCM_ErrRTime_WrongFunctionSignature_MASK	signature error category bitmask
16#9851_2021	eCM_ErrRTime_IllegalBuffer	illegal data buffer

code (HRESULT)	symbol	description / solution
16#9851_2022	eCM_ErrRTime_IllegalSubarraySize	illegal size of subarray
16#9851_2023	eCM_ErrRTime_IllegalInput	illegal input signature
16#9851_2024	eCM_ErrRTime_IllegalInputArgnum	input data has illegal argument number
16#9851_2025	eCM_ErrRTime_IllegalInputDimensionNumber	input data has illegal number of dimensions
16#9851_2026	eCM_ErrRTime_IllegalInputShape	input data has illegal shape
16#9851_2027	eCM_ErrRTime_IllegalInputValue	illegal value in input data stream
16#9851_2028	eCM_ErrRTime_IllegalInputDataType	illegal element type of input data stream
16#9851_2029	eCM_ErrRTime_IllegalInputShapeZero	input data has illegal shape of Zero
16#9851_202A	eCM_ErrRTime_IllegalInputCombineParameterMismatch	parameters of objects do not match
16#9851_202B	eCM_ErrRTime_IllegalInputNoArray	no multiarray passed as input parameter
illegal output buffer parameters (can occur in fixed-buffer ADS calls)		
16#9851_2030	eCM_ErrRTime_IllegalOutput	general invalid output buffer parameters
16#9851_2031	eCM_ErrRTime_IllegalOutputArgnum	output data has illegal argument number
16#9851_2032	eCM_ErrRTime_IllegalOutputDimensionNumber	output buffer has illegal number of dimensions
16#9851_2033	eCM_ErrRTime_IllegalOutputShape	output buffer has illegal shape
16#9851_2034	eCM_ErrRTime_IllegalOutputDataType	illegal element type of output data buffer
16#9851_2035	eCM_ErrRTime_IllegalOutputNoArray	no multiarray passed as output parameter
16#9851_2036	CM_ErrRTime_IllegalOutputNoData	Multiarray has no data (product of dimension sizes is zero)
16#9851_2040	eCM_ErrRTime_FloatPoint	general floating point error during computation
16#9851_204F	eCM_ErrRTime_FloatPoint_MASK	general floating point error category bitmask
16#9851_2041	eCM_ErrRTime_FloatPointDivisionByZero	division by zero attempted
16#9851_2042	eCM_ErrRTime_FloatPointOverflow	overflow in computation
16#9851_2043	eCM_ErrRTime_FloatPointUnderflow	arithmetic underflow
16#9851_2044	eCM_ErrRTime_FloatPointOutOfRealDomain	result is no real number
16#9851_2045	eCM_ErrRTime_FloatPointLogarithmOfZero	logarithm of zero attempted
16#9851_2050	eCM_ErrRTime_NumericNoUsefulResult	cannot compute useful result now (e.g.), variance from single sample)
16#9851_2051	eCM_ErrRTime_NumericInsufficientData	amount of valid data is insufficient for evaluation
16#9851_2052	eCM_ErrRTime_MissingTraining	a classification method was invoked without prior training
16#9851_2053	eCM_ErrRTime_RecursionDepthExceeded	the maximum recursion depth of an algorithm was exceeded
16#9851_2054	eCM_ErrRTime_OutOfRouterMemory	amount of available router memory was exceeded
errors in interaction with periphery		
16#9851_2360	eCM_ErrRTime_OSError	error in general OS call
16#9851_236F	eCM_ErrRTime_OSError_MASK	OS call error bitmask
note: codes corresponding to file operations are placeholders currently not used		
16#9851_2370	eCM_ErrRTime_IOError	I/O operation failed (e.g. file operation)
16#9851_237F	eCM_ErrRTime_IOError_MASK	I/O error category bitmask

code (HRESULT)	symbol	description / solution
16#9851_2371	eCM_ErrRTime_IOError_FileOpenFailed	failure on attempt to open file
16#9851_2372	eCM_ErrRTime_IOError_FileReadFailed	failure on attempt to read from filee
16#9851_2373	eCM_ErrRTime_IOError_FileWriteFailed	failure on attempt to write to file
16#9851_2374	eCM_ErrRTime_IOError_FileCloseFailed	failure on attempt to close file
16#9851_2375	eCM_ErrRTime_IOError_FileBadState	file is in bad state for operation
ADS service errors		
16#9851_2400	eCM_ErrRTime_CMService	Error in communication with CM service
16#9851_24FF	eCM_ErrRTime_CMService_MASK	CM server error category bitmask
user errors (probably due to looped initialization)		
16#9851_2410	eCM_ErrRTime_CMServiceUser	CM service usage errors
16#9851_241F	eCM_ErrRTime_CMServiceUser_MASK	CM service usage error category bitmask
16#9851_2411	eCM_ErrRTime_CMServiceUser_ResourcesExhausted	the ADS service has insufficient resources
16#9851_2412	eCM_ErrRTime_CMServiceUser_NoHandlesLeft	the maximum number of handles is exhausted
16#9851_2413	eCM_ErrRTime_CMServiceUser_UnknownHandle	no instance with this handle is currently registered
16#9851_2414	eCM_ErrRTime_CMServiceUser_HandleAlreadyFreed	illegal request: this handle was already freed
16#9851_2415	eCM_ErrRTime_CMServiceUser_InstanceExists	illegal request: this handle is already used
protocol errors (probably from mismatch of client/server software)		
16#9851_2420	eCM_ErrRTime_CMServiceProtocol	protocol error (internal error in communication)
16#9851_242F	eCM_ErrRTime_CMServiceProtocol_MASK	protocol error category bitmask
16#9851_2421	eCM_ErrRTime_CMServiceProtocol_UnknownFunction	logic error: no function with this class id is known
16#9851_2422	eCM_ErrRTime_CMServiceProtocol_IllegalRequest	illegal request to service
16#9851_2423	eCM_ErrRTime_CMServiceProtocol_IllegalRequestSyntax	illegal syntax in request
16#9851_2424	eCM_ErrRTime_CMServiceProtocol_IllegalHandle	the handle number is inconsistent by internal checks
16#9851_2425	eCM_ErrRTime_CMServiceProtocol_InvalidNullpointer	a NULL pointer was passed to store return values
16#9851_2426	eCM_ErrRTime_CMServiceProtocol_UnknownParameterID	the used parameter ID is unknown
16#9851_2427	eCM_ErrRTime_CMServiceProtocol_IllegalValType	type of passed value does not match
16#9851_2428	eCM_ErrRTime_CMServiceProtocol_ClientServerMismatch	protocol error due to client / server mismatch
16#9851_2429	eCM_ErrRTime_CMServiceProtocol_ResultBufferTooSmall	ADS read buffer is too small to pass return data correctly
16#9851_242A	eCM_ErrRTime_CMServiceProtocol_InputBufferTooSmall	ADS write buffer is too small to pass argument data correctly
16#9851_242B	eCM_ErrRTime_CMServiceProtocol_UnknownMethod	logic error: no action with this code is known

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.5.2 E_CMA_ErrorCode

These error codes are necessary in the realtime context only. Please, note that the analysis function blocks must be allocated in the PLC declaration part correctly.
The configuration errors should be solved first, followed by the initialization errors.
For example: If any instances throw the error eCMA_ErrConfig_InvalidOwnID this has to be solved first. Runtime errors at other function blocks can be subsequent errors.

code (HRESULT)	symbol	description / solution
0	eCMA_OK	No Error, everything is OK
configuration errors		
16#9852_0101	eCMA_ErrConfig_InvalidOwnID	invalid transfer own ID was allocated
16#9852_0102	eCMA_ErrConfig_InvalidDestID	invalid transfer destination IDs were allocated
16#9852_0103	eCMA_ErrConfig_InvalidBufferNumber	invalid number of MultiArrays which should be initialized for results
16#9852_0104	eCMA_ErrConfig_InvalidTimeout	invalid timeout. condition: 0us << tTransferTimeout << task cycle time
initialization errors		
16#9852_0201	eCMA_ErrInit_IllegalInitContext	initialization not possible. Illegal initialization context or internal members uninitialized.
16#9852_0202	eCMA_ErrInit_InitTransferTrayFailed	Initialization of transfer tray has been failed. Check TcCOM object states and router memory (see Memory management [▶ 62]). Check installed TwinCAT version (see System requirements [▶ 53]).
16#9852_0203	eCMA_ErrInit_NoStreamAllocated	The analysis chain is incorrect. Check all OwnIDs and DestIDs.
16#9852_0204	eCMA_ErrInit_StreamOverrun	Not enough streams available. Adjust ST_CM_TransferTray_InitPars
runtime errors		
16#9852_0301	eCMA_ErrRTime_InvalidPointer	NULL pointer was allocated
16#9852_0302	eCMA_ErrRTime_InvalidDataBufferSize	invalid size of data buffer was allocated
16#9852_0303	eCMA_ErrRTime_InvalidElementType	invalid element type was allocated
16#9852_0304	eCMA_ErrRTime_InvalidElementCnt	element count does not match. (check number of elements, MultiArray buffer size and start index)
16#9852_0305	eCMA_ErrRTime_InvalidStartIndex	invalid pStartIndex was allocated (check buffer sizes)
16#9852_0311	eCMA_ErrRTime_MissingConfiguration	Argument not configured. Call method Configure() first.
16#9852_0321	eCMA_ErrRTime_NoMultiArrayAvailable	no multiarray available for result. Check analysis chain, task cycle times and the number of MultiArrays (usually at least 3 in each ring)

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM

5.3.5.3 E_MA_ErrorCode

code (HRESULT)	symbol	description / solution
0	eMA_OK	No Error, everything is OK
16#9871_0100	eMA_ErrLogic	general logic error
16#9871_0110	eMA_ErrLogic_LackOfInitialization	algorithm was not initialized correctly
16#9871_1000	eMA_ErrConfig	general configuration error
16#9871_1100	eMA_ErrConfig_OutOfMemory	memory allocation failed => increase router memory (see chapter Memory management [▶ 62]).
16#9871_1800	eMA_ErrConfig_IllegalParameter	configuration parameter is not valid
16#9871_2000	eMA_ErrRTime	general runtime error
16#9871_2011	eMA_ErrRTime_IllegalPointer	illegal interface pointer or memory address
16#9871_2012	eMA_ErrRTime_EmptyArray	multiarray has no data (product of dimension sizes is zero)
16#9871_2021	eMA_ErrRTime_IllegalBuffer	illegal data buffer
16#9871_2022	eMA_ErrRTime_IllegalSubarraySize	illegal size of subarray
16#9871_2023	eMA_ErrRTime_IllegalInput	illegal input signature
16#9871_2024	eMA_ErrRTime_IllegalInputArgnum	input data has illegal argument number
16#9871_2025	eMA_ErrRTime_IllegalInputDimensionNumber	input data has illegal number of dimensions
16#9871_2026	eMA_ErrRTime_IllegalInputShape	input data has illegal shape
16#9871_2027	eMA_ErrRTime_IllegalInputValue	illegal value in input data stream
16#9871_2028	eMA_ErrRTime_IllegalInputDataType	illegal element type of input data stream

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.3.6 InitPars structures

5.3.6.1 ST_CM_AnalyticSignal_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_AnalyticSignal_InitPars EXTENDS ST_MA_InitPars :
STRUCT
    nFFT_Length            : UDINT := 512;                         (* length of FFT *)
    nWindowLength          : UDINT := 400;                         (* length of analysis window *)
    eWindowType            : E_CM_WindowType := eCM_HannWindow;    (* window type of window analys
is *)
    eOverlapWindowType     : E_CM_WindowType := eCM_HannWindow;    (* window type for output synth
esis *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT of the envelopes. It must be greater than one and an integral power of two.
- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.
- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 166]). A good default value is the window type `eCM_HannWindow`.
- `eOverlapWindowType` defines the window function used for the inverse transformation. It is freely selectable. A good default value is the window type `eCM_HannWindow`. Windowing can switched off by using the type `eCM_RectangularWindow`.

Avoiding artefacts

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. In order to avoid artefacts in the calculation, `nFFT_Length` should be at least 30 to 50% larger than nWindowLength. An increase in the FFT length in relation to the window length makes sense with this function block in order to avoid circular aliasing.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.2 ST_CM_ArgSort_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_ArgSort_InitPars EXTENDS ST_MA_InitPars :
STRUCT
    nInLength        : UDINT := 256;    (* length of input data array *)
    bSortDownward    : BOOL  := FALSE;  (* if true, sort in descending order (largest values first)*)
    bShiftNaNsToEnd  : BOOL  := TRUE;   (* if true, sorts correctly even if NaN values occur, shiftin
g these to the end *)
    fScaleFactor     : LREAL := 1.0;    (*
scaling factor to transform index values, for example to frequency values *)
END_STRUCT
END_TYPE
```

- `nInLength` is the length of the input array.
- `bSortDownward` is a flag with which you can select whether the data are to be sorted in ascending or descending order. If `bSortDownward` is TRUE, then the largest values are placed at the front.
- `bShiftNaNsToEnd` can be set to TRUE in order to sort possible NaN values to the end.
- `fScaleFactor` can be used in order to directly display, for example, the amplitude with associated frequencies instead of the index position (fScaleFactor = 1).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.3 ST_CM_CrestFactor_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_CrestFactor_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels          : UDINT := 32;          (* number of channels *)
    nBufferLength      : UDINT := 250;         (* number of time values *)
    fDecibelThreshold  : LREAL := cCM_MinArgLog10;     (* lower limit for logarithm argument *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must always be bigger than zero.
- `nBufferLength` is the number of values of the input vector. This must always be an integer bigger than zero.
- `fDecibelThreshold` is a very small floating point value bigger than zero. Values that are less than this number are replaced with this value before the transformation to the decibel scale is done. (The purpose is the avoidance of value range errors. The logarithm of zero is not defined and strives infinitely towards minus for the limit value of small arguments. The same applies to the argument of the number zero, arg(0). The smallest possible value is 3.75e-324, which is equivalent to the constant `cCM_MinArgLog10`.)

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.4 ST_CM_ComplexFFT_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_ComplexFFT_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length        : UDINT := 256;       (* length of FFT *)
    bForward           : BOOL := TRUE;
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT. It must be greater than one and an integral power of two.
- `bForward` is a Boolean parameter indicating the direction of the FFT. If the value is "true", the normal FFT is calculated. Otherwise the inverse FFT is used.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.5 ST_CM_DiscreteClassification_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_DiscreteClassification_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels      : UDINT :=  10;      (* number of input channels *)
    nMaxClasses    : UDINT :=   3;      (* number of configurable classes *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must always be bigger than zero.
- `nMaxClasses` is the maximum number of configured classes. This must always be bigger than zero.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.6 ST_CM_EmpiricalMoments_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_EmpiricalMoments_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels              : UDINT :=  512;    (* number of input channels *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must be greater than zero.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022	PC or CX (x86, x64)	Tc3_CM_Base >= v1.1.10

5.3.6.7 ST_CM_Envelope_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_Envelope_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;           (* length of FFT *)
    nWindowLength       : UDINT := 400;           (* length of analysis window *)
    eWindowType         : E_CM_WindowType := eCM_HannWindow;  (* window type of window analysis *)
    eOverlapWindowType   : E_CM_WindowType := eCM_HannWindow; (* window type for output synthesis *)
END_STRUCT
END_TYPE
```

- `nFFT_Length`has to be bigger than one. It has to be a power of two.
- `nWindowLength` is the length of the analysis window in samples. The length must be bigger than one and an even number.
- `eWindowType` defines the used window function. Refer to type definition E_CM_WindowType. Because the signal will be transformed backwards intothe time domain and the sum of the overlapping window factors needs to be one at every time, only these window functions can be used:
  - Hann-Window (`eCM_HannWindow`),
  - Bartlett-Window (`eCM_BartlettWindow`) und
  - Bartlett-Hann-Window (`eCM_BartlettHannWindow`)

    In most cases the eCM_HannWindow window type is a meaningful default value.
- `eOverlapWindowType` defines the used window function for the inverse transformation (of type E_CM_WindowType). Good standard is the eCM_HannWindow. The windowing can be switched off by using the `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions.

Avoid artefacts

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. In order to avoid artefacts during the calculation, `nFFT_Length` should be at least 30% to 50% greater than nWindowLength. An increase in the FFT length in relation to the window length makes sense with this block in order to avoid circular aliasing.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.8 ST_CM_EnvelopeSpectrum_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_EnvelopeSpectrum_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_length_Envelope   : UDINT := 512;     (* length of Envelope FFT *)
    nFFT_length_Spectrum   : UDINT := 512;     (* length of Spectrum FFT *)
    nWindowLength          : UDINT := 400;     (* length of analysis window *)
    bTransformToDecibel    : BOOL  := TRUE;
    fDecibelThreshold      : LREAL := cCM_MinArgLog10;
    eWindowType            : E_CM_WindowType := eCM_HannWindow;   (* window type of window analysis *)
    eOverlapWindowType     : E_CM_WindowType := eCM_HannWindow;   (* window type for output synthesis
*)
    eScalingType           : E_CM_ScalingType := eCM_DiracScaling;
END_STRUCT
END_TYPE
```

- `nFFT_length_Envelope` has to be bigger than one. It has to be a power of two.
- `nFFT_length_Spectrum` has to be bigger than one. It has to be a power of two.
- `nWindowLength` is the length of the analysis window in samples. The length has to be bigger than one and an even number.
- `bTransformToDecibel` is a boolean value to specify if the FFT result should be transformed to decibel scale using the transformation x → 20 * log10(x).
- `fDecibelThreshold` is a very small floating point value bigger than zero. Values below this threshold will be replaced by that value before transformation to decibel scale. This is caused by the mathematic fact, that a logarithm of zero is not defined.
- `eWindowType` defines the used window function (of type E_CM_WindowType [▸ 166]). Default is `eCM_HannWindow`.
- `eOverlapWindowType` defines the used window function for the inverse transformation. Good standard is the eCM_HannWindow. The windowing can be switched off by using the `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions (in work).
- `eScalingType` offers a scaling selection (refer E_CM_ScalingType [▸ 165]) if an absolute scaling is required. Default is `eCM_DiracScaling`.

Avoid artefacts

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. In order to avoid artefacts during the calculation, `nFFT_Length` should be at least 30% to 50% greater than nWindowLength. An increase in the FFT length in relation to the window length makes sense with this block in order to avoid circular aliasing .

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.9 ST_CM_HistArray_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_HistArray_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels     : UDINT :=  512;      (* number of input channels *)
    nBins         : UDINT :=  100;      (* number of bins in interval *)
    fMinBinned    : LREAL := -120;      (* minimum binned value *)
    fMaxBinned    : LREAL :=  100;      (* maximum binned value *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must be greater than zero.
- `fMinBinned` is the lower limit value for which samples are counted in the regular histogram bins.
- `fMaxBinned` is the upper limit value for which samples are counted in the regular histogram bins. `fMaxBinned` must be greater than `fMinBinned`.
- `nBins` is the number of histogram bins. It must be at least one. In many cases it makes sense to choose values between 10 and 20. The two special bins for values that lie below `fMinBinned` or above `fMaxBinned` are not included in this value.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.10 ST_CM_InstantaneousFrequency_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_InstantaneousFrequency_InitPars EXTENDS ST_MA_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                       (* length of FFT *)
    nWindowLength        : UDINT := 400;                       (* length of analysis window *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;  (* window type of window analysis *)
    eOverlapWindowType   : E_CM_WindowType := eCM_HannWindow;  (* window type for output synthesis *)
    fMagnitudeThreshold  : LREAL := cCM_MinArgDiv;
    fSampleRateHz        : LREAL := 50000;                     (* sample rate in Hertz *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT. It must be greater than one and an integral power of two.
- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.
- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions.
- `eOverlapWindowType` defines the window function used for the inverse transformation. It is freely selectable. A good default value is the window type `eCM_HannWindow`. Windowing can switched off by using the type `eCM_RectangularWindow`.
- `fMagnitudeThreshold` defines the limit value for the numerical calculability of the instantaneous frequency. The limit value refers to the value

$$\mathrm{conj}\{\mathcal{H}\{x[n]\}\}\mathcal{H}\{x[n+1] < \mathrm{cCMMinArgDiv}$$

- `fSampleRateHz` Sampling rate of the incoming time signal. The value is used for scaling the result in Hz.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.11 ST_CM_InstantaneousPhase_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_InstantaneousPhase_InitPars EXTENDS ST_MA_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                       (* length of internal  FFT *)
    nWindowLength        : UDINT := 400;                       (* length of analysis window *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;  (* window type of window analysis *)
    eOverlapWindowType   : E_CM_WindowType := eCM_HannWindow;  (* window type for output synthesis *)
    eUnwrapMethod        : E_CM_UnwrapMethod := eCM_ThresholdUnwrapping; (* Unwrap phase values *)
    fPhaseThreshold      : LREAL := cCM_MinArgDiv;
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT. It must be greater than one and an integral power of two.

- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.
- `eWindowType` defines the used window function (of the type E_CM_WindowType [▸ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions.
- `eOverlapWindowType` defines the window function used for the inverse transformation. It is freely selectable. A good default value is the window type `eCM_HannWindow`. Windowing can switched off by using the type `eCM_RectangularWindow`.
- `eUnwrapMethod` defines the method used for *phase unwrapping*.
- `fPhaseThreshold` Limit value for calculating the instantaneous phase. The value refers to the signal envelope. Interpretation: If the signal level is too low, the phase calculation is numerically too uncertain and cannot be evaluated reliably. In this case 0 is output as the phase.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.12 ST_CM_IntegratedRMS_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_IntegratedRMS_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;    (* length of FFT *)
    nWindowLength        : UDINT := 400;    (* length of FFT window *)
    fSampleRate          : LREAL := 20000;  (* sample rate *)
    fLowerFrequencyLimit: LREAL := 20.0;    (* lower frequency limit of measurement *)
    fUpperFrequencyLimit: LREAL := 1000.0; (* upper frequency limit of measurement *)
    nOrder               : UDINT := 2;      (* maximum order of integration (0 = acceleration, 1 = ve
locity, 2 = place *)
    nChannels            : UDINT := 2;      (* number of input channels *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow; (* windowing function used *)
    bTransformToDecibel : BOOL  := TRUE;    (* transform result to decibel *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10; (* log threshold for decibel transformation *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` has to be bigger than one. It has to be a power of two.
- `nWindowLength` is the length of the analysis window in samples. The length has to be bigger than one and an even number.
- `fSampleRate` Sampling rate of the incoming time signal.
- `fLowerFrequencyLimit` Lower limit of the considered frequency interval. The lower limit frequency must be at least the sampling rate divided by the FFT length.
- `fUpperFrequencyLimit` Upper limit of the considered frequency interval. The upper limit frequency must be no greater than half the sampling rate and greater than the lower limit frequency.
- `nOrder` is the maximum order of the integration. This must be an integer between zero and two. The number of the values determined per channel is (nOrder+1).
- `nChannels` is the number of independent channels. This must be greater than zero.
- `eWindowType` defines the used window function (of type E_CM_WindowType [▸ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions (in work).
- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation x → 20 * log10(x).
- `fDecibelThreshold` is a very small floating point value greater than zero. Values that are less than this number are replaced with this value before any transformation to the decibel scale, since the logarithm of zero is not defined mathematically.

Window length

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 3/4 of the FFT length is often used as the window length.

If `nFFT_Length` is greater than `nWindowLength`, the frequency resolution of the FFT is increased. The length difference is filled with zeros before the Fourier Transform. This can be useful for achieving a higher frequency resolution or, e.g. when calculating with inverse transformation in the time domain, in order to avoid circular aliasing. Despite the higher frequency resolution, however, the result contains no more information.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.13 ST_CM_MagnitudeSpectrum_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_MagnitudeSpectrum_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;
    nWindowLength        : UDINT := 400;
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;
    bTransformToDecibel  : BOOL  := TRUE;
    eWindowType          : E_CM_WindowType := eCM_HannWindow;
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;
END_STRUCT
END_TYPE
```

- `nFFT_Length` has to be bigger than one. It has to be a power of two.
- `nWindowLength` is the length of the analysis window in samples. The length has to be bigger than one and an even number.
- `fDecibelThreshold` is a very small floating point value greater than zero. Values that are less than this number are replaced with this value before any transformation to the decibel scale, since the logarithm of zero is not defined mathematically.
- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation x → 20 * log10(x).
- `eWindowType` defines the used window function (of type E_CM_WindowType [▶ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions (in work).
- `eScalingType` offers a choice of scaling (refer E_CM_ScalingType [▶ 165]), if absolute scaling is required. The default value is `eCM_DiracScaling`. The standard value `eCM_DiracScaling` adapts the scaling to the common scaling of the FFT. When selecting the scaling the type of signal should be considered: either deterministic signals or wide-band signals with stochastic portion. Both types require different scalings. More precise explanations are given in section Scaling factors.

Window length

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 3/4 of the FFT length is often used as the window length.

If `nFFT_Length` is greater than `nWindowLength`, the frequency resolution of the FFT (and therefore also the length of the return values vector) is increased. The length difference is filled with zeros before the Fourier transform . This can be useful for achieving a higher frequency resolution or, e.g. when calculating with inverse transformation in the time domain , in order to avoid circular aliasing . Despite the higher frequency resolution, however, the result contains no more information.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.14 ST_CM_MomentCoefficients_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_MomentCoefficients_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels                : UDINT :=  512;    (* number of input channels *)
    nOrder                   : E_CM_MCoefOrder := E_CM_MCoefOrder.eCM_Kurtosis;
    bPopulationEstimates     : BOOL  := FALSE;   (* apply Bessel's correction to results *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must be greater than zero.
- `nOrder` is the maximum order of the moment coefficients (E_CM_MCoefOrder [▶ 165]) that are calculated. This must be an integer between one and four. The order numbers are: 0 = counter, 1 = average value, 2 = standard deviation, 3 = skew, 4 = excess kurtosis. The number of determined coefficients is (nOrder+1).
- `bPopulationEstimates` is a Boolean value that indicates, whether the corresponding Bessel's correction is applied to the sample variance, skew and excess.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.15 ST_CM_MultiBandRMS_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_MultiBandRMS_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length            : UDINT := 512;     (* length of FFT *)
    nWindowLength          : UDINT := 400;     (* length of FFT window *)
    fSampleRate            : LREAL := 20000;   (* sample rate *)
    nMaxBands              : UDINT := 10;      (* maximum number of bands *)
    nChannels              : UDINT := 10;      (* number of input channels *)
    eWindowType            : E_CM_WindowType := E_CM_WindowType.eCM_HannWindow; (* windowing function 
used *)
    bTransformToDecibel    : BOOL  := TRUE;    (* transform result to decibel *)
    fDecibelThreshold      : LREAL := cCM_MinArgLog10; (* log threshold for decibel transformation *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT. It must be greater than one and an integral power of two.
- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.
- `fSampleRate` specifies the sampling rate (samples per second) of the input signal.
- `nMaxBands` specifies the maximum number of bands for which the RMS is calculated.
- `nChannels` is the number of independent channels. This must be greater than zero.
- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions.

- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation x → 20 * log10(x).
- `fDecibelThreshold` is a very small floating point value greater than zero. Values that are less than this number are replaced with this value before any transformation to the decibel scale, since the logarithm of zero is not defined mathematically. The smallest possible value is 3.75e-324, which is equivalent to the constant `cCM_MinArgLog10` .

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.16 ST_CM_PowerCepstrum_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_PowerCepstrum_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length   : UDINT := 512;              (* length of FFT *)
    nWindowLength : UDINT := 400;              (* length of analysis window *)
    eWindowType   : E_CM_WindowType := eCM_HannWindow; (* window type of window analysis *)
    fLogThreshold : LREAL := cCM_MinArgLogN;   (* threshold for logarithm computation *)
    eScalingType  : E_CM_ScalingType := eCM_DiracScaling;
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT. It must be greater than one and an integral power of two.
- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.
- `eWindowType` defines the used window function (of the type E_CM_WindowType [▸ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions.
- `fLogThreshold` is a very small floating point value greater than zero. The smallest possible value is 3.75e-324, which is equivalent to the constant `cCM_MinArgLogN`.
  Spectral values with absolute values that are smaller than this number are replaced with this value before the spectrum is logarithmized. The purpose is the avoidance of value range errors. The logarithm of zero is not defined and strives infinitely towards minus for the limit value of small arguments. The same applies to the argument of the number zero, arg(0).
- `eScalingType` offers a choice of scaling (see E_CM_ScalingType [▸ 165]), if absolute scaling is required. The default value `eCM_DiracScaling` adapts the scaling of the PowerSpectrum function block to the common scaling of the FFT. For further information see notes in the introductory "Frequency analysis" section [▸ 35], section "Absolute scaling of the power spectrum", and the Scaling factor table [▸ 227].
  When selecting the scaling the type of signal should be considered: either deterministic signals or wide-band signals with stochastic portion. Both types require different scalings.

Window length

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 3/4 of the FFT length is often used as the window length.

If `nFFT_Length` is greater than `nWindowLength`, the frequency resolution of the FFT (and therefore also the length of the return values vector) is increased. The length difference is filled with zeros before the Fourier transform. This can be useful for achieving a higher frequency resolution or, e.g. when calculating with inverse transformation in the time domain, in order to avoid circular aliasing . Despite the higher frequency resolution, however, the result contains no more information.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.17 ST_CM_PowerSpectrum_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_PowerSpectrum_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;
    nWindowLength       : UDINT := 400;
    fDecibelThreshold   : LREAL := cCM_MinArgLog10;
    bTransformToDecibel: BOOL  := TRUE;
    eWindowType         : E_CM_WindowType := eCM_HannWindow;
    eScalingType        : E_CM_ScalingType := eCM_DiracScaling;
END_STRUCT
END_TYPE
```

- `nFFT_Length` has to be bigger than one. It has to be a power of two.
- `nWindowLength` is the length of the analysis window in samples. The length has to be bigger than one and an even number.
- `fDecibelThreshold` is a very small floating point value greater than zero. Values that are less than this number are replaced with this value before any transformation to the decibel scale, since the logarithm of zero is not defined mathematically.
- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation x → 20 * log10(x).
- `eWindowType` defines the used window function (of type E_CM_WindowType [▸ 166]). A good default value is the window type `eCM_HannWindow`. The windowing can be switched off by the use of the window type `eCM_RectangularWindow`. Further explanations and the list of possible window functions can be found in the introductory section Window functions (in work).
- `eScalingType` offers a choice of scaling (refer E_CM_ScalingType [▸ 165]), if absolute scaling is required. The default value is `eCM_DiracScaling`. The standard value `eCM_DiracScaling` adapts the scaling to the common scaling of the FFT. When selecting the scaling the type of signal should be considered: either deterministic signals or wide-band signals with stochastic portion. Both types require different scalings. More precise explanations are given in section Scaling factors.

Window length

The value of `nFFT_Length` must be equal or greater the value of `nWindowLength`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 3/4 of the FFT length is often used as the window length.

If `nFFT_Length` is greater than `nWindowLength`, the frequency resolution of the FFT (and therefore also the length of the return values vector) is increased. The length difference is filled with zeros before the Fourier transform. This can be useful for achieving a higher frequency resolution or, e.g. when calculating with inverse transformation in the time domain, in order to avoid circular aliasing . Despite the higher frequency resolution, however, the result contains no more information.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.18 ST_CM_Quantiles_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_Quantiles_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels         : UDINT :=  512;        (* number of input channels *)
    fMinBinned        : LREAL := 120;         (* minimum binned value *)
    fMaxBinned        : LREAL :=  100;        (* maximum binned value *)
    nBins             : UDINT :=  100;        (* number of bins in interval *)
    nMaxQuantiles     : UDINT :=  10;         (* maximum number of quantile values for *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must always be bigger than zero.
- `fMinBinned` is the lower limit value for which samples are counted in the regular histogram bins.
- `fMaxBinned` is the upper limit value for which samples are counted in the regular histogram bins. `fMaxBinned` must be greater than `fMinBinned`.
- `nBins` is the number of bins in the Histogram. It must be at least one. In many cases it makes sense to set the value between 10 and 20. Take care about fMinBinned and fMaxBinned. The two special bins for values below `fMinBinned` and above `fMaxBinned` are not included in this value.
- `nMaxQuantiles` is the number of quantiles to be calculated for each channel. This must be an integer bigger than zero.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.19 ST_CM_RealFFT_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_RealFFT_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;       (* length of FFT *)
    bForward            : BOOL := TRUE;
    bHalfSpec           : BOOL := TRUE;
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT. It must be greater than one and an integral power of two.
- `bForward` is a Boolean parameter indicating the direction of the FFT. If the value is "true", the normal FFT is calculated. Otherwise the inverse FFT is used.
- `bHalfSpec` is a Boolean parameter, that specifies the parameters of the result buffer. If the value is "true", the algorithm outputs half the spectrum (`nFFT_Length`/2 + 1).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.20 ST_CM_RMS_InitPars

Function-block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_RMS_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels           : UDINT := 4;
    nBufferLength       : UDINT := 2000;
    fDecibelThreshold   : LREAL := cCM_MinArgLog10;
    bTransformToDecibel : BOOL  := TRUE;
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must be greater than zero.
- `nBufferLength` is the number of input values per channel to be held in the internal buffer.
- `fDecibelThreshold` is a very small floating point value greater than zero. Values that are less than this number are replaced with this value before any transformation to the decibel scale, since the logarithm of zero is not defined mathematically. The smallest possible value is 3.75e-324, which is equivalent to the constant `cCM_MinArgLog10` .
- `bTransformToDecibel` is a boolean value that specifies whether the result is to be transformed to the decibel scale, according to the transformation x → 20 * log10(x).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.21 ST_CM_WatchUpperThresholds_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_WatchUpperThresholds_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nChannels       : UDINT := 10;    (* number of input channels *)
    nMaxClasses     : UDINT := 3;     (* number of configurable threshold classes *)
    bMemorize       : BOOL :=TRUE;    (* retain largest result until Reset() call *)
END_STRUCT
END_TYPE
```

- `nChannels` is the number of independent channels. This must always be bigger than zero.
- `nMaxClasses` is the maximum number of configured classes. This must always be bigger than zero.
- `bMemorize` is a boolean variable. If `FALSE` the function block recalculates the number of the highest category and the corresponding channel for each call. So only current input data influence the output. If `TRUE` the result values are stored and considered at next calculation until the ResetData() method is called. Default is `TRUE`.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6.22 ST_MA_MultiArray_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_MA_MultiArray_InitPars :
STRUCT
    eTypeCode      : E_MA_ElementTypeCode := eMA_TypeCode_LREAL;
    nDims          : UDINT := 1;
    aDimSizes      : ARRAY[0.. 15] OF UDINT := [1, 1, 1, 1,  1, 1, 1, 1,  1, 1, 1, 1,  1, 1, 1, 1]; (
* size for each dimension *)
END_STRUCT
END_TYPE
```

- `eTypeCode` This parameter specifies the element type (E_MA_ElementTypeCode [▶ 166]) of the MultiArray buffer elements.
- `nDims` This parameter specifies the number of dimensions of the MultiArray buffer.

- `aDimSizes` The size of each dimension is specified by this array.
  If the required shape of the input buffer of an following algorithm is given as 'm x n' (in its input stream) the MultiArray buffer has to be specified with aDimSizes := [m,n].

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.3.6.23 ST_MA_TransferTray_InitPars

Function block specific structure for initialization parameters.

```
TYPE ST_CM_TransferTray_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nStreams            : UDINT :=   64;     (* number of independent Queue channels *)
    nMaxEntries         : UDINT :=   32;     (* minimum space in queue *)
    nQueueSize          : UDINT :=  256;     (* reserved space of queue (power-of-two) *)
    bLockFree           : BOOL  :=  TRUE;    (* use lock free structures in place of interrupts *)
    nUpdatePeriod       : UDINT :=    2;     (* frequency of updates to queue indices *)
END_STRUCT
END_TYPE
```

- `nStreams` This parameter indicates how many streams (i.e. independently functioning queues) the function block FB_CM_TransferTray provides. There should be a separate queue for each task-spanning data stream. Additional channels do not require any system resources.
- `nMaxEntries` This parameter indicates the maximum number of elements that the queues can contain. For the communication of data buffers it usually makes sense for all buffers that come into question to have space in the queue so that no buffer overrun conditions can occur. A value of one can also be selected.
- `bLockFree` If this parameter is `TRUE`, a modern lock-free implementation is used for the queues. This is the preset state. Otherwise a classic implementation with interrupt disable is used. The lock-free implementation can achieve a better time behavior of the overall system, but may lead to higher latencies under an extremely high load.
- `nQueueSize` The reserved length of the queues. This value must be larger than `nMaxEntries` and in addition must be a power of two.
- `nUpdatePeriod` This parameter indicates how often internal intermediate results are refreshed. The frequency of complex operations can be reduced by a value greater than one. Values of two (preset) or three are usually practical.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.4 Global constants

5.4.1 GVL_CM

Option flags for option mask:

Each method lists its supported option flags.

```
cCMA_Option_MarkInterruption : DWORD := 16#0000_0001;
```

Mark first buffer element (of each channel) as invalid (NaN) if time series collection has been interrupted. Hint: It isn't possible to compute correct spectra from chopped time series.

For further information see chapter Handling with NaN values [▸ 65].

Analysis block constants:

```
cCMA_MaxDest : UDINT := 20;
```

Maximum number of destinations for one analysis block.

```
cCMA_MaxID : UDINT := 200;
```

Maximum ID which can be used (=maximum number of analysis blocks).

Transfer Tray parameter:

The internal Transfer Tray used for buffer transfer between analysis blocks is initialized with these constants.

```
cCMA_InitParsTransferTray  : ST_MA_TransferTray_InitPars := (
nStreams      := 1024,
                                   nMaxEntries   := 10,
                                   nQueueSize    := 64,
                                   bLockFree     := TRUE,
                                   nUpdatePeriod := 2 );
```

nStreams : number of independent FIFO streams
nMaxEntries : maximum entries in FIFO
nQueueSize : size of FIFO (power of two > nMaxEntries)
bLockFree : if true use lock-free data structure | if false use interrupt-locks
nUpdatePeriod : update frequency of internal indices

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM

5.4.2 GVL_CM_Base

Threshold constants:

```
cCM_MinArgLog10 : LREAL :=  2.3E-308;    (* approximate minimum argument of decadic logarithm *)
cCM_MinArgLogN  : LREAL :=  2.3E-308;    (* approximate minimum argument of natural logarithm *)
cCM_MinArgDiv   : LREAL :=  2.3E-308;    (* minimum argument of division *)
```

The purpose is the avoidance of value range errors. The logarithm of zero is not defined and strives infinitely towards minus for the limit value of small arguments. The same applies to the argument of the number zero, arg(0).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.4.3 Global_Version

This global constant contains the library version information.

All libraries have a specific version. This version is shown in the PLC library repository too.

```
VAR_GLOBAL CONSTANT
    stLibVersion_Tc3_CM : ST_LibVersion;
END_VAR
```

Type definition of this global constant structure: ST_LibVersion

To compare the existing version to a required version the function F_CmpLibVersion (defined in Tc2_System library) is offered.

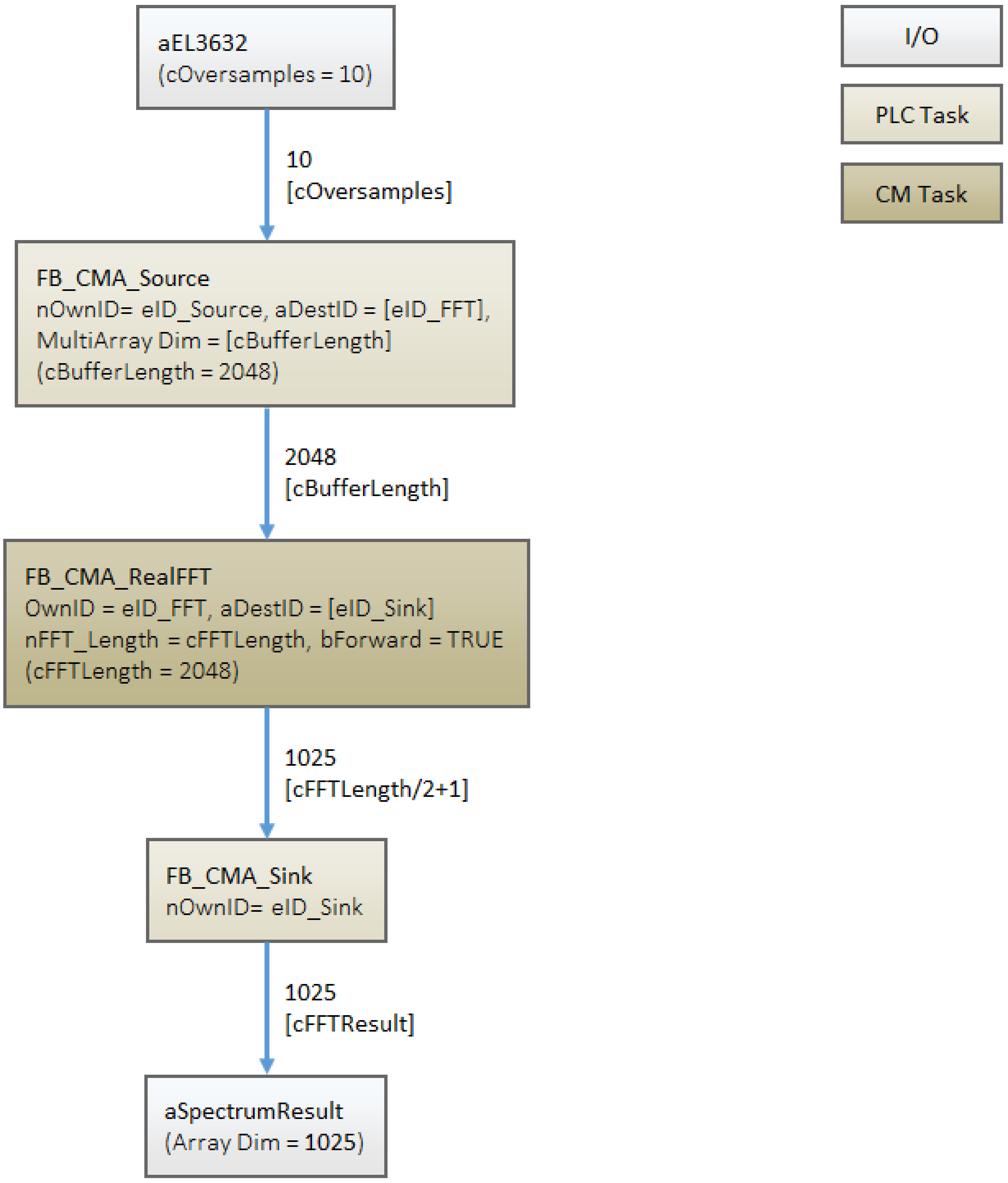
6 Samples

6.1 FFT with real-value input signal

The sample illustrates the implementation of a spectrum calculation with the function block FB_CMA_RealFFT [▸ 145].

The sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649248523.zip

Block diagram



Program parameters

The table below shows a list of important parameters for the configuration of the

RealFFT function block.

FFT length	2048
Forward calculation	TRUE

Requirements

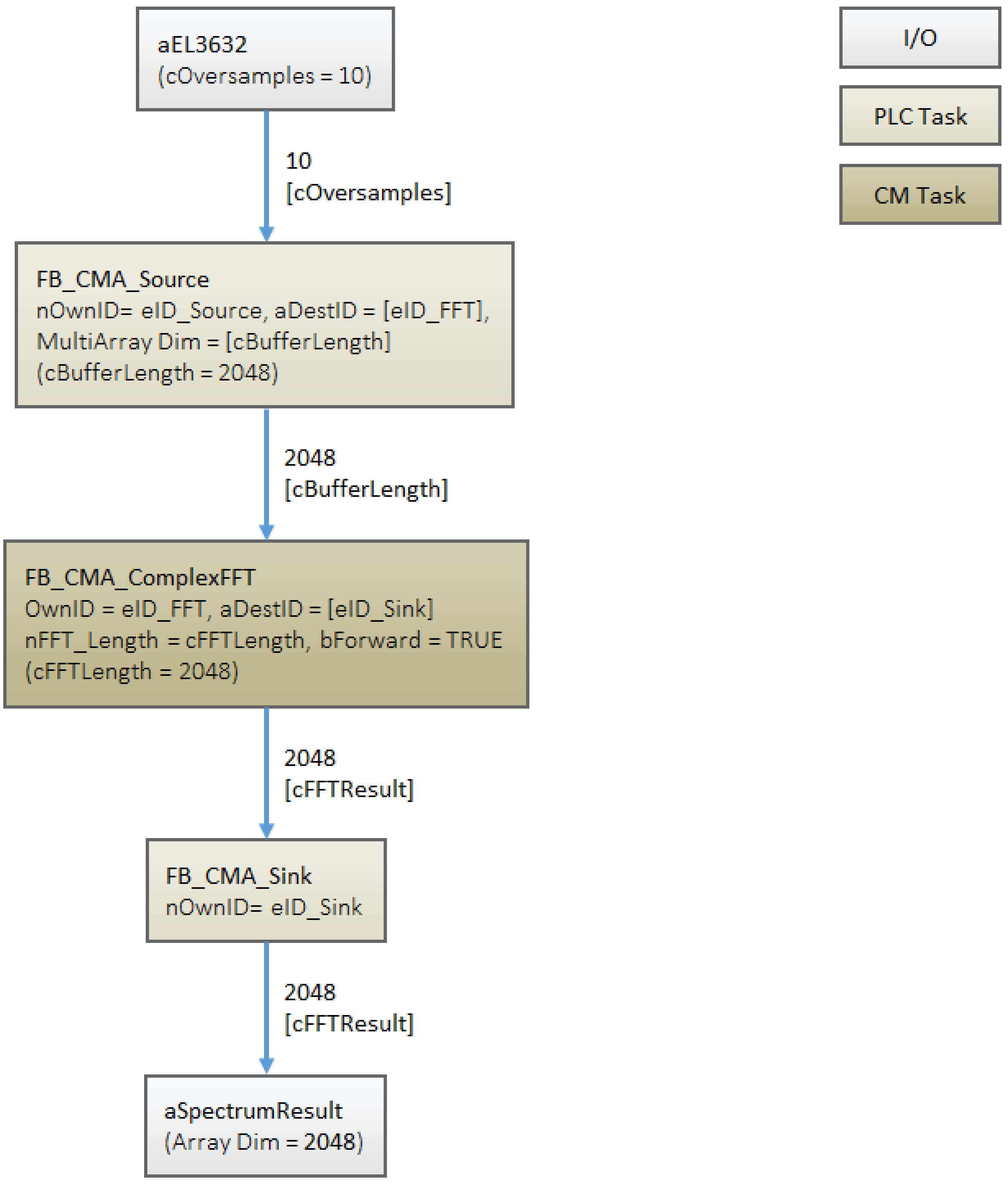
Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.2 FFT with complex-value input signal

The sample illustrates the implementation of a spectrum calculation with the function block FB_CMA_ComplexFFT [▸ 86]. In contrast to the function block FB_CMA_RealFFT [▸ 145], the data type LCOMPLEX is used for the required multi-array.

The sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649220747.zip

Block diagram



Program parameters

The table below shows a list of important parameters for the configuration of the

ComplexFFT function block.

Type code	eMA_TypeCode_LCOMPLEX
FFT length	2048

Forward calculation	TRUE

Requirements

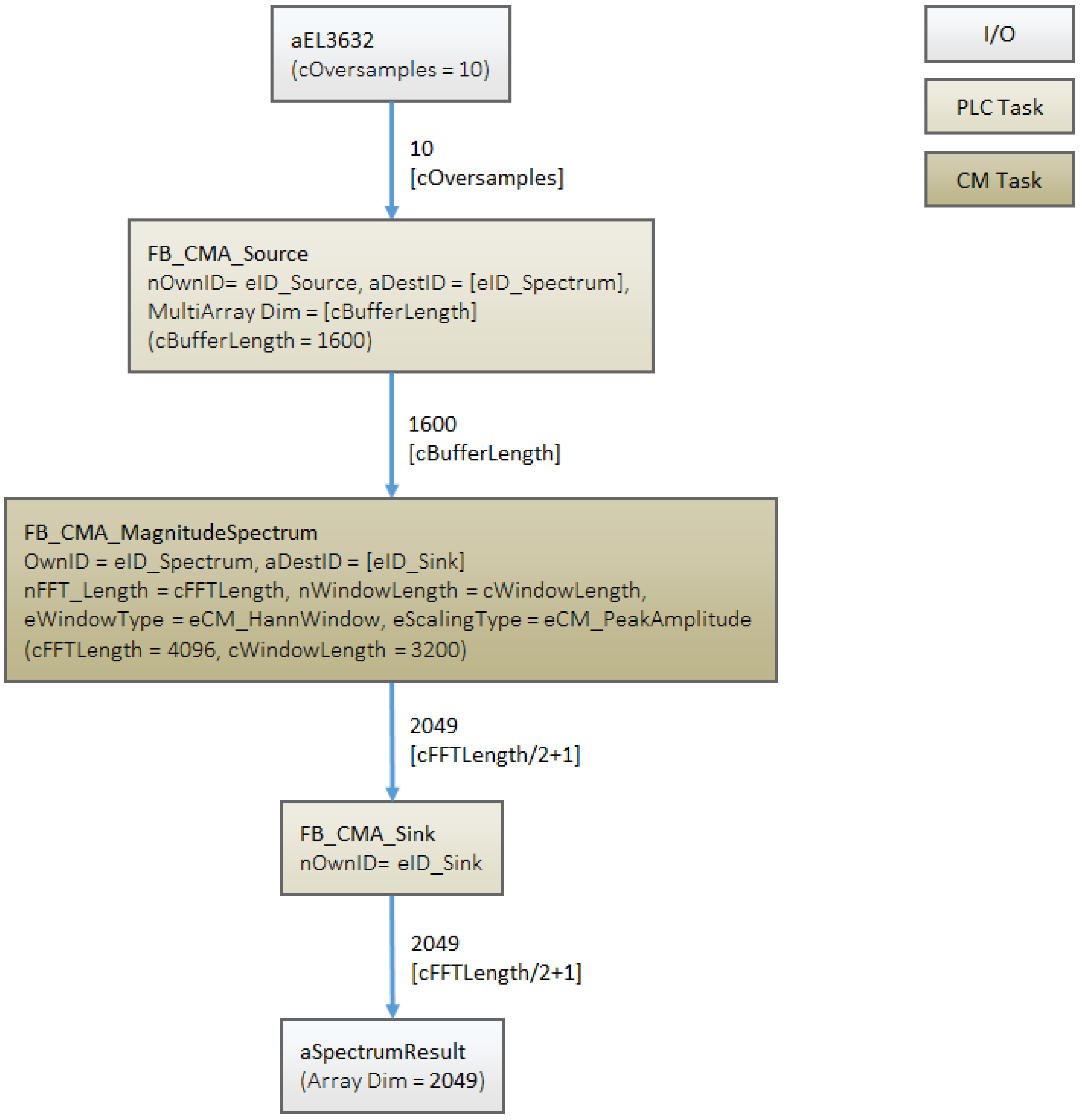
Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.3 Magnitude spectrum:

This sample implements a single-channel magnitude spectrum. The code is split into two tasks: a control task, which collects the discrete input signal of a hardware module, e.g. EL3632, and a CM task, which calculates the spectrum. The block diagram below shows the analysis chain implemented in the sample.

The source code for the sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649315851.zip

Block Diagram


aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_Spectrum],
MultiArray Dim = [cBufferLength]
(cBufferLength = 1600)
1600
[cBufferLength]
FB_CMA_MagnitudeSpectrum
OwnID = eID_Spectrum, aDestID = [eID_Sink]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
eWindowType = eCM_HannWindow, eScalingType = eCM_PeakAmplitude
(cFFTLength = 4096, cWindowLength = 3200)
2049
[cFFTLength/2+1]
FB_CMA_Sink
nOwnID= eID_Sink
2049
[cFFTLength/2+1]
aSpectrumResult
(Array Dim = 2049)
I/O
PLC Task
CM Task

Program parameters

The table below shows a list of important parameters for the configuration of the magnitude spectrum function block.

FFT-length	4096
Window size	3200
Buffer size	1600
Window type	eCM_HannWindow
Scaling type	eCM_ROOT_POWER_SUM
Scaling in decibels (dB)	FALSE

Global Constants

These parameters are defined as constants in the list of global variables.

```
VAR_GLOBAL CONSTANT
    cOversamples        : UDINT := 10;      // oversampling factor
    cBufferLength       : UDINT := 1600;    // size of buffer for spectrum
    cWindowLength       : UDINT := 3200;   // size of window
    cFFTResult          : UDINT := 2049;   // size of spectrum result
    cFFTLength          : UDINT := 4096;   // spectrum lines
END_VAR
```

Code for Control Task

Following code snippet shows the declaration in MAIN program:

```
PROGRAM MAIN

VAR CONSTANT
    cInitSource    : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL, nDims := 1, aDim
Sizes := [cBufferLength]);
END_VAR

VAR
     nInputSelection    : UDINT := 1; // Switch between hardware and function generator
     nSample   : UDINT;
     aEl3632 AT %I*     : ARRAY[1..cOversamples] OF INT; // Input from hardware e.g. EL3632
     aBuffer   : ARRAY[1..cOversamples] OF LREAL;

     fbSource       : FB_CMA_Source :=( stInitPars := cInitSource, nOwnID := eID_Source, aDestIDs :=
[eID_Spectrum]); // Initialize source buffers
     fbSink         : FB_CMA_Sink := (nOwnID := eID_Sink);
     aSpectrumResult    : ARRAY[1..cFFTResult] OF LREAL; // Copy result
END_VAR
```

Method calls in MAIN program:

```
fbSource.Input1D(pDataIn := ADR(aBuffer),
               nDataInSize := SIZEOF(aBuffer),
               eElementType := eMA_TypeCode_LREAL,
               nWorkDim := 0,
               pStartIndex := 0,
               nOptionPars := cCMA_Option_MarkInterruption);

fbSink.Output1D(pDataOut := ADR(aSpectrumResult),
               nDataOutSize := SIZEOF(aSpectrumResult),
               eElementType := eMA_TypeCode_LREAL,
               nWorkDim := 0,
               nElements := 0,
               pStartIndex := 0,
               nOptionPars := 0,
               bNewResult => bCalculate);
```

Code for CM Task

Declaration in MAIN_CM program:

```
PROGRAM MAIN_CM

VAR CONSTANT
     cInitSpectrum : ST_CM_MagnitudeSpectrum_InitPars := (nFFT_Length := cFFTLength,
                                    nWindowLength := cWindowLength,
                                    bTransformToDecibel:= FALSE,
                                    eWindowType := eCM_HannWindow,
                                    eScalingType := eCM_RMS);
END_VAR
VAR
     fbSpectrum : FB_CMA_MagnitudeSpectrum :=(stInitPars := cInitSpectrum,
                                   nOwnID := eID_Spectrum,
                                   aDestIDs := [eID_Sink]);
END_VAR
```

Method calls in MAIN_CM program:

```
fbSpectrum.Call();
```

The result of the sample code can be checked for a sinusoidal signal of arbitrary amplitude and frequency as the input signal. The variable, fRmsValue above should be exactly equal to amplitude/SQRT(2).

Each frequency value can be assigned to the corresponding array index of the spectrum result. Calculation formula:
sample rate = oversampling factor / sampling task cycle time
index = frequency * (FFT length / sample rate)

Requirements

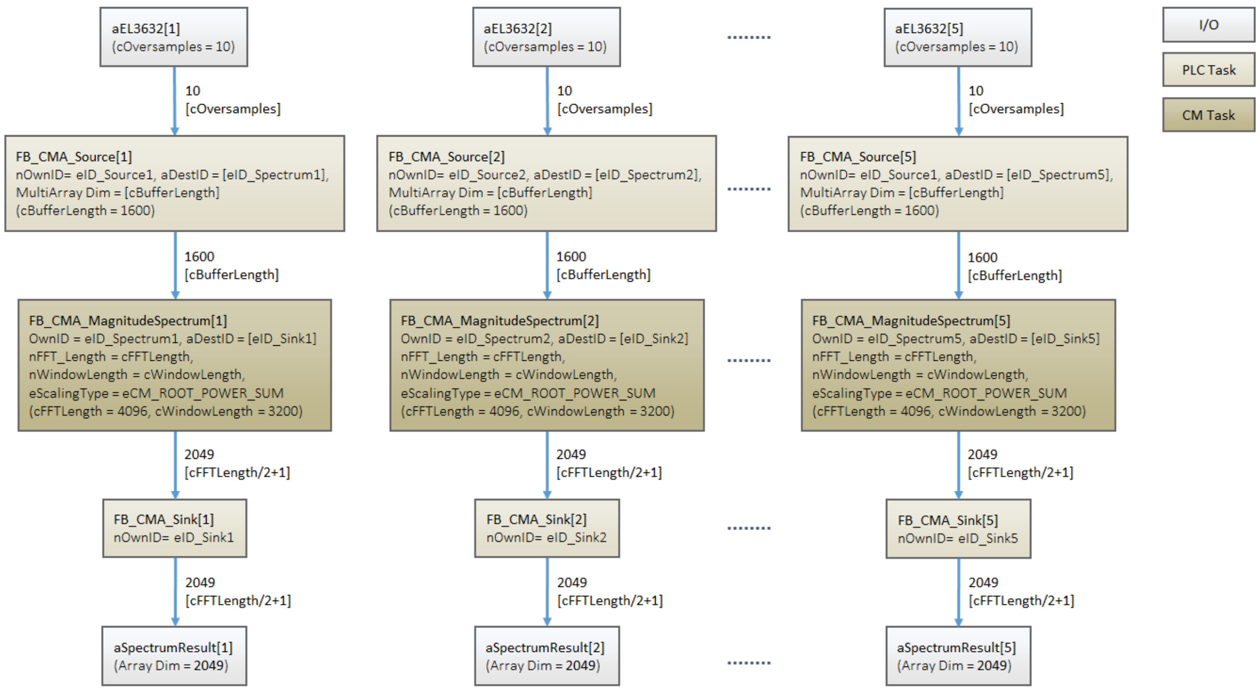
Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.4 Multi-channel magnitude spectrum

This sample implements the magnitude spectrum for 5 input channels simultaneously. The code is split into two tasks: a control task, which collects the input samplings of a hardware module, e.g. EL3632, and a CM task, which calculates the spectrum. The block diagram below shows the analysis chain implemented in the program.

The source code for the sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649317515.zip

Block diagram


aEL3632[1] (cOversamples = 10)
10 [cOversamples]
FB_CMA_Source[1] nOwnID= eID_Source1, aDestID = [eID_Spectrum1], MultiArray Dim = [cBufferLength] (cBufferLength = 1600)
1600 [cBufferLength]
FB_CMA_MagnitudeSpectrum[1] OwnID = eID_Spectrum1, aDestID = [eID_Sink1] nFFT_Length = cFFTLength, nWindowLength = cWindowLength, eScalingType = eCM_ROOT_POWER_SUM (cFFTLength = 4096, cWindowLength = 3200)
2049 [cFFTLength/2+1]
FB_CMA_Sink[1] nOwnID= eID_Sink1
2049 [cFFTLength/2+1]
aSpectrumResult[1] (Array Dim = 2049)
aEL3632[2] (cOversamples = 10)
FB_CMA_Source[2] nOwnID= eID_Source2, aDestID = [eID_Spectrum2], MultiArray Dim = [cBufferLength] (cBufferLength = 1600)
FB_CMA_MagnitudeSpectrum[2] OwnID = eID_Spectrum2, aDestID = [eID_Sink2] nFFT_Length = cFFTLength, nWindowLength = cWindowLength, eScalingType = eCM_ROOT_POWER_SUM (cFFTLength = 4096, cWindowLength = 3200)
FB_CMA_Sink[2] nOwnID= eID_Sink2
aSpectrumResult[2] (Array Dim = 2049)
aEL3632[5] (cOversamples = 10)
FB_CMA_Source[5] nOwnID= eID_Source1, aDestID = [eID_Spectrum5], MultiArray Dim = [cBufferLength] (cBufferLength = 1600)
FB_CMA_MagnitudeSpectrum[5] OwnID = eID_Spectrum5, aDestID = [eID_Sink5] nFFT_Length = cFFTLength, nWindowLength = cWindowLength, eScalingType = eCM_ROOT_POWER_SUM (cFFTLength = 4096, cWindowLength = 3200)
FB_CMA_Sink[5] nOwnID= eID_Sink5
aSpectrumResult[5] (Array Dim = 2049)
I/O
PLC Task
CM Task

Note that this block diagram shows one of the input channels configured for calculating the magnitude spectrum. The chain is replicated exactly for the other channels.

Program parameters

The table below shows a list of important parameters for the configuration of the magnitude spectrum function blocks.

Channels	5
FFT-length	4096
Window size	3200
Buffer size	1600
Window type	eCM_HannWindow
Scaling type	eCM_ROOT_POWER_SUM
Conversion to decibels	FALSE

Global Constants

These parameters are defined in the global variable list as constants.

```
VAR_GLOBAL CONSTANT
    cOversamples       : UDINT := 20;    // oversampling factor
    cMaxChannels       : UDINT := 5;     // number of channels
    cBufferLength           : UDINT := 1600;  // size of buffer for spectrum
    cFFTResult         : UDINT := 2049;  // size of spectrum result
    cWindowLength           : UDINT := 3200;  // size of window
    cFFTLength         : UDINT := 4096;  // spectrum lines
END_VAR
```

Code for Control Task

Following code snippet shows the declaration in MAIN program:

```
PROGRAM MAIN

VAR CONSTANT
    cInitParsResultBuffer : ST_MA_MultiArray_InitPars :=(eTypeCode := eMA_TypeCode_LREAL, aDimSizes
:= [cBufferLength]); // buffer length
END_VAR

VAR
     nInputSelection   : UDINT := 1; // select input, 0: hardware, 1: function generator
     nChnIdx       : UDINT; // channel index
     nSample       : UDINT;

     aEl3632 AT %I*    : ARRAY [1..cMaxChannels] OF ARRAY [1..cOversamples] OF INT; // connect senso
r signal here
     aBuffer       : ARRAY [1..cMaxChannels] OF ARRAY [1..cOversamples] OF LREAL;

     fbSource : ARRAY[1..cMaxChannels] OF FB_CMA_Source // source collecting multi-
channel input signal
          := [     (stInitPars := cInitParsResultBuffer, nOwnID := eID_SourceChn1, aDestIDs := [eID_
SpectrumChn1]),
               (stInitPars := cInitParsResultBuffer, nOwnID := eID_SourceChn2, aDestIDs := [eID_Spec
trumChn2]),
               (stInitPars := cInitParsResultBuffer, nOwnID := eID_SourceChn3, aDestIDs := [eID_Spec
trumChn3]),
               (stInitPars := cInitParsResultBuffer, nOwnID := eID_SourceChn4, aDestIDs := [eID_Spec
trumChn4]),
               (stInitPars := cInitParsResultBuffer, nOwnID := eID_SourceChn5, aDestIDs := [eID_Spec
trumChn5])];

     fbSink : ARRAY[1..cMaxChannels] OF FB_CMA_Sink
          := [     (nOwnID := eID_SinkChn1),
               (nOwnID := eID_SinkChn2),
               (nOwnID := eID_SinkChn3),
               (nOwnID := eID_SinkChn4),
               (nOwnID := eID_SinkChn5)];

     aNewResult : ARRAY[1..cMaxChannels] OF BOOL;
END_VAR
```

Method calls in MAIN program:

```
FOR nChnIdx :=1 TO cMaxChannels DO

// Collect input data in source
     fbSource[nChnIdx].Input1D(      pDataIn := ADR(aBuffer[nChnIdx]),
                    nDataInSize := SIZEOF(aBuffer[nChnIdx]),
                    eElementType := eMA_TypeCode_LREAL,
                    nWorkDim := 0,
                    pStartIndex := 0,
                    nOptionPars := cCMA_Option_MarkInterruption );
// Push results to sink
     fbSink[nChnIdx].Output1D(  pDataOut := ADR(aSpectrumResult[nChnIdx]),
                    nDataOutSize := SIZEOF(aSpectrumResult[nChnIdx]),
                    eElementType := eMA_TypeCode_LREAL,
                    nWorkDim := 0,
                    nElements := 0,
                    pStartIndex := 0,
                    nOptionPars := 0,
                    bNewResult => aNewResult[nChnIdx]);
END_FOR
```

Code for CM Task

Declaration in MAIN_CM program:

```
PROGRAM MAIN_CM

VAR CONSTANT
     cInitParsSpectrum : ST_CM_MagnitudeSpectrum_InitPars
               :=(      nFFT_Length := cFFTLength,
                    nWindowLength := cWindowLength,
                    fDecibelThreshold := cCM_MinArgLog10,
                    bTransformToDecibel := FALSE,
                    eWindowType := eCM_HannWindow,
                    eScalingType := eCM_ROOT_POWER_SUM); // Configure spectrum analyzer here
END_VAR

VAR
     fbSpectrum : ARRAY[1..cMaxChannels] OF FB_CMA_MagnitudeSpectrum
          := [      (stInitPars := cInitParsSpectrum, nOwnID := eID_SpectrumChn1, aDestIDs := [eID_Si
nkChn1]),
               (stInitPars := cInitParsSpectrum, nOwnID := eID_SpectrumChn2, aDestIDs := [eID_SinkCh
n2]),
               (stInitPars := cInitParsSpectrum, nOwnID := eID_SpectrumChn3, aDestIDs := [eID_SinkCh
n3]),
               (stInitPars := cInitParsSpectrum, nOwnID := eID_SpectrumChn4, aDestIDs := [eID_SinkCh
n4]),
               (stInitPars := cInitParsSpectrum, nOwnID := eID_SpectrumChn5, aDestIDs := [eID_SinkCh
n5])];
     nChnIdx : UDINT;
END_VAR
```

Method calls in MAIN_CM program:

```
FOR nChnIdx:=1 TO cMaxChannels DO
     // Call Spectrum
     fbSpectrum[nChnIdx].Call();
END_FOR
```

The result of the sample code can be checked for sinusoidal signals of arbitrary amplitude and frequency. The RMS values are stored in the array aRmsValue according to the corresponding channel number. The result should be exactly equal to the peak amplitude of each sinusoidal signal divided by /SQRT(2). The sample code can be extended for more than 5 channels depending on the requirements and the resources of the target system.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

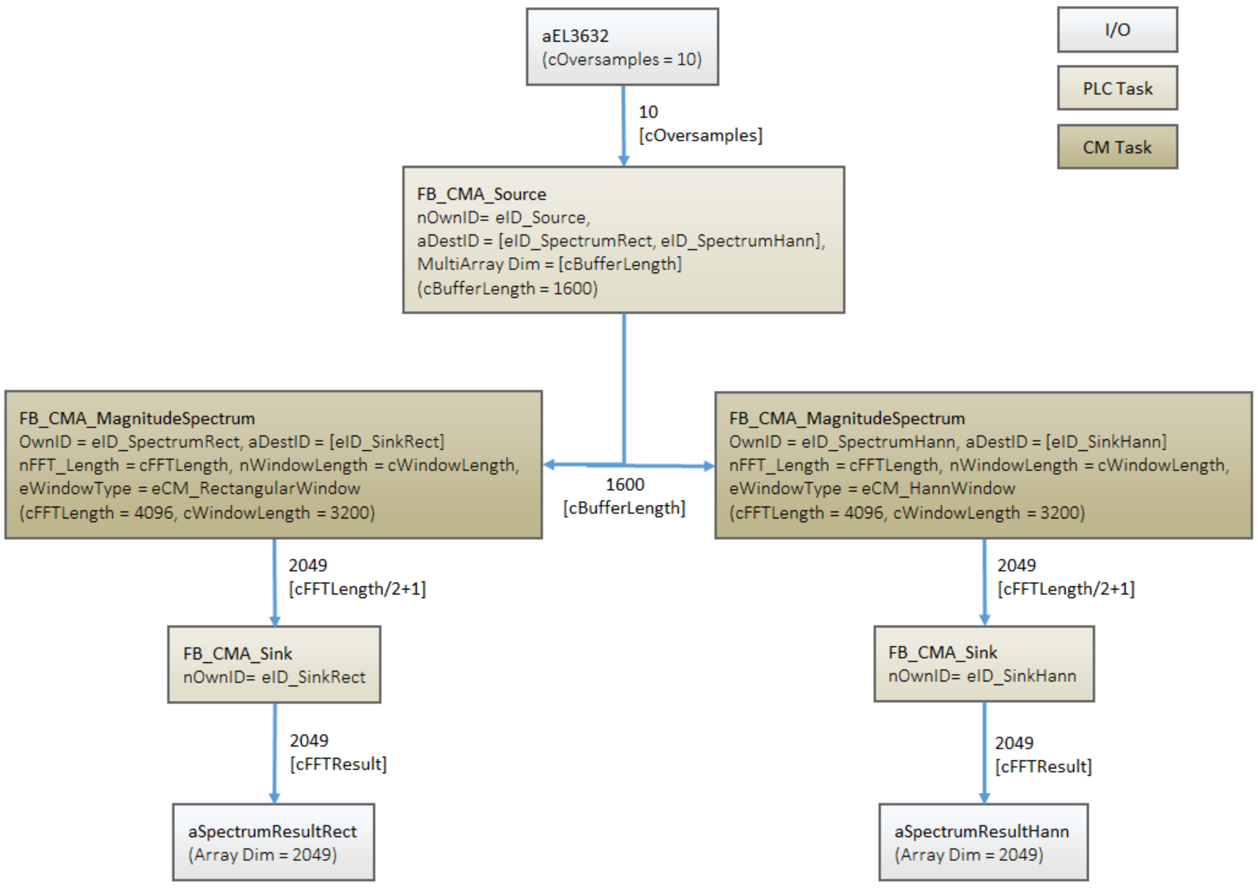
6.5 Window functions

This sample implements a single-channel magnitude spectrum and compares the application of different window functions. For better illustration, the frequency range of the scope is limited to 0 Hz to 1000 Hz.

The source code for the sample is available for download from here:

http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/5261536139.zip

Block diagram


aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source,
aDestID = [eID_SpectrumRect, eID_SpectrumHann],
MultiArray Dim = [cBufferLength]
(cBufferLength = 1600)
I/O
PLC Task
CM Task
FB_CMA_MagnitudeSpectrum
OwnID = eID_SpectrumRect, aDestID = [eID_SinkRect]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
eWindowType = eCM_RectangularWindow
(cFFTLength = 4096, cWindowLength = 3200)
1600
[cBufferLength]
FB_CMA_MagnitudeSpectrum
OwnID = eID_SpectrumHann, aDestID = [eID_SinkHann]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
eWindowType = eCM_HannWindow
(cFFTLength = 4096, cWindowLength = 3200)
2049
[cFFTLength/2+1]
FB_CMA_Sink
nOwnID= eID_SinkRect
2049
[cFFTResult]
aSpectrumResultRect
(Array Dim = 2049)
2049
[cFFTLength/2+1]
FB_CMA_Sink
nOwnID= eID_SinkHann
2049
[cFFTResult]
aSpectrumResultHann
(Array Dim = 2049)

Program parameters

The table below shows a list of important parameters for the configuration of the magnitude spectrum function block.

FFT length	4096
Window size	3200
Buffer size	1600
Window type	eCM_HannWindow / eCM_RectangularWindow
Scaling type	eCM_ PeakAmplitude

Requirements

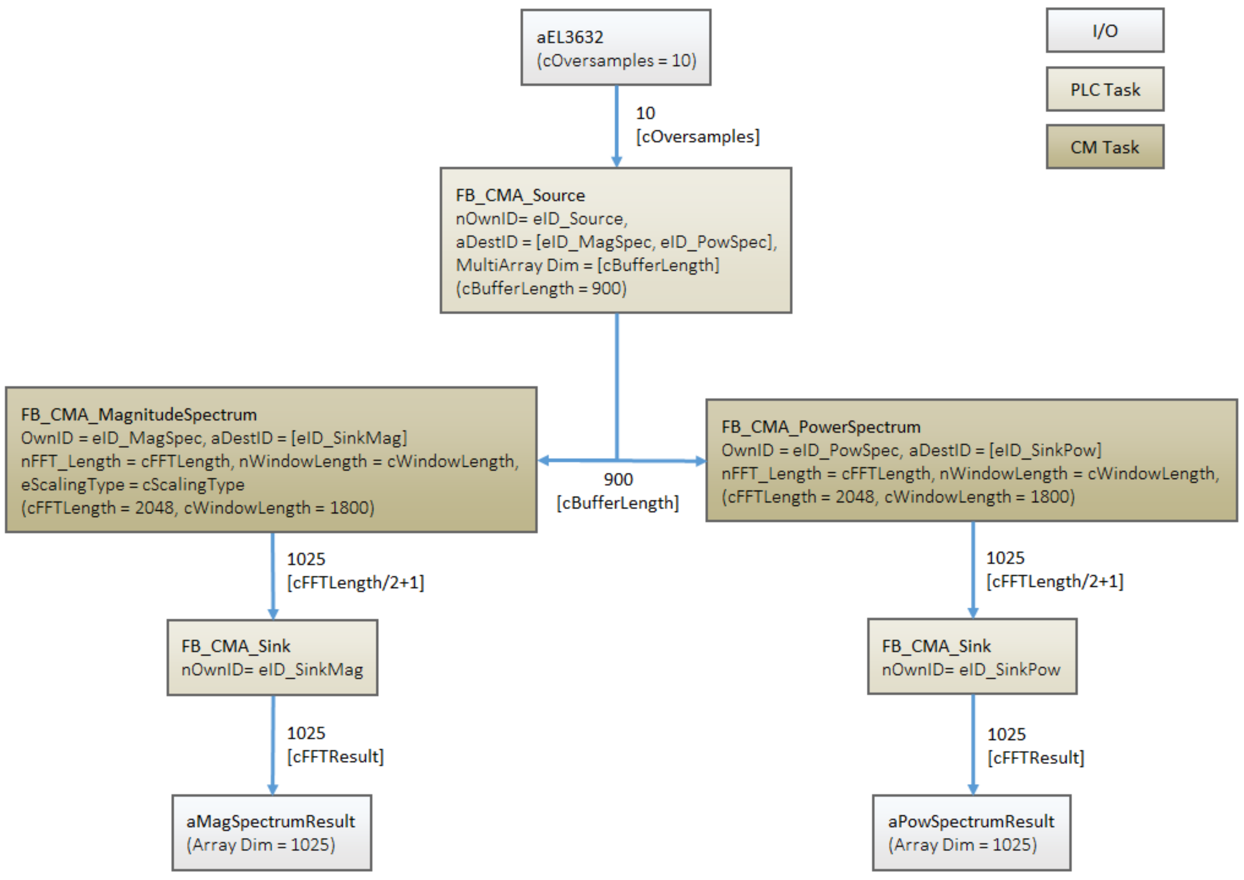
Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.6 Scaling of spectra

As described under Scaling of spectra [▸ 22], the Condition Monitoring library offers a number of different options for scaling of spectra. This tutorial enables examination of various prepared scalings by means of a simple sine wave, and to deepen the theoretical understanding. The scopes are limited to the range 0 Hz to 400 Hz, in order to be able to show the differences more clearly.

The tutorial is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649251851.zip

Block diagram


aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source,
aDestID = [eID_MagSpec, eID_PowSpec],
MultiArray Dim = [cBufferLength]
(cBufferLength = 900)
900
[cBufferLength]
FB_CMA_MagnitudeSpectrum
OwnID = eID_MagSpec, aDestID = [eID_SinkMag]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
eScalingType = cScalingType
(cFFTLength = 2048, cWindowLength = 1800)
FB_CMA_PowerSpectrum
OwnID = eID_PowSpec, aDestID = [eID_SinkPow]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
(cFFTLength = 2048, cWindowLength = 1800)
1025
[cFFTLength/2+1]
FB_CMA_Sink
nOwnID= eID_SinkMag
FB_CMA_Sink
nOwnID= eID_SinkPow
1025
[cFFTResult]
aMagSpectrumResult
(Array Dim = 1025)
aPowSpectrumResult
(Array Dim = 1025)
I/O
PLC Task
CM Task

Program parameters

The table below shows a list of important parameters for the configuration of the used function blocks FB_CMA_MagnitudeSpectrum [▸ 127] and FB_CMA_PowerSpectrum [▸ 142].

GVL_Constant contains three scenarios, which you can test by commenting or uncommenting the selected code segments and enabling the configuration. The expected behavior of the scenarios is documented in the GVL as a comment.

FFT length	2048
Window size	1800
Conversion to decibels	TRUE / FALSE
Window type	eCM_HannWindow
Scaling type	eCM_PeakAmplitude / eCM_RMS

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

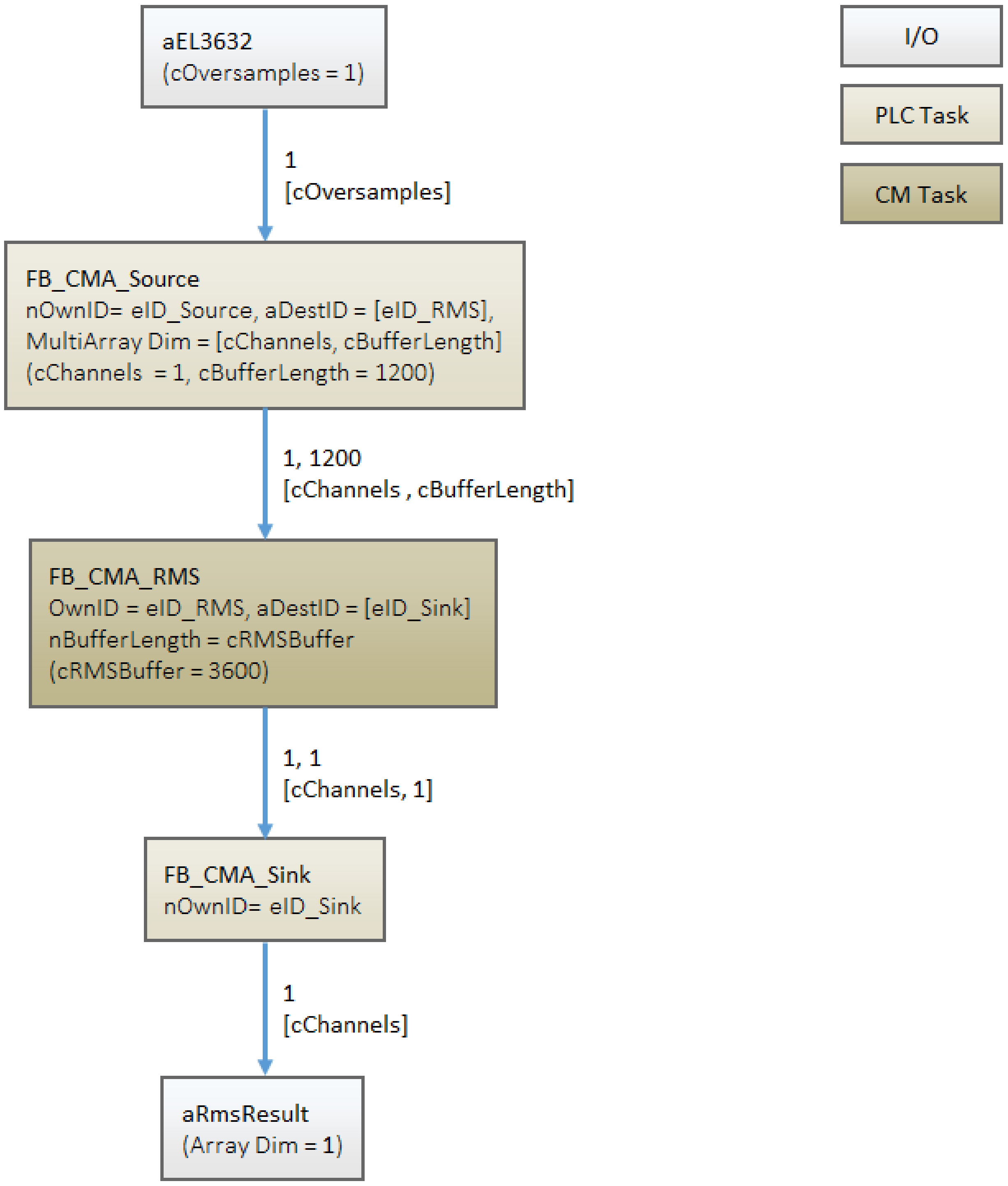
6.7 Time-based RMS

The sample illustrates the implementation of a time-based RMS calculation for a signal with the function block FB_CMA_RMS [▶ 152].

The sample is available for download from here:

http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649250187.zip

Block diagram


aEL3632
(cOversamples = 1)
1
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_RMS],
MultiArray Dim = [cChannels, cBufferLength]
(cChannels = 1, cBufferLength = 1200)
1, 1200
[cChannels , cBufferLength]
FB_CMA_RMS
OwnID = eID_RMS, aDestID = [eID_Sink]
nBufferLength = cRMSBuffer
(cRMSBuffer = 3600)
1, 1
[cChannels, 1]
FB_CMA_Sink
nOwnID= eID_Sink
1
[cChannels]
aRmsResult
(Array Dim = 1)
I/O
PLC Task
CM Task

Program parameters

The table below shows a list of important parameters for the configuration of the program block for calculating the time-based RMS of a signal.

Channels	1
Buffer size	1200
Conversion to decibels	FALSE

Requirements

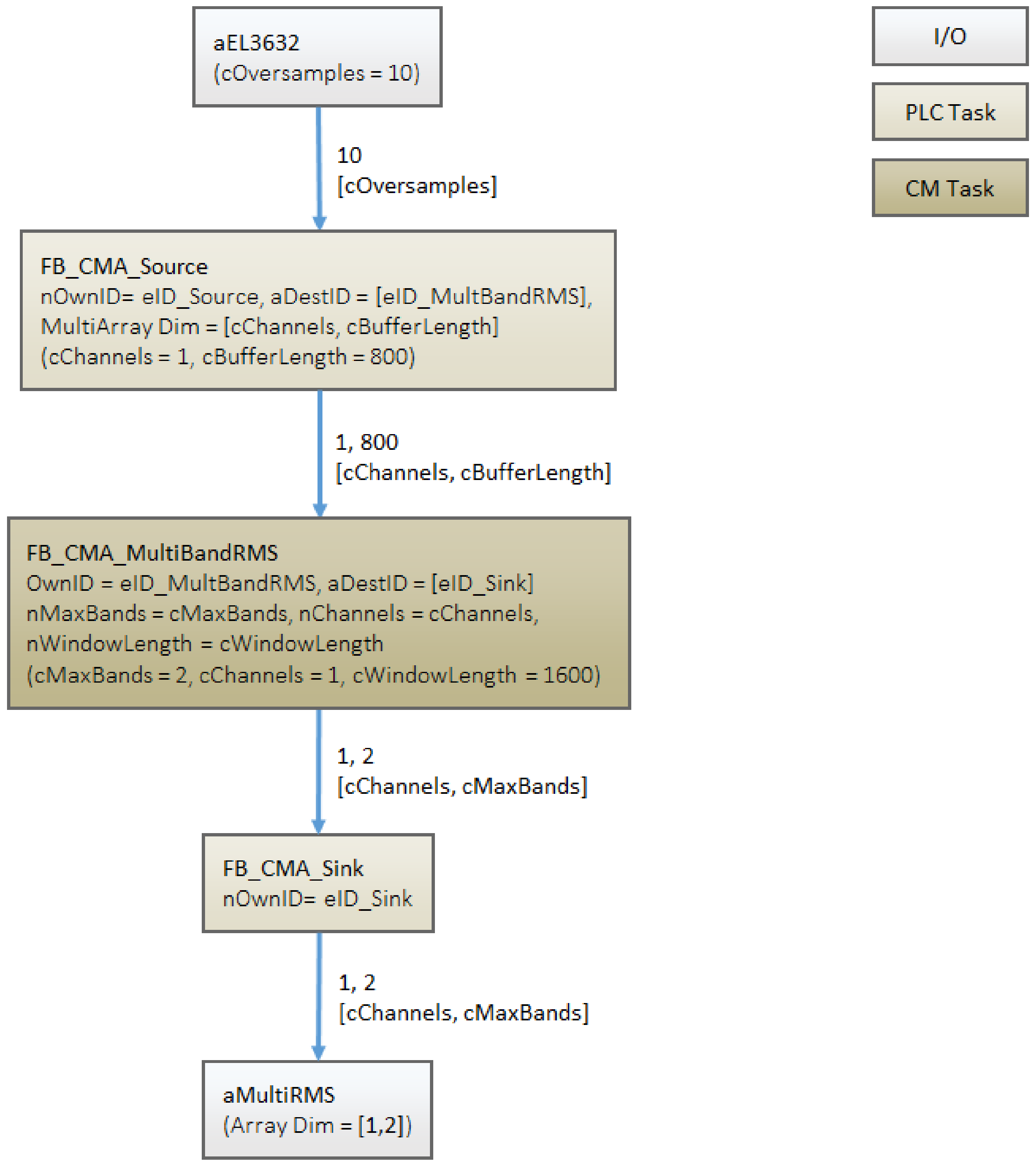
Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.8 Multi-band RMS

The sample illustrates a calculation implementation for several frequency band-limited RMS values of a signal with the function block FB_CMA_MultiBandRMS [▶ 135].

The sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649246859.zip

Block diagram


aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_MultBandRMS],
MultiArray Dim = [cChannels, cBufferLength]
(cChannels = 1, cBufferLength = 800)
1, 800
[cChannels, cBufferLength]
FB_CMA_MultiBandRMS
OwnID = eID_MultBandRMS, aDestID = [eID_Sink]
nMaxBands = cMaxBands, nChannels = cChannels,
nWindowLength = cWindowLength
(cMaxBands = 2, cChannels = 1, cWindowLength = 1600)
1, 2
[cChannels, cMaxBands]
FB_CMA_Sink
nOwnID= eID_Sink
1, 2
[cChannels, cMaxBands]
aMultiRMS
(Array Dim = [1,2])
I/O
PLC Task
CM Task

Program parameters

The table below shows a list of important parameters for the configuration of the function block for calculating several frequency band-limited RMS values of a signal

Size of the FFT	2048
Window size	1600
Sampling rate	10000
Frequency bands	2
Channels	1

Window type	eCM_HannWindow
Conversion to decibels	FALSE

Requirements

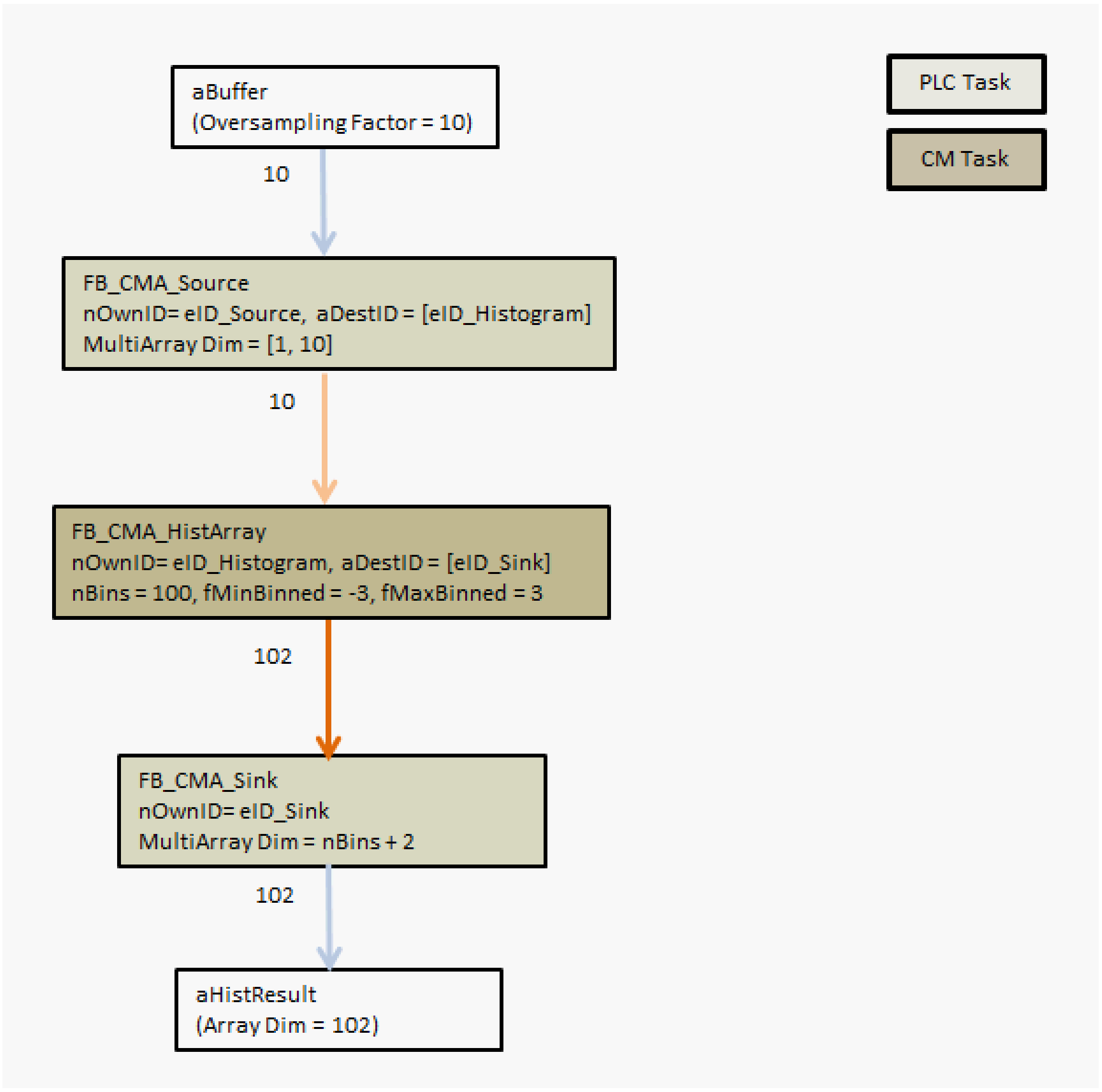
Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.9 Histogram

This sample implements a histogram. The code is divided into two tasks: a control task that collects the input data, e.g. from EL3632, and a so-called CM task that calculates the histogram. The block diagram below shows the analysis chain.

The source code for the sample is available for download from here: http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/18014401904055179.zip

Block diagram


aBuffer
(Oversampling Factor = 10)
10
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_Histogram]
MultiArray Dim = [1, 10]
10
FB_CMA_HistArray
nOwnID= eID_Histogram, aDestID = [eID_Sink]
nBins = 100, fMinBinned = -3, fMaxBinned = 3
102
FB_CMA_Sink
nOwnID= eID_Sink
MultiArray Dim = nBins + 2
102
aHistResult
(Array Dim = 102)
PLC Task
CM Task

Program parameters

The table below shows the most important parameters for the configuration of the histogram function block:

Histogram Bins	100
Appended Datasets	10
Oversamples	10
Max. Bin Limit	+3 or +5
Min. Bin Limit	-3 or -5
Channels	1
Buffer Length	100

Global Constants

The parameters specified above can be defined as global constants:

```
VAR_GLOBAL CONSTANT
    cBufferLength : UDINT := 100;
    cChannels     : UDINT := 1;
    cOversamples  : UDINT := 10;
    cMaxBins      : UDINT := 100;
    cAppendedData : UDINT := 10;
    cBinLimit_1   : LREAL := 3;
    cBinLimit_2   : LREAL := 5;
END_VAR
```

Code for the MAIN task

The following code snippet shows the declaration in the MAIN program:

```
PROGRAM MAIN
VAR CONSTANT
    cInitSource    : ST_MA_MultiArray_InitPars
    := (eTypeCode := eMA_TypeCode_LREAL, nDims := 2, aDimSizes := [1, cBufferLength]);
END_VAR
VAR
    nInputSelection : UDINT := 1;
    nSample         : UDINT;
    aEl3632 AT %I* : ARRAY [1..cOversamples] OF INT;
    aBuffer         : ARRAY [1..cOversamples] OF LREAL;
    fbSource        : FB_CMA_Source := (stInitPars := cInitSource, nOwnId
:= eID_Source, aDestIDs := [eID_Histogram]);
    fbSink          : FB_CMA_Sink := (nOwnID := eID_Sink);
    aHistReulst     : ARRAY [1..cMaxBins+2];
END_VAR
```

The following code snippet shows the method calls in the MAIN program:

```
fbSource.Input2D(pDataIn  := ADR(aBuffer),
                aDataInSize  := SIZEOF(aBuffer),
                eElementType := eMA_TypeCode_LREAL,
                nWorkDim0    := 0,
                nWorkDim1    := 1,
                pStartIndex  := 0,
                nOptionPars  := cCMA_Option_MarkInterruption);

fbSink(pDataOut := ADR(aHistResult),
        nDataOutSize := SIZEOF(aHistResult),
        eElementType := eMA_TypeCode_UINT64,
        nWorkDim0    := 0,
        nWorkDim1    := 1,
        nElements    := 0,
        pStartIndex  := 0,
        nOptionPars  := 0);
```

Code for the CM task

The variable declaration in the MAIN_CM program:

```
VAR CONSTANT
    cInitHistArray : ST_CM_HistArray_InitPars := (nChannels := cChannels, nBins := cMaxBins,
fMinBinned := -cBinLimit_1, fMaxBinned := cBinLimit_1);
END_VAR
```

The method calls in the MAIN_CM program:

```
fbHistArray.CallEx(nAppendData := cAppendData, bReset := );

IF bConfig then
    fbHistArray.Configure(pArg := ADR(aHisArrayConfig), nArgSize := SIZEOF(aHistArrayConfig)
END_IF
```

The Configure method is optional, but it enables the fine setting of the parameters fMinBinned and fMaxBinned at runtime.

Random Number Generator

A histogram is very often used as a visual help in order to understand the underlying distribution of all measured values, e.g. the peaks in the vibration signal. The function generator contained in the sample code is extended for this purpose. The function generator can simulate the usual and practically oriented random numbers and their distributions. Using the variable E_DistributionType you can select a distribution such as exponential, normal (or Gaussian), Chi-squared or gamma. By default the random numbers are generated from a uniform distribution.

Note

Please note that every distribution requires one or more parameters in order to determine the propagation of the random numbers or their range. This can be done using the input variable aRange.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4016.12	PC or CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base, Tc3_MultiArray

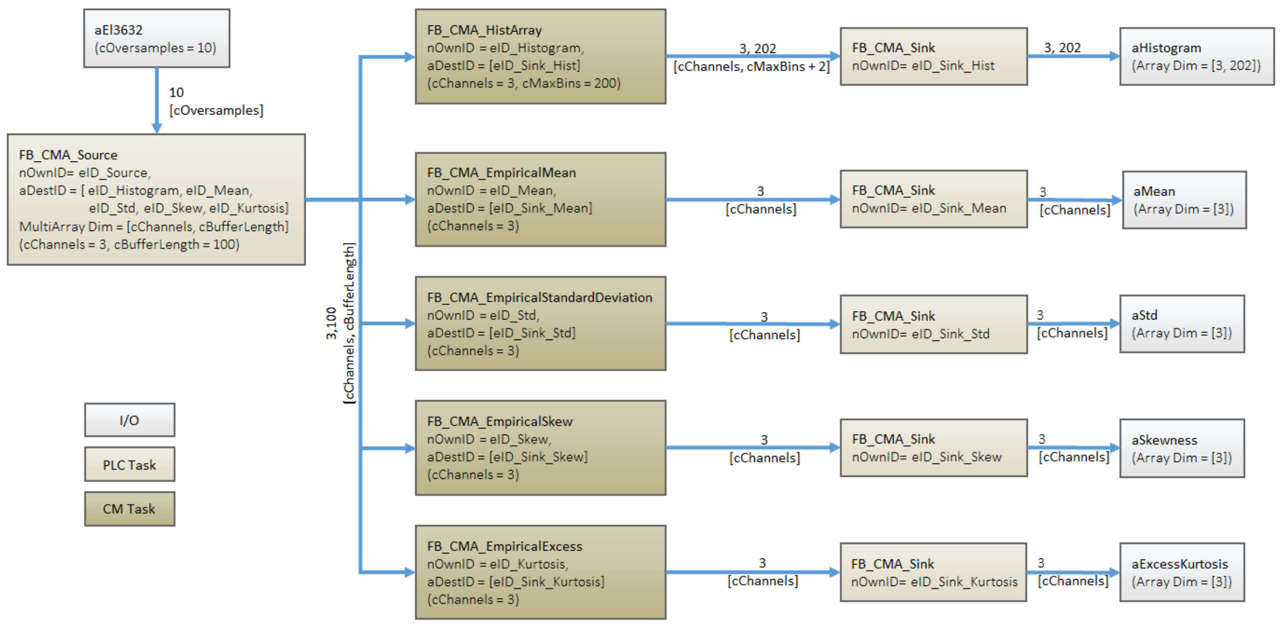
6.10 Statistical methods

This sample illustrates the options for statistical evaluation of Condition Monitoring Library data. Statistical evaluations for standard normal and gamma-distributed signal data and a sine signal are processed. The function blocks FB_CMA_HistArray [▸ 115], FB_CMA_EmpiricalMean [▸ 97], FB_CMA_EmpiricalStandardDeviation [▸ 105], FB_CMA_EmpiricalSkew [▸ 101] and FB_CMA_EmpiricalExcess [▸ 93] are used.

The sample is available for download from here:

http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/5261532811.zip

Block diagram



Program parameters

The table below shows a list of important parameters for the configuration of the function blocks that are used.

Buffer size	100
Channels	3
Frequency bins	200

Requirements

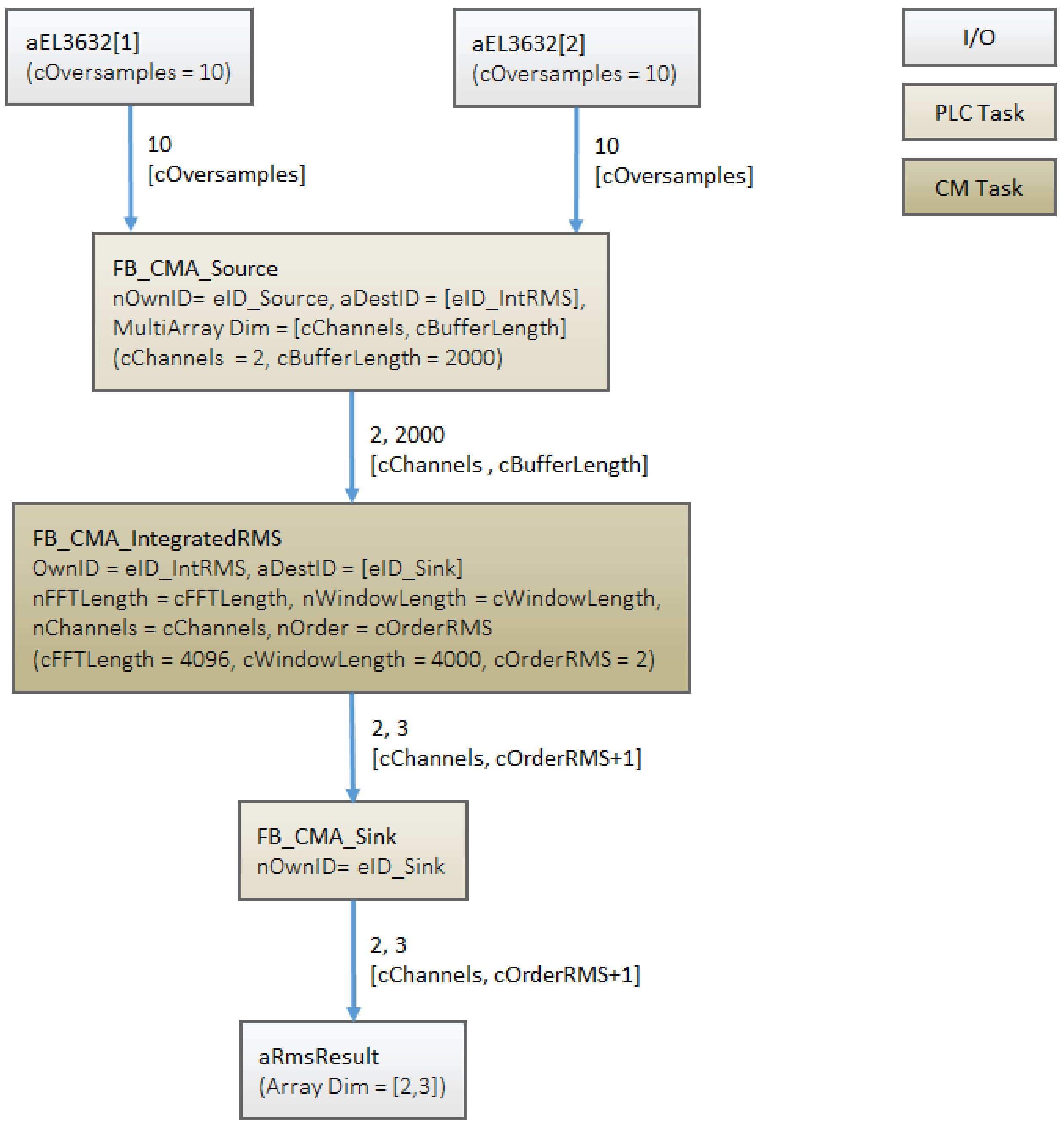
Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022	PC or CX (x86, x64)	Tc3_CM (>= 1.0.22), Tc3_CM_Base (>= 1.1.10), Tc3_MultiArray

6.11 Vibration assessment according to ISO 10816-3

Vibration assessment based on ISO 10816-3 is explained in more detail in section Application concepts, see Vibration assessment [▶ 31]. The classification based on the calculated RMS values is done directly in the MAIN program. Alternatively, the function blocks FB_CMA_WatchUpperThresholds [▶ 162] or FB_CMA_DiscreteClassification [▶ 89] could be used.

The sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/18014401903994507.zip

Block diagram



aEL3632[1]
(cOversamples = 10)
10
[cOversamples]
aEL3632[2]
(cOversamples = 10)
10
[cOversamples]
I/O
PLC Task
CM Task
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_IntRMS],
MultiArray Dim = [cChannels, cBufferLength]
(cChannels = 2, cBufferLength = 2000)
2, 2000
[cChannels , cBufferLength]
FB_CMA_IntegratedRMS
OwnID = eID_IntRMS, aDestID = [eID_Sink]
nFFTLength = cFFTLength, nWindowLength = cWindowLength,
nChannels = cChannels, nOrder = cOrderRMS
(cFFTLength = 4096, cWindowLength = 4000, cOrderRMS = 2)
2, 3
[cChannels, cOrderRMS+1]
FB_CMA_Sink
nOwnID= eID_Sink
2, 3
[cChannels, cOrderRMS+1]
aRmsResult
(Array Dim = [2,3])

Program parameters

The table below shows a list of important parameters for the configuration of the function blocks that are used.

Buffer size	2000
Channels	2
FFT length	4096
Window size	4000
Sampling rate	10000
Lower frequency bound	10

Upper frequency bound	1000
Order (RMS)	2
Window type	eCM_HannWindow
Conversion to decibels	FALSE

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

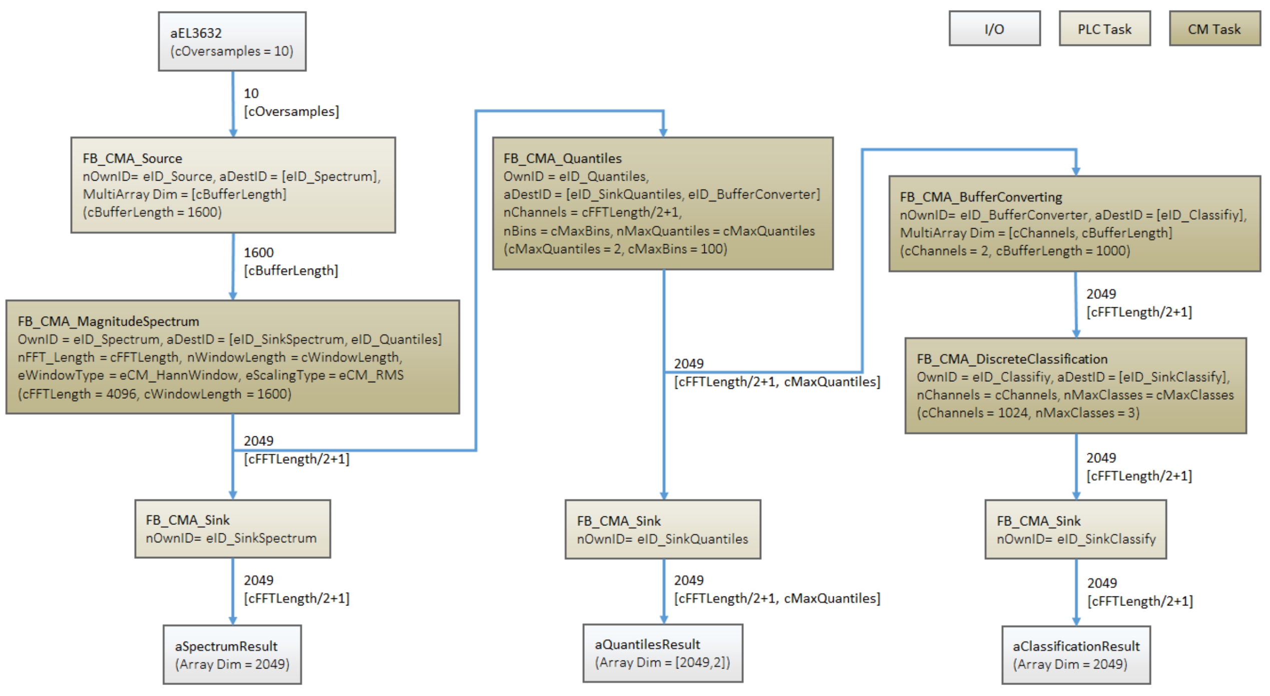
6.12 Condition Monitoring with frequency analysis

This tutorial configures a complete monitoring application, based on the TwinCAT3 Condition Monitoring API. It facilitates creation of a workflow for Condition Monitoring applications, including data collection and adding high-performance analysis algorithms. The block diagram below illustrates the arrangement of the application. For a better understanding of the programming tasks, the document is subdivided into design steps.

The sample is available for download from here: http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649312523.zip

It can be modified and extended as required.

Block Diagram



aEL3632 (cOversamples = 10)
10 [cOversamples]
FB_CMA_Source nOwnID= eID_Source, aDestID = [eID_Spectrum], MultiArray Dim = [cBufferLength] (cBufferLength = 1600)
1600 [cBufferLength]
FB_CMA_MagnitudeSpectrum OwnID = eID_Spectrum, aDestID = [eID_SinkSpectrum, eID_Quantiles] nFFT_Length = cFFTLength, nWindowLength = cWindowLength, eWindowType = eCM_HannWindow, eScalingType = eCM_RMS (cFFTLength = 4096, cWindowLength = 1600)
2049 [cFFTLength/2+1]
FB_CMA_Sink nOwnID= eID_SinkSpectrum
2049 [cFFTLength/2+1]
aSpectrumResult (Array Dim = 2049)
FB_CMA_Quantiles OwnID = eID_Quantiles, aDestID = [eID_SinkQuantiles, eID_BufferConverter] nChannels = cFFTLength/2+1, nBins = cMaxBins, nMaxQuantiles = cMaxQuantiles (cMaxQuantiles = 2, cMaxBins = 100)
2049 [cFFTLength/2+1, cMaxQuantiles]
FB_CMA_Sink nOwnID= eID_SinkQuantiles
2049 [cFFTLength/2+1, cMaxQuantiles]
aQuantilesResult (Array Dim = [2049,2])
I/O
PLC Task
CM Task
FB_CMA_BufferConverting nOwnID= eID_BufferConverter, aDestID = [eID_Classifiy], MultiArray Dim = [cChannels, cBufferLength] (cChannels = 2, cBufferLength = 1000)
2049 [cFFTLength/2+1]
FB_CMA_DiscreteClassification OwnID = eID_Classifiy, aDestID = [eID_SinkClassify], nChannels = cChannels, nMaxClasses = cMaxClasses (cChannels = 1024, nMaxClasses = 3)
2049 [cFFTLength/2+1]
FB_CMA_Sink nOwnID= eID_SinkClassify
2049 [cFFTLength/2+1]
aClassificationResult (Array Dim = 2049)

Step 1: Application specification

The first step for the design of a condition monitoring application is to determine the main aims of the application, e.g. automatic warning in the event of excessive vibrations or in the event of a malfunction in the bearing, based on frequency analysis. It is also important to consider other technical factors such as measuring sensors, the sampling rate of the controller and the expected accuracy. The aim of this tutorial is to detect small and large errors in the input signal with the aid of the magnitude spectrum and its quantile distribution. In addition, a classifier is used for predicting the general state as "normal state", "warning state" or "alarm state". The table below shows a list of the function blocks used in this tutorial.

Function block
FB_CMA_Souce
FB_CMA_Sink
FB_CMA_MagnitudeSpectrum
FB_CMA_Quantiles
FB_CMA_DiscreteClassification

For a more detailed description of the algorithm selection for specific issues such as bearing analysis [▸ 39], gear unit analysis [▸ 47] or frequency analysis [▸ 35], we refer to the solutions described elsewhere. Since the aim of the tutorial is to detect general changes in the input signal, a magnitude spectrum with a resolution of 4098 lines is sufficient. The 50 % and 90 % quantile of the spectral values are calculated, and the result is classified as "normal state", "warning state" or "alarm state".

Step 2: Configuration of the PLC tasks

Since condition monitoring and analysis is comprised of a data acquisition stage, a calculation stage and an analysis stage, the task has to be structured according to the calculation requirements for each step. Here [▸ 63] you can find additional information on this topic. The aim of this tutorial is to calculate the magnitude spectrum of 4098 lines, for which approx. 3200 data samples are required. The means that, during the data collection stage, a source multi-array has to handle 1600 data samples, considering overlapping. With 10x oversampling, 160 cycles are required, or 160 ms with 1 ms trigger, to fill a single multi-array. The following setting is therefore recommended for the calculation task:

Calculation cycle time < (data collection cycle time * buffer size / oversampling factor)*0.5

For the tutorial the calculation cycle time is set to 10 ms. For the actual application it is important to consider the computation load, which is affected by other tasks that run simultaneously on the same controller, such as visualization or network communication. Further information on task settings can be found here [▸ 63] in the task cycle time section. Make sure that adequate router memory capacity [▸ 62] is allocated before starting to build a condition monitoring application. This tutorial was set for working with a router memory capacity adjusted to 32 MB.

Step 3: Configuration of the function blocks

In this step the function blocks listed above are configured according to the application requirements. As already mentioned, the source multi-array collects 1600 data samples for calculating a spectrum. The aDimSizes parameter is therefore set to 1600. Since the tutorial only considers one channel, nDims is set to 1.

```
PROGRAM CM_Worker

VAR CONSTANT
      cInitSourceSpectrum       : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL, nD
ims := 1, aDimSizes := [1600]);
END_VAR
```

In the calculation task the magnitude spectrum for calculating a spectrum of 4098 lines is configured, indicated by cFFTLength. A so-called window function is used, since the spectrum calculation is associated with periodic processing of discrete segments of a continuous signal. A correctly selected window function improves the signal transformation efficiency, reduces fluctuations thanks to the overlap-add method and improves the spectral resolution. In practical applications the window function also reduces the leakage effect near critical frequencies. In the tutorial Hann window was selected. The magnitude spectrum function block offers a wide range of scaling options as shown here [▸ 227], out of which the RMS value was selected. The reason is that for time-varying physical signals, an RMS value is a preferred indicator of the mean signal power, in contrast to the peak value, for example. In the vibration acceleration spectrum, individual lines indicate the effective values of the vibrations at the corresponding frequency and can be expressed directly in the corresponding units such as mm/s² or g.

```
PROGRAM MAIN_CM

VAR CONSTANT
     cInitSpectrum : ST_CM_MagnitudeSpectrum_InitPars := (      nFFT_Length := 4096,
                              nWindowLength := 2*1600,
                              bTransformToDecibel:= FALSE,
                              eWindowType := eCM_HannWindow,
                              eScalingType := eCM_RMS);
END_VAR
```

The result of the magnitude spectrum is copied to an array via a sink function block, with specified array length of nFFT_Length/2+1. In the next step of the analysis chain, a quantile function block for calculating the 50 % and 90 % quantiles of the spectral values is configured. In many cases the spectral values fluctuate strongly, so that an evaluation is difficult if the values are too low or too high. Using the quantiles it is possible to determine the maximum, minimum or indeed the average value over a specified time interval. This type of range-based evaluation is often more reliable and easier to handle. A 50 % quantile value ($Q_{0.5}$) indicates that almost 50 % of the values in a distribution are smaller than $Q_{0.5}$. It is also referred to as median value. Similarly, a 90 % quantile ($Q_{0.9}$) indicates that 90 % of the values in a distribution are smaller than $Q_{0.9}$.

```
VAR CONSTANT
     cInitQuantiles : ST_CM_Quantiles_InitPars := ( nChannels := (4096/2+1),
                         fMinBinned := -10,
                         fMaxBinned := 10,
                         nBins := 100,
                         nMaxQuantiles := 2);
END_VAR
```

In the program the quantiles are configured as follows:

```
(*--------- Configure quantile args ---------*)
IF bConfigureQuantile THEN
     FOR nChannel := 1 TO (cFFTLength/2+1) DO
          aQuantilesArg[nChannel,1]:= 0.50; // 50% quantile
          aQuantilesArg[nChannel,2]:= 0.90; // 90% quantile
     END_FOR
     fbQuantiles.Configure( pArg := ADR(aQuantilesArg), nArgSize := SIZEOF(aQuantilesArg));
     bConfigureQuantile := FALSE;
END_IF
```

Here [▶ 148] you can find a more detailed description of the function block. Note that the parameters fMinBinned and fMaxBinned define the expected input signal range and nBins indicates the number of Bins into which the signal range is divided. These parameters depend on the respective input signal. The signal state is classified based on the quantiles information. The discrete function block can process several channels simultaneously, therefore the quantile output is sent directly to the block. The classifier is set to distinguish between three states and to display the corresponding state via the nMaxClasses parameter.

```
VAR CONSTANT
     cInitClassification    : ST_CM_DiscreteClassification_InitPars := (nChannels:= (4096/2+1),
                                          nMaxClasses := 3);
END_VAR
```

Remark: The output of the quantiles function block is a 2D array, which in this case is the number of spectral lines over the number of quantiles. But the discrete classifier only allows a one-dimensional array, which contains the number of input channels. In order to avoid a dimension conflict, the buffer converter of FB_CMA_BufferConverting should be used. This function block converts a two-dimensional multi-array to a one-dimensional array without any data loss. The code snippet describes the corresponding application.

```
VAR CONSTANT
     cInitBuffer       : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL,
                              nDims := 1,
                              aDimSizes := [(4096/2+1)]);
END_VAR
VAR
     fbBufferConverter : FB_CMA_BufferConverting := (stInitPars := cInitBuffer, nOwnID := eID_Buffer
Converter, aDestIDs := [eID_Classify]);
END_VAR
```

The buffer converter calls a method:

```
fbBufferConverter.Copy1D(nWorkDimIn := 0,
               nWorkDimOut := 0,
               nElements := 0,
               pStartIndexIn := 0,
               pStartIndexOut := 0,
               nOptionPars := 0);
```

Further information on this function block can be found under FB_CMA_BufferConverting [▸ 81]. To complete the function block configuration, each sink function block must be linked to PLC arrays with correct dimensions.

Step 4: Fine-tuning of the application parameters

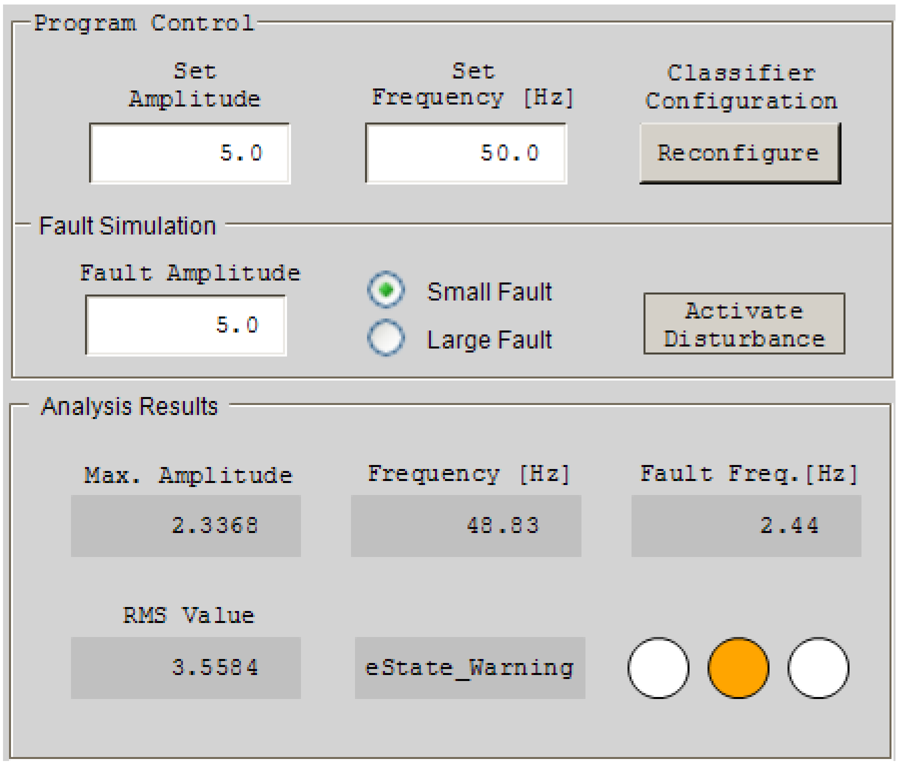
Before starting the analysis, it is important to configure the discrete classifier with regard to its limit values. A classification limit or threshold value enables the discrete classifier to monitor incoming channels continuously and determine whether one of the input channels exceeds this threshold value. The threshold values depend on the respective application, the accuracy requirements, the permitted detection tolerances, etc. The aim of this tutorial is to detect small errors, which are comparable to random noise, and also large-sized errors, which occur at a specific frequency (e.g. 200 Hz). The threshold values fWarning and fAlarm are determined. If the amplitude of the input channels exceeds fWarning, the general state switches to warning state. If fAlarm is exceeded, an alarm message is issued. If the threshold values are not exceeded, the channel state is in the normal range.

```
(*--------- Configure classifier args ---------*)
IF bConfigureClassifier THEN

   fWarning := (fMonitoringLevel/100)*1.5;
   fAlarm := (fMonitoringLevel/100)*2.5;

   fbTeachTimer(IN := TRUE, PT := T#15S);
   IF fbTeachTimer.Q THEN

      fbTeachTimer(IN := FALSE);

      FOR nChannel := 1 TO (cFFTLength/2+1) DO
     aClassArgs[nChannel, 1] := (fMonitoringLevel/100)*aQuantilesCopy[nChannel,1];
     aClassArgs[nChannel, 2] := fWarning*aQuantilesCopy[nChannel,1];
     aClassArgs[nChannel, 3] := fAlarm*aQuantilesCopy[nChannel,1];
      END_FOR

      fbClassification.Configure(pArg := ADR(aClassArgs), nArgSize := SIZEOF(aClassArgs));
      bConfigureClassifier := FALSE;

   END_IF

END_IF
```

The code snippet above describes the configuration of this discrete classifier, so that a timer block allows a normal operating window to pass through a so-called teaching phase, during which the discrete classifier is configured. It is assumed that the input signal behaves normally during this time, i.e. within the permissible range. The warning threshold is 150 % of the “normal” 50 % quantile, the alarm threshold is 250 % of the normal 50 % quantile. Since the 50% quantile describes the average behavior, this threshold value configuration is suitable for applications whose inputs only have few outliers. The 90 % quantile can also be determined as threshold value, if it is assumed that the input signal is likely to fluctuate strongly. It is also possible to configure another variable, fMonitoringLevel, which can be used to apply a certain tolerance range around the permissible value, in order to control the number of false alarms. This parameter can be used to fine-tune the threshold values. Note that the threshold values for the discrete classifier can be specified individually for all input channels.

Step 5: Starting the application

Compile the code, download it to the target system and start the PLC, in order to execute the tutorial. A small prepared visualization, referred to as Dashboard, can be found in the Solution Explorer under the VISU node, which can be used for a quick test. For the simulation the input signal is linked to a function generator, which was configured for generating a sinusoidal 50 Hz signal with an amplitude of 5. Other available signals such as pulse, triangle or saw tooth, or indeed a hardware module such as EL3632, can be applied to the input. Once the application has been started, the display fields show the maximum amplitude, the RMS amplitude and the frequency at the maximum amplitude of the PLC in real-time.



The diagram illustrates that the state of the application is shown in the corresponding display field. A small-sized error can be simulated by pressing the Add Fault button. You can see how the RMS value of the input signal slowly increases beyond the threshold value and how the state changes accordingly. To simulate a large-sized error, press the Small/Large button. Similar to the previous error the RMS value will increase, but this time the “Fault Frequency” field shows the frequency of the fault signal, in this case 200 Hz.

Step 6: Monitoring

Once the PLC has started, the display fields show the current values. Initially the Reconfigure button is shown as pressed, and the signal in the right-hand corner is disabled. The means that the limit values are in the process of being specified for the discrete classifier. Once the configuration is complete, the Reconfigure button resets itself, and the machine status is shown as “normal state”. The signal switches to green, which has the same meaning.

To simulate an error, leave the option field at “Small Fault” and press the Activate Disturbance button. The machine will switch between “Normal” and “Warning” state, and the signal switches between green and orange. If a large error is simulated by switching the option field, the machine state switches to “Alarm” state, and the signal switches to red. To prevent the fault, release the Activate Disturbance button. The signal state returns to green. Note that a change in the signal amplitude also results in an error state. If this is undesirable, press the Reconfigure button again, in order to adjust the discrete classifier to this new signal state.

Requirements

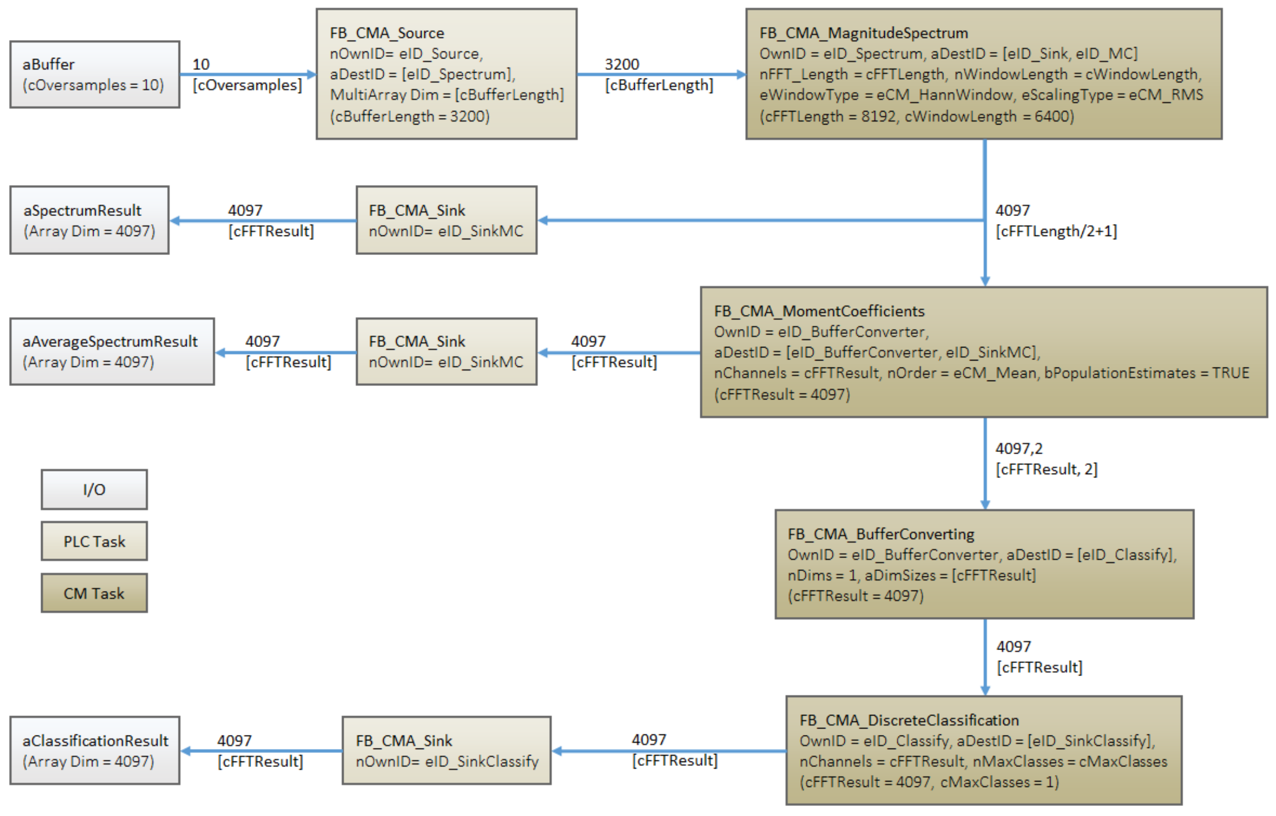
Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.13 Threshold value consideration for averaged magnitude spectra

This sample illustrates an analysis chain for threshold value consideration, as explained in the Frequency analysis [▶ 35] application concept. The analysis chain implements the calculation of a magnitude spectrum, averaging of several magnitude spectra and subsequent threshold value consideration for exemplary frequency bands. For a better illustration of the threshold value consideration around 50 Hz, the scope is limited to the frequency range from 0 Hz to 100 Hz.

The sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649236875.zip

Block diagram for the analysis chain:


aBuffer
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source,
aDestID = [eID_Spectrum],
MultiArray Dim = [cBufferLength]
(cBufferLength = 3200)
3200
[cBufferLength]
FB_CMA_MagnitudeSpectrum
OwnID = eID_Spectrum, aDestID = [eID_Sink, eID_MC]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
eWindowType = eCM_HannWindow, eScalingType = eCM_RMS
(cFFTLength = 8192, cWindowLength = 6400)
aSpectrumResult
(Array Dim = 4097)
4097
[cFFTResult]
FB_CMA_Sink
nOwnID= eID_SinkMC
4097
[cFFTLength/2+1]
aAverageSpectrumResult
(Array Dim = 4097)
4097
[cFFTResult]
FB_CMA_Sink
nOwnID= eID_SinkMC
4097
[cFFTResult]
FB_CMA_MomentCoefficients
OwnID = eID_BufferConverter,
aDestID = [eID_BufferConverter, eID_SinkMC],
nChannels = cFFTResult, nOrder = eCM_Mean, bPopulationEstimates = TRUE
(cFFTResult = 4097)
4097,2
[cFFTResult, 2]
I/O
PLC Task
CM Task
FB_CMA_BufferConverting
OwnID = eID_BufferConverter, aDestID = [eID_Classify],
nDims = 1, aDimSizes = [cFFTResult]
(cFFTResult = 4097)
4097
[cFFTResult]
aClassificationResult
(Array Dim = 4097)
4097
[cFFTResult]
FB_CMA_Sink
nOwnID= eID_SinkClassify
4097
[cFFTResult]
FB_CMA_DiscreteClassification
OwnID = eID_Classify, aDestID = [eID_SinkClassify],
nChannels = cFFTResult, nMaxClasses = cMaxClasses
(cFFTResult = 4097, cMaxClasses = 1)

Program parameters

The table below shows a list of important parameters for the configuration of the function blocks that are used.

FFT length	8192
Window size	6400
Buffer size	3200
Window type	eCM_HannWindow
Scaling type	eCM_RMS
Coefficient order	eCM_Mean
Maximum number of classes	1

Requirements

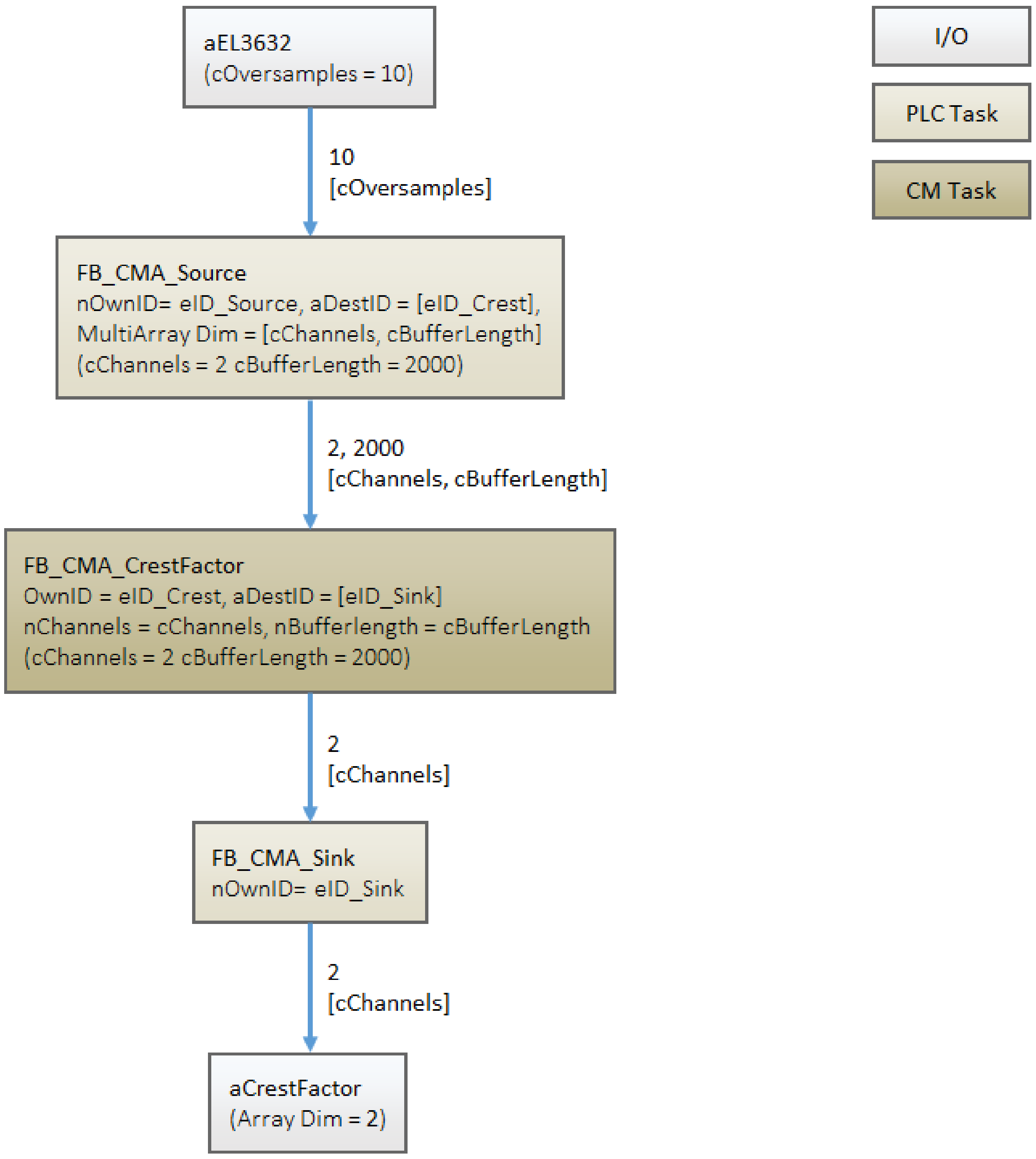
Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.14 Crest factor

This sample calculates the crest factor for an input signal. Although the function block FB_CMA_CrestFactor [▶ 83] is able to process several channels, for the purpose of illustration only a single channel will be considered. The block diagram below shows the analysis chain implemented in the program.

The source code for the sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649285259.zip

Block Diagram


aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_Crest],
MultiArray Dim = [cChannels, cBufferLength]
(cChannels = 2 cBufferLength = 2000)
2, 2000
[cChannels, cBufferLength]
FB_CMA_CrestFactor
OwnID = eID_Crest, aDestID = [eID_Sink]
nChannels = cChannels, nBufferlength = cBufferLength
(cChannels = 2 cBufferLength = 2000)
2
[cChannels]
FB_CMA_Sink
nOwnID= eID_Sink
2
[cChannels]
aCrestFactor
(Array Dim = 2)
I/O
PLC Task
CM Task

Program parameters

The table below shows a list with important configuration parameters for the function block for calculating the crest factor.

Channels	2
Buffer size	1600

Global constants

These parameters are defined in the global variable list as constants.

```
VAR_GLOBAL CONSTANT
    cOversamples  : UDINT := 10;    // oversampling factor
    cChannels          : UDINT := 2;     // number of channels
    cBufferLength      : UDINT := 2000;  // size of buffer
END_VAR
```

Code for Control Task

Following code snippet shows the declaration in MAIN program:

```
VAR CONSTANT
     cInitSource       : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL, nDims := 2,
 aDimSizes := [cChannels, cBufferLength]);
END_VAR

VAR
     nInputSelection : UDINT := 1;
     aCrestFactor      : ARRAY[1..cChannels] OF LREAL;
     nSampleIdx    : UDINT;
     nChannelIdx       : UDINT;
     aEl3632 AT %I*    : ARRAY[1..cChannels] OF ARRAY[1..cOversamples] OF INT; // input from hardwar
e e.g. EL3632
     aBuffer  : ARRAY[1..cChannels] OF ARRAY[1..cOversamples] OF LREAL;
     fbSource      : FB_CMA_Source := (stInitPars := cInitSource, nOwnID := eID_Source, aDestIDs :=
[eID_Crest]); // Initialize source
     fbSink        : FB_CMA_Sink := (nOwnID := eID_Sink);
END_VAR
```

Method calls in Main program:

```
// Collect data in a source
fbSource.Input2D(pDataIn  := ADR(aBuffer),
          nDataInSize := SIZEOF(aBuffer),
          eElementType := eMA_TypeCode_LREAL,
          nWorkDim0  := 0,
          nWorkDim1  := 1,
          pStartIndex := 0,
          nOptionPars := 0 );

// Push results to sink
fbSink.Output1D(pDataOut  := ADR(aCrestFactor),
          nDataOutSize  := SIZEOF(aCrestFactor),
          eElementType  := eMA_TypeCode_LREAL,
          nWorkDim := 0,
          nElements  := 0,
          pStartIndex  := 0,
          nOptionPars  := 0,
          bNewResult => bNewResult);
```

Code for CM Task

Declaration in MAIN_CM program:

```
VAR CONSTANT
     cInitCrest : ST_CM_CrestFactor_InitPars := ( nChannels := cChannels, nBufferLength := cBufferLe
ngth );
END_VAR

VAR
     fbCrest : FB_CMA_CrestFactor := (stInitPars := cInitCrest, nOwnID:= eID_Crest, aDestIDs:= [eID_
Sink]); // Initialize crest
END_VAR
```

Method calls in MAIN_CM program:

```
fbCrest.Call();
```

The result of the sample code can be checked for a sinusoidal signal of arbitrary amplitude and frequency as the input signal. The crest factor, in this case first element of aCrestFactor, must be equal to 3.01 dB.

Requirements

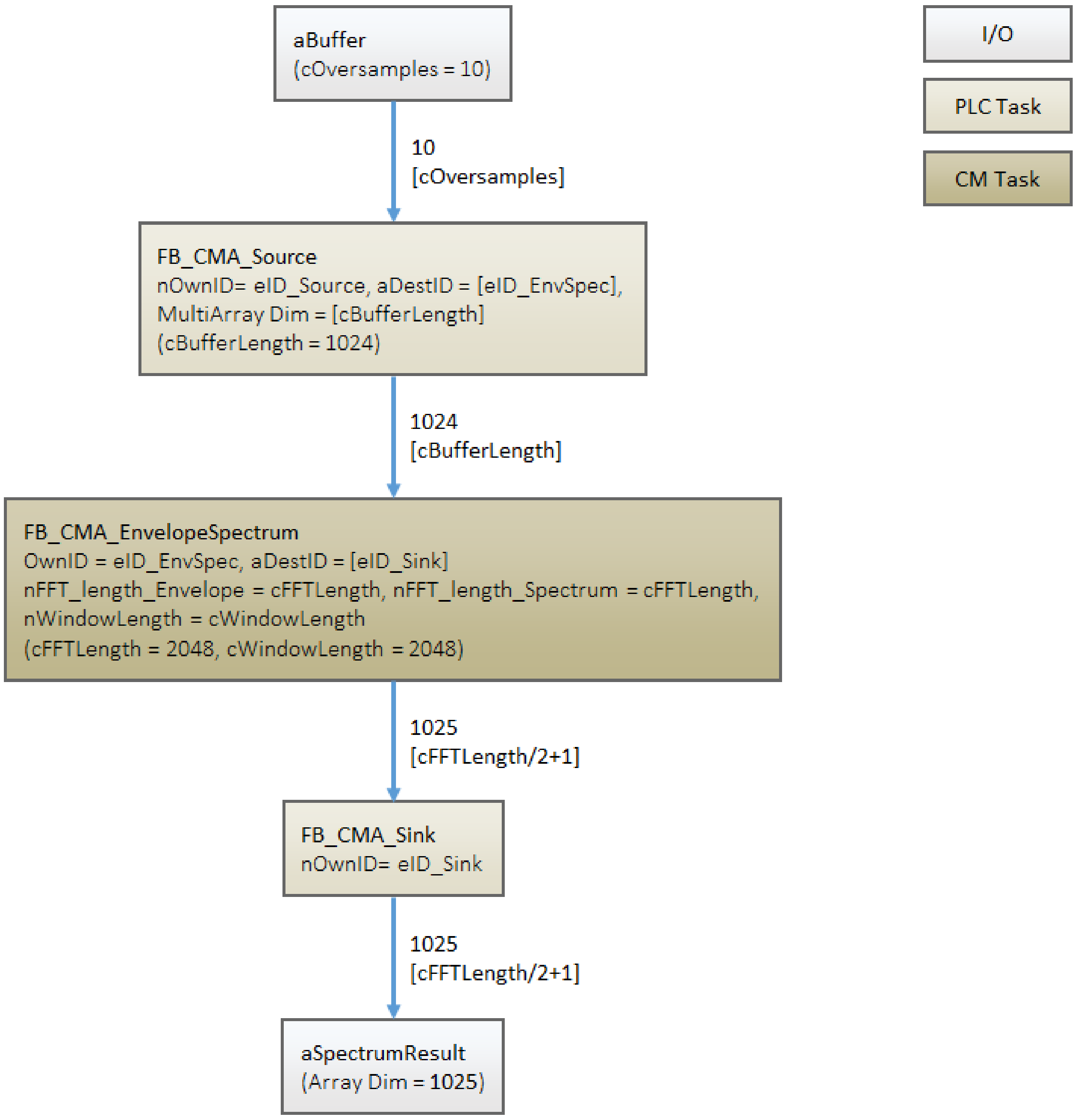
Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.15 Envelope spectrum

The sample illustrates a calculation implementation for an envelope spectrum with the function block FB_CMA_EnvelopeSpectrum [▸ 112]. The input signal is generated with a function generator. It corresponds to the superposition of two sine waves with 120 Hz and 230 Hz. For a better illustration of the result, the scope is limited to the frequency range from 0 Hz to 300 Hz.

The sample is available for download from here:
http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/9007202649235211.zip

Block diagram


aBuffer
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source, aDestID = [eID_EnvSpec],
MultiArray Dim = [cBufferLength]
(cBufferLength = 1024)
1024
[cBufferLength]
FB_CMA_EnvelopeSpectrum
OwnID = eID_EnvSpec, aDestID = [eID_Sink]
nFFT_length_Envelope = cFFTLength, nFFT_length_Spectrum = cFFTLength,
nWindowLength = cWindowLength
(cFFTLength = 2048, cWindowLength = 2048)
1025
[cFFTLength/2+1]
FB_CMA_Sink
nOwnID= eID_Sink
1025
[cFFTLength/2+1]
aSpectrumResult
(Array Dim = 1025)
I/O
PLC Task
CM Task

Program parameters

The table below shows a list with important configuration parameters for the function block for calculating the envelope spectrum.

FFT length envelope	2048
FFT length spectrum	2048
Window size	2048
Conversion to decibels	FALSE
Window type	eCM_HannWindow
Scaling type	eCM_RMS

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

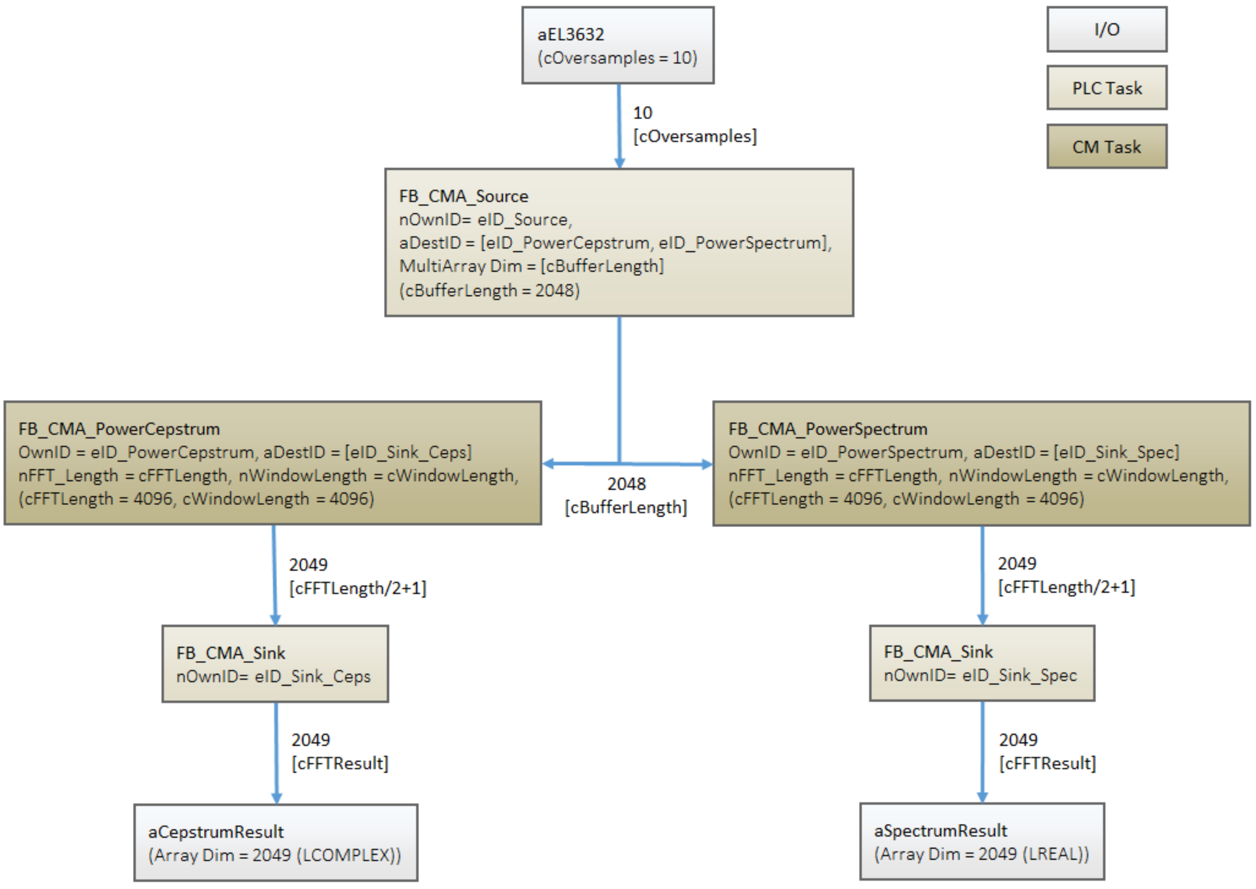
6.16 Power cepstrum

This sample implements the calculation of power cepstrum and power spectrum. The signal under consideration is generated by amplitude modulation based on two sine waves, a carrier frequency and a modulation frequency.

The source code for the sample is available for download from here:

http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/5261531147.zip

Block diagram


aEL3632
(cOversamples = 10)
10
[cOversamples]
FB_CMA_Source
nOwnID= eID_Source,
aDestID = [eID_PowerCepstrum, eID_PowerSpectrum],
MultiArray Dim = [cBufferLength]
(cBufferLength = 2048)
I/O
PLC Task
CM Task
FB_CMA_PowerCepstrum
OwnID = eID_PowerCepstrum, aDestID = [eID_Sink_Ceps]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
(cFFTLength = 4096, cWindowLength = 4096)
2048
[cBufferLength]
FB_CMA_PowerSpectrum
OwnID = eID_PowerSpectrum, aDestID = [eID_Sink_Spec]
nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
(cFFTLength = 4096, cWindowLength = 4096)
2049
[cFFTLength/2+1]
2049
[cFFTLength/2+1]
FB_CMA_Sink
nOwnID= eID_Sink_Ceps
FB_CMA_Sink
nOwnID= eID_Sink_Spec
2049
[cFFTResult]
2049
[cFFTResult]
aCepstrumResult
(Array Dim = 2049 (LCOMPLEX))
aSpectrumResult
(Array Dim = 2049 (LREAL))

Program parameters

The table below shows a list of important parameters for the configuration of the function blocks that are used.

FFT length	4096
Window size	4096
Buffer size	2048

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

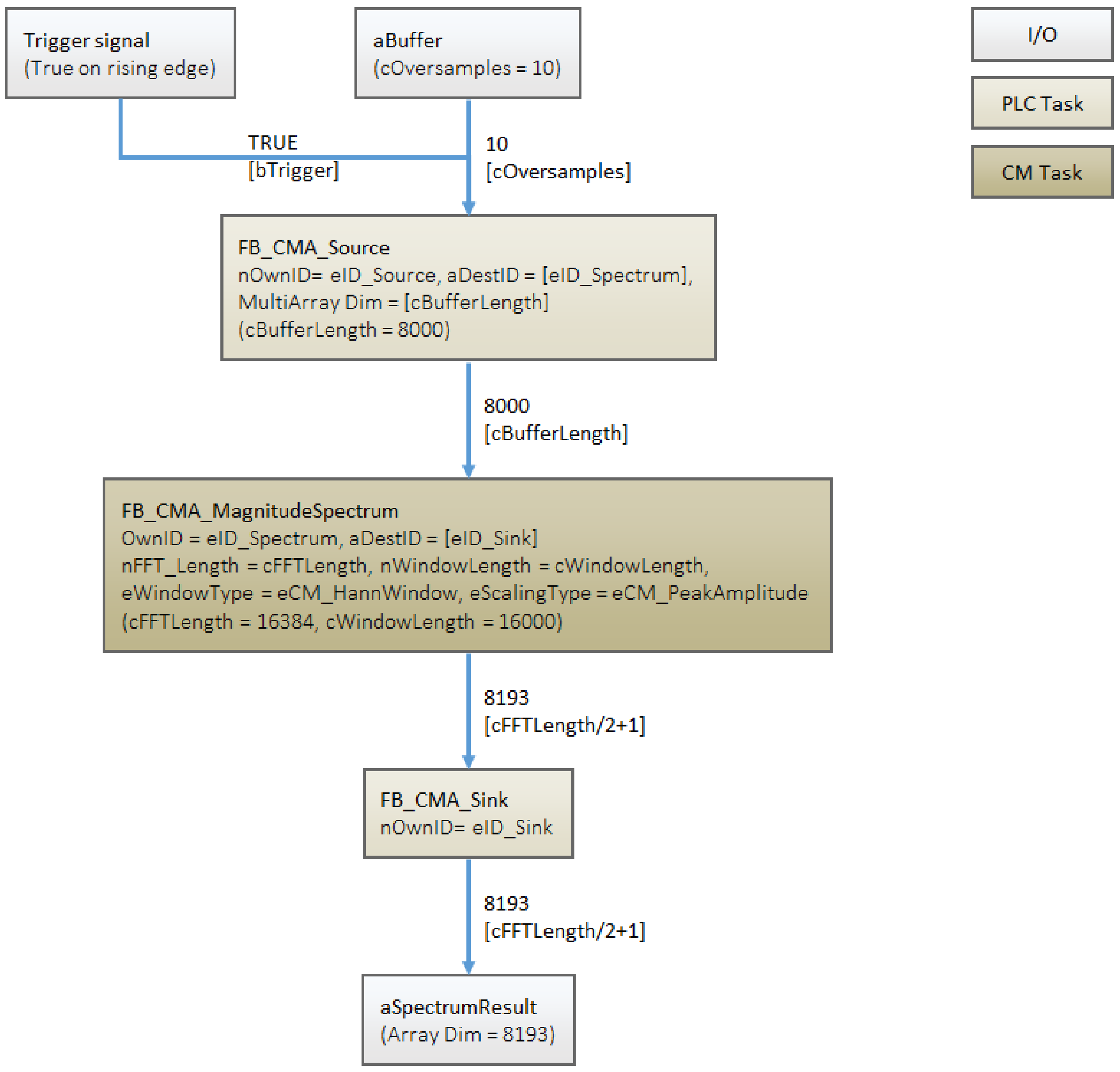
6.17 Event-based frequency analysis

This sample implements an event based frequency analysis. The generated signal consists of a noisy sine signal with a frequency of 200 Hz and pure noise, which alternate every two seconds. Buffering of the signal begins when a rising edge is detected in the (generated) input signal. The collected data are then relayed via FB_CMA_Source [▶ 158] to the function block FB_CMA_MagnitudeSpectrum [▶ 127].

The source code for the sample is available for download from here:

http://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/zip/5261425419.zip

Block diagram



Program parameters

The table below shows a list of important parameters for the configuration of the magnitude spectrum function block.

FFT length	16384
Window size	16000
Buffer size	8000
Window type	eCM_HannWindow
Scaling type	eCM_ PeakAmplitude

Event-based buffering of the input signal

The program block `CollectData` controls the event-based sampling of the input signal. The input parameters are defined as follows:

```
PROGRAM CollectData
VAR_INPUT
    bTrigger     : BOOL;     // Trigger signal, start with rising edge
END_VAR
VAR_IN_OUT
    aInputSignal : ARRAY[1..cOversamples] OF LREAL;  // input time signal
END_VAR
```

The inverse of the trigger signal `bTrigger_` and the current state of buffer are stored locally.

```
VAR
    bTrigger_           : BOOL := FALSE;
    nSourceState        : UINT := 0;
    nActualBuffersSent : ULINT := 0;
    nBuffersToSent     : ULINT := 2;

    // ...

END_VAR
```

Event-controlled sampling of the signal takes place when the trigger signal has a rising edge and the buffer is ready, i.e. state `0`.

```
IF (bTrigger AND NOT bTrigger_) AND nSourceState = 0 THEN
    nActualBuffersSent := fbSource.nCntResults;  // check number of sent MultiArrays from fbSource
    fbSourceState := 1;
END_IF
bTrigger_ := bTrigger;
```

The following code shows the actual event-based buffering of the signal via the source function block.

```
CASE nSourceState OF

    1: // if <nBuffersToSent> MultiArrays has been sent, stop buffering

       fbSource.Input1D( pDataIn      := ADR(aInputSignal),
                         nDataInSize  := SIZEOF(aInputSignal),
                         eElementType := eMA_TypeCode_LREAL,
                         nWorkDim     := 0,
                         pStartIndex  := 0,
                         nOptionPars := cCMA_Option_MarkInterruption );

       IF (fbSource.nCntResults-nActualBuffersSent) = nBuffersToSent THEN
           nSourceState := 2;
       END_IF

    2: // reset Source Buffer and wait for next trigger hit

       fbSource.ResetData();
       nSourceState := 0;

END_CASE;
```

The buffered signal data is subsequently relayed to the magnitude spectrum function block. The buffered signal is processed in the same way as shown in the Magnitude spectrum: [▸ 192] sample.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4018	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

7 Appendix

7.1 Error Codes Overview

The following error codes may occur.

16#9811_0000 - 16#9811_FFFF	listed in TwinCAT (ADS) Error Codes [▸ 225] (there without high-order WORD). Further notes below on this page.
16#9851_0000 - 16#9851_FFFF	Condition Monitoring Error Codes are listed under E_CM_ErrorCode [▸ 167]
16#9852_0000 - 16#9852_0FFF	Condition Monitoring Analysis Error Codes are listed in E_CMA_ErrorCodes [▸ 171]
16#9871_0000 - 16#9871_FFFF	MultiArray Error Codes can be found in E_MA_ErrorCode [▸ 172]

If an error occurs during initialization, the function block cannot be used.

Further information on standard TwinCAT Error Codes:

error value	symbol	Error description	Remedy option
16#9811_070A	NOMEMORY	No memory	Incorrect memory settings => increase router memory (see chapter Memory Management [▸ 62])
16#9811_0719	TIMEOUT	Device has a timeout	A timeout may occur during buffer memory transfers. Usually this is non-critical for the CM analysis chain. The response to the error depends on the type of algorithm and the precise location where the error occurred. The timeout input should only be increased if it matches the task cycle time. See section Parallel processing [▸ 66].

Error handling

Error codes are returned by type HRESULT.

declaration	error	ok	ok but with info	check functions
hrErrorCode :HRESULT;	< 0	>= 0	> 0	SUCCEEDED(),FAILED()

A test for non-zero value is insufficient for values of type HRESULT.

In some cases error handling with error codes is not the best choice, particularly if the actions result in an undefined value with regard to non-standard, but possible input data. Or if values were excluded from the process. In this case missing values and partially undefined results can be described by the special constant NaN (see chapter NaN values [▸ 65]). This is used in case of errors whose appearance does not depend on the program logic, but on certain input data.

7.2 ADS Return Codes

Error codes: 0x000 [▶ 225]..., 0x500 [▶ 225]..., 0x700 [▶ 225]..., 0x1000 [▶ 227]...

Global Error Codes

Hex	Dec	Description
0x0	0	no error
0x1	1	Internal error
0x2	2	No Rtime
0x3	3	Allocation locked memory error
0x4	4	Insert mailbox error
0x5	5	Wrong receive HMSG
0x6	6	target port not found
0x7	7	target machine not found
0x8	8	Unknown command ID
0x9	9	Bad task ID
0xA	10	No IO
0xB	11	Unknown ADS command
0xC	12	Win 32 error
0xD	13	Port not connected
0xE	14	Invalid ADS length
0xF	15	Invalid ADS Net ID
0x10	16	Low Installation level
0x11	17	No debug available
0x12	18	Port disabled
0x13	19	Port already connected
0x14	20	ADS Sync Win32 error
0x15	21	ADS Sync Timeout
0x16	22	ADS Sync AMS error
0x17	23	ADS Sync no index map
0x18	24	Invalid ADS port
0x19	25	No memory
0x1A	26	TCP send error
0x1B	27	Host unreachable
0x1C	28	Invalid AMS fragment

Router Error Codes

Hex	Dec	Name	Description
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	No locked memory can be allocated
0x501	1281	ROUTERERR_RESIZEMEMORY	The size of the router memory could not be changed
0x502	1282	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages. The current sent message was rejected
0x503	1283	ROUTERERR_DEBUGBOXFULL	The mailbox has reached the maximum number of possible messages. The sent message will not be displayed in the debug monitor
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	Unknown port type
0x505	1285	ROUTERERR_NOTINITIALIZED	Router is not initialized
0x506	1286	ROUTERERR_PORTALREADYINUSE	The desired port number is already assigned
0x507	1287	ROUTERERR_NOTREGISTERED	Port not registered
0x508	1288	ROUTERERR_NOMOREQUEUES	The maximum number of Ports reached
0x509	1289	ROUTERERR_INVALIDPORT	Invalid port
0x50A	1290	ROUTERERR_NOTACTIVATED	TwinCAT Router not active

General ADS Error Codes

Hex	Dec	Name	Description
0x700	1792	ADSERR_DEVICE_ERROR	error class <device error>
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by server

Hex	Dec	Name	Description
0x702	1794	ADSERR_DEVICE_INVALIDGRP	invalid index group
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	invalid index offset
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	reading/writing not permitted
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	parameter size not correct
0x706	1798	ADSERR_DEVICE_INVALIDDATA	invalid parameter value(s)
0x707	1799	ADSERR_DEVICE_NOTREADY	device is not in a ready state
0x708	1800	ADSERR_DEVICE_BUSY	device is busy
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	invalid context (must be in Windows)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	out of memory
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	invalid parameter value(s)
0x70C	1804	ADSERR_DEVICE_NOTFOUND	not found (files, ...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	syntax error in command or file
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	objects do not match
0x70F	1807	ADSERR_DEVICE_EXISTS	object already exists
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	symbol not found
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVAL	symbol version invalid
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	server is in invalid state
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	no more notification handles
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	size for watch too big
0x718	1816	ADSERR_DEVICE_NOTINIT	device not initialized
0x719	1817	ADSERR_DEVICE_TIMEOUT	device has a timeout
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	query interface failed
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	wrong interface required
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	class ID is invalid
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	object ID is invalid
0x71E	1822	ADSERR_DEVICE_PENDING	request is pending
0x71F	1823	ADSERR_DEVICE_ABORTED	request is aborted
0x720	1824	ADSERR_DEVICE_WARNING	signal warning
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	invalid array index
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	symbol not active
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	access denied
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	missing license
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	license expired
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	license exceeded
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	license invalid
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	license invalid system id
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	license not time limited
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	license issue time in the future
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOLONG	license time period to long
0x72c	1836	ADSERR_DEVICE_EXCEPTION	exception occured during system start
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	invalid signature
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	public key certificate
0x740	1856	ADSERR_CLIENT_ERROR	Error class <client error>
0x741	1857	ADSERR_CLIENT_INVALIDPARM	invalid parameter at service
0x742	1858	ADSERR_CLIENT_LISTEMPTY	polling list is empty
0x743	1859	ADSERR_CLIENT_VARUSED	var connection already in use
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	invoke ID in use
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	timeout elapsed
0x746	1862	ADSERR_CLIENT_W32ERROR	error in win32 subsystem
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	ads-port not opened
0x750	1872	ADSERR_CLIENT_NOAMSADDR	internal error in ads sync

Hex	Dec	Name	Description
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	hash table overflow
0x752	1874	ADSERR_CLIENT_ADDHASH	key not found in hash
0x753	1875	ADSERR_CLIENT_REMOVEHASH	no more symbols in cache
0x754	1876	ADSERR_CLIENT_NOMORESYM	invalid response received
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	sync port is locked

RTime Error Codes

Hex	Dec	Name	Description
0x1000	4096	RTERR_INTERNAL	Internal fatal error in the TwinCAT real-time system
0x1001	4097	RTERR_BADTIMERPERIODS	Timer value not vaild
0x1002	4098	RTERR_INVALIDTASKPTR	Task pointer has the invalid value ZERO
0x1003	4099	RTERR_INVALIDSTACKPTR	Task stack pointer has the invalid value ZERO
0x1004	4100	RTERR_PRIOEXISTS	The demand task priority is already assigned
0x1005	4101	RTERR_NOMORETCB	No more free TCB (Task Control Block) available. Maximum number of TCBs is 64
0x1006	4102	RTERR_NOMORESEMAS	No more free semaphores available. Maximum number of semaphores is 64
0x1007	4103	RTERR_NOMOREQUEUES	No more free queue available. Maximum number of queue is 64
0x100D	4109	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied
0x100E	4110	RTERR_EXTIRQNOTDEF	No external synchronization interrupt applied
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	The apply of the external synchronization interrupt failed
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in system BIOS
0x1019	4121	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension
0x101A	4122	RTERR_VMXENABLEFAILS	Enabling Intel VT-x fails

TCP Winsock Error Codes

Hex	Dec	Description
0x274d	10061	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
0x2751	10065	No connection could be made because the target machine actively refused it. This error normally occurs when you try to connect to a service which is inactive on a different host - a service without a server application.
0x274c	10060	No route to a host. A socket operation was attempted to an unreachable host
		Further Winsock error codes: Win32 Error Codes

7.3 Spectrum Scaling Options

This page provides an overview of the scaling options for spectral calculations. The following table shows symbols and important parameters for the scaling.

Symbol	Function block parameters	Meaning
N	nFFT_Length	Number of input values of the FFT
F_s		Sampling frequency
Σw_n	eWindowFunction, nWindowLength	Sum of the values of the window function
Σw_n^2	eWindowFunction, nWindowLength	Sum of the squared values of the window function
SQRT(x)		Square root of X
MAX($\|X_n\|$)		Maximum of the spectral values X_n
RMS(x_n) = SQRT([Σ $(x_n)^2$] / N)		Root mean square (effective) value of a signal

Symbol	Function block parame-ters	Meaning
PSD(X_n)		Power Spectral Density
LSD(X_n)		Linear Spectral Density
A		Amplitude of a reference sine signal

The following table lists default scaling options (of type E_CM_ScalingType [▸ 165]), which can be selected by the blocks FB_CM_PowerSpectrum [▸ 142] and FB_CMA_MagnitudeSpectrum [▸ 127] and blocks derived from these. The resulting factors do not have to be evaluated by the user. They are given in the second column in order to be able to include further parameters if necessary. The values x_n denote the input values of the function block and the values X_k the spectral value for the frequency channel k resulting from the scaling.

Scaling option	Contained factor	Description
Deterministic signals		
`eCM_PeakAmplitude`	2 / Σw_n	This scaling adapts the magnitude values in such a way that an input sine signal with the amplitude A reaches a maximum value of A. The result is independent of the type of window function. The unit of the magnitude value is the same as the unit of the input signal. MAX($\|X_k\|$) = A However, the maximum values of the spectrum do not enable a robust estimation of the amplitude, since so-called Scalloping Losses may occur.
`eCM_ROOT_POWER_SUM`	2 / SQRT(N * $\Sigma w_n{}^2$)	This scaling adapts the spectral values in such a way that for an input sine signal with the amplitude A, the square of the sum of the power values has the value A. Accordingly the square root of the sum of the squares of the magnitude values can also be used. The result is thus equal to the RMS value of the input signal multiplied by SQRT(2). SQRT($\Sigma\|X_k\|^2$) = A This scaling is suitable for the evaluation of narrow-band signals. Since the summing via neighboring frequency bands reduces scalloping losses, it is considerably more robust than `eCM_PeakAmplitude`.
`eCM_RMS`	SQRT(2/N * $\Sigma w_n{}^2$)	This scaling results in power values and the square root of their sum is equal to the RMS value of the input signal. A sine signal with the amplitude A results in a value of A/SQRT(2): SQRT($\Sigma\|X(k)\|^2$) = RMS(x_n) = A * SQRT(1/2) Like `eCM_ROOT_POWER_SUM` this scaling is also robust and suitable for the evaluation of narrow-band signals. In addition the RMS value is also well-defined for broadband signals.
Stochastic and broadband signals		

Scaling option	Contained factor	Description
`eCM_PowerSpectralDensity`	SQRT(2 / Σw_n^2)	This scaling determines the Power Spectral Density (PSD). For broadband and stochastic signals this is independent of the parameters of the FFT and window function. PSD(X_k) = $\lvert X_k \rvert^2/F_S$ In order to determine a physically correct power spectral density, the result must additionally be divided by the sampling rate of the input signal in Hertz. If the input signal has the unit Volt, then the unit 1 V/Hz is obtained for the magnitude and the unit 1 V^2/Hz for the power density. Division by the *root of the sampling rate* must take place for the Linear Spectral Density; the unit is then 1 V/(1 Hz)$^{1/2}$: LSD(X_k) = $\lvert X_k \rvert$/ SQRT(F_S)
`eCM_UnitaryScaling`	SQRT(1 / Σw_n^2)	This scaling determines power densities similar to an FFT, which is divided by the value SQRT(N). It therefore corresponds to a so-called unitary FFT, for which the same factors apply for the transformation and inverse transformation.
Elementary		
`eCM_DiracScaling`	sqrt(N / Σw_n^2)	This scaling normalizes the power spectrum in such a way that the broadband signal is equal to the unscaled FFT (with the definition given above). The influence of window type and window length is thus eliminated. However, the effect of the FFT length N exists just as it does with the unscaled FFT.
`eCM_NoScaling`	1	No scaling. The result consists of the application of the window function (which always has a maximum of one in accordance with convention) followed by the FFT.
`eCM_GainCorrection`	SQRT(Σw_n^2 / $(\Sigma w_n)^2$)	This scaling divides the signal by the processing gain of the window function, which is the reciprocal value of the Effective Noise Bandwidth.

Glossary

Acceleration Spectral Density (ASD)

is the name given to the physical variable represented by the output values of the Fourier transformation if the input signal is an acceleration signal such as is measured, for example, by a piezoelectric vibration pick-up. If integrated over a frequency interval, the acceleration density produces a frequency-specific acceleration in much the same way as the power density. The usual unit is 1 millimetre per second squared per Hertz = 1 mm ⁄ s2 ⁄ Hz.

Aliasing

is an error that occurs if frequencies occur in a signal that are higher than half the sampling rate. In this case the signal from the sampling can no longer be clearly reconstructed (Nyquist theorem). These frequencies are reflected in the spectrum as so-called image frequencies.

Angle of contact

is the angle between the line along which the balls of a ball bearing touch the running surface and the plane that is perpendicular to the axis of the bearing. While the angle of contact is always close to zero in the case of bearings designed exclusively for radial loads, it can be significantly larger with bearings that also bear axial loads. It therefore depends both on the geometry and on the current load on the bearing and has an effect on the observable damage frequencies due to the pitch diameter. These are therefore not constant in the case of bearings for axial loads.

Artefacts

unwanted changes in the signal that result from errors in the processing, for example due to aliasing.

Bessel's correction

correction that takes into account the number of the degrees of freedom when estimating statistical moment coefficients from a series of data. Specifically, for example, the standard deviation is corrected by multiplying it by the factor sqrt(n/(n-1)), the skew by sqrt(n*(n-1)/(n-2)) and so on. The factor is generally negligible if n is a larger number.

Bin

designates one channel of a multi-channel signal output. The designation is used in particular with transformations that convert signals, such as the FFT or the formation of the histogram.

Cepstrum

is a transformation based on frequency analysis that emphasises periodic elements in the spectrum due to harmonics or amplitude modulations. Distinction is made between the power cepstrum and the complex cepstrum.

Circular aliasing

is an artefact that can occur when signals are modified in the frequency domain and then transformed back into the time domain by means of an inverse FFT (Overlap-Add method). The modification can be described as a multiplication in the frequency domain, which generally corresponds to filtering in the time domain. This is equivalent to a cyclically defined folding with the pulse response of the filtering. If the pulse response is too long, then signal portions belonging to the beginning of the time period appear at the end of the section and vice versa. The reason for this is the cyclic definition of the discrete Fourier transformation. Extensive modifications in the frequency domain can thus lead to artefacts. As countermeasure the time signal can be supplemented by zeros before processing (zero padding), so that a reserve is created for the extension of the signal.

Complexity

in this case: specification of the required resources of an algorithm (computing time and, if necessary, memory space). Condition Monitoring functions are called with vastly different data quantities; while a short-term FFT may be called with only 32 values, it may be useful, for example, to calculate a cepstrum for 16000 values. Therefore, in the case of a variable number of input data n, the algorithm is observed to see how it behaves with an increasing amount of data; in computer science this is normally described by the notation O(f(n)) (also called 'Landau Notation'). This notation states that the complexity does not grow significantly faster than a function f(n) as n increases. An algorithm with the computing time complexity O(n) thus requires, for example, eight times the computing time for eight times the data amount n, while an algorithm with the complexity O(n2) already requires sixty-four times the computing time. An FFT of the complexity O(n * log2 n) conversely requires 112 times the computing time for n=16384 compared to n=256. With small amounts of data the computing time is usually dominated by a portion that is independent of the number of input data.

Crest factor

relationship between the peak value and the RMS value of a signal, normally expressed in decibels.

Damage frequencies

are characteristic frequencies that occur when certain machine elements are damaged. For example, certain frequencies are assigned to damage to the rolling elements, inner race, outer race and cage in roller bearings and these frequencies are proportional to the speed of rotation of the axis, depending on the angle of contact.

Decibel or dB

logarithmic scale for evaluating the intensity of oscillations or of intensity ratios. A decibel (symbol dB) is defined as one tenth of the auxiliary unit of measurement Bel. If x is a power value, then the value y in decibels = 20 * log10(x/x0). The value 1 or a defined reference value is used for x0.

FFT

or Fast Fourier Transformation: Fast Fourier Transformation, a calculation method for calculating the discrete Fourier transformation. Strictly speaking several such calculation methods exist, wherein the common implementations permit only power-of-two numbers as the input length (Cooley-Tukey algorithm). The common feature is a complexity of the order O(n * nlog(n)), i.e. the calculation of an FFT with 2048 points is a little more than four times as complex as for 512 points.

Fourier transformation

is a transformation that enables a time signal to be decomposed into different frequency portions, thus forming the basis for many frequency analysis methods. Instead of the continuous Fourier transformation, which represents a continuous function of an infinite signal, the discrete Fourier transformation (DFT) is normally used in practice as it is defined for a discrete, periodic signal. An efficient implementation of the discrete Fourier transformation, which is of great practical importance, is the Fast Fourier Transformation (FFT).

Frequency domain

or frequency space is the name given to the representation of a signal on the basis of the values of the FFT. Since the complex Fourier spectrum of every signal can be clearly represented and can be transformed back into an equivalent time signal without losses, frequency domain and time domain (as so-called ‘orthonormal bases’ in the function space) represent equivalent representations of the same signal. Many operations for the analysis of signals can be performed more simply and efficiently in the frequency domain than in the time domain.

Harmonics

are oscillations that occur as integer multiples of a basic frequency. They are characteristic of pulse-type excitations and non-linear effects at the origin of the oscillation and in this case can typically be recognised by groups of lines in the spectrum with a constant distance between one another.

Hilbert Transformation

transformation that efficiently determines the ninety-degree phase-shifted signal from an oscillation signal. The Hilbert Transformation is used, for example, for the calculation of the analytical signal.

Kurtosis

(sometimes also curtosis or curvature): indicator of the ‘impulsiveness’ or ‘peakness’ of a statistical distribution of values, determined from the fourth central statistical moment. For better evaluation of distributions, often the distance between curtosis of the measured distribution and curtosis of the normal distribution (value is 3) is used. This is then called excess curtosis. A Gaussian distribution accordingly has the excess curtosis zero, a distribution with many outliers achieves a value much greater than zero.

Machine protection

is the name given to methods that aim to automatically switch a plant off as quickly as possible if monitoring parameters exceed a critical threshold. In this way accidents and damage can be avoided.

Moment coefficients

is a collective term for statistical values such as mean value, standard deviation, skew and kurtosis of statistical variables. They are called that because they can be calculated from the central statistical moments of the probability distributions or histograms of these variables.

NaN (Not a Number)

is a symbolic constant that marks invalid or missing values according to the IEC 745 standard. The following points rank among the main characteristics of NaN values: All arithmetic operations that use NaN as input data return NaN as the result. All relational operators =, !=, > < >= <= always return the value False if at least one of the operands is NaN. The standard function isnan or _isnan returns the value True if the argument has the value NaN. The expression isnan(a) is equivalent to the expression !(a == a) or NOT(a = a). The fact that NaN values reproduce themselves when used in further calculations is advantageous in that invalid values cannot be overlooked

Nyquist theorem or sampling theorem

a theorem from communication technology and signal processing that states, slightly simplified, that a continuous signal must be sampled with a frequency greater than double that of the highest frequency contained in the signal so that the original signal can be reconstructed without loss of information or ambiguity from the time-discrete signal obtained in this way. This maximum frequency is called the Nyquist frequency. In practice filters are integrated into most D/A convertors that limit the maximum frequency of the input signal to a value smaller than half the sampling rate.

Overlap-Add method

a method that enables a signal to be decomposed initially into short-term spectra without loss of information, then to process it further (e.g. to filter it) in the frequency domain and then to reconstruct it as a continuous time signal again.

Quantile or percentile

is the designation of a value that is determined from a statistical variable. First of all its empirical frequency distribution (density function) is determined and from this the cumulative frequency distribution (also called cumulative distribution function) is calculated. The value of the percentile q is the maximum value which the random variable reaches in q percent of all cases, but does not exceed. This value is determined by the formation of the inverse function of the cumulative frequency distribution. The only difference between quantiles and percentiles is that quantiles use the decimal fraction instead of the corresponding percentages. The value of the 50-percent percentile is also called the median.

Quefrency

is the name given to the time axis that results from the calculation of the cepstrum. As a ‘scrambled’ reversal of the term ‘Frequency’, the name suggests the operations ‘inversion’ and ‘re-sorting’ which are characteristic of the cepstrum. As a result of two successive Fourier transformations, a transformation into the frequency domain initially results, with the assigned unit 1 Hertz. The second transformation leads in turn to a time domain in which, however, it is no longer the absolute time that lies on the axis, but the periodic durations determined by means of the cepstrum. The unit of quefrency is a second.

RCFA or Root Cause Failure Analysis

name for the analysis for the determination of primary causes of damage. This is of particular importance in the case of roller bearings, since primary damage leads to more complex consequential damage. Determination of the causes allows the emergence of damage to be effectively reduced.

Sampling frequency

is the frequency with which the analog signal is originally sampled and converted into digital values. This conversion takes place in steps of a constant length of time called the sampling period. The inverse value of the sampling period is called the sampling frequency and is expressed in Hertz. See also ‘Nyquist theorem’.

Scalloping

is the effect that the precise spectral value of narrow-band signals (for instance a sine signal or that of a calibrator) depends on which part of the FFT channel the frequency of the channel lies. The extent of the effect depends on the window function.

Skew

measurement of the asymmetry of a statistical distribution, determined from the third central statistical moment. A symmetrical distribution has a skew of zero.

Time domain

denotes the representation of a signal using the temporally sampled values, as is originally available following a measurement. Since the Fourier spectrum of every signal can also be clearly represented and can be transformed back into an equivalent time signal without losses, the time domain and frequency domain (as so-called ‘orthonormal bases’ in the function space) represent equivalent representations of the same signal.

Tooth engagement frequencies

or meshing frequencies denotes the frequency with which the pairs of teeth in a gearbox touch each other. This contact causes the so-called meshing oscillation.

Window functions

functions that are used, for example, in conjunction with a frequency analysis (Welch method) to decompose long input signals without the addition of artificial jumps. As standard the Hann window can be used in almost all cases. The choice of window function affects the frequency and time resolution of the frequency analysis.

Windowing

is the name given to the calculation step of the multiplication by a window function (see above).

Zero Padding

denotes a processing step that is applied when an FFT with a certain length is to be calculated from a smaller number of samples. To do this the values of the time series are filled at the front and rear with zeros until the desired number of the values is attained. This usually requires the windowing of the signal e.g. according to the Welch method, so that no false jumps are created in the time series. Zero padding increases the frequency resolution of an FFT, which is equal to the sampling rate divided by the number of FFT points, but the information content of the original signal is, of course, not increased.