**BECKHOFF** New Automation Technology

Manual | EN

# TF3600

TwinCAT 3 | Condition Monitoring
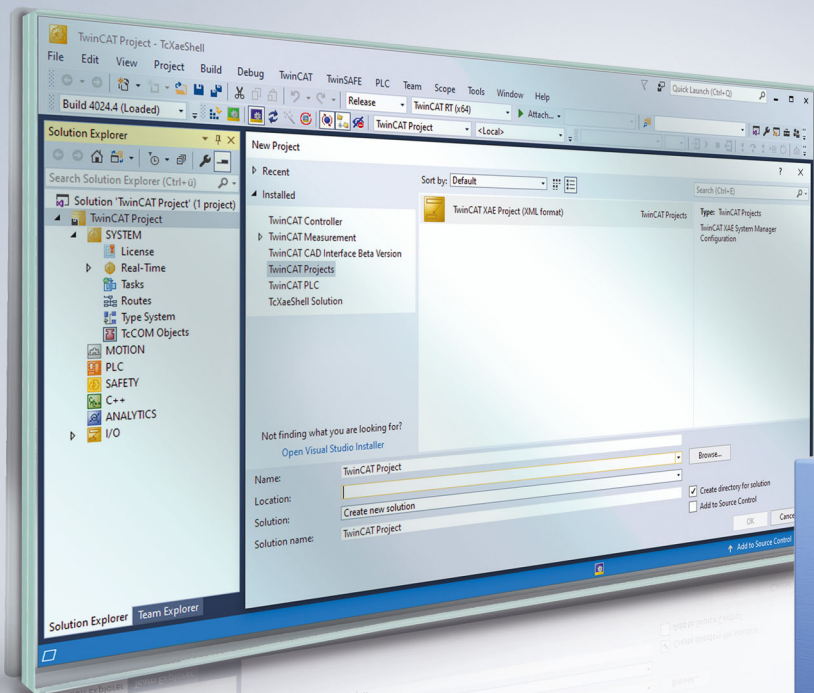
# Table of contents

# 1 Foreword

## 1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all 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 notice.
No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

**Trademarks**

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

**Patents**

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

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

**Copyright**

© Beckhoff Automation GmbH & Co. KG, Germany.
The distribution and reproduction of this document as well as the use and communication of its contents without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

## 1.2 For your safety

**Safety regulations**

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

**Exclusion of liability**

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

**Signal words**

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

**Personal injury warnings**

| ⚠ DANGER |
|---|
| Hazard with high risk of death or serious injury. |

| ⚠ WARNING |
|---|
| Hazard with medium risk of death or serious injury. |

| ⚠ CAUTION |
|---|
| There is a low-risk hazard that could result in medium or minor injury. |

**Warning of damage to property or environment**

| *NOTICE* |
|---|
| The environment, equipment, or data may be damaged. |

**Information on handling the product**

ℹ This information includes, for example:
recommendations for action, assistance or further information on the product.

# 1.3    Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

# 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: http:// www.beckhoff.com/software . 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
- **Driver:** TcCM.sys and TcMultiArray.sys

In addition, the product contains the components of **TF3680 TC3 Filter**. The TF3600 Condition Monitoring Level 1 license contains the TF3680 license, so that all constituent parts of the TC3 Filter product can be used.
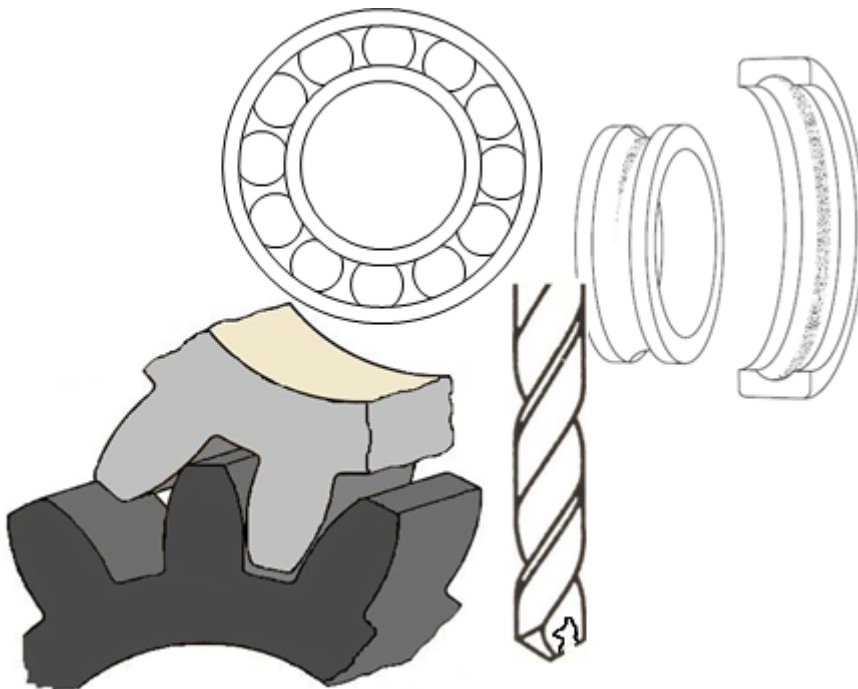
**Product features**

This table shows the functions 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 | ✔ |
| Sparse 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 | ✔ |
| Crest Factor Plus | ✔ |
| 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 | ✔ |
| Vibration assessment (according to ISO 10816-3) | ✔ |
| Digital filter | ✔ (component TC3 Filter) |

# 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:

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

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. Conversely, from the magnitude spectrum |*X(w)*|, i.e. the magnitude of the Fourier-transformed, it is easy to see with appropriate scaling (see Scaling of spectra [▶ 26]), that there are two oscillations – 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: [▶ 303] sample, the magnitude spectrum is calculated and displayed for a signal of this form.
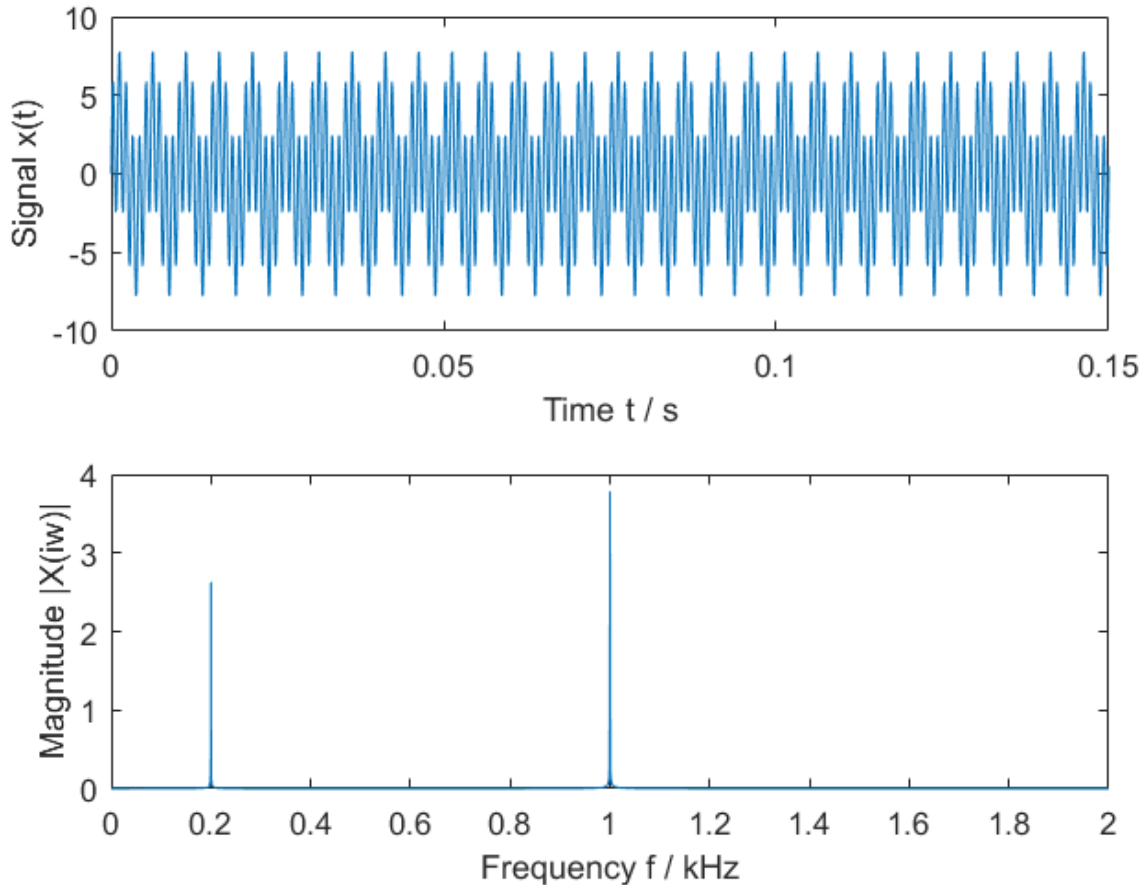
There are two processes that influence the vibration signals originating in a machine during sound transfer. Firstly transfer via machine components that attenuate the vibrations to different degrees depending on the frequency, and secondly superposition with vibrations from other machine components, 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 $c_s$:

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

Sample: 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.

> **ℹ 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.
> The Task Cycle Time can be queried in the PLC via the global data type `PlcTaskSystemInfo`.
> See: INFOSYS Global Data Types

**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.

> **ℹ 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 the ELM3xxx modules have several filter levels that can be designed for anti-aliasing filtering as well as for wanted-signal filtering, see also TC3 Filter Designer.

**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 $\Delta f$ of the discrete frequency axis $k \cdot \Delta f$. 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 vibration. This results in the following relationship between the resolution $\Delta f$ 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 <u>Magnitude spectrum: [▶ 303]</u> sample:

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

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 $\Delta f$. 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 $\Delta f$, it does not change the measuring resolution. A refinement of the measuring resolution can only be realized through a correspondingly long measuring time (often called *WindowLength* in the Condition Monitoring library for FFT-based algorithms). For practical applications, the key factor is usually the frequency resolution of the measurement, which influences the differentiability between two closely adjacent signal frequencies.

**i** ● **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 $\Delta f$ = 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 $\Delta f$ = 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 saved in a PLC array with the index $m$, which runs from 1.. $N$, the following frequency axis results for $X[k]$

```
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 <u>Fourier analysis [▶ 16]</u>.

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

Positive frequencies · Negative frequencies
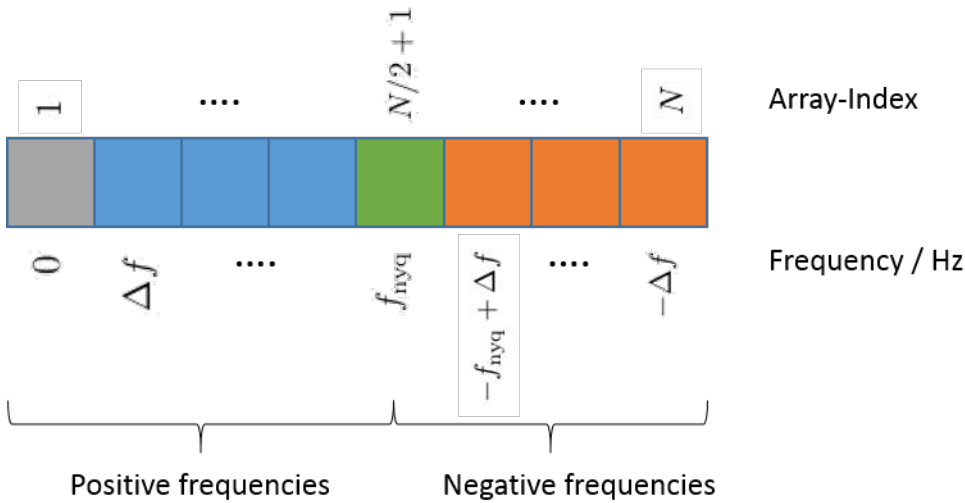
**Efficient calculation through FFT algorithms**

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 $O(N * \log N)$. 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 FB_CMA_MagnitudeSpectrum [▶ 168] or FB_CMA_PowerSpectrum [▶ 197], 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 $k > N/2$ are therefore redundant for real input sequences, the Fourier-transformed for real sequences is usually limited to the first $n/2$ values (applies to $k = 0..N—1$).

**Overview of the calculation of spectral values**

The Condition Monitoring library offers a variety of ways to calculate spectral values. The following table offers an overview. The relationships of the FFT with (overlapping) windows will be considered in detail in the next chapter.

| Function block | Input | | | Output | | | Note |
|---|---|---|---|---|---|---|---|
| | di-men-sion | real | cplx | di-men-sion | Real | cplx | |
| FB_CMA_ComplexFFT [▶ 109] | N | - | ✔ | N | - | ✔ | • FFT for complex-valued input signals<br>• No windowing, overlapping and scaling<br>• Calculation of the entire spectrum |
| FB_CMA_MagnitudeSpectrum [▶ 168] | L | ✔ | - | N/2+1 | ✔ | - | • Absolute value of the spectral values<br>• Windowing and scaling adjustable<br>• Calculation of the entire spectrum |
| FB_CMA_OrderPowerSpectrum [▶ 189] | V, P | ✔ | - | N/2+1 | ✔ | - | • Order spectrum<br>• Windowing and scaling adjustable<br>• Calculation of the entire spectrum |
| FB_CMA_PowerSpectrum [▶ 197] | L | ✔ | - | N/2+1 | ✔ | - | • Performance-related spectral values<br>• Windowing and scaling adjustable<br>• Calculation of the entire spectrum |
| FB_CMA_PowerCepstrum [▶ 193] | L | ✔ | - | N/2+1 | ✔ | - | • Power cepstrum<br>• Windowing and scaling adjustable<br>• Calculation of the entire cepstrum |
| FB_CMA_RealFFT [▶ 211] | N | ✔ | - | N/2+1, N | - | ✔ | • FFT for real-valued input signals<br>• No windowing, overlapping and scaling<br>• Calculation of the entire spectrum |
| FB_CMA_SlidingDFT [▶ 218] | * | ✔ | - | nBins | - | ✔ | • Online DFT, i.e. sample-wise update<br>• For minimum latency<br>• Frequency ranges to be calculated are configurable |
| FB_CMA_SparseSpectrum [▶ 223] | M | ✔ | - | nBins | ✔ | ✔ | • Calculation of individual configurable spectral lines<br>• Windowing and scaling adjustable |
| FB_CMA_SpikeEnergySpectrum [▶ 227] | L | ✔ | - | N/2+1 | ✔ | - | • Detection of periodically occurring high-frequency pulses<br>• Windowing and scaling adjustable |
| FB_CMA_ZoomFFT [▶ 259] | N | ✔ | ✔ | N/(2D)+1 | - | ✔ | • Calculation of a frequency excerpt of the spectrum<br>• No windowing, overlapping and scaling |

Here, `N` corresponds to the FFT length, `L` to the window length (or overlapping respectively) and `nBins` to the number of configured spectral values. Furthermore, `D` designates the decimation factor. The length of the input data buffer of the function block FB_CMA_SlidingDFT is not restricted to a fixed size. The lengths `L`, `V` of the order spectrum must be specifically adapted to vibration and position data.

## 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.
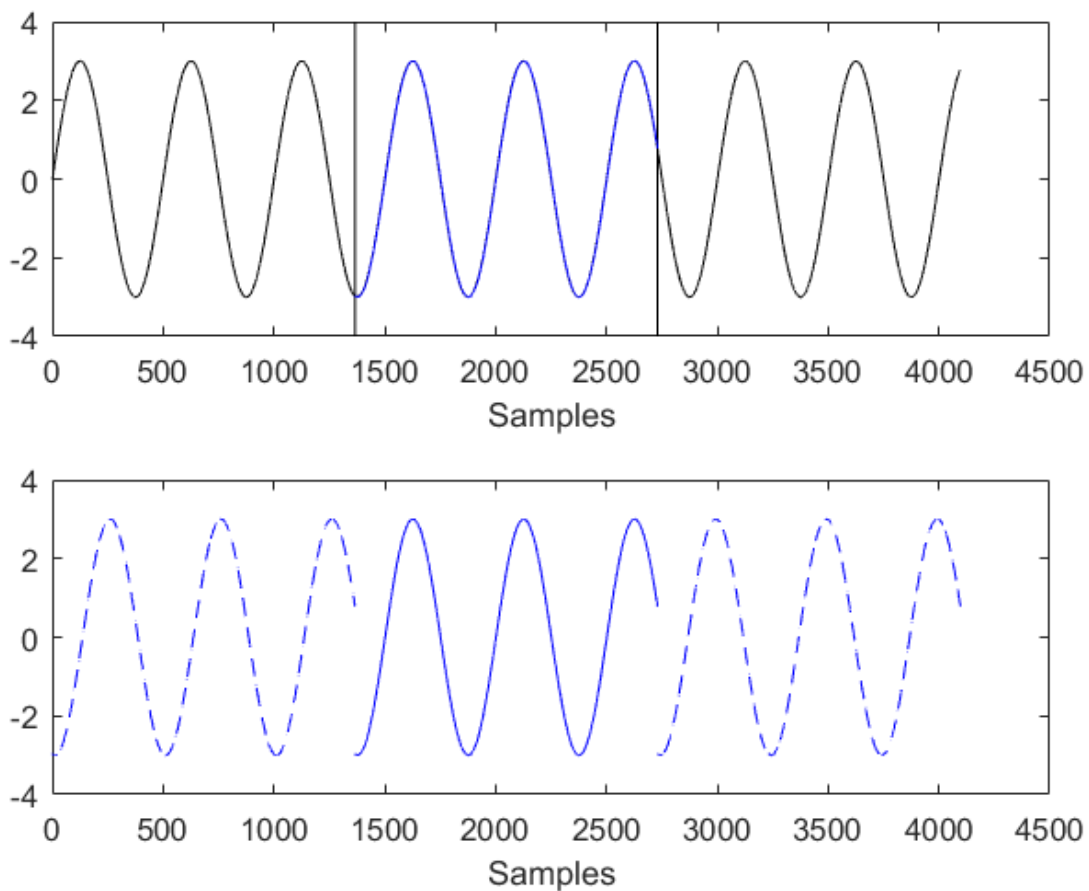


The situation can be rectified by weighting the signal sections before the transformation with a suitable window function (for details see next section). In a suitable window, time values near the start and the end are multiplied with a factor zero or closed to zero. The following diagram shows the same 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.
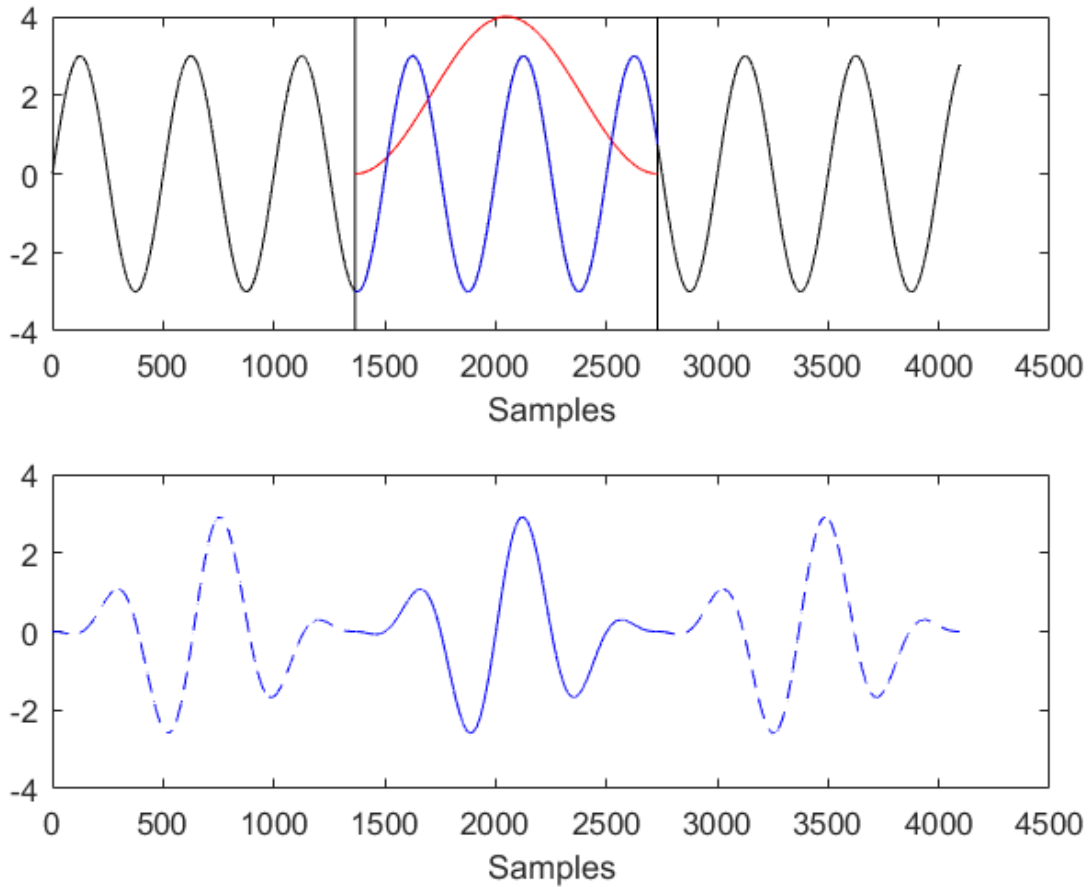
A problematic aspect of windowing is that values at the edge of the window are hardly taken into account in the spectrum. In situations where this region contains signal characteristics that indicate possible damage, key information may be lost. In order to prevent the loss of information, the TwinCAT Condition Monitoring Library uses overlapping signal sections for the windowing procedure. A 50% overlap is recommended with the Hann window, which is used as standard. As a result, samples that are at the edge of one window section are in the center of the next window section.

The following picture schematically illustrates the procedure for an FFT analysis from a data stream with 50% overlap.
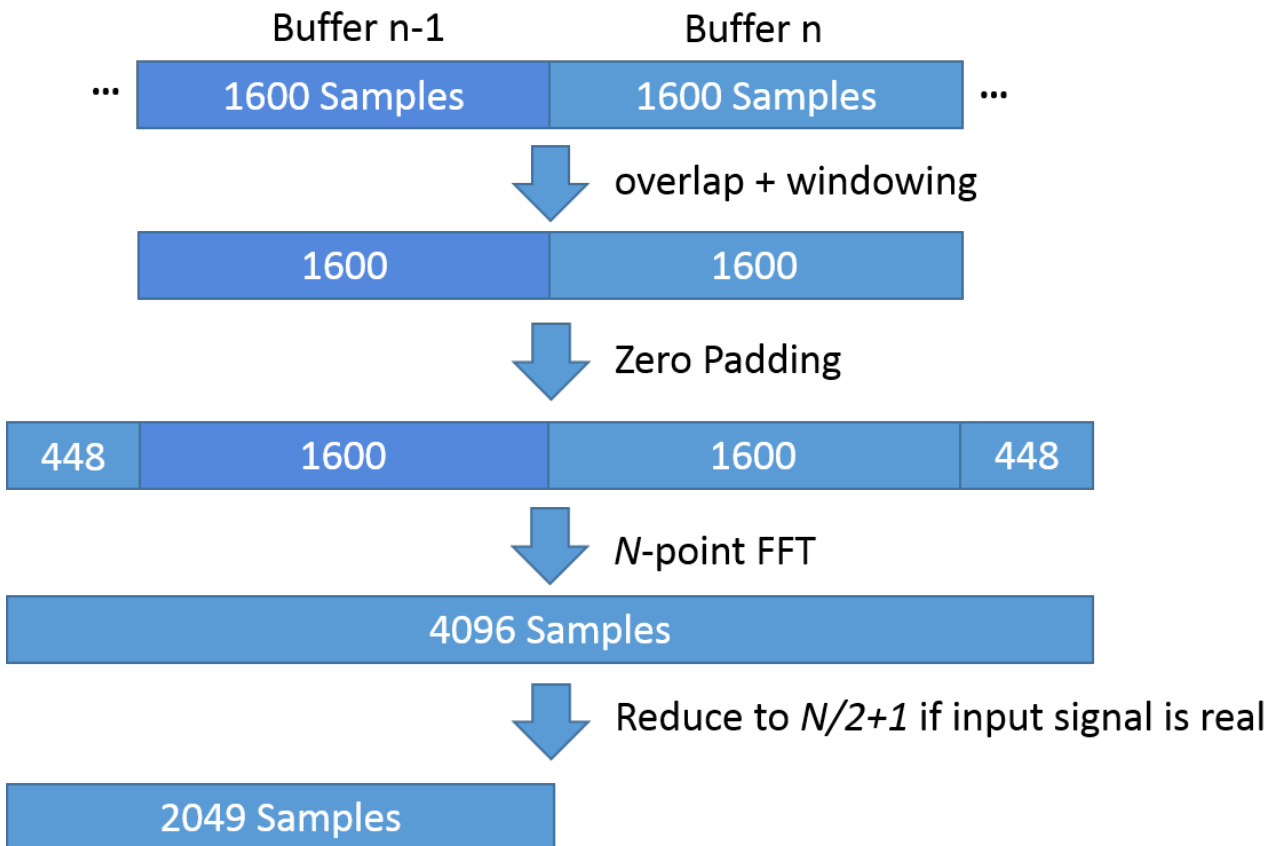
Initially, buffers with a defined length of 1600 values are filled from the data stream. The previous buffer is included in the evaluation of the data from buffer n, so that the data packet that is windowed now contains 3200 values. The maximum of the window function is precisely in the middle between 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 Image frequencies [▶ 16]).

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. As user, only a configuration of the parameters (length of the buffer, length of the FFT, window function, overlap, etc.) and the provision of the data buffer are necessary. See for example: ST_CM_PowerSpectrum_InitPars [▶ 288].

**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 $x[n] \cdot w[n]$, 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 3200 samples, followed by calculation of the magnitude spectrum (with the Spectrum Scaling Options [▶ 361] eCM_PeakAmplitude). In the following illustration, the magnitude spectrum is compared when using the available window functions.

A sample for the reconstruction of the following graphic can be found here: Window functions [▶ 310]
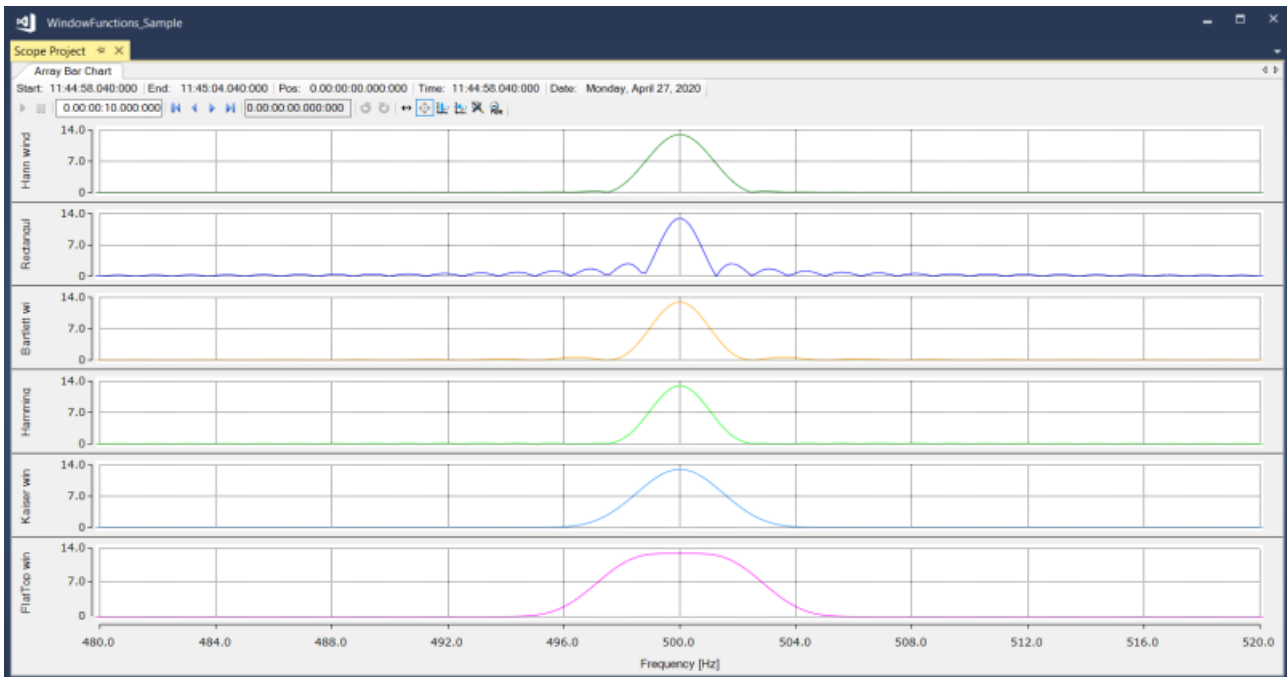
It illustrates two key features of window functions:

- 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 amplitude accuracy of harmonic signals, compare the maximum values when specifying a vibration with an amplitude of 13.

The different window types each form a trade-off between these three essential properties.

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. The *curvature* of the *mainlobes* in the range -0.5 bins to +0.5 bins around the maximum characterizes the maximum amplitude error that can occur with harmonic signals (*scalloping losses*).

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. In addition, the maximum possible amplitude error is very high at -36.34%. The **Hann window** considerably reduces the *sidelobes*, but it also reduces the attainable frequency resolution, which nevertheless represents a good compromise as long as no exact determination of the amplitudes of sinusoidal signals is required. The maximum possible amplitude error here is -15.12%. The Hann window is one of the best-known window functions and is therefore set by default in the Condition Monitoring library. If amplitude accuracy is required with harmonic signals, a **flat-top window** (SFT5M) is to be used with the flattest possible curvature in the central *main lobe* area (maximum amplitude error -0.045%). However, the *mainlobe* is very wide here, as a result of which this window is only recommended for the analysis of purely harmonic signals.

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 $\Delta f$ is derived from the FFT-length $N$ and the sampling rate $f_s$ (see <u>Fourier analysis [▶ 12]</u>). The expression in the equation before the $\Delta f$ is defined via the properties of the window and is 1 for a rectangular window and, for example, 1.5 for the Hann window and 3.885 for the SFT5M.

The selection of the window to be used, the associated parameters and the overlap to be used is realized for the function block concerned via the respective function block-specific structure with initialization parameters, e.g. <u>ST_CM_MagnitudeSpectrum_InitPars [▶ 281]</u>.

**Available window functions**

For a window of the length N, the following window functions are available in the Condition Monitoring library. Here, $I_0$ designates the modified Bessel function of the first type and zero order. The `eCM_KaiserWindow` and the `eCM_FlatTopWindow` are parameterized via T_CM_WindowParameters [▶ 268] in the initPars structures. The last column specifies the recommended overlap in percent of the window length. By way of

example, the Kaiser-4 window with $a_0 = \beta = 4\pi$ has been selected for `eCM_KaiserWindow`

and the SFT5M window with

$$a_0 = 0.209671, a_1 = -0.407331, a_2 = 0.281225, a_3 = -0.092669, a_4 = 0.0091036$$

for `eCM_FlatTopWindow`.

| Identifier | PID | Definition | Overlapping |
|---|---|---|---|
| `eCM_HannWindow` | 16#05300 901 | $w[n] = 0.5 \cdot (1 - \cos(2\pi n/N))$ | 50% |
| `eCM_Rectangular Window` | 16#05300 902 | $w[n] = 1$ | 0% |
| `eCM_BartlettWin dow` | 16#05300 905 | $w[n] = 1 - |2n/N - 1|$ | 50% |
| `eCM_HammingWind ow` | 16#05300 906 | $w[n] = 0.54 - 0.45\cos(2\pi n/N)$ | 50% |
| `eCM_KaiserWindo w` | 16#05300 907 | $w[n] = I_0\left(\beta\sqrt{1 - (2n/N - 1)^2}\right)/I_0(\beta)$ | 67% (Kaiser-4) |
| `eCM_FlatTopWind ow` | 16#05300 917 | $w[n] = a_0 + \sum_{k=1}^{4}\left\{(-1)^k a_k \cos(2k\pi n/N)\right\}$ | 76% (SFT5M) |

**Calculation of the recommended overlap**

The recommended overlap of the windowed signal sections results from the consideration of the following properties of the window function:

- *amplitude flatness*: flatness of the overlapping window function. i.e. the weighting of the individual samples.
- *overlap correlation*: correlation between the individual samples of the overlapping functions.

The recommended overlap lies at the point where the difference between the two properties is largest. In other words, the best possible flatness is achieved with the smallest possible correlation. This is shown in the following graphic for the SFT5M window (`eCM_FlatTopWindow`).
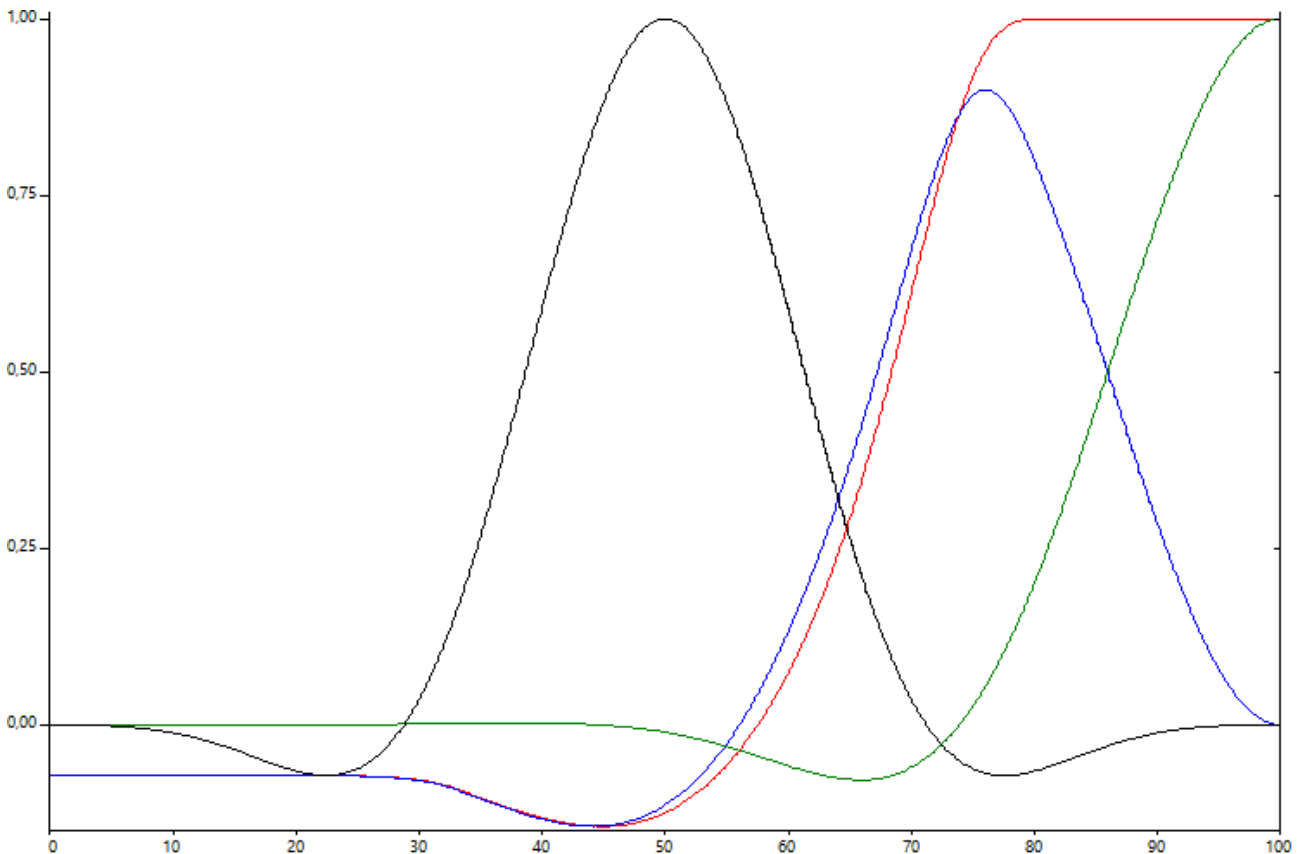
Window function (black), *amplitude flatness* (red), *overlap correlation* (green), recommended overlap (blue).

For further information, see:

- G. Heinzel et.al.: Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows. 2002.

- In the Condition Monitoring library the calculation of the recommended overlap takes place via F_CM_CalculateRecommendedOverlap [▶ 263]. The function can be used explicitly; if configured appropriately, the algorithms also use it implicitly depending on the set window function. Of course, a freely selectable overlap can also be set. Compare, for example, eWindowType, eWindowTypeParameters and nOverlap in ST_CM_PowerSpectrum_InitPars [▶ 288].

**Overlap-Add Method**

Some function blocks of the Condition Monitoring library work by manipulating the spectrum of the input signal, i.e. as described above, the input signal is first decomposed into overlapping partial signals and Fourier-transformed. A manipulation of the spectrum is then carried out and an inverse Fourier transformation is calculated. Depending on the window function used, a correction function is required to compensate for the influence of the window. The individual results, which also overlap, are then summed at the output of the function block, so that a data stream is created again at the output of the function block. This procedure is called the overlap-add method and is shown in the following graphic as an example on the basis of the calculation of the signal envelope (FB_CMA_Envelope [▶ 143]).
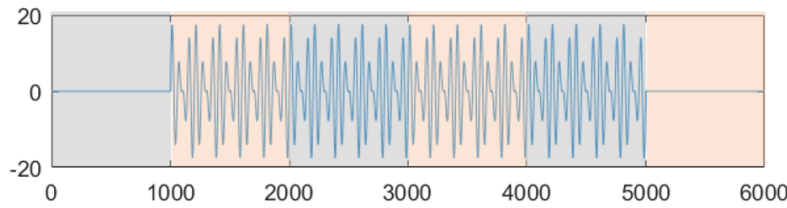
complete data stream
sectioned into buffers
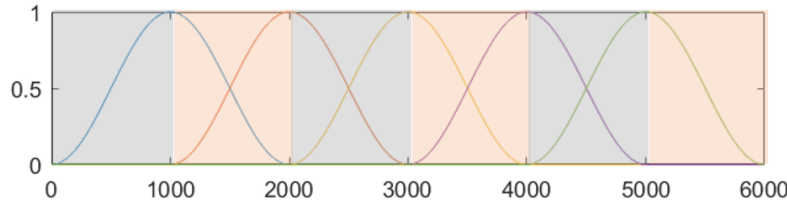
windowing of 2 data buffers
with 50% overlap
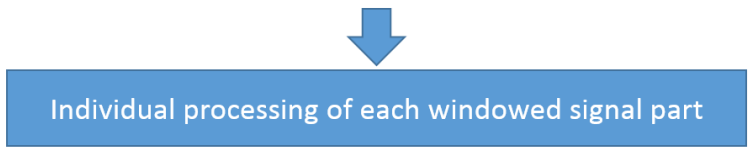
(1) Apply FFT
(2) Manipulation of signal
(3) Apply inverse FFT
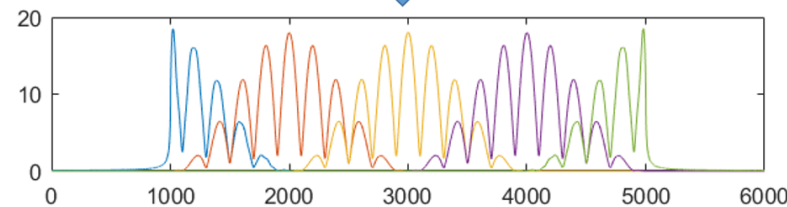
result of each
windowed signal part
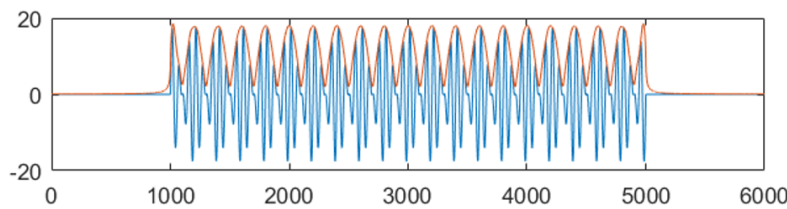
overlap add for reconstruction
into stream analysis

Individual processing of each windowed signal part

---

**ⓘ** **Overlap-Add in TwinCAT 3**

The procedure is used within some function blocks of the library and does not have to be implemented by the user himself. 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. Only two points need to be observed when configuring the analysis chain:

- Sufficient data (a sufficiently large window of data) must be sent to the analysis chain via the FB_CMA_Source in order for a valid result to be calculated. For example, the window length of an FFT-based algorithm must be observed.

- 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. The FB_CMA_Source offers the method `ResetAnalysisChain()` for this. Alternatively, the method `ResetData()` can be used on every individual analysis function block; the method then works only on the individual analysis function block.

**Sample implementation**

A sample based on a synthetic signal is described below. The synthetic signal consists of a background noise and an additively superimposed sinusoidal signal with a frequency of 200 Hz and an amplitude of 2. The sinusoidal signal is always switched on and off for 2 seconds in alternation.

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 sinusoidal signal is switched on. The schematic diagram below shows the described synthetic signal and the amplitude spectrum based on the indicated evaluation window.

Time domain signal



start          window of interest          stop
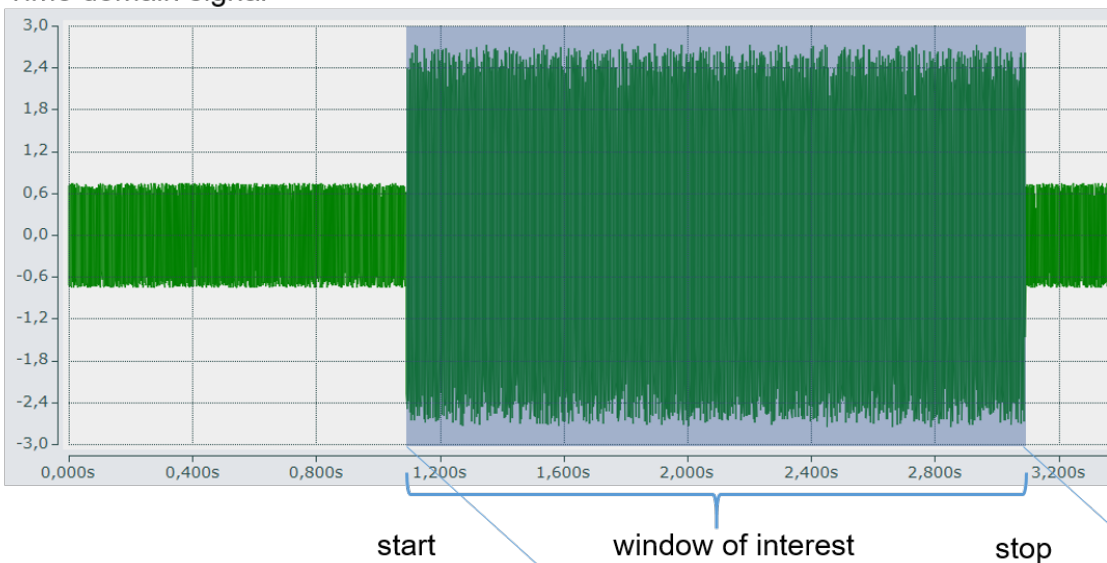
Magnitude Spectrum
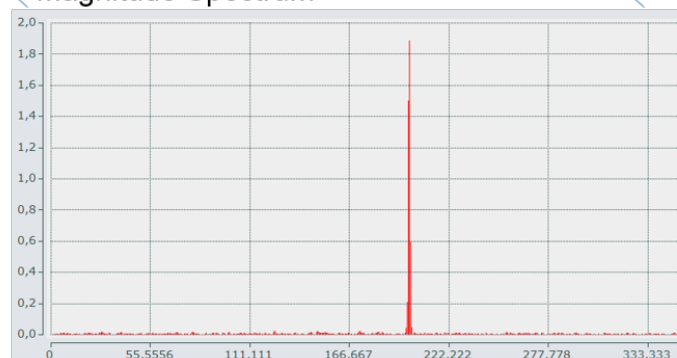
The source code and a more detailed description of the sample can be found here: <u>Event-based frequency</u> <u>analysis</u> [▶ 339].

## 2.1.4    Scaling of spectra

**Magnitude and power spectrum**

There are several common ways of evaluating the spectrum:

- The <u>FB_CMA_MagnitudeSpectrum</u> [▶ 168], which uses linearly scaled magnitude values of the complex-valued spectral values $|X[k]|$. It is also called the amount spectrum or amplitude spectrum.
- The <u>FB_CMA_PowerSpectrum</u> [▶ 197], 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 allows RMS values, for example, to be calculated for limited frequency ranges of a signal. Practical <u>Spectrum Scaling Options</u> [▶ 361] 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**

The conversion of values from the linear to the logarithmic "decibel scale" is common in vibration analysis and machine acoustics. The decibel scale allows a good interpretation if both very large values and very small values occur in a spectrum and are to be evaluated in areas of both large and small values. The magnitude spectrum is converted to the decibel scale via:

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

The decibel scale can be calculated both over 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 [▶ 168] and FB_CMA_PowerSpectrum [▶ 197] is therefore identical in the decibel scale.

The conversion of results to the decibel scale is conveniently activated by the Condition Monitoring Library via a Boolean variable in the function block initialization parameters, see e.g. ST_CM_PowerSpectrum_InitPars [▶ 288].

**Scaling options according to signal type**

By selecting a suitable Spectrum Scaling Options [▶ 361], the spectral values calculated by the FB_CMA_PowerSpectrum [▶ 197] or FB_CMA_MagnitudeSpectrum [▶ 168] function block can be automatically adjusted to a desired reference variable. The correct interpretation of the reference variable is of particular importance here.

In practice, and assuming a stationary signal, it is important first of all with scaling options to distinguish between deterministic and stochastic signals.

**Deterministic signals** consist of periodic vibrations with a defined frequency. Decisive here is that the frequency resolution (ENBW) is wider than a harmonic frequency. Thus, the entire power of this frequency component of the signal is consolidated in this frequency channel. Therefore, the spectral values are directly scalable to an amplitude (Spectrum Scaling Options [▶ 361] eCM_PeakAmplitude) or an RMS value of an equivalent sinusoidal signal. If the signal does not fall in the center of the frequency channel, so-called scalloping losses occur, compare section Window functions [▶ 20], which reduce the observed maximum amplitude. Apart from the use of a flat-top window, this can be compensated retroactively by the use of a Hann window, for example, in which the power values from neighboring frequency channels are evaluated as a sum, see Spectrum Scaling Options [▶ 361] eCM_ROOT_POWER_SUM and eCM_RMS.

**Stochastic or broadband signals** necessitate the evaluation 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. Logically, they must be referenced to this bandwidth in order to obtain results that are independent of the evaluation parameters. Because the effective width of the frequency channels when using window functions depends on the length and shape of the window function, the above-mentioned Equivalent Noise Bandwidth (ENBW) must be used in this case, see Spectrum Scaling Options [▶ 361] eCM_PowerSpectralDensity.

Scaling on the basis of the PSD does not enable consistent scaling of the "DC 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. One example would be the evaluation of a signal that is composed of a harmonic sine wave and band-limited noise. If the amplitude of the harmonic sine wave is to be evaluated, then scaling for deterministic signals must be carried out. If one strives to assess the stochastic background noise, then scaling must be carried out as PSD or LSD.

> ℹ **Scaling of spectra with the Condition Monitoring library**
>
> Various scaling options are already implemented in the Condition Monitoring library and can be parameterized via the function-block-specific structure with initialization parameters. See E_CM_ScalingType [▶ 266] and Spectrum Scaling Options [▶ 361]. A tutorial on this can be found here: Scaling of spectra [▶ 312].

**Referencing**

**Classification of the scaling**

While comparison of absolute measured values is very important for measurement technology, for vibration assessment [▶ 34] 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.
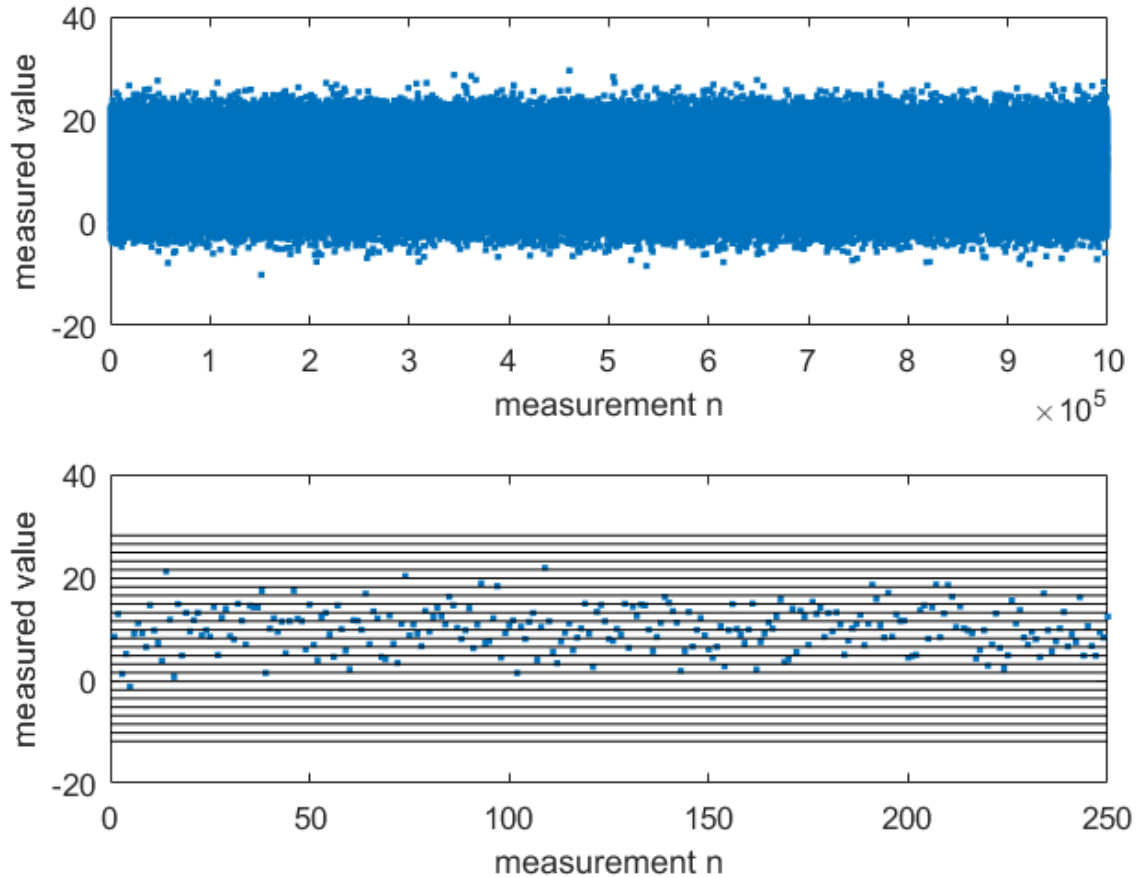
The absolute frequency distribution is then simply results from the number of measured values that lie within a class (bin), see diagram below, left. The distribution is parameterized based on the number of considered classes – the more classes, the finer the distribution. The relative frequency distribution 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%.
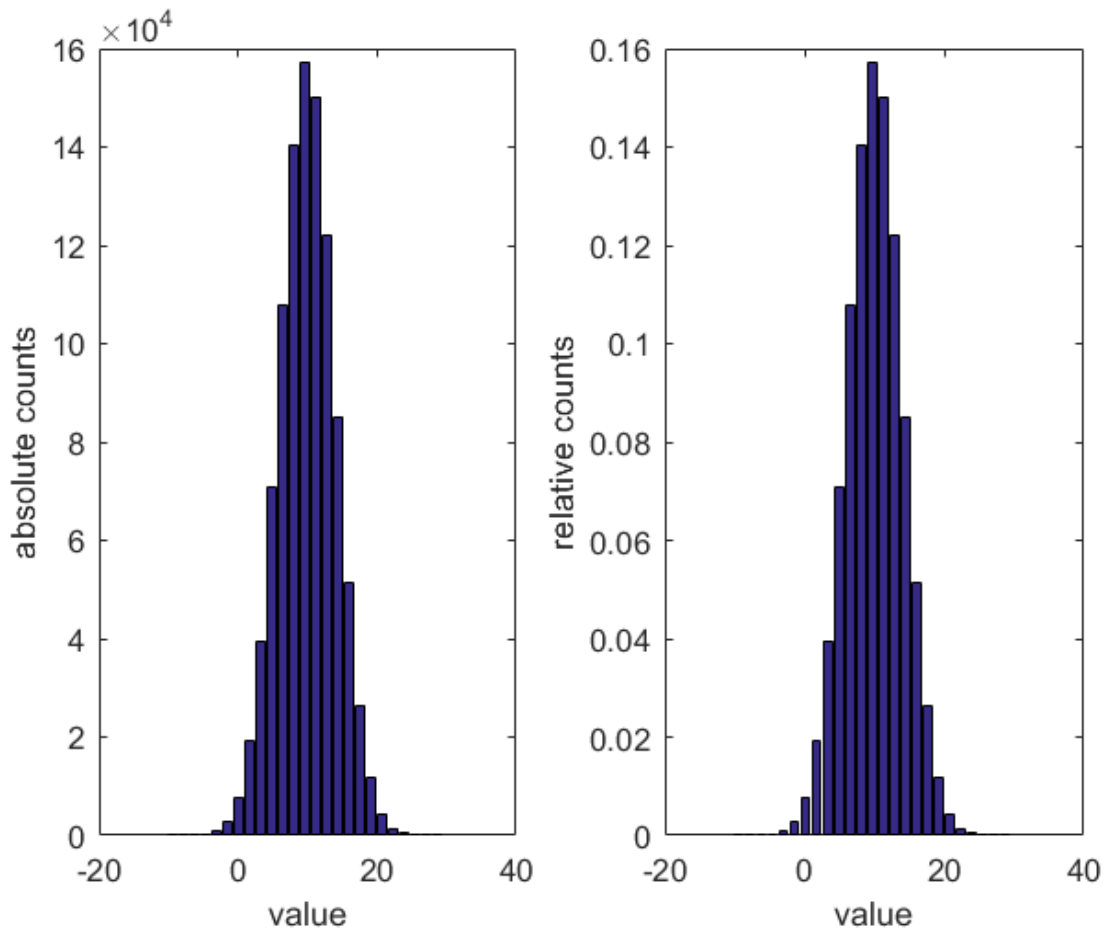
An experimentally investigated process can initially be visually assessed quite easily on the basis of a frequency distribution. Three questions can be explored:

- 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 [▶ 151]. 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 Histogram [▶ 317].
>
> The Statistical methods [▶ 319] sample illustrates further Condition Monitoring Library options for statistical data evaluation.

**Ordinary and 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).

For example, the moments are very well suited for calculating statistical variables from a given sample: average value, variance, skew, kurtosis etc. are particularly suitable for calculating statistical parameters from a given number of operations. While the average value provides a suitable estimate, the other 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 root of the variance corresponds to the standard deviation and is 4.0013 V. If the distribution of the measured values is normally distributed as in this case, then the distribution of the measured values is completely described with these two 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.

---

**ℹ** **Calculation of moments in TwinCAT 3**

With the Condition Monitoring library, the function block FB_CMA_MomentCoefficients [▶ 179] can be used to calculate the first to fourth order moments (mean, variance, skew, kurtosis) of a sample. The function block only has to be parameterized in terms of the sample size used.

---

**Quantile**

The p-quantile $Q_p$ of a random variable *x* is the value for which $Q_p$ is greater than *x* for the portion *p* of all realizations of *x*. A little more descriptively formulated: If a finite number of values is given, then 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 mean of a sample.

The value of *p* can lie between zero and one. If *p* is specified in percent, then percentiles are concerned. $Q_{0.5}$ thus corresponds exactly to the median, while $Q_{0.9}$ is the 90% percentile and $Q_1$ is the maximum of an observed sequence of values.
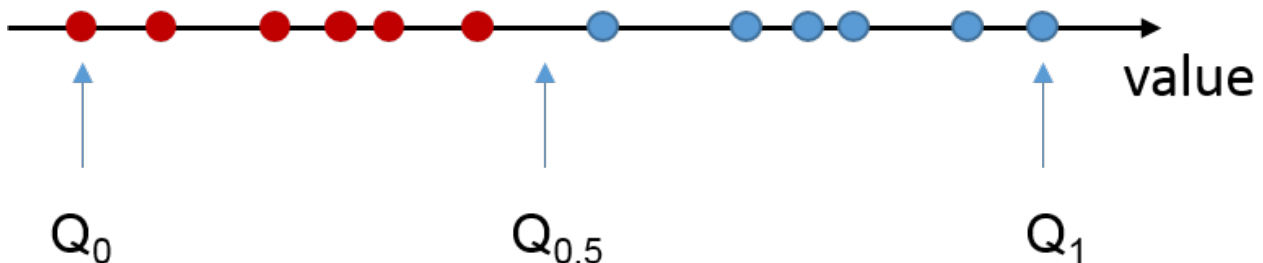


The closer *p* approaches the value of one, the more $Q_p$ is determined by outliers and extreme individual values, and the closer *p* approaches the value of 0.5, the more $Q_p$ approaches the median, which is very robust against outliers. The value of *p*, which can be configured in TwinCAT at runtime,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

To illustrate the basic idea of quantiles, the following graphic shows a sequence of 1000 values scattered around a mean value of 13.

---
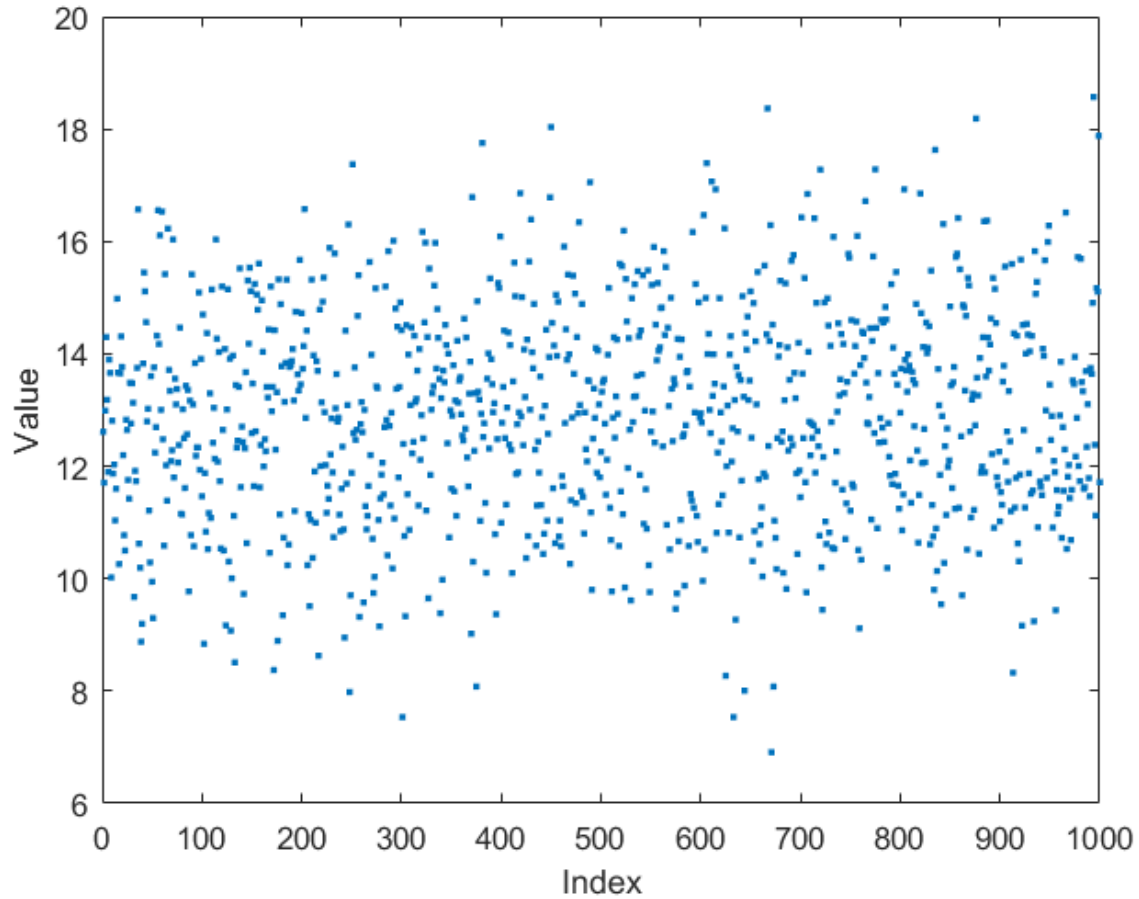
From the value sequence, a histogram can be calculated that indicates how often a value occurs in the sequence under consideration (sample). By integrating the absolute frequency shown in the histogram and referencing to the total number of values in the sequence under consideration (in this case 1000), the empirical sum frequency distribution can be calculated, on the basis of which the quantiles can easily be read. In this case, for example, the 25% quantile is 11.8, i.e. at least 25% of the individual values of the sample of 1000 values under consideration lie below this value.
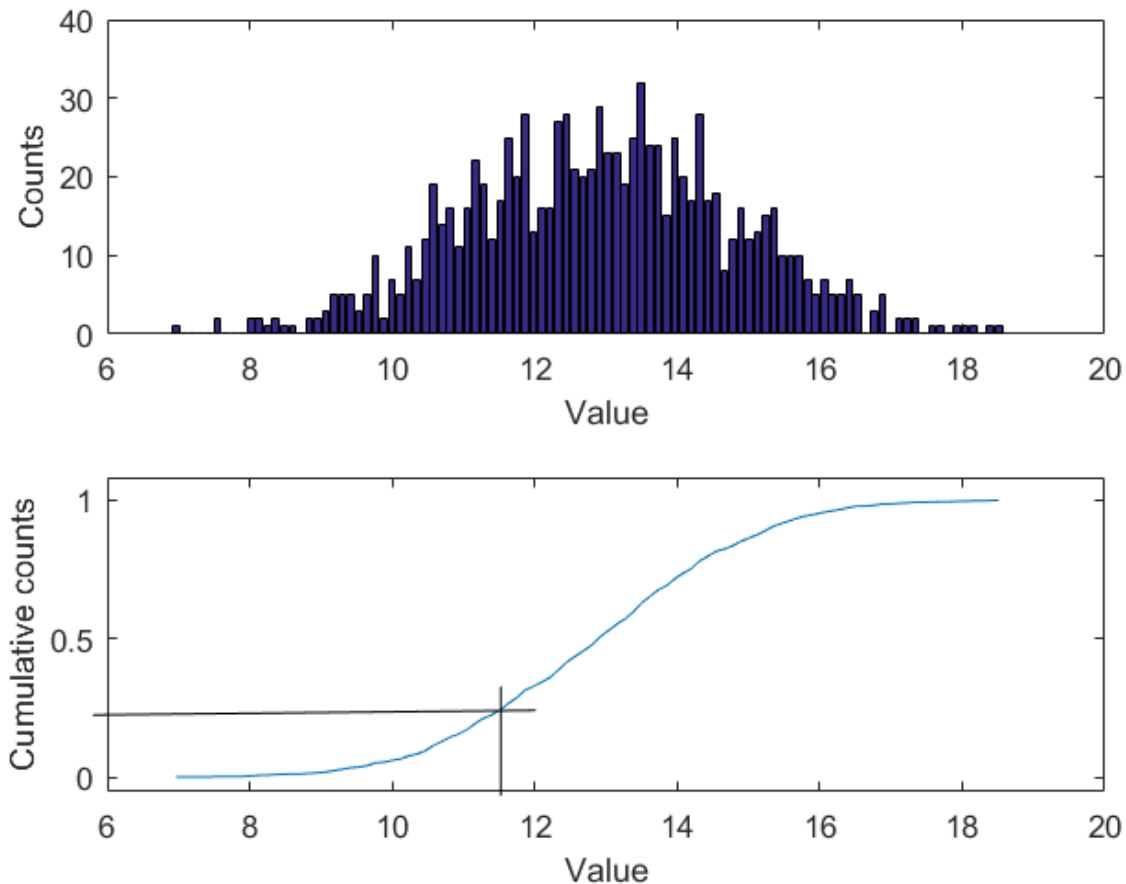
The library function block for calculating FB_CMA_Quantiles [▶ 201] works in two sub-steps, which can be called together or in separate sub-steps. In the first step values are added to an internal histogram, whose parameters can be configured in advance. This step requires very little computational effort. In the second step, the previously selected quantiles are calculated from the stored histogram. Depending on the configuration this second operation is computationally much more intensive, because it is defined by more complex operations, but it needs to be performed much less frequently.

> ● **Calculation of quantiles in TwinCAT 3**
>
> **i** The function block FB_CMA_Quantiles [▶ 201] can be used for the calculation of quantiles. Several quantiles can be calculated with just one function block call. The function block is parameterized like the histogram function block, as well as the quantiles to be calculated and the sample size to be used.

## 2.2 Application concepts

This part of the introduction provides an overview of the basic patterns of applications and solutions for tasks in Condition Monitoring. Details of the programming and interfaces are not yet dealt with; instead, some fundamental strategies and concepts are explained. At the end of each concept an implementation scheme is executed with the Condition Monitoring library, so that an overview of the possibilities of the library is created.

What you will learn:

- 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 roller bearings structured?
- How is Condition Monitoring for gear units structured?
- How do you implement damage monitoring?

## 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 |

The described vibration assessment according to ISO 10816-3 is implemented in three different examples. See:

-

-

-

## 2.2.2    Frequency analysis

**Motivation**

One of the most important methods in the diagnostic/analytic machine monitoring is the recording of vibrations with accelerometers and a frequency analysis based on that. The reason is that machines are made of metal and thus of elastic structures that are virtually always subjected to periodic forces. This leads to vibrations in which the frequencies and excitation 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 propagate in the machine components in the form of sound waves (structure-borne sound). As machines consist of a large number of parts, which transmit vibrations elastically to other components on the one hand and oscillate themselves on the other, filtering and superimposition of the individual vibration components occur. A vibration signal correspondingly consists of several signal components which, with different time delays and attenuations that depend on the distance traveled, add up to the total signal. Individually sought vibration components are therefore no longer recognizable in the time sequence of the total signal without further processing. 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 divided into:

- Calculation of the spectrum
- Statistical assessment of the result
- Threshold value monitoring

**Practical elements of the frequency analysis**

The main aspects of the Fourier analysis have already been dealt with in the section Fourier analysis [▶ 12]. At this point we will look again at the most important practical aspects.

In designing the parameters of the function block for Fourier analysis (e.g. FB_CMA_MagnitudeSpectrum [▶ 168] or FB_CMA_PowerSpectrum [▶ 197]), the following questions are of paramount importance.

- How high is the maximum frequency that is to be analyzed?
  The sampling frequency – via the terminal's oversampling factor – and the associated task cycle time are to be set accordingly. Attention must also be paid to the setting of an anti-aliasing filter. See section Samplinig theorem [▶ 14].
- What are the requirements for the frequency resolution?
  The length of the measuring time (size of the input array) must be set accordingly; see Frequency resolution [▶ 14]. The deterioration of the frequency resolution when applying a window function should also be taken into account, see Window function [▶ 20].
- The FFT length must be larger than the length of the input array, and it must be a power of two. The surplus elements are filled with zeros; see Zero Padding or Frequency resolution [▶ 14].
- Suitable scaling of the spectrum must be selected, see Scaling of spectra [▶ 26].

ℹ The above points refer in particular to the use of the FB_CMA_MagnitudeSpectrum and FB_CMA_PowerSpectrum. When using special variants of the spectral calculation, several points can be optimized. Refer to the section Overview of the calculation of spectral values [▶ 17].

**Statistical assessment**

The Fourier spectrum is very sensitive to noise and interference in the signal. Therefore, the Fourier-transformed real noisy signals are usually not well suited for direct analysis or evaluation. To compensate for this, the magnitude spectrum is usually averaged or evaluated by quantiles, see Statistical analysis [▶ 28]. This approach presupposes the temporal stability or cyclic repetition of the signal to be analyzed. The parameters determined in this way are significantly more robust against interference and easier to assess visually. In the following, an evaluation based on the mean of several spectra is considered as an example.

ℹ **Statistical evaluation 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 analysis more reliable.

An alternative method is the averaging of the calculated Fourier coefficients via the frequency, i.e. the averaging of adjacent frequency bins. For this purpose, the FFT should understandably be calculated at a higher frequency resolution than in the method of successive averaging of spectra over time described above. The averaging of adjacent frequency bins is largely equivalent to the averaging of spectra over time, but more computationally complex.

**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.
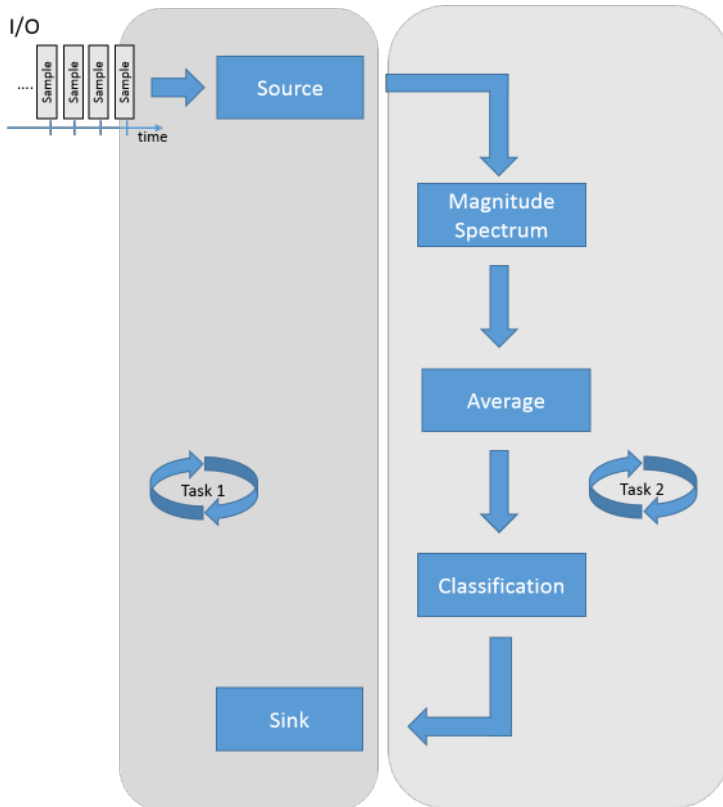


Threshold value monitoring of averaged magnitude spectra is to be implemented. The following components of the Condition Monitoring Library with the described functions are used:

- 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:
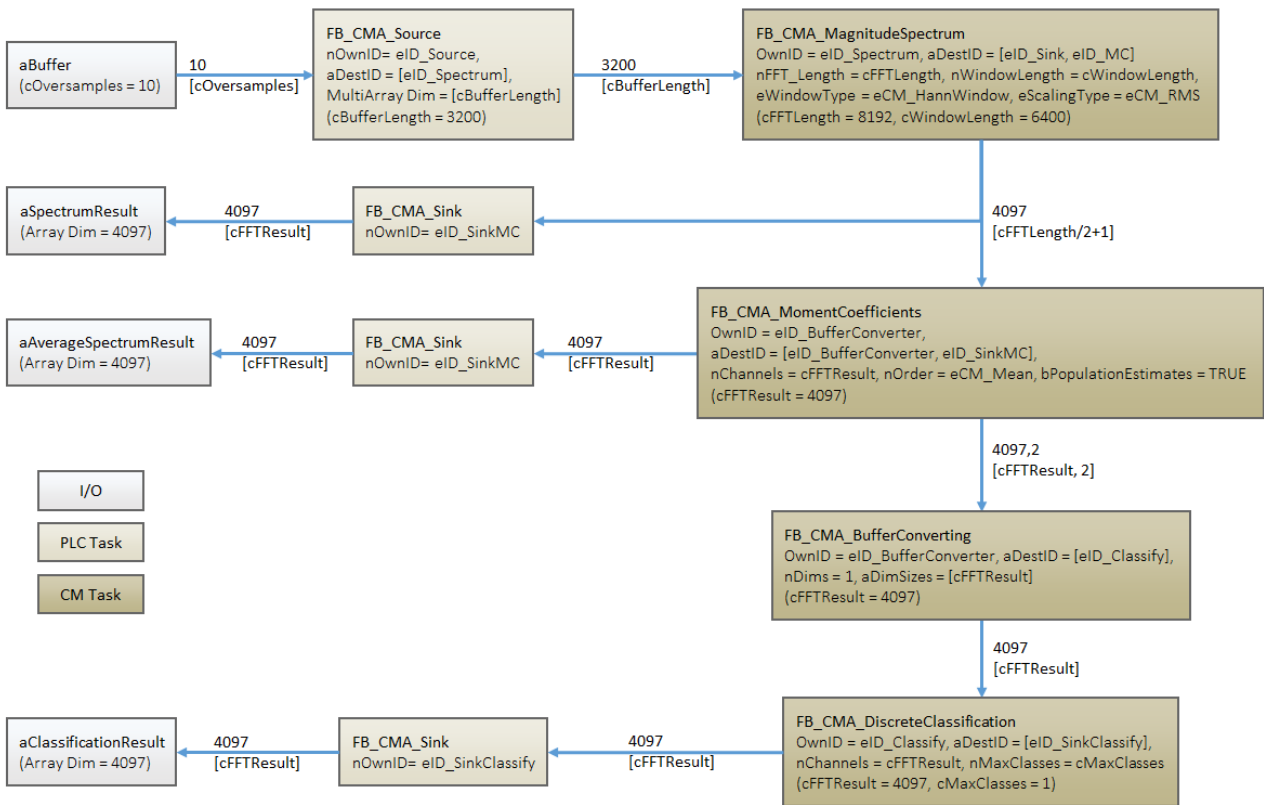
**The sample project for the concept shown here can be downloaded from here:** Threshold value consideration for averaged magnitude spectra [▶ 332]**.**

**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 Frequency resolution [▶ 14], 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 a 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 wave with a frequency of 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 <u>Condition Monitoring with frequency analysis [▶ 327]</u> 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.

# 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 fatigue 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 fatigue 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 fatigue life. The fatigue 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:
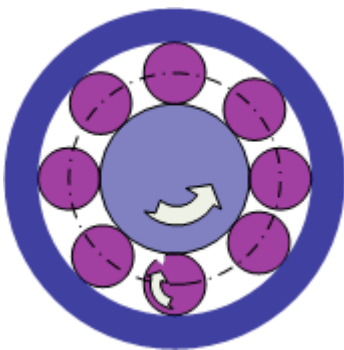


The critical parts of a roller bearing are moving surfaces in contact with each other. These are the rolling element surface, the contact surface of the inner race and the contact surface of the outer race. Rolling over local damage in a contact surface results in a shock pulse, which can be picked up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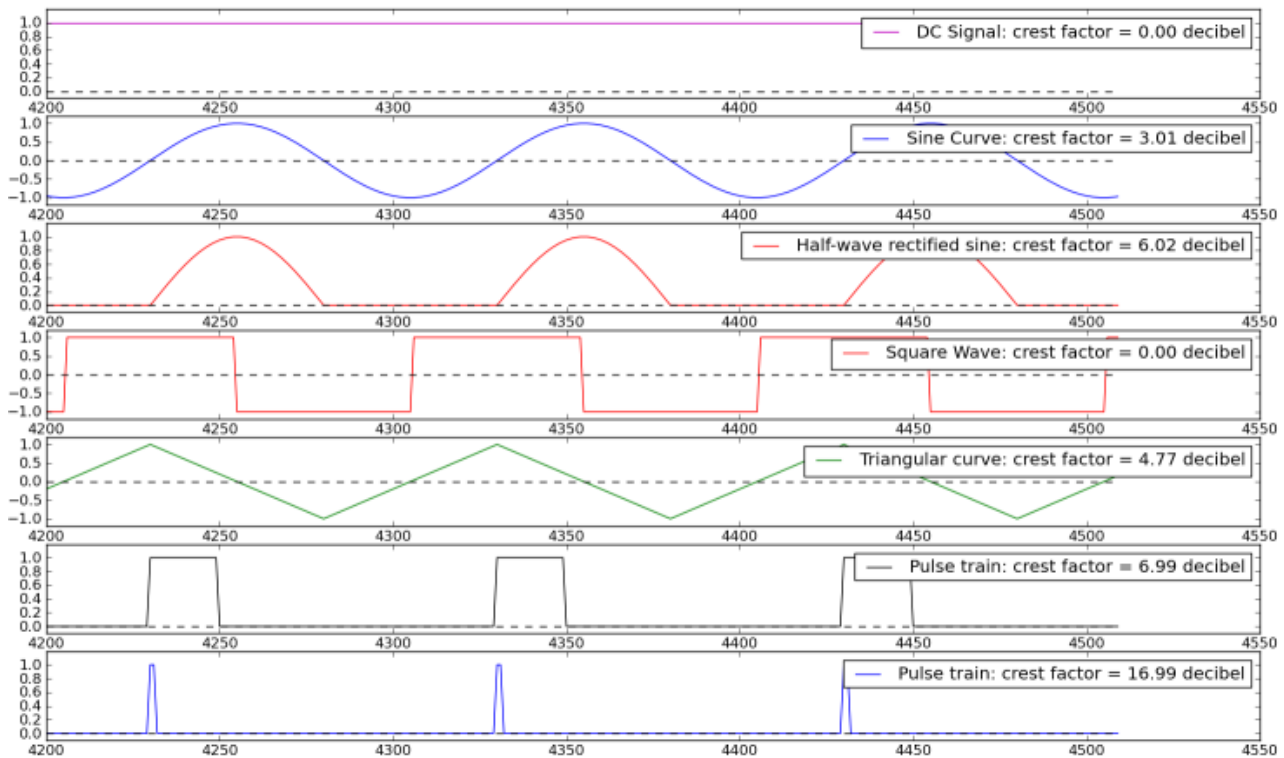


The bottom diagram shows the typical strong increase in the crest factor when acute shock pulses are encountered. An increase in the crest factor is usually a good indicator for damage. That makes this variable a suitable tool for the early recognition of damage and for trend analyses.

Signals from a roller bearing can be interpreted as follows.

BECKHOFF
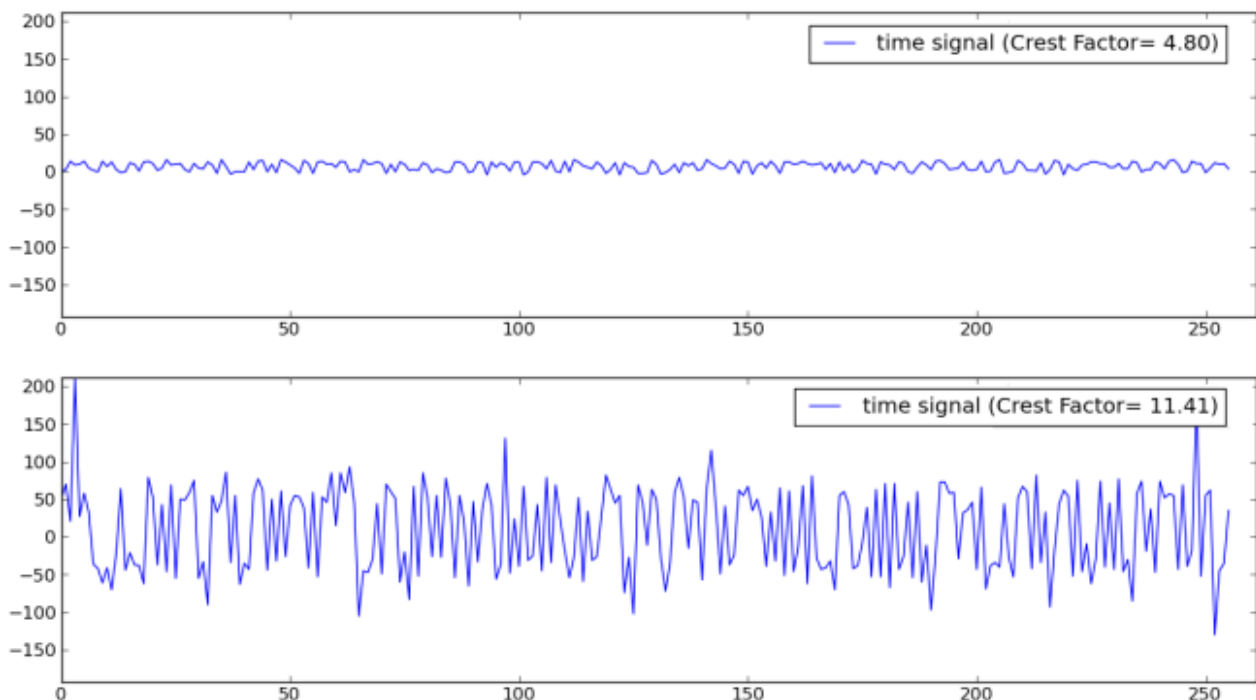
The diagram above shows two vibration signals from bearings with different wear, in each case with the corresponding crest factor. The signal sequence at the bottom clearly shows peaks resulting from damage. While the undamaged bearing has a crest factor of 4.8 dB, the damaged component has a value of 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 [▶ 179] function block.

**Processing concepts**
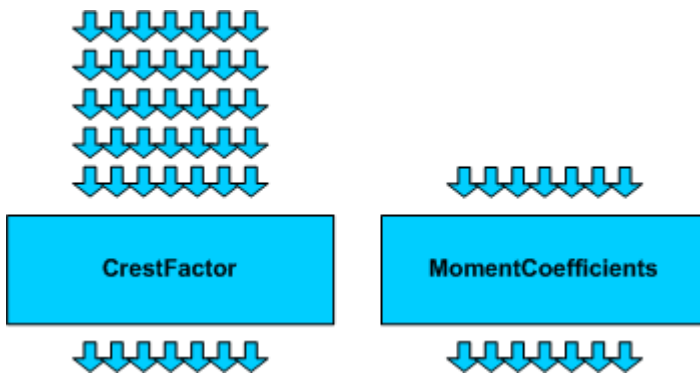


The above diagram shows the function blocks available for calculating the crest factor and kurtosis. The FB_CMA_CrestFactor [▶ 97] function block can evaluate data from several sensors at the same time, provided that the number of individual values is the same for each channel. The return value consists 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 FB_CMA_MomentCoefficients [▶ 179] 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.
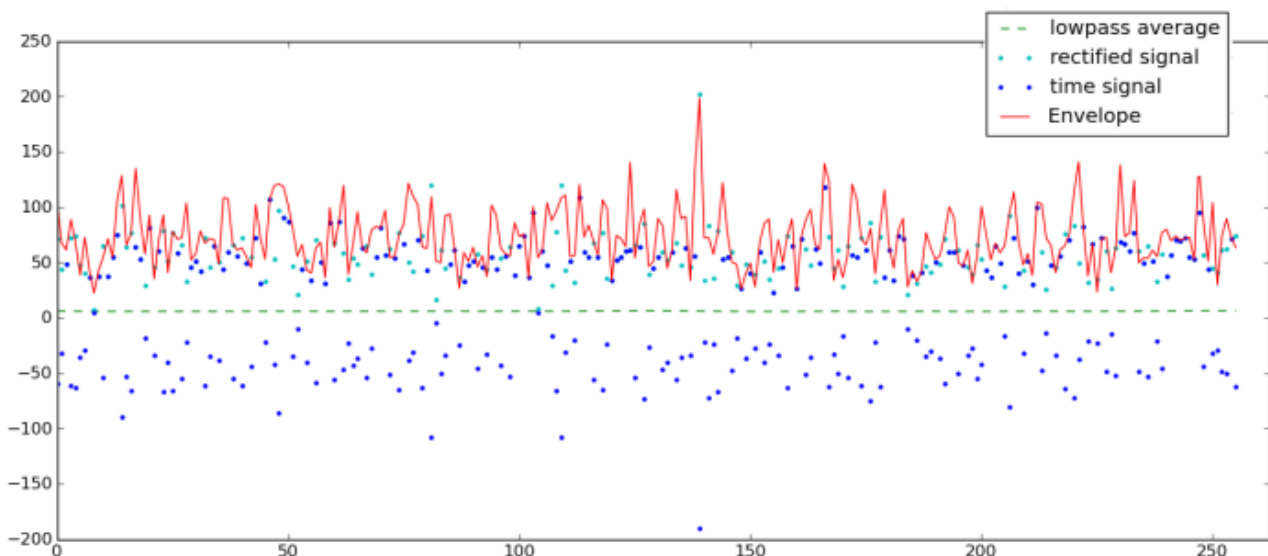


The above diagram shows the envelope of the signal of a damaged roller bearing already used before. The time signal is marked by the blue points, the envelope by the red line. For better understanding a green dotted line plots the sliding mean value of the time signal, which is not exactly zero, and 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 <u>Frequency resolution [▶ 14]</u>. 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
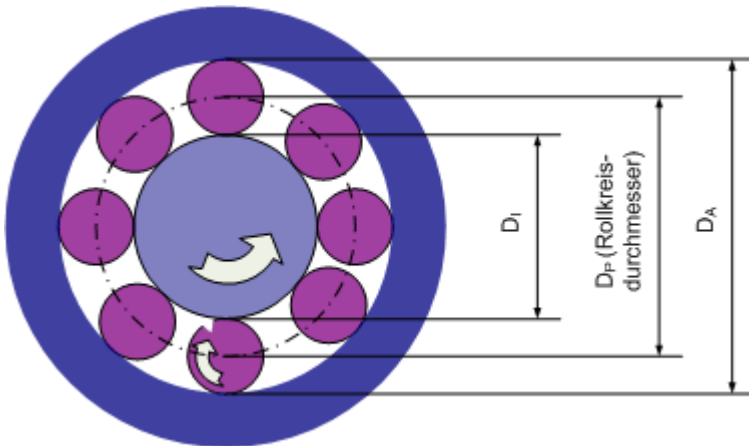


Assume that the speed $f_{red}$ of an axle connected to the inner ring is measured and the diameters of the bearing parts are as follows: Diameter of the inner race $D_I$, diameter of the balls $D_B$, diameter of the outer race $D_A$. The number of balls is Z. $D_I$ and $D_A$ give rise to the **pitch diameter**: $D_P = (D_I + D_A)/2$. Now if the inner race rotates with a speed of $f_{rot}$, then the pulse frequencies can be determined from this. The following acronyms are common for the designation of the frequencies:

- **BPFO**: Rolling element pass frequency outer race.
- **BPFI**: Rolling element pass frequency inner race.
- **BSF**: Bearing spin frequency rolling elements.
- **BPF**: Ball pass frequency.
- **FTF**: Fundamental Train frequency.

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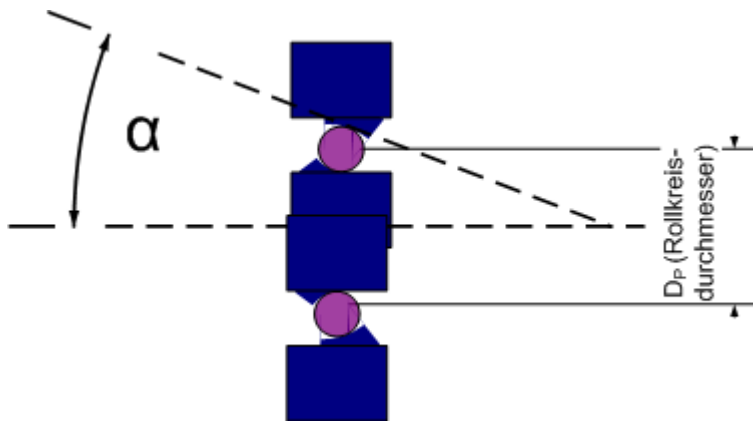
For an accurate calculation in the case of bearings that bear axial loads, the diameter of the balls is to be corrected with the contact angle $\alpha$ with which the balls touch the running surface: $D_b = \cos(\alpha) D_B$. For radial bearings, this angle is 0°, which results in the following formulas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

$BPFO = Z * f_{red} / 2 * (1 - D_b / D_P)$
$BPFI = Z * f_{red} / 2 * (1 + D_b / D_P)$
$BSF = f_{red} / 2 * D_P/D_B * (1 - (D_b/D_P)^2)$
$BPF = 2 * BSF$
Rotating inner race:
$FTF = f_{red} / 2 * (1 - D_b/D_P)$
Rotating outer race:
$FTF = f_{red} / 2 * (1 + D_b/D_P)$ Where $(BPFI + BPFO) / f_{red}$ is always equal to the number of rolling elements Z. There are slight deviations from these formulas in practice, as the contact angle $\alpha$ can vary under load, for example. As a simple rule of thumb, the value

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

is used as an indicator frequency for a defect of the inner race, and

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

for a defect in the outer race. For the determination of the bearing geometry it is useful to refer to the bearing manufacturer's data. It can also be helpful to use calculation programs, some of which the manufacturers offer on the Internet.

---

**i** 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:
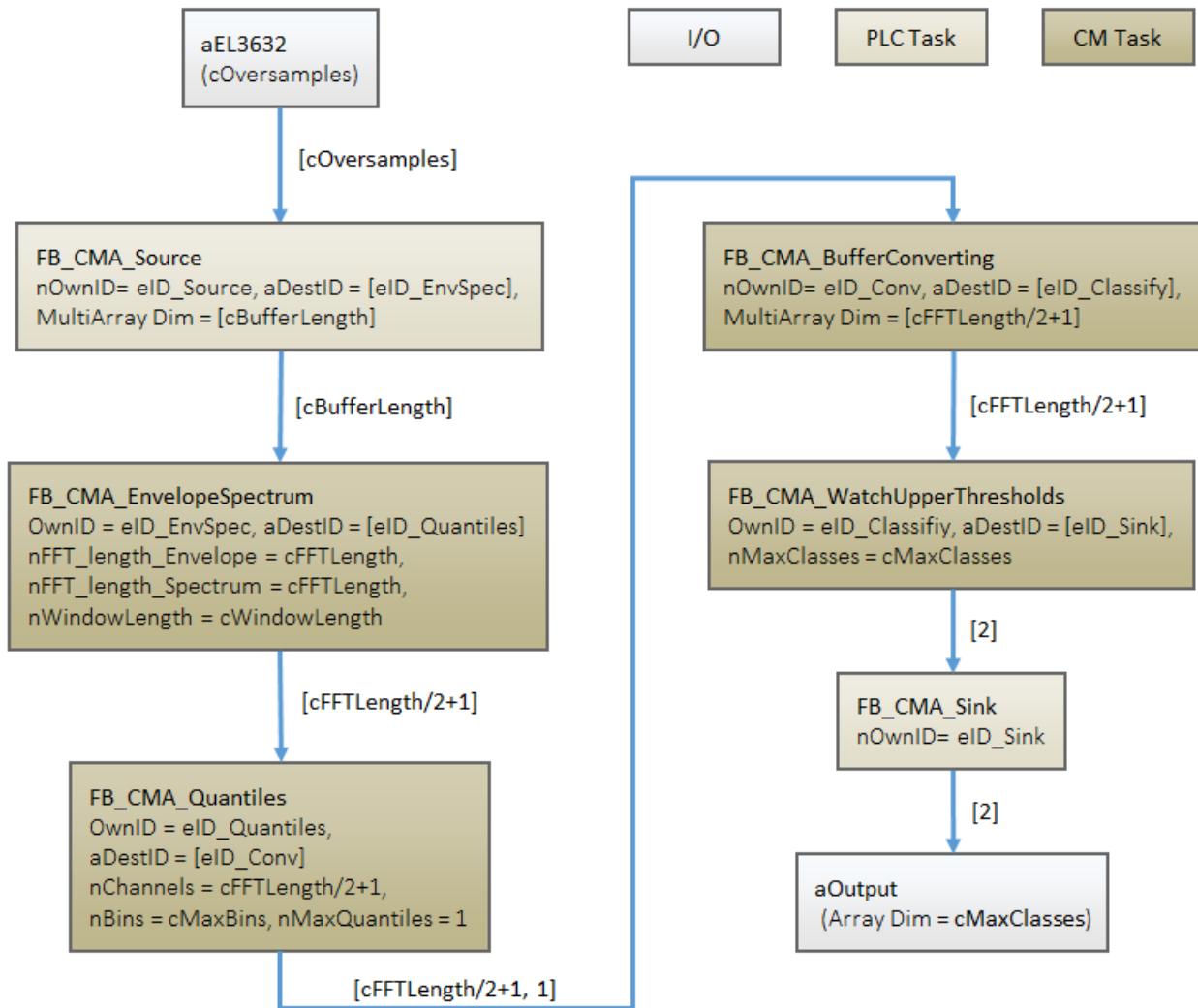


The above diagram shows the processing steps for the envelope spectrum as well as the function blocks that can be used here. First of all the envelope is calculated using the Envelope [▶ 143] function block. Subsequently the power spectrum is calculated (PowerSpectrum [▶ 197]) function block, in the 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 [▶ 255] 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 fatigue 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
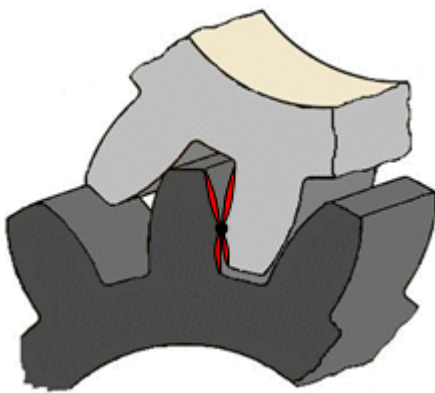


In a gearbox the gear wheels roll off one another, in the course of which the individual teeth periodically come into contact, transfer force and then separate from each another again. While it is possible for this to take place with a precisely constant transmission ratio and largely constant force 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
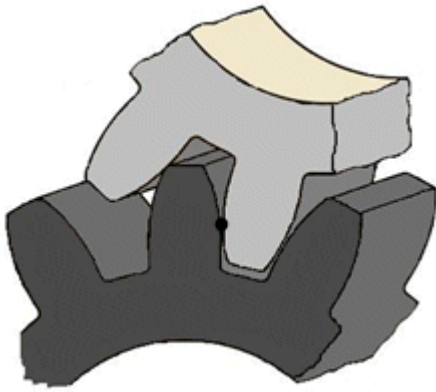
With increasing wear the tooth profiles deviate more and more from the ideal shape, since material is removed by the sliding of the surfaces over one another. This happens more and more intensively the further away the surface is from the center of the tooth flank, as the diagram above shows. The sliding motion itself therefore increases and the torque varies more strongly, whereby the meshing oscillations and in particularly 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.
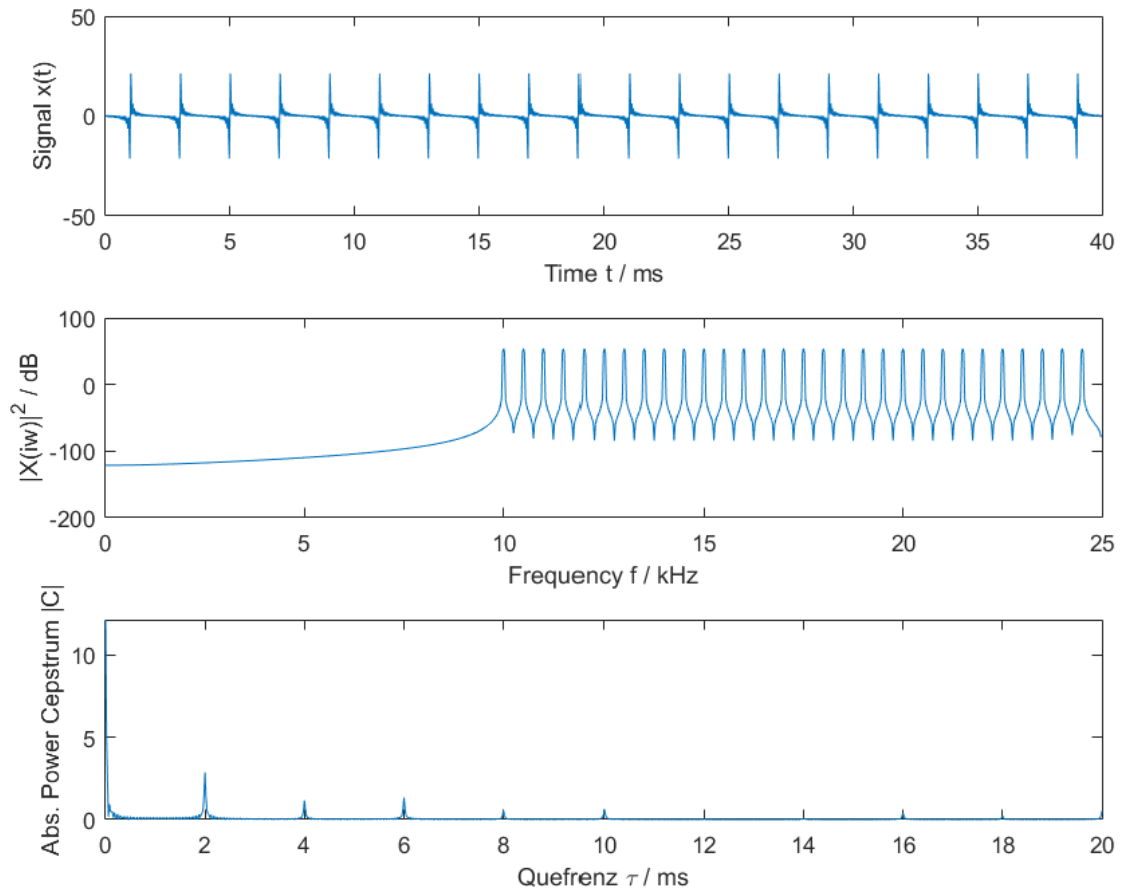
**Processing concept (calculation steps)**
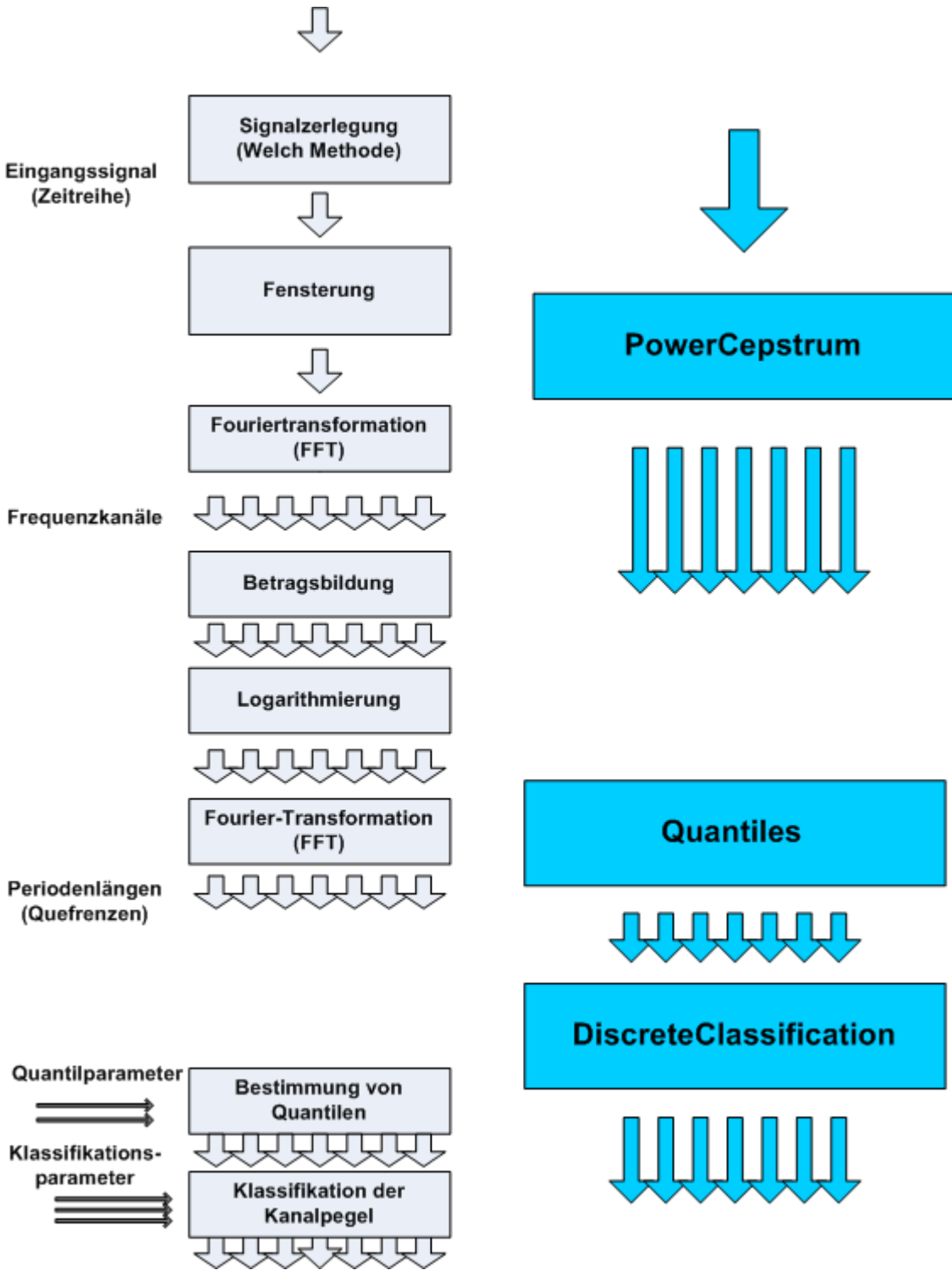


**Calculating the power cepstrum**

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 [▶ 18], 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: <u>Power cepstrum [▶ 338]</u>

**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 <u>Quantiles [▶ 201]</u> 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 <u>DiscreteClassification function block [▶ 118]</u> sketched in here.

- For tasks such as machine protection with limited scope for individual analysis, the <u>WatchUpperThreshold function block [▶ 255]</u> 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.

# 2.2.5 Fatigue life analysis and damage calculation

**Motivation**

How long is the fatigue life of a structural component that is exposed to cyclic loads, such as a wind turbine tower, bridge grider, load crane or machine component?

Computational fatigue analyses provide the answer to these questions. They use operating loads, geometry and material data to determine the fatigue strength of a structure. If the component or the structure is in operation, continuous monitoring of the real operating loads can be used to realize an estimate of the remaining fatigue life.

Above all else, the goal of lightweight construction is the resource-saving use of raw materials and energy in production and operation. Lighter constructions not only allow a reduction in costs, but also the realization of new products with more complex geometries. These goals are naturally not restricted to the field of lightweight construction, but are extending more and more across all areas of mechanical designs. Formerly over-dimensioned components are adapted to the loads expected in operation. Accordingly, the safety margins with respect to loads are reduced – the components are thus no longer designed as *fatigue-resistant*. A fatigue-optimized design, conversely, attempts to design a component so that it achieves the desired fatigue life without failure if used for the defined purpose and with a defined load profile.

As shown in the <u>Fatigue Analysis [▶ 343]</u> area in the <u>Samples [▶ 299]</u> section, the software for fatigue life analysis is simple to implement with TwinCAT 3 Condition Monitoring. Moreover, the Beckhoff system offers an excellent platform for recording the necessary measured variables (e.g. ELM35xx), communicating them even over long distances by EtherCAT, processing them on a reliable IPC and finally forwarding them to higher-level systems for storage or display.

The following section imparts background knowledge and serves the classification of the subject area. The description of the relevant function blocks and the prepared sample serve as examples of the practical implementation.

**Background to fatigue life analysis**

In this section you will learn the following:

- How do you calculate the fatigue life of a component and what is a damage accumulation?

- What are rainflow cycle counts?

- What is a Wöhler curve?
- How are Wöhler curves determined?

**Basic concepts**

When fatigue calculation began around 200 years ago, well before the present-day standard use of simulation tools and measurement technology, the first cases of component failure were investigated in detail: Wilhelm Albert, an engineer from Clausthal, had carried out initial investigations into fatigue of steel chains as early as 1837. His observation that the chains failed not only due to overloading, but also due to frequent cyclic loading at lower amplitudes, laid the foundations for the systematic investigation of fatigue strength. Around 1860, Albrecht Wöhler, a German railway engineer, formulated a mathematical equation for the fatigue strength of metals, the so-called **Wöhler curve, sometimes referred as S-N curve (stress-life curve)**. To do this, he subjected material samples to cyclic loads and ran the tests until failure. From these measurements, August Wöhler was able to plot the mechanical stress over the number of cycles to failure in a diagram and derive a characteristic curve for the material under test.

Supplementary to these basic observations, the influence of mean stresses on the fatigue life, e.g. a constant mechanical preload, was examined in detail by Gerber and Goodman. They developed calculation methods to take the mean stress effect into account in the calculations.

The hypothesis of the linear **damage accumulation** was put forward by A. Palmgren in 1924 and published by M.A. Miner in 1945. It states that the component under examination fails at a total damage of $D = 1$.

$$D = \sum_{i=1}^{k} \frac{n_i}{N_i}$$

$N_i$ describes the maximum number of load cycles for a given cycle stress range until the component fails, while $n_i$ describes the number of load cycles to which the component has already been subjected. The ratio $n_i/N_i$ is referred to as partial damage and only takes into account the damage caused by a fixed stress range. The sum of all partial damages over all occurring stress ranges is then the total damage $D$. The index $i$ accordingly runs over $k$ classes, each of which defines a stress range.

To define mean stress and stress range: If a component is cyclically stressed with a harmonic signal of a constant amplitude yp, the stress range of each cycle equals 2*yp, with a mean stress of ym = 0. When an additional prestress ym is applied to the harmonic vibration, the resulting stress ranges do not change, but the mean stress for each cycle is changing to ym. Depending on direction (tension or compression) and amplitude of the mean stress, the resulting fatigue life of the component can significantly change. Here, mean stress corrections come into play to correct the cycle stress range to damage-equivalent cycles with zero mean stress.

In the following example, a component is stressed with a stress range of 200 MPa. For this stress, the Wöhler curve defines a number of cycles to failure of 100,000 (1E5) cycles. Once cycle thus "consumes" 1/100,000 of the fatigue life. The Wöhler curve is plotted double-logarithmically.
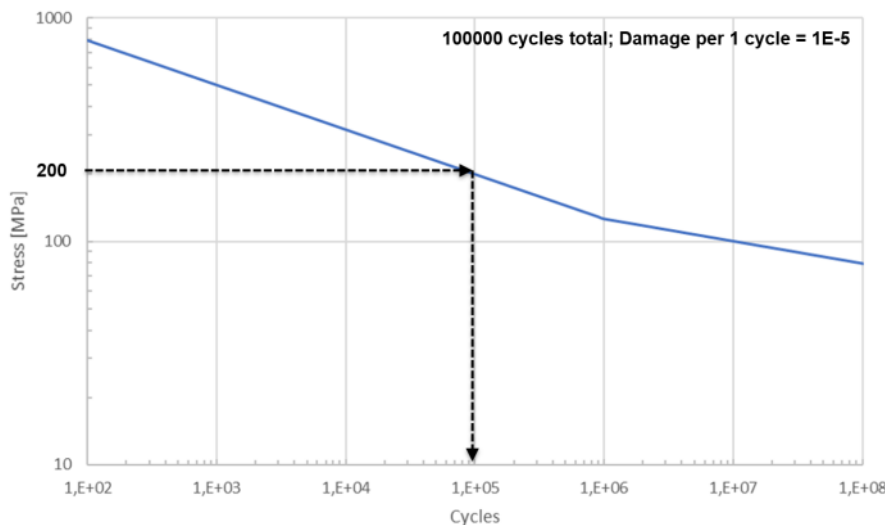
As opposed to material tests, stresses on real components have almost arbitrary signal curves. Therefore, it is necessary to bin the time series of the measured signals in classes using a suitable method, which can then be converted to the damage domain and accumulated according to the above Miner method.

M. Matsuishi and Tatsuo Endo presented a method in 1968 – the **rainflow cycle count** – that represents the time signal on a time axis rotated by 90°. The method got its name from the similarity to a pagoda roof, over which rain runs off at the reversal points. First of all, half cycles, e.g. in the pulling direction, are counted and combined with matching half cycles in the opposite stress direction, e.g. pushing direction, to form closed cycles. This method is standardized in ASTM E 1049-85 (Standard Practices for Cycle Counting in Fatigue Analysis) and is in widespread use.
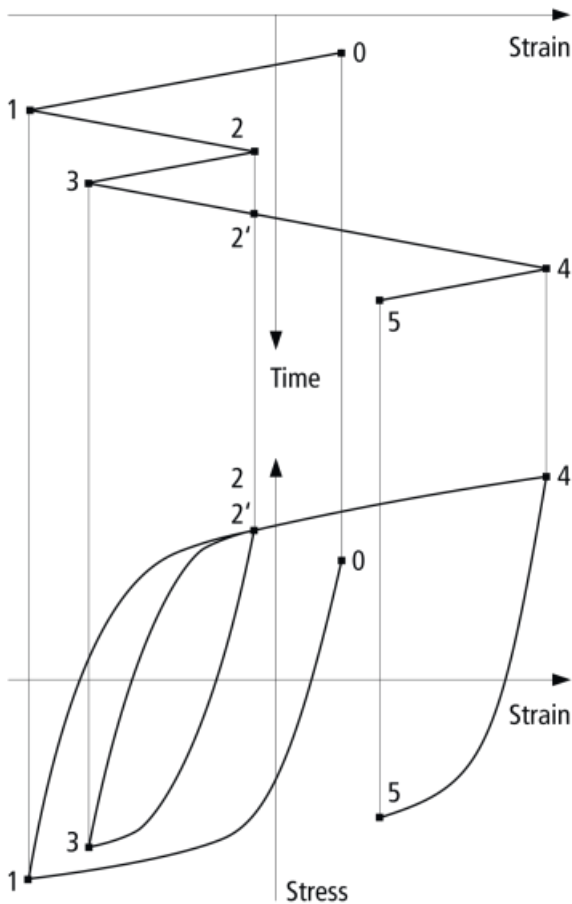


**Determination and meaning of Wöhler curves**

To determine the Wöhler curve, material samples with a defined sample geometry (frequently a round sample with a diameter of 10 mm) are clamped in a testing machine. The testing machine introduces cyclic forces into the sample, usually oscillating symmetrically around the zero value (mean stress-free) with a constant maximum force. However, variations in the cyclic compressive or tensile range are possible in order to determine material parameters with other mean stress values. The test is executed at a constant maximum amplitude up to failure (= fatigue fracture) or up to the technical incipient crack and the number of cycles reached for this load stage are entered in the diagram.

A material reacts in the case of an external stress, i.e. the introduced force $F$, on account of its geometry with cross section $A$, with a stress that is defined as the mechanical stress. With a known (initial) sample diameter and a known acting force, the resulting component stress can thus be determined as the quotient of $F$ and $A$. It is usually specified in the unit Megapascal (MPa).

Accordingly, a mechanical stress value is plotted in the Wöhler curve over the achievable number of cycles. In order to determine the natural variance of the tests, several repetitions are carried out per load stage with equivalent material and the same sample geometry.

The execution of fatigue strength tests is standardized in DIN 50100 among others. The ASTM has also published a standard procedure for fatigue tests with constant amplitude in ASTM E466-15.

**BECKHOFF**

By way of example, a Wöhler curve for aluminum in a simple logarithmic representation is shown in the following.

**S-N Curve for Brittle Aluminum with a UTS of 320 MPa**
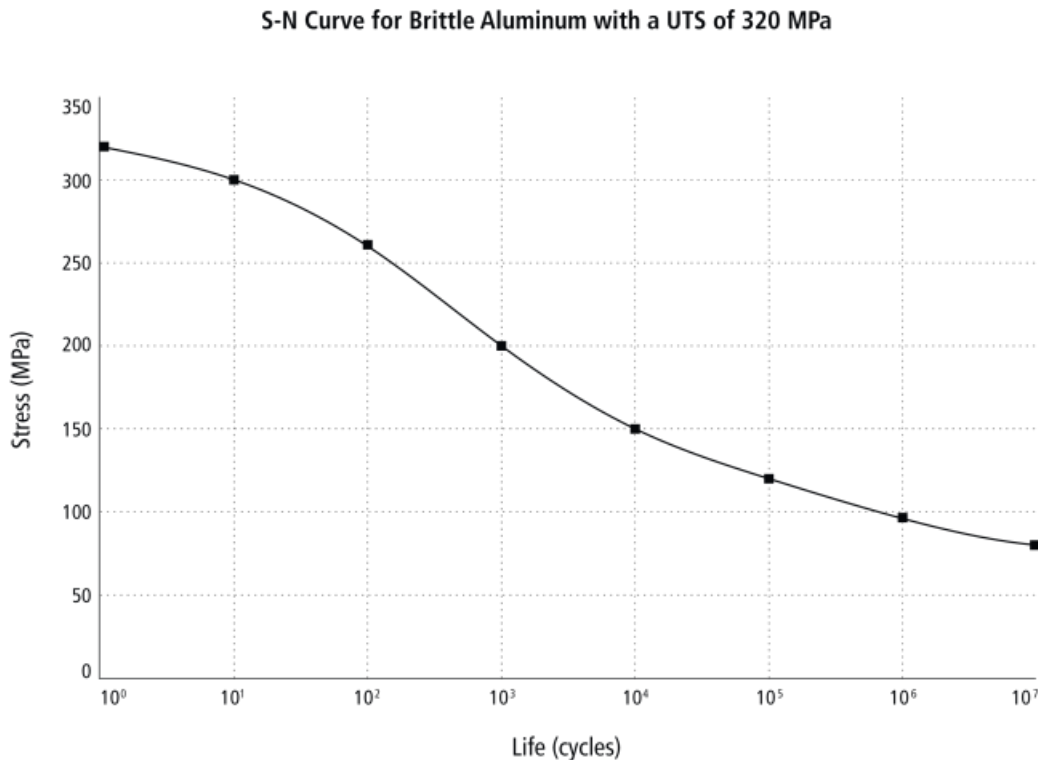
The Wöhler curve can be divided into three areas:

**Low Cycle Fatigue**, LCF: Area of high stress amplitudes
(N = 1 … approx. $10^4$ cycles). Shown in yellow in the graphic below.

- High stress takes place in this area, in which strong plastic deformation of the material occurs. In this area, strain-controlled material tests illustrate the behavior much better, so-called strain Wöhler lines are used.
- In practice, only a few components are designed for the low cycle fatigue area.

**High Cycle Fatigue**, HCF: Area of a constant slope when the Wöhler curve is plotted logarithmically in the x and y-axes ([N = approx. $10^4$ … $10^6$). Shown in green in the graphic below.

- Classic area of operationally reliable component design.
- In widespread use in industry and research and in many applications.
- Many material parameters available in literature.

**Very High Cycle Fatigue**, VHCF: Area with no significant drop of the Wöhler curve from a certain amplitude [N > approx. $10^6$]. Shown in blue in the graphic below.

- Theoretical stress limit below which no fatigue is observed (nevertheless, signs of fatigue appear e.g. due to corrosion and high temperatures).

Fatigue regions

Stress range [MPa, linear scale]

Cycles to failure [no unit, log. scale]
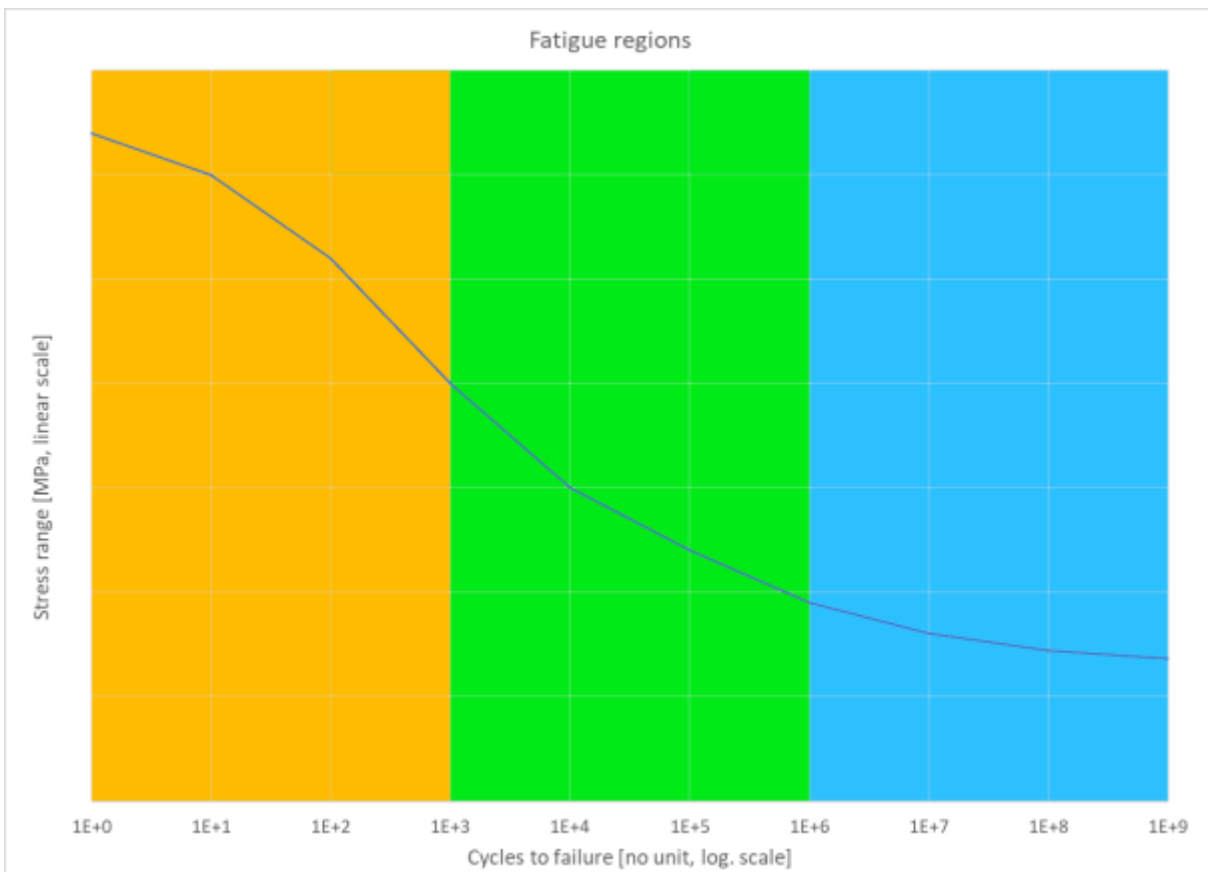
A usable approximation of a Wöhler curve on the basis of the above three areas is illustrated in the following. For this, the Wöhler curve is converted to a double logarithmic representation, which is then subdivided into three linear sections. This representation is frequently used and can be created from just a few material parameters.

Wöhler curve

Stress [MPa]

Number of Cycles

—●— Resulting Wöhler Curve      ● Transition Points
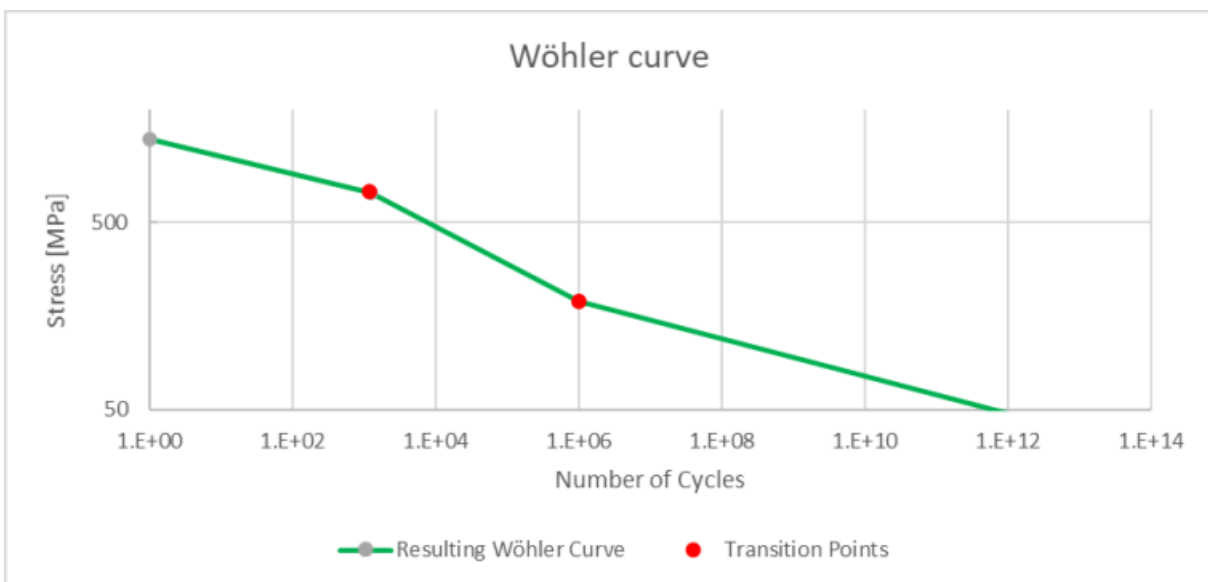
Common values for the slope k of the Wöhler curve are:

- Welded components: k = 3
- Electronic components: k = 4
- Components without stress concentrations: k = ~10

The flatter the slope of a Wöhler curve is (high k values), the less sensitive the material is in terms of its fatigue behavior. Smaller stress ranges thus cause less damage to the material, i.e. the tolerable number of cycles to failure up to a certain stress range is larger than with a material with a steeper slope of the Wöhler curve.

A steeper slope is suitable for components that are still susceptible to fatigue even with smaller stress ranges: welded constructions, components with stress concentrations or electronic components. In the following example, two Wöhler curves with the slopes k = 8 and k = 6 are plotted with the same stress intersection. The tolerable numbers of cycles to failure differ significantly (46,500 to 1.68*10^6).
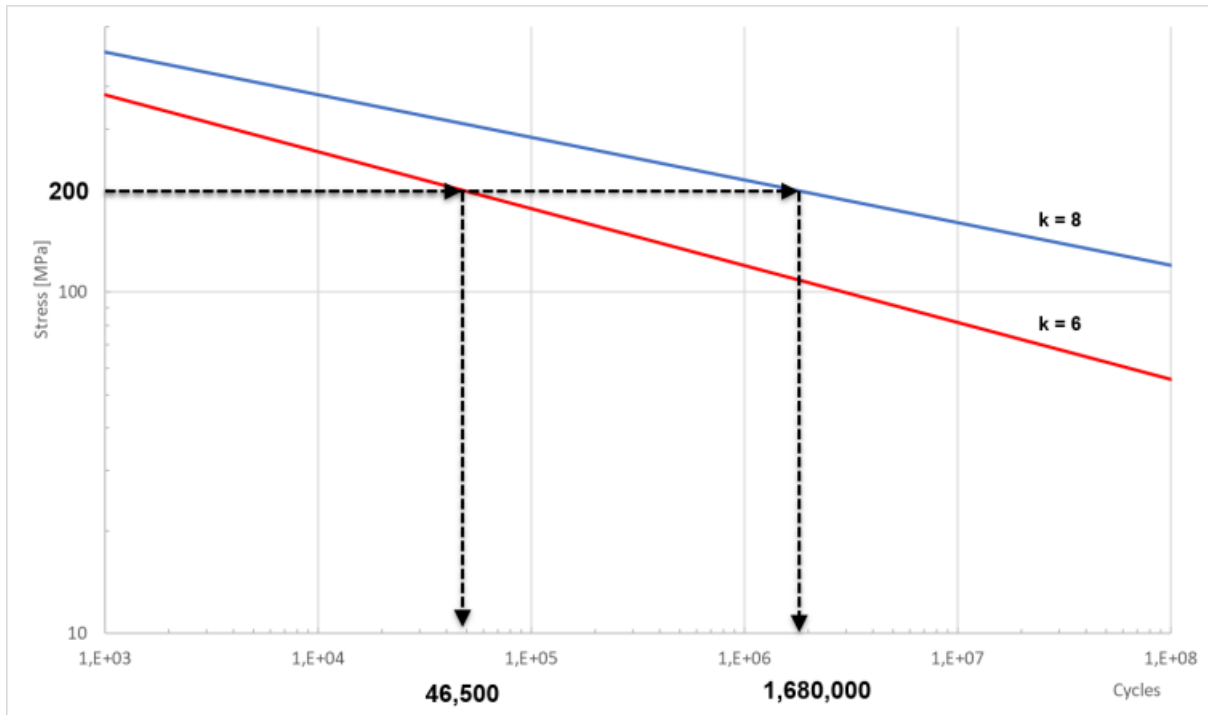


**Practical notes on interpreting the Wöhler curve**

When working in the topical area of material fatigue, you should always bear in mind that the calculations are always of a statistical nature. Wöhler curves are frequently defined over a 50 % survival probability. This means that at the specified point of a p50 Wöhler curve, 50 % of the test specimens tested have already been destroyed. If a higher survival probability is required, curves with corresponding survival probabilities are to be derived from the material tests.

Not only the determination of the material parameters abstracts the scattered measurement results in a few simple curve parameters, but also material samples differ greatly from real components. In particular the shortened lifetime of components due to stress concentrations plays an important role.

Geometry:

- Test specimens have no sharp edges and are ground (no notches, blowholes, etc.)
- Sharp edges and punch-outs in the component cause local stress increases and cracking

Material quality:

- Test specimens have a high material quality
- The quality of real materials in production often fluctuates (blowholes, variation of manufacturing processes, oxidation, electrocorrosion, damage due to deformation, etc.)

Machining processes:

- Welded joints in real components
- Machining processes result in sharp edges, cut-outs and notches. Reworking of the surface or a design change can reduce these stress concentrations.

Stress concentrations lead to significantly earlier material failure. In the following, the influence of a *stress concentration factor* on the Wöhler curve is shown, which leads to a lowering of the curve and reduces the tolerable number of cycles to failure by a factor of 6.
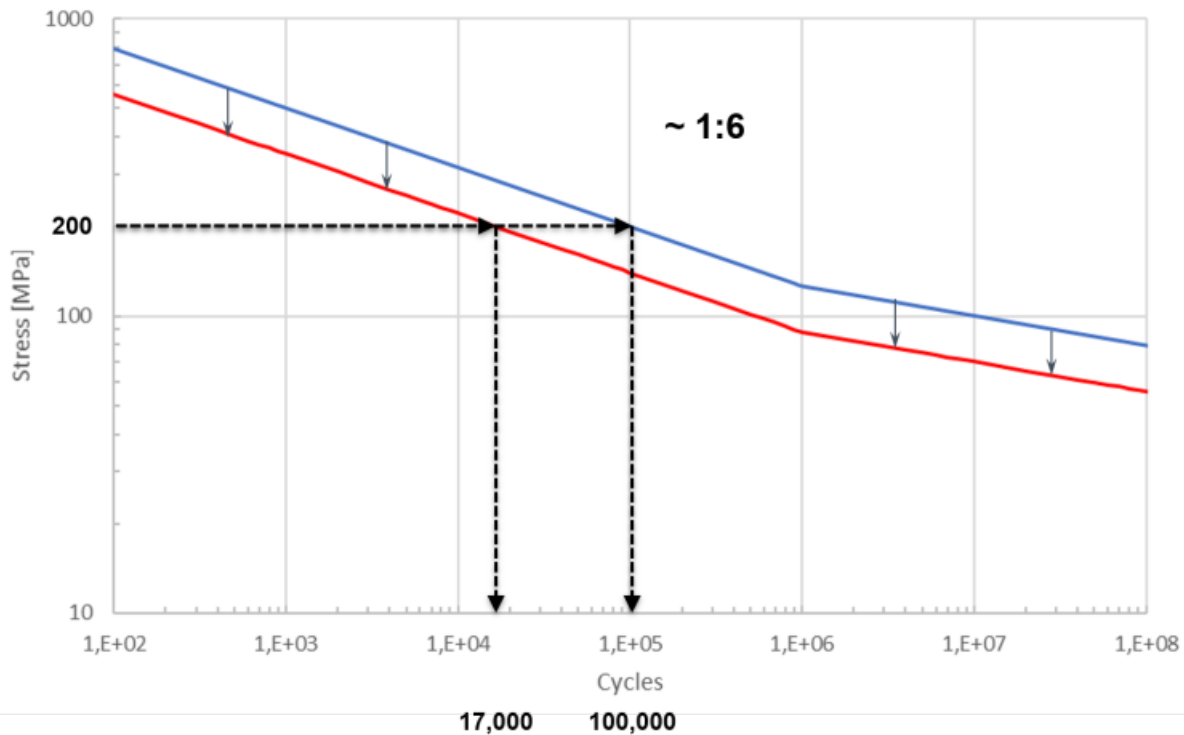
If a precise statement on fatigue strength is required for a structural part/component, it is possible – depending on the component cost, size and availability of the component and test equipment – to determine a so-called "component Wöhler curve". In addition to the high costs for testing, the disadvantage here is that, although the significance for the current component is high, this can only be used to a limited extent if the component design changes.

**Monitoring strategy: Fatigue or service life analysis**

The load data is determined directly on the test specimen, e.g. by measurement with strain, force or pressure sensors. After the cycle counting with the rainflow algorithm, all the necessary information is available from the point of view of the loads. With regard to the fatigue assessment according to Miner, the material properties (Wöhler curve) are now to be consulted. The determination of the fatigue properties of the material used is often difficult, often there are few or no material characteristics available or they have to be estimated.

Practically, the following considerations help:

- Limitation to a relative comparison of identical components
- Use of literature data and corresponding conversion into an approximated Wöhler curve

**Relative comparison of components**

Generic Wöhler curves with defined slopes are suitable especially for relative comparisons of identical components, e.g. machine parts in series machine construction. The absolute damage sum is not meaningful here, but statements such as "Machine part B is 40 % more damaged than machine part A" can be derived and thus maintenance decisions can be supported.

**Use of literature sources**

For the determination of absolute damage values, Wöhler curves are usually determined from real material samples in testing machines. However, with Eurocode 3 and the FKM Directive, sources of material parameters of steel are also available from which Wöhler curves can be calculated.

> ℹ In the following, stress ranges are always specified and not, as is usual in some sources, peak amplitudes. The stress range is twice as large as the peak amplitude.

The **FKM Directive** (Mechanical Engineering Research Curatorship, Association in the VDMA) specifies conservative material characteristics for different types and conditions of materials. An example from the FKM Directive (for more, see section Appendix [▶ 59]).

| Sort | UTS [Stress range, MPa] | Stress range at very high cycle fatigue [MPa] | k | NC | SRI [MPa] |
|------|-------------------------|-----------------------------------------------|---|-----|-----------|
| S185 | 620 | 280 | 5 | 1E+06 | 4438 |

Translation into the material parameters for the function F_CM_CalculateWoehlerCurve [▶ 263] results in:

```
fSRI : LREAL := 4438;
fUTS : LREAL := 620;
bUseUTSCorrection : BOOL := TRUE;
fK1 : LREAL := 5;
fK2 : LREAL := 0;
nNC1 : ULINT := 18790;
nNC2 : ULINT := 1E+06;
```

`fSRI` and `fUTS` can therefore be directly applied. The slope is 5 as specified and refers to the area between the transition from the UTS correction to the very high cycle fatigue limit. The very high cycle fatigue limit is defined for NC, which is entered as `nNC2`. The slope in the very high cycle fatigue range is of course `fK2 = 0`. Only as an input into the function F_CM_CalculateWoehlerCurve [▶ 263], the transition point from the UTS correction to the area with slope 5 must additionally be calculated via the equation

$$NC1 = \left(\frac{UTS}{SRI}\right)^{-k}$$

.

**Eurocode 3** (BS EN1993-1-9: 2005) defines different Wöhler curves for steel in different design details, in particular rolled/extruded products, welded and bolted connections and different geometries, such as T-beams. All Wöhler curves have these three section definitions in common:

- $N \leq 5*10^6$ where k = 3
- $5*10^6 \leq N \leq 10^8$ where k = 5
- $N \geq 10^8$ where k=0 (not fatigue critical)

An example from Eurocode 3 (see section Appendix [▶ 59] for more information).

| Design detail | Δσc [stress range, MPa] | k1 | k2 | N at Δσc | NC1 | ND (very high cycle fatigue limit) | SRI at N = 1 [MPa] | ΔσNC1 at NC1 [MPa] | Δσl at ND [MPa] |
|---------------|-------------------------|----|----|----------|-----|-----------------------------------|--------------------|--------------------|-----------------|
| 160 | 160 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 20159 | 118 | 65 |

A rolled or extruded product, manufactured as a plate, hollow cylinder or similar shape, is defined in the Eurocode as design detail 160.

Translation into the material parameters from the provided Fatigue Analysis [▶ 343] results in:

```
fSRI : LREAL := 2580;
fUTS : LREAL := 20159;
bUseUTSCorrection : BOOL := TRUE;
fK1 : LREAL := 5;
fK2 : LREAL := 0;
nNC1 : ULINT := 5E+06;
nNC2 : ULINT := 1E+08;
```

The parameterization was converted here in such a way that the requirement for three defined zones with slope 3, 5, and 0 results when using the function F_CM_CalculateWoehlerCurve [▶ 263]. The UTS correction is used here in such a way that a slope of 3 results in the first section. This is why `fUTS` is also larger than `fSRI`. The value for `fSRI` with $k_2$ = 5 is then

$$fSRI = \Delta\sigma_{NC1} NC1^{(1/k_2)}$$

`fUTS` is then set to SRI from the Eurocode. This results in a slope of 3 in the first section. Setting `fK1` = k2 and `fK2` = 0 results in the other two sections.

## Summary

The **input values** of a fatigue life analysis can be divided into three groups:

- **Material properties** describe the fatigue strength of the material, usually as a Wöhler curve, supplemented by surface texture and processing.
- **Geometrical factors** are used to adapt the material parameters to local geometrical properties that affect the fatigue life (e.g. notches or sharp edges).
- **Loads** are signals acting on the component that lead to stress. They are often measured with strain gauges or with force or torque transducers.

The **analysis part** contains the following components, depending on the degree of complexity:

- **Corrections** of material parameters and use of geometrical factors
- **Cycle count** for the conversion of time signals into classes (mean stress, stress range) for processing in damage accumulation
- **Mean stress corrections** to take into account the reduction of fatigue life due to mean stress effects
- **Damage accumulation** for the summation of the individual damage components from the cycle matrix, material parameters and correction factors

As a result, the damage sum is available for further comparisons, for example of identical components, or as a basis for warning/alarm messages. The monotonously increasing course of the damage sum is well suited for extrapolation to defined limits in order to identify critically stressed components and to be able to plan maintenance operations better.

This method is not only suitable for the classic fatigue analysis to calculate an absolute damage number, but also for relative comparisons of the damage intensity: Force or pressure signals, temperature cycles or other vibrating signal forms can be used to extend static methods (extreme values, crest factor, kurtosis, etc.) by meaningful information.
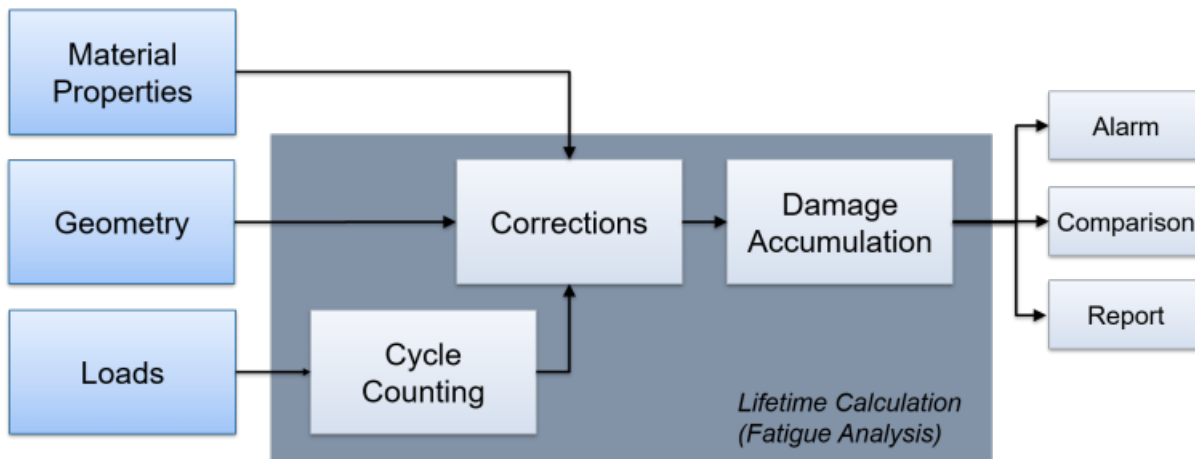


The three elements "Cycle Counting", "Corrections" and "Damage Accumulation", which are the central elements in the above graphic, are implemented in the TwinCAT 3 Condition Monitoring function with FB_CMA_RainflowCounting [▶ 206], FB_CMA_MeanStressCorrection [▶ 171] and FB_CMA_MinersRule [▶ 175].

## Appendix: Eurocode and FKM Directive

**Eurocode 3** (EN 1993-1-9:2005 - Design of steel structures)

| Design detail | Δσc [stress range, MPa] | k1 | k2 | N at Δσc | NC1 | ND (very high cycle fatigue limit) | SRI at N = 1 [MPa] | ΔσNC1 at NC1 [MPa] | Δσl at ND [MPa] |
|---|---|---|---|---|---|---|---|---|---|
| 160 | 160 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 20159 | 118 | 65 |
| 140 | 140 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 17639 | 103 | 57 |

| Design detail | Δσc [stress range, MPa] | k1 | k2 | N at Δσc | NC1 | ND (very high cycle fatigue limit) | SRI at N = 1 [MPa] | ΔσNC1 at NC1 [MPa] | Δσl at ND [MPa] |
|---|---|---|---|---|---|---|---|---|---|
| 125 | 125 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 15749 | 92 | 51 |
| 112 | 112 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 14111 | 83 | 45 |
| 100 | 100 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 12599 | 74 | 40 |
| 90 | 90 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 11339 | 66 | 36 |
| 80 | 80 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 10079 | 59 | 32 |
| 71 | 71 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 8945 | 52 | 29 |
| 63 | 63 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 7938 | 46 | 25 |
| 56 | 56 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 7056 | 41 | 23 |
| 50 | 50 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 6300 | 37 | 20 |
| 45 | 45 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 5670 | 33 | 18 |
| 40 | 40 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 5040 | 29 | 16 |
| 36 | 36 | 3 | 5 | 2E+06 | 5E+06 | 1E+08 | 4536 | 27 | 15 |

**FKM Directive**

The FKM directive specifies conservative material characteristics for different types and conditions of material. The following values for unalloyed structural steel, unalloyed fine grained structural steel, heat treatable steel and cast iron apply to a survival probability of 97.5 %.

Unalloyed structural steel

| Sort | UTS [Stress range, MPa] | Stress range at very high cycle fatigue [MPa] | k | NC | SRI [MPa] |
|---|---|---|---|---|---|
| S185 | 620 | 280 | 5 | 1E+06 | 4438 |
| S235 | 720 | 320 | 5 | 1E+06 | 5072 |
| S275 | 860 | 390 | 5 | 1E+06 | 6181 |
| S355 | 1020 | 460 | 5 | 1E+06 | 7291 |
| S450 | 1100 | 500 | 5 | 1E+06 | 7924 |
| E295 | 980 | 440 | 5 | 1E+06 | 6974 |
| E335 | 1180 | 530 | 5 | 1E+06 | 8400 |
| E360 | 1380 | 620 | 5 | 1E+06 | 9826 |

Tempered steel, tempered state

| Sort | UTS [Stress range, MPa] | Stress range at very high cycle fatigue [MPa] | k | NC | SRI [MPa] |
|---|---|---|---|---|---|
| C22 E/R | 1000 | 450 | 5 | 1E+06 | 7132 |
| C35 E/R/- | 1260 | 570 | 5 | 1E+06 | 9034 |
| C40 E/R/- | 1300 | 590 | 5 | 1E+06 | 9351 |
| C45 E/R/- | 1400 | 630 | 5 | 1E+06 | 9985 |
| C50 E/R/- | 1500 | 680 | 5 | 1E+06 | 10777 |
| C55 E/R/- | 1600 | 720 | 5 | 1E+06 | 11411 |
| C60 E/R/- | 1700 | 770 | 5 | 1E+06 | 12204 |
| 28Mn6 | 1600 | 720 | 5 | 1E+06 | 11411 |
| 38Cr2 | 1600 | 720 | 5 | 1E+06 | 11411 |
| 46Cr2 | 1800 | 810 | 5 | 1E+06 | 12838 |
| 34Cr4 | 1800 | 810 | 5 | 1E+06 | 12838 |
| 37Cr4 | 1900 | 860 | 5 | 1E+06 | 13630 |
| 41Cr4 | 2000 | 900 | 5 | 1E+06 | 14264 |

| Sort | UTS [Stress range, MPa] | Stress range at very high cycle fatigue [MPa] | k | NC | SRI [MPa] |
|---|---|---|---|---|---|
| 25CrMo4 | 1800 | 810 | 5 | 1E+06 | 12838 |
| 34CrMo4 | 200 | 450 | 5 | 1E+06 | 7132 |
| 42CrMo4 | 2200 | 990 | 5 | 1E+06 | 15690 |
| 50CrMo4 | 2200 | 990 | 5 | 1E+06 | 15690 |
| 34CrNiMo8 | 2400 | 1080 | 5 | 1E+06 | 17117 |
| 30CrNiMo8 | 2500 | 1130 | 5 | 1E+06 | 17909 |
| 35NiCr6 | 1760 | 790 | 5 | 1E+06 | 12521 |
| 36NiCrMo16 | 2500 | 1130 | 5 | 1E+06 | 17909 |
| 39NiCrMo3 | 1960 | 880 | 5 | 1E+06 | 13947 |
| 30NiCrMo16-6 | 2160 | 970 | 5 | 1E+06 | 15373 |
| 51CrV4 | 2200 | 990 | 5 | 1E+06 | 15690 |

# 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 system requirements for the Condition Monitoring product for engineering and/or runtime systems. The Condition Monitoring installation must be carried out on the engineering and the runtime system.

---

**●**
**ⓘ** **Version-dependent range of functions**

The full range of functions described (see section PLC API) is only ensured with the installation of the current versions of the driver and PLC libraries. The use of previous versions is possible with a limited range of functions. A detailed overview of the functionality extended by Version CM3.2 can be found in the section Compatibility [▶ 64].

The Overview [▶ 10] also contains a list of library functions that depends on the product level.

---

**Engineering**

Engineering system is the term used to describe a computer that is used for the development of program code, but does not run any program code. An engineering system must meet the following requirements:

- TwinCAT 3.1 XAE (engineering installation) build 4022.25 or higher*

**Runtime**

Runtime system is the term used to describe an industrial or embedded PC on which the program code is run. A runtime system must meet the following requirements:

- TwinCAT 3.1 XAR (runtime installation) build 4022.25 or higher*
- Operating systems: Win 7, Win 10, Windows Embedded Standard 7
- A license for TC1200 PLC and for TF360x Condition Monitoring

---

**●**
**ⓘ** A 7-day trial license can be activated repeatedly for testing purposes

---

**Engineering and runtime on the same system**

In the case where engineering and runtime are to be used on one system, the following system requirements must be met:

- TwinCAT 3.1 XAE (engineering installation) build 4022.25 or higher*
- A license for TC1200 PLC and for TF360x Condition Monitoring

---

**●**
**ⓘ** A 7-day trial license can be activated repeatedly for testing purposes

---

*: Use with a limited range of functions is possible from TwinCAT 3.1 Build 4018.

## 3.2 Compatibility

The TwinCAT Condition Monitoring library has been on the market for many years. The CM version 3.1.x has been used successfully in many applications. In order to meet the latest requirements and options in the areas of algorithmics and TwinCAT, a new version 3.2.x is available. In its development we have paid particular attention to compatibility with existing applications. It is thus possible that you will have to adapt your existing application for the use of the version 3.2.x in very few special cases.

- The `Enums` used in the InitPars structures are `externalTypes` and can only be used without library namespace.

- CM `Enum` values may not be used without type qualifiers in libraries implemented by the user.

**Overview of released version of the Condition Monitoring library**

| TcCM 3.1.x | 3.1.16 | 3.1.17 | 3.1.18 | |
|---|---|---|---|---|
| TcCM 3.2.x | | | | 3.2.20 |

**Changes with Version CM 3.2 (CM Setup Version)**

The changes/extensions to the Condition Monitoring library explained below require the following minimum versions of the PLC libraries and drivers:

| Tc3_CM | Tc3_CM_Base | Tc3_MultiArray | TcCM.sys | TcMultiArray.sys |
|---|---|---|---|---|
| 2.0.30.0 | 2.1.18.0 | 2.0.14.0 | 3.4.17.0 | 3.4.17.0 |

**Fundamental changes**

- Multichannel capability of the algorithms, i.e. sequential processing of several channels by means of a single instance. This simplifies the implementation of PLC applications.
- Extension of the choice of window functions by `eCM_BartlettWindow`, `eCM_KaiserWindow` and `eCM_FlatTopWindow` as well as the option to freely set the overlap when using the Welch method. Refer here to the section Window functions [▶ 20].
- Use of the TC3 EventLogger for outputting errors and information. Refer here to the section Online view [▶ 83].
- Results on most function blocks can now be viewed directly in the Tc3 Scope View. Refer here to the section Online view [▶ 83].
- Overall increase in the performance and improvement of the numerics.

**Application-oriented algorithms**

- Vibration assessment [▶ 34] on the basis of ISO 10816-3 for the vibration-based classification of the machine state with the function block FB_CMA_VibrationAssessment [▶ 251].

**Extension of existing function blocks**

- FB_CMA_Source [▶ 238]: Simplified methods for the supply of single and multi-channel input data. The function block also offers the option of triggering a reset of the complete analysis chain.
- FB_CMA_Sink [▶ 231]: Method for outputting 3D data, required for the function blocks FB_CMA_ArgSort [▶ 91] and FB_CMA_IntegratedRMS [▶ 163], as well as simplified variants of the methods for outputting results.
- FB_CMA_IntegratedRMS [▶ 163]: Calculation (optional) of integrated RMS values on configured frequency bands.
- FB_CMA_MomentCoefficients [▶ 179]: Option to calculate the excess in two variants: If `bExcessKurtosis := TRUE`, the kurtosis is reduced by three.

# 3.3 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**.
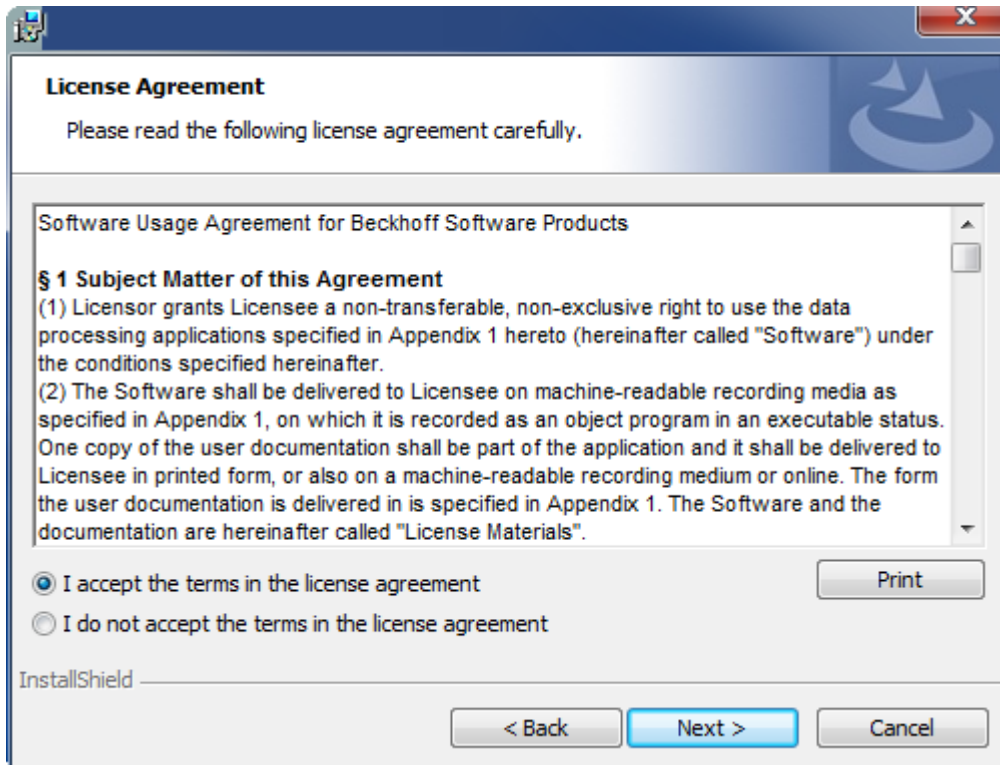


3. Enter your user data.
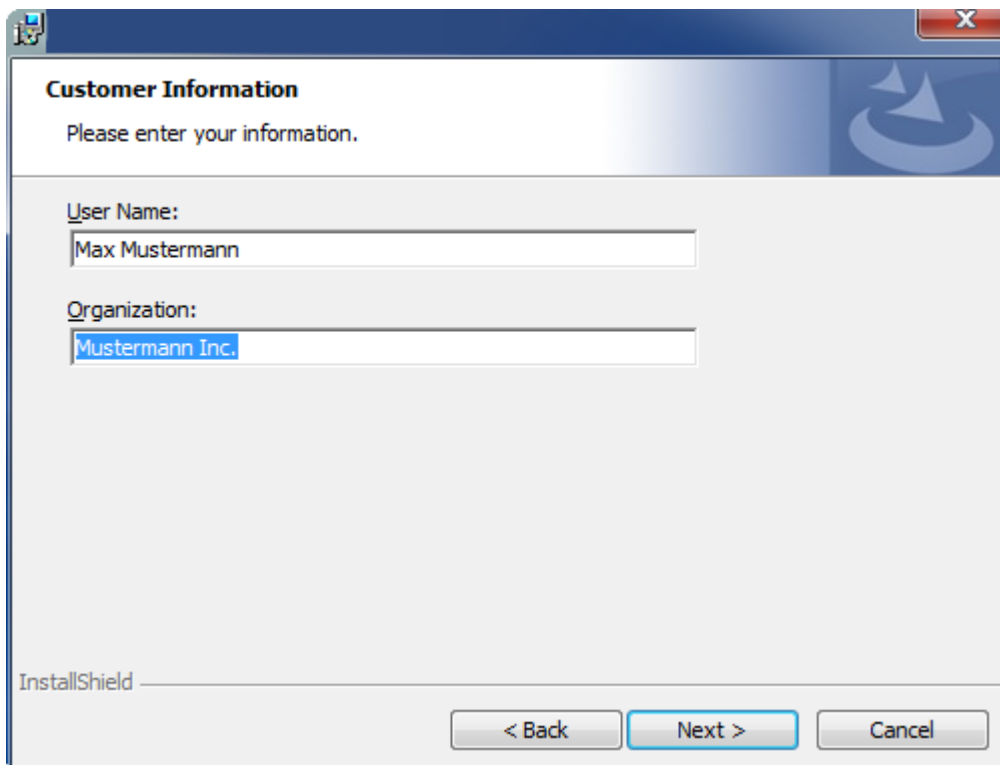
4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.
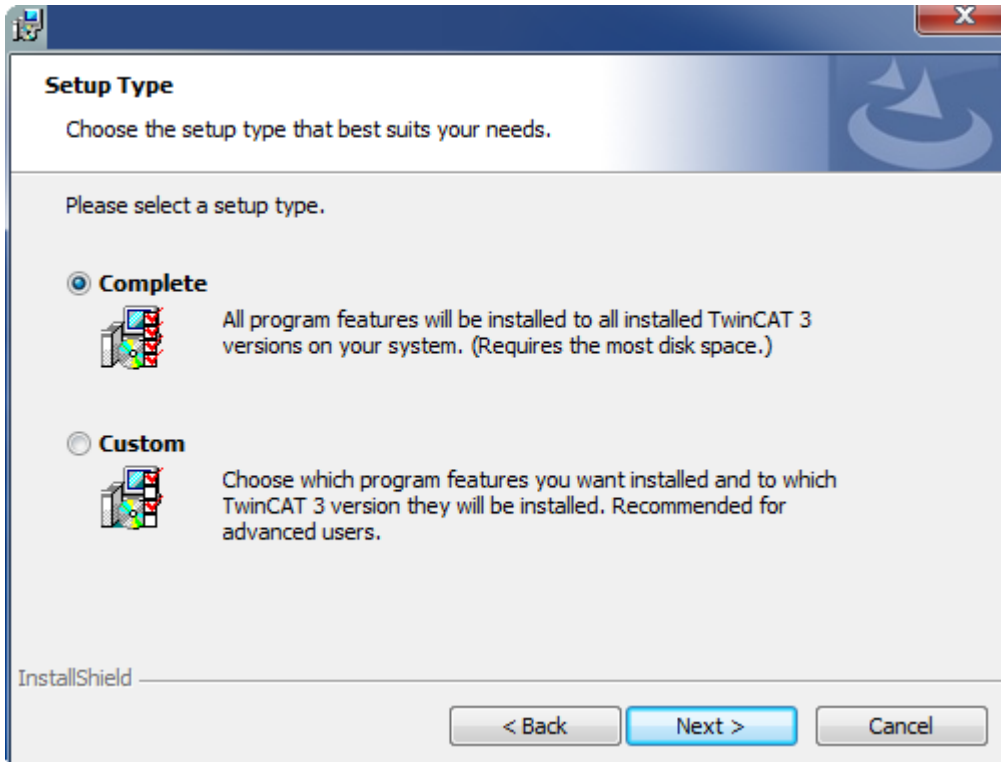


5. Select **Next**, then **Install** to start the installation.
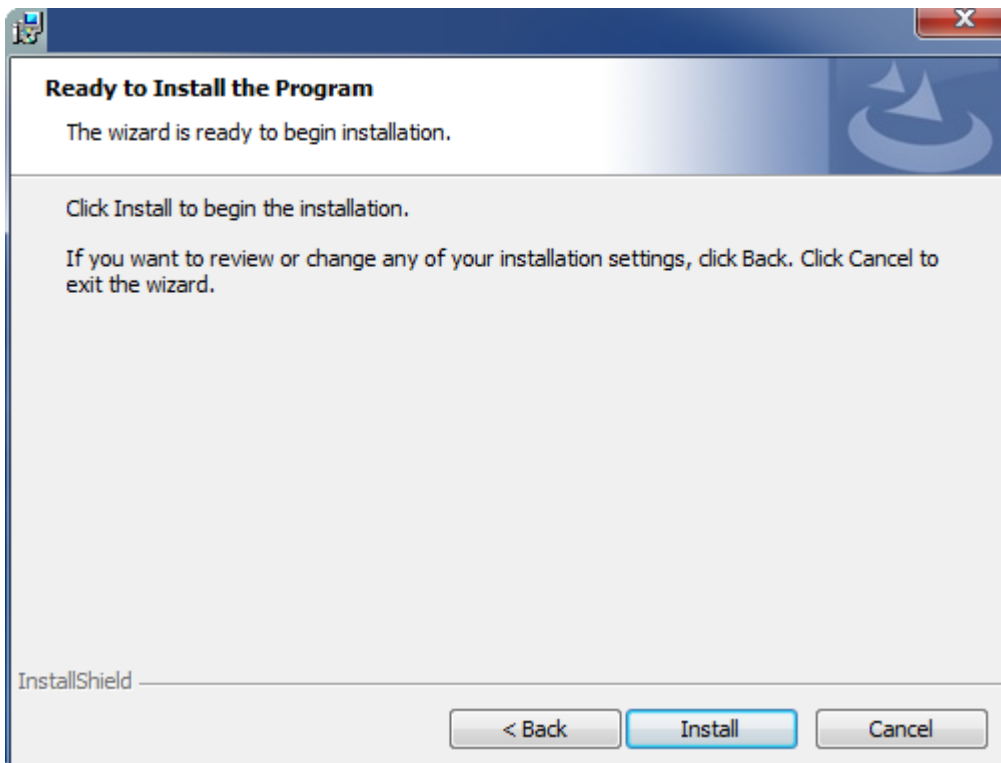


⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

6. Confirm the dialog with **Yes**.
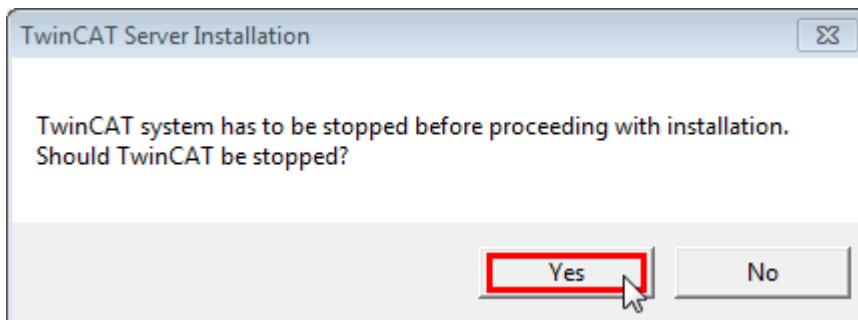
7. Select **Finish** to exit the setup.
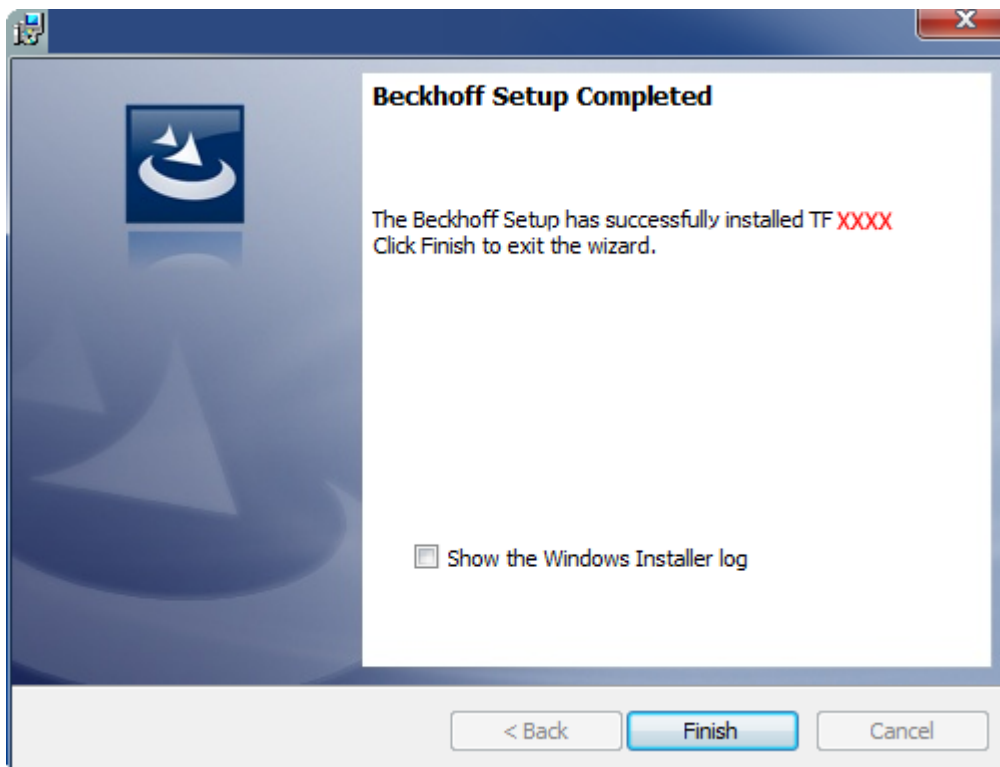
⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▶ 68]).

# 3.4 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).

**Licensing the full version of a TwinCAT 3 Function**

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

**Licensing the 7-day test version of a TwinCAT 3 Function**

ℹ A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

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.
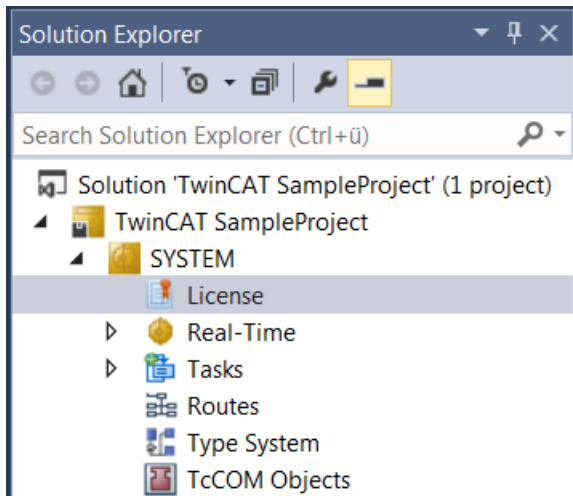


   ⇨ The TwinCAT 3 license manager opens.

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. "TF4100 TC3 Controller Toolbox").
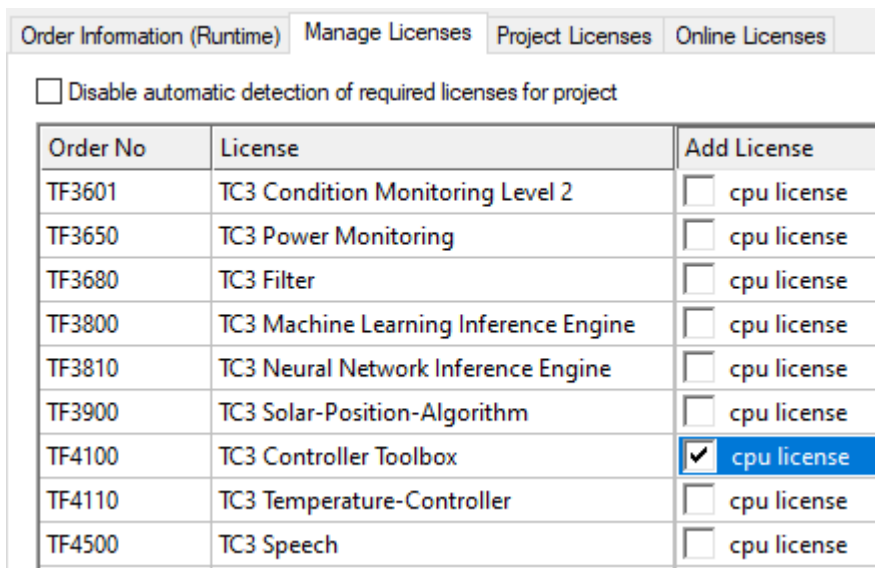


6. Open the **Order Information (Runtime)** tab.

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

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



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



8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.
⇨ The 7-day trial version is enabled.

# 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.
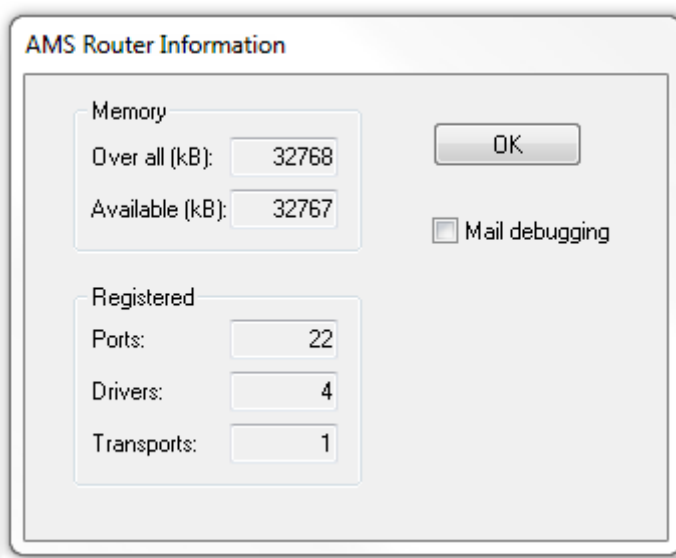


To increase the router memory capacity, a value in MB is entered in the TwinCAT configuration under System\ Real-Time\ Settings and the configuration is activated.

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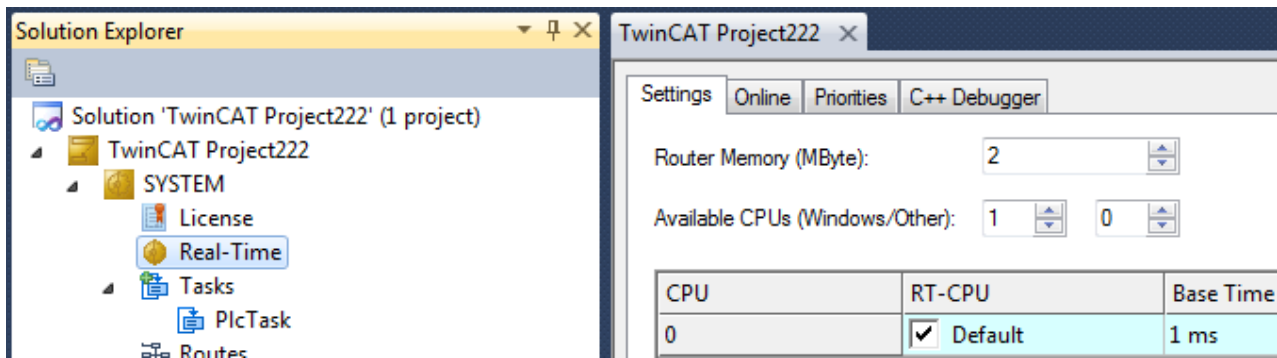
## 4.2 Task Setting

**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 <u>section "Parallel processing" [▶ 75]</u>.

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.

**Sample**: The data collection is stored in buffers, the size of which was declared as 1600 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.

---

**ⓘ Guide value for the upper limit of the calculation cycle time**

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 function 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**

**i** 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.

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" [▶ 74].

| ⚠ CAUTION |
|---|
| **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 [▶ 357] 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 IEEE 754 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:

- If insufficient valid values are available for a statistical evaluation, the result is output with NaN.
- If certain values are to be excluded from a statistical evaluation, this is achieved by the function block ignoring NaN values at the input.
- If the input vector of the frequency analysis of a time series contains one or more NaN values, the entire output vector 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. Because it is not possible to calculate correct spectra on the basis of fragmented time series.

The ability of a function block to create NaN values is noted in its description.

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 C 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.

| ⚠ CAUTION |
|---|
| **Software malfunctions** |
| NaN values may only be used in 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! |

| ⚠ CAUTION |
|---|
| **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 [▶ 72].

# 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 [▶ 72]".

**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 [▶ 77]) 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 $\Delta f$ and sampling rate $f_s$ is: $N = f_s / \Delta f$ (for simplification, a rectangular window is assumed here). The result is an FFT length of $N = 125000$. In addition, the FFT length $N'$ must be a power of two; hence, with $\log_2(125000) = 16.93$, it follows that the signal of length $N$ to $N' = 2^{17} = 131072$ is filled with zeros.

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 / 20 = 6250, 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 [▶ 72], a maximum cycle time of 1,562.5 ms is allowed for the calculating task. This requirement is clearly met with the value of 20 ms.

Legend:

$x[n]$ 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** — **TcTransferTray** — **Magnitude Spectrum**

$x[n]$

1. Write access by 1ms Task
2. Read access by 20ms Task
3. Ready to reuse

$x_w[n]$

**Sink**

$|X[m]|$

1. Write access by 20ms Task
2. Read access by 1ms Task
3. Ready to reuse

$|X[m]|$

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.

**ℹ 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.

---

ℹ **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 downstream 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 [▶ 77]".

# 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 [▶ 75].

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.

---

ℹ **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 MultiArray 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.

> ℹ **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 fatigue 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 [▶ 296] 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 [▶ 238] (or FB_CMA_Sink [▶ 231], FB_CMA_BufferConverting [▶ 94]) 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.

---

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 FB_CMA_MomentCoefficients [▶ 179]. `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.

---

aBuffer [1..cChannels] of [1..cOversamples]
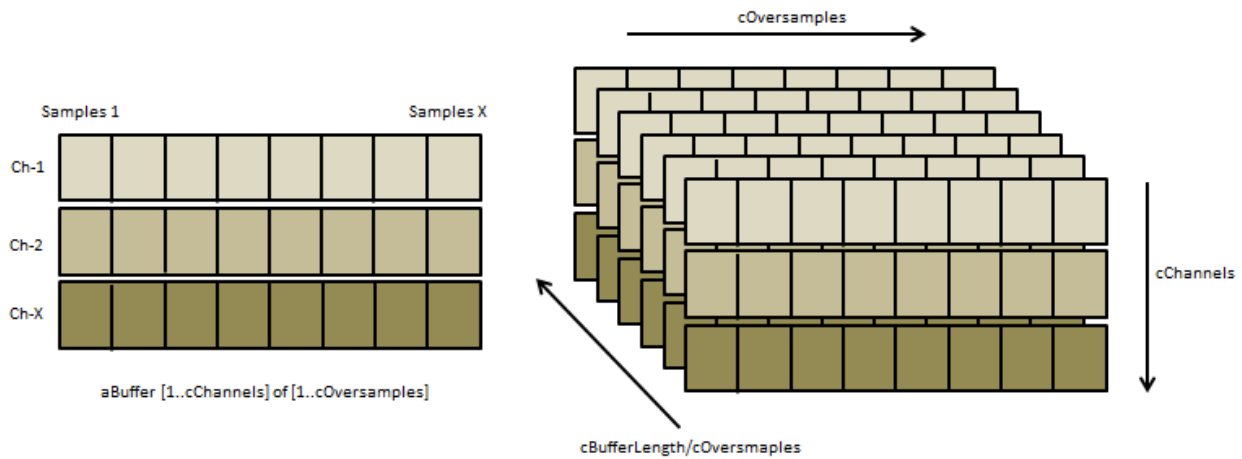
Because the **MultiArray** is **two-dimensional**, this is done by calling the **Input2D() method**.

```
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, for example, the variable `aBuffer : ARRAY [1.. cOversamples] OF ARRAY [1.. 3] OF LREAL` could be declared and the necessary transposition 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`).
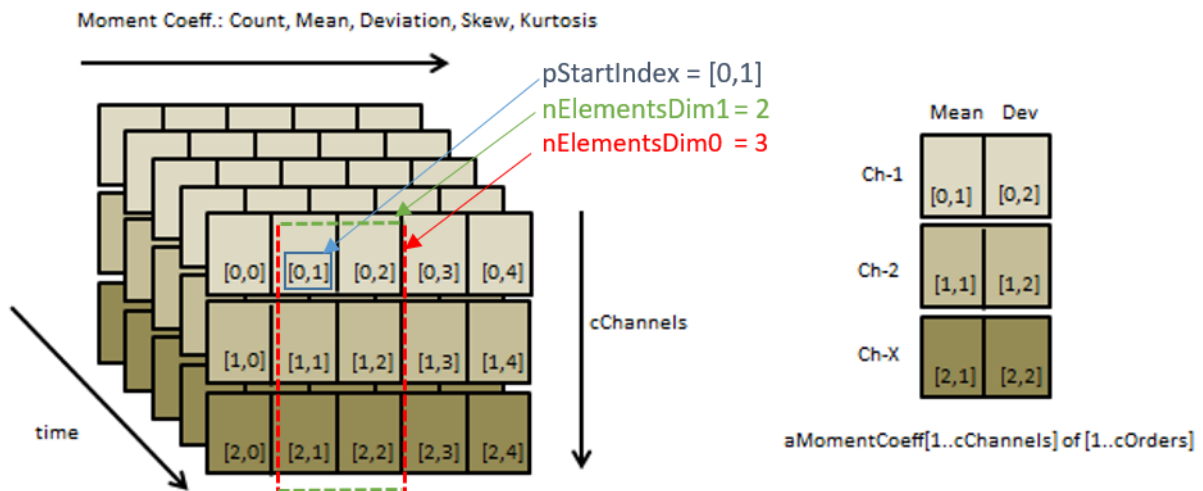


In the configuration shown, the Output2D() method will only copy one segment of the MultiArray into the PLC variable `aMomentCoef`. The segment to be copied is configured with the parameters `nWorkDim0`, `nWorkDim1`, `nElementsDim0`, `nElementsDim1` and `pStartIndex` as explained above.

# 5  PLC API

The TwinCAT3 Condition Monitoring Library provides analysis options in a TwinCAT PLC application. Please refer to our product description [▶ 10] 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 [▶ 75]). 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.
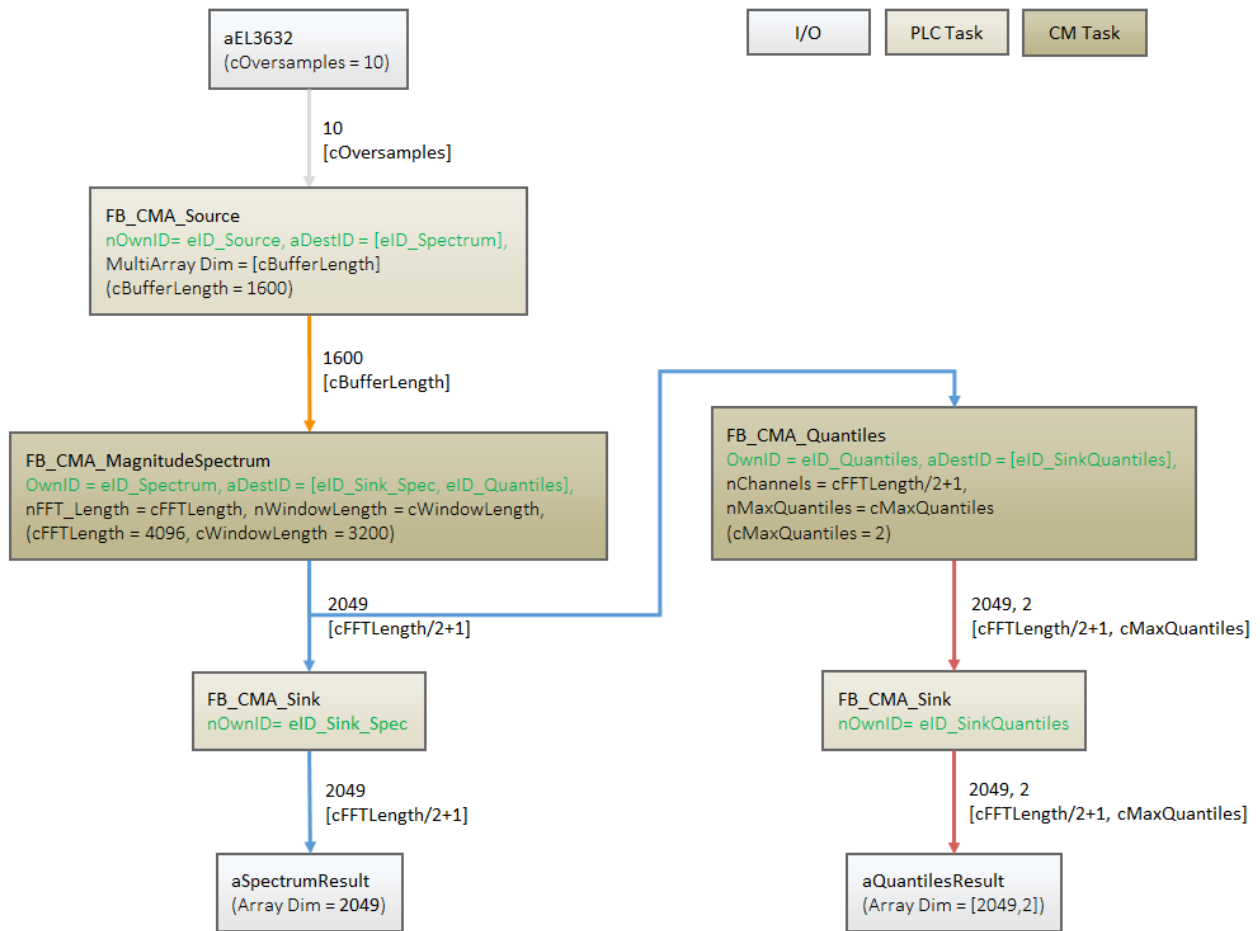
> **ℹ Cyclic call**
>
> As long as the functionality of FB_CMA_Source is called and signal data is transferred to a target function block, all other modules of the analysis chain must be called cyclically. See description of the internal communication principle in chapter Parallel processing. If not 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.

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

**Online View**

The function blocks of the Condition Monitoring library hold significant information about the current state of the function block in the Online View. These can be used for the programming of the application (initialization and configuration of the function blocks, adjustment of the buffer sizes in the analysis chain, etc.) as well as the analysis and evaluation (error analysis, graphic display).

The graphic below shows the Online View of the function block FB_CMA_MagnitudeSpectrum. Some nodes and their possible uses are explained in more detail below.

| Expression | Type | Value |
|---|---|---|
| ⊟ ◈ fbAlgorithm | FB_CMA_MagnitudeSpectrum | |
| ◈ bError | BOOL | FALSE |
| ◈ hrErrorCode | HRESULT | 16#00000000 |
| ⊞ ◈ ipErrorMessage | I_TcMessage | 16#FFFFBB8691299868 |
| ◈ nCntResults | ULINT | 0 |
| ◈ nOwnID | UDINT | 2 |
| ⊞ ◈ aDestIDs | ARRAY [1..cCMA_MaxDest] OF UDINT | |
| ◈ nResultBuffers | UDINT | 4 |
| ◈ tTransferTimeout | LTIME | LTIME#500us |
| ⊞ ◈ stInitPars | ST_CM_MagnitudeSpectrum_InitPars | |
| ⊞ ◈ stInitParsResultBuffer | ST_MA_MultiArray_InitPars | |
| ⊞ ◈ fbImplementation | FB_CMA_Implementation | |
| ⊞ ◈ fbBaseAlgorithm | FB_CM_MagnitudeSpectrum | |
| ◈ eTraceLevel | TCEVENTSEVERITY | Critical |

**Return values**

In case of error, `bError := TRUE` is set and the return value `hrErrorCode` contains the error code; see Error Codes Overview [▶ 357].

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

The reading of the error list and an example of the handling can be found in the example Multi-channel magnitude spectrum [▶ 306].

**Error description and information**

More detailed information about the current return value can be found in `ipErrorMessage. sEventText` shows the text description of an error that has occurred. Depending on the setting of TcEventSeverity (initial parameter `eTraceLevel,` see also Param [▶ 298] setting), this message is output via the TC3 EventLogger.

| ⊟ ◈ ipErrorMessage | I_TcMessage | 16#FFFFBB8691299868 |
|---|---|---|
| ◈ eSeverity | TCEVENTSEVERITY | Error |
| ⊞ ◈ ipSourceInfo | I_TcSourceInfo | 16#FFFFBB86912996B0 |
| ◈ nEventId | UDINT | 8230 |
| ◈ sEventClassName | STRING(255) | 'TcMultiArrayEventClass' |
| ◈ sEventText | STRING(255) | 'Input data has illegal shape.' |

**Initial parameters**

The initial parameters of the function block, which are adopted in the course of the initialization phase of the PLC, are stored under the node `stInitPars`.

For all function blocks that have an adjustable overlap (`nOverlap` parameter), the calculated recommended value (in the case of `nOverlap := cCM_OverlapRecommended`) can be read in the named node after the login of the PLC. This can subsequently be used for the adjustment of the data buffer (Input/Output Shape) in the analysis chain. It is not necessary to start an application for this.

**ADS access to result data**

Each analysis function block has the sub-node `pResultData`, which is only visible in the TwinCAT Target Browser. The results (MultiArrays) sent between various function blocks in an analysis chain can thus be read via ADS. This enables the simple graphic display of the results of a function block by means of the Overview without the results first having to be converted from a multi-array into a PLC array via a sink function block.

ⓘ The function described requires TC3 Scope View from Version 1.4.3141 (available among other things with TwinCAT from Version 3.1.4024.7).

| Name | Type | Size | Subitems |
|---|---|---|---|
| ⊟ pResultData | POINTER TO ARRAY [1..<aResultDataLen>] OF ARRAY [1..0] OF LREAL | 8 | 0 |
| ⊟ pResultData^ | ARRAY [1..<aResultDataLen>] OF ARRAY [1..0] OF LREAL | 41000 | 5 |
| ⊞ pResultData^[1] | ARRAY [1..0] OF LREAL | 8200 | 1025 |
| ⊞ pResultData^[2] | ARRAY [1..0] OF LREAL | 8200 | 1025 |
| ⊞ pResultData^[3] | ARRAY [1..0] OF LREAL | 8200 | 1025 |
| ⊞ pResultData^[4] | ARRAY [1..0] OF LREAL | 8200 | 1025 |
| ⊞ pResultData^[5] | ARRAY [1..0] OF LREAL | 8200 | 1025 |
| ⊟ aResultDataLen | ARRAY [0..1] OF UDINT | 8 | 2 |
| aResultDataLen[0] | UDINT | 4 | 0 |
| aResultDataLen[1] | UDINT | 4 | 0 |
| hrResultDataSymbol | HRESULT | 4 | 0 |

| Name | Type | | |
|---|---|---|---|
| ⊟ pResultData | POINTER TO ARRAY [1..<aResultDataLen>] OF ARRAY [1..0] OF LREAL | | |
| ⊟ pResultData^ | ARRAY [1..<aResultDataLen>] OF ARRAY [1..0] OF LREAL | | |

# 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**

Algorithms and tools for the processing of signals in the range:

| Signal processing | Statistics | Classification | Buffer handling |
|---|---|---|---|
| FB_CMA_AnalyticSignal [▶ 87] | FB_CMA_HistArray [▶ 151] | FB_CMA_DiscreteClassification [▶ 118] | FB_CMA_BufferConverting [▶ 94] |
| FB_CMA_ArgSort [▶ 91] | FB_CMA_MinersRule [▶ 175] | FB_CMA_RainflowCounting [▶ 206] | FB_CMA_ComplexDataHandling [▶ 105] |
| FB_CMA_ComplexFFT [▶ 109] | FB_CMA_MomentCoefficients [▶ 179] | FB_CMA_VibrationAssessment [▶ 251] | FB_CMA_Sink [▶ 231] |
| FB_CMA_Correlation [▶ 113] | FB_CMA_Quantiles [▶ 201] | FB_CMA_WatchUpperThresholds [▶ 255] | FB_CMA_Source [▶ 238] |
| FB_CMA_CrestFactor [▶ 97] | FB_CMA_EmpiricalMean [▶ 128] | | FB_CMA_Normalization |
| FB_CMA_CrestFactorPlus [▶ 101] | FB_CMA_EmpiricalStandardDeviation [▶ 138] | | |
| FB_CMA_Downsampling [▶ 122] | FB_CMA_EmpiricalSkew [▶ 133] | | |
| FB_CMA_Envelope [▶ 143] | FB_CMA_EmpiricalExcess [▶ 123] | | |
| FB_CMA_EnvelopeSpectrum [▶ 147] | | | |
| FB_CMA_InstantaneousFrequency [▶ 156] | | | |
| FB_CMA_InstantaneousPhase [▶ 160] | | | |
| FB_CMA_IntegratedRMS [▶ 163] | | | |
| FB_CMA_MagnitudeSpectrum [▶ 168] | | | |
| FB_CMA_MultiBandRMS [▶ 184] | | | |
| FB_CMA_PowerCepstrum [▶ 193] | | | |
| FB_CMA_PowerSpectrum [▶ 197] | | | |
| FB_CMA_OrderPowerSpectrum [▶ 189] | | | |
| FB_CMA_RealFFT [▶ 211] | | | |
| FB_CMA_RMS [▶ 215] | | | |
| FB_CMA_SlidingDFT [▶ 218] | | | |
| FB_CMA_SparseSpectrum [▶ 223] | | | |

**Thematic structuring of the signal processing**

Algorithms for signal analysis in the:

# 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

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

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 Welch method. It is subsequently multiplied by a window function (window type used: `eCM_HannWindow`). 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. Subsequently, the result is transformed back into the time range with an inverse FFT. The time signal obtained in this way is added up overlapped with the help of the Overlap-Add method. During the signal synthesis, a windowing is carried out again (window type used: `eCM_HannWindow`). The result is thus a continuous output signal without jumps.

**ⓘ** **Note the window length**

The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**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 (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

**ⓘ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nChannels = 1`) | LREAL, 1, nWindowLength/2 | LCOMPLEX, 1, nWindowLength/2 |
| Multi-channel version (`nChannels > 1`) | LREAL, 2, nChannels x nWindowLength/2 | LCOMPLEX, 2, nChannels x 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 [▶ 272]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

```
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 ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_AnalyticSignal_InitPars [▶ 272]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_Envelope [▶ 143] calculates the envelope of a time series.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 5.1.2    FB_CMA_ArgSort

**Sorts the incoming arguments**

The incoming arguments are sorted optionally in ascending or descending order. In the process, any NaN values occurring in the input data are moved to the end of the output data. 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. Via a scaling factor, the frequency can be displayed directly, for example, instead of the index. The function block calculates internally with "0"-based arrays. This must be taken into account in the evaluation.

**NaN occurrence**

Possible NaN values in the input data are sorted at the end of the output data in order to obtain a sorting result that is always valid.

● **Handling of NaN values**

ℹ️ 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nChannels = 1`) | LREAL, 1, nInLength | LREAL, 2, nInLength x 2 |
| Multi-channel version (`nChannels > 1`) | LREAL, 2, nChannels x nInLength | LREAL, 3, nChannels x nInLength x 2 |

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_ArgSort_InitPars [▶ 273]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;     // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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 ini
tialized for results (0 for no initialization)
END_VAR
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_ArgSort_InitPars [▶ 273]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is

sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

### Similar function blocks

None.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

> ℹ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.3    FB_CMA_BufferConverting

**Copies data from one MultiArray to another MultiArray.**

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 one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **output stream** | eTypeCode | nDims | aDimSizes |

```
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
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type <u>ST_MA_MultiArray_InitPars [▶ 296]</u>. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section <u>Parallel processing [▶ 75]</u>.

**Output parameters**

```
VAR_OUTPUT
    bError        : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode   : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;    // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults   : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 <u>List of error codes [▶ 357]</u>.

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section <u>Error description and information [▶ 84]</u>. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Copy1D():**

Copies one-dimensional data from one MultiArray to another MultiArray.

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ  If a timeout occurs or no MultiArray buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

**Transpose():**

Transposes two dimensions so that the output buffer is the transposition of the input buffer. This operation is only possible for one and two-dimensional input buffers.

```
METHOD Transpose : HRESULT
VAR_INPUT
    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
```

- **bNewResult**: The output is TRUE every time an output MultiArray is calculated and sent to the TransferTray.

- **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 [▶ 357]. This output is identical to the return value of the method.

ⓘ  If a timeout occurs or no MultiArray buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

```
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
```

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type <u>ST_MA_MultiArray_InitPars [▶ 296]</u>. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus transferred to a target block, all further function blocks of the analysis chain have to be called cyclically as explained in the <u>API PLC Reference [▶ 82]</u>.

Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the <u>communication ring [▶ 75]</u>. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | 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 single and 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). It allows, for example, conclusions to be drawn about 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.

To avoid value range errors, each calculated value is compared to a minimum value before the logarithm is applied and, if it is less than that, replaced by that 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**

For the calculation of the RMS value, `nBufferLength` values of the time series are stored internally per channel/subchannel. 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. If the input buffer is larger than the internal buffer, the latter is filled with the latest values for the calculation.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

The output vector is filled with NaN until the internal buffer has been completely filled with new, valid values.

> **ⓘ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link: Crest factor [▶ 333].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version for several data sets (`nSubChannels = 0`) | LREAL, 2, nChannels x *not specified**  | LREAL, 1, nChannels |
| Multi-channel version for several data sets (`nSubChannels > 0`) | LREAL, 3, nChannels x nSubChannels x *not specified** | LREAL, 2, nChannels x nSubChannels |

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

```
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
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_CrestFactor_InitPars [▶ 273]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;     // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

---

ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_CrestFactor_InitPars [▶ 273]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs():**

As long as an FB_CMA_Source instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the PassInputs() method in place of the Call() method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 201] block calculates the quantiles of an empirical distribution, which enable the frequency of outliers to be assessed.

The FB_CMA_MomentCoefficients [▶ 179] 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

> **i** Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 5.1.5 FB_CMA_CrestFactorPlus

**Calculates the crest factor plus for each channel for single and multi-channel time series.**

This is defined as a linear combination of the magnitude of the peak value of a signal, the RMS value and the ratio between the two variables:

$$C^+ = c_1 \max\{|x(t)|\} + c_2 \operatorname{rms}\{x(t)\} + c_3 \frac{\max\{|x(t)|\}}{\operatorname{rms}\{x(t)\}}, \qquad C_{\mathrm{dB}}^+ = 20\log_{10}\left(C^+\right), \qquad c_1, c_2, c_3 \in \mathbb{R}$$

The default values of the constants are selected such that the crest factor plus corresponds to the crest factor without a separate configuration of the parameters.

In contrast to the crest factor, the value increases with appropriate configuration in the progressive course of damage to a roller bearing. Thus, a retrospective knowledge of the bearing condition is not required. Compare FB_CMA_CrestFactor [▶ 97].

To avoid value range errors in the calculation, each calculated value is compared to a minimum value before the logarithm is applied and, if it is less than that, replaced by that minimum value.

Since the crest factor plus depends on the ratio of the peak value to the RMS mean value, the result can be strongly influenced by individual maxima. This can lead to unexpected results.

**Memory properties**

For the calculation of the RMS value, `nBufferLength` values of the time series are stored internally per channel/subchannel. 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. If the input buffer is larger than the internal buffer, the latter is filled with the latest values for the calculation.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

The output vector is filled with NaN until the internal buffer has been completely filled with new, valid values.

> ℹ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version for several data sets (`nSubChannels = 0`) | `LREAL, 2,` nChannels x *not specified** | `LREAL, 1,` nChannels |
| Multi-channel version for several data sets (`nSubChannels > 0`) | `LREAL, 3,` nChannels x nSubChannels x *not specified** | `LREAL, 2,` nChannels x nSubChannels |

*: The length of this dimension can be selected as desired and can thus adapt itself to the application or to 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_CrestFactorPlus_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_CrestFactorPlus_InitPars [▶ 274]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

---

ℹ️ If a timeout occurs or no MultiArray 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 weights of the magnitude of the peak value, the RMS value and the ratio between the two variables can be reconfigured. The corresponding PLC array must be defined as follows.

- **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.

```
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
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`. The three parameters to be configured per channel and subchannel are `c1`; `c2`; `c3`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | `LREAL, 1,` 3 |
| Channel-specific configuration (`nSubChannels = 0`) | `LREAL, 2,` nChannels x 3 |
| Subchannel-specific configuration (`nSubChannels > 0`) | `LREAL, 2,` nSubChannels x 3 |
| Channel and subchannel-specific configuration (`nSubChannels > 0`) | `LREAL, 3,` nChannels x nSubChannels x 3 |

**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.

- **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 [▶ 357].

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_CrestFactorPlus_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
```

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_CrestFactorPlus_InitPars [▶ 274]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

### PassInputs():

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

### Similar function blocks

The function block FB_CMA_CrestFactor [▶ 97] calculates the standard variant of the crest factor.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

> **i** Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.6    FB_CMA_ComplexDataHandling

**Conversion of complex input data into real output data**

The function block FB_CMA_ComplexDataHandling extracts either the real or imaginary part from complex input data or calculates the absolute value or the phase.

## NaN occurrence

If the input vector contains one or more NaN (Not a Number) values, the associated elements in the output vector are filled with NaN. See separate section for further information on NaN values [▶ 74].

> ℹ **Handling of NaN 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.

### Behavior when processing multi-channel input data

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

### Inputs and outputs

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard variant<br>(`nChannels = 1`) | LCOMPLEX, 1,<br>nInLength | LREAL, 1,<br>nInLength |
| Multi-channel variant spectrum<br>(`nChannels > 1`) | LCOMPLEX, 2,<br>nChannels x nInLength | LREAL, 2,<br>nChannels x nInLength |

### 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_ComplexDataHandling_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_ComplexDataHandling_InitPars [▶ 274]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError          : BOOL;                           // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                        // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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 method used to handle the complex input data can be changed at runtime. The data type E_CM_ComplexDataHandling [▶ 265] can be used to select the 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.

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the Error Codes Overview [▶ 357].

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input stInitPars.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | `UDINT, 1,` 1 |
| Channel-specific configuration (`nChannels > 1`) | `UDINT, 2,` nChannels x 1 |

**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.

- **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 [▶ 357].

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_ComplexDataHandling_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)
END_VAR
```

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_ComplexDataHandling_InitPars [▶ 274]. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_BufferConverting [▶ 94] enables the transposition or extraction of low-dimensional data from multidimensional input data.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 5.1.7     FB_CMA_ComplexFFT

**Calculation of the Fast Fourier Transform (FFT) for complex-valued input signals.**

The function block FB_CMA_ComplexFFT calculates the Fourier transform of the complex-valued input signal x[n] present at the function block. A high-performance FFT algorithm is used. It is possible to calculate both the forward and inverse transform. The setting is made via the input `stInitPars` (see inputs and outputs).

Definition of the forward Fourier transform in DFT notation:

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

Definition of the inverse Fourier transform in DFT notation:

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

The highest frequency of a component in the input signal should be at the most around half of the sampling rate of the input signal so that aliasing effects are avoided.

The FFT is defined as the transform of a cyclically continued signal. This can lead to the determination of jumps as soon as the cyclically continued signal is not exactly continuous, i.e. the same at the beginning and end. The function blocks FB_CMA_PowerSpectrum [▶ 197] and FB_CMA_MagnitudeSpectrum [▶ 168] avoid these difficulties by an analysis in overlapping sections multiplied by a window function.

**Scaling**

For the quantitative evaluation of the spectrum by measurement means, the calculated spectrum is to be weighted with $1/\text{nFFT\_Length}$ for the DC component, i.e. the first array element of the output, and with $2/\text{nFFT\_Length}$ for all other elements of the output. The function block scales during the forward and inverse transformation in such a way that, in the case of successive forward and inverse transformations, the original input signal is directly calculated again at the output.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

**i** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: FFT with complex-value input signal [▶ 301].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(`nChannels = 1`) | LCOMPLEX, 1,<br>nFFT_Length | LCOMPLEX, 1,<br>nFFT_Length |
| Multi-channel version<br>(`nChannels > 1`) | LCOMPLEX, 2,<br>nChannels x nFFT_Length | LCOMPLEX, 2,<br>nChannels x nFFT_Length |

```
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
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_ComplexFFT_InitPars [▶ 275]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- **ipErrorMessage**: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_ComplexFFT_InitPars [▶ 275]. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_RealFFT [▶ 211] calculates the Fourier transform of a real-valued signal.

The function block FB_CMA_PowerSpectrum [▶ 197] calculates the power spectrum of a continuous time signal.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.8 FB_CMA_Correlation

**Calculation of the correlation function for single and multi-channel real-valued time series.**

The FB_CMA_Correlation function block calculates the correlation function of real-valued time series.

The correlation function of two signals describes their similarity at different temporal shifts in relation to each other. The time-discrete displacement is designated *m* in the following.

The input data consist of a tuple of one (or more) test signals *x*[*n*] and one (or more) reference signals *y*[*n*]. The input data are introduced into the analysis chain via the function block FB_CMA_SourcePaired [▶ 245]. Please note that when specifying a single reference signal, all channels of the test signals are correlated with the same reference signal. Conversely, when specifying several reference signals in pairs, the reference signal of channel *i* is correlated with the test signal of channel *i*. The number of channels `nChannels` must match accordingly.

There are many possible applications for the correlation function. For example, it is possible to record a signal reference of a process step and then continuously compare it with repetitions of this process step. The measure of similarity (value of the maximum of the correlation function) can then be used to monitor the process step. Likewise, the correlation function is a good tool for determining temporal shifts. We aren't looking here for the value of the maximum, but rather its position on the displacement axis *m*. This can be used, for example, to determine runtime differences of signals, for example, when locating leaks by means of several microphones. In general, autocorrelation, i.e. the correlation of a signal with itself, can be used for signal recognition in highly noisy signals. If the noise is random and the wanted signal is a deterministic signal, the result of the autocorrelation contains the noise component only at *m*=0 and the deterministic signal becomes visible.

Sampling theorem: Like the discrete Fourier transform, the sampling theorem is valid in the same way. The temporally continuous curve of an autocorrelation function can be reconstructed in discrete time series if the sampling rate is at least twice as large as the highest frequency occurring in the signal.

The calculation of the correlation function is based on one of the following configurable definitions:

Base

$$C_{xy}[m] = \sum_{n=0}^{N-1} x[n-m]\, y[n]$$

Normed

$$\hat{C}_{xy}[m] = \frac{1}{N} \sum_{n=0}^{N-1} x[n-m]\, y[n]$$

Covariance

$$cov_{xy}[m] = \left(\frac{1}{N} \sum_{n=0}^{N-1} x[n-m]\, y[n]\right) - \left(\frac{1}{N} \sum_{n=0}^{N-1} x[n-m]\right)\left(\frac{1}{N} \sum_{n=0}^{N-1} y[n]\right)$$

Covariance (Bessel correction)

$$\widetilde{cov}_{xy}[m] = \frac{N}{N-1}\, cov_{xy}[m]$$

Pearson

$$\rho_{xy}[m] = \frac{cov_{xy}[m]}{\sigma_x[m]\, \sigma_y[m]}, \qquad \sigma_x^2[m] = cov_{xx}[m]$$

The above definitions are selected via the initial parameter `eCorrelationMode` of type E_CM_CorrelationMode [▶ 265]. When quantitatively assessing the similarity of signals, "Pearson" is recommended. Conversely, the standardization is irrelevant when looking for a shift time.

The calculation is done via a selected timeframe, which is set by `eWindowMode` of the type E_CM_WindowMode [▶ 267]. There is a choice of three variants: a sliding window, a fixed window and a calculation based on all data collected since the last `ResetData()`.

**Configuration**

The initialization parameters are used to define the calculation variant and the window. They can be individually adjusted for each channel with the `Configure()` method.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> **ℹ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: Correlation function [▶ 354]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`. The rule for the processing of a single channel (`nChannels = 1`) is:

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **input stream A** | LREAL | 1 | *not specified** |
| **input stream B** | LREAL | 1 | *not specified** |
| **output stream** | LREAL | 1 | `1+ (nPositiveShift+nNe gativeShift)/ nStepsize` |

The rule for the processing of multiple channels (`nChannels > 1`) is:

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **input stream A** | LREAL | 2 | `nChannels x` *not specified** |
| **input stream B** | LREAL | 1 | *not specified** |
| **output stream** | LREAL | 2 | `nChannels x (1+ (nPositiveShift+nNe gativeShift)/ nStepsize)` |

or for the pairwise calculation of the correlation coefficients:

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **input stream A** | LREAL | 2 | `nChannels x` *not specified** |
| **input stream B** | LREAL | 2 | `nChannels x` *not specified** |

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **output stream** | LREAL | 2 | nChannels x (1+ (nPositiveShift+nNe gativeShift)/ nStepsize) |

```
VAR_INPUT
    stInitPars     : ST_CM_Correlation_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
```

**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.

- stInitPars: Function block-specific structure with initialization parameters of the type ST_CM_Correlation_InitPars [▶ 275]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;     // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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 scaling of the correlation coefficients (initial parameter eCorrelationMode) as well as the underlying calculation window (initial parameter eWindowMode) can be changed at runtime.

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.

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the Error Codes Overview [▶ 357].

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input stInitPars.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | UDINT, 1, 2 |
| Channel-specific configuration (nChannels > 1) | UDINT, 2, nChannels x 2 |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_Correlation_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 in
itialized for results (0 for no initialization)
END_VAR
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_Correlation_InitPars [▶ 275]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 5.1.9 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

Two sample implementations are available under the following links: Threshold value consideration for averaged magnitude spectra [▶ 332] and Condition Monitoring with frequency analysis [▶ 327].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (nSubChannels = 0) | LREAL, 1, nChannels | DINT (32bit), 1, nChannels |
| Multi-channel version (nSubChannels > 0) | LREAL, 2, nChannels x nSubChannels | DINT (32bit), 2, nChannels x nSubChannels |

```
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
```

**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.

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_DiscreteClassification_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | `LREAL, 1,`<br>nMaxClasses |
| Channel-specific configuration<br>(`nSubChannels = 0`) | `LREAL, 2,`<br>nChannels x nMaxClasses |
| Subchannel-specific configuration<br>(`nSubChannels > 0`) | `LREAL, 2,`<br>nSubChannels x nMaxClasses |
| Channel and subchannel-specific configuration<br>(`nSubChannels > 0`) | `LREAL, 3,`<br>nChannels x nSubChannels x nMaxClasses |

```
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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_DiscreteClassification_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_WatchUpperThresholds [▶ 255] calculates the highest category of multi-channel input data and returns it together with the number of the associated channel.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ⓘ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 5.1.10    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 [▶ 238].

**Inputs and outputs**

**Input parameters**

```
VAR_INPUT
    nDivider   : UDINT := 1;     // given input signal is sampled down by the specified divider. (1 =
 no downsampling)
    nChannels  : UDINT;          // number of channels in data buffer (=1:onedimensional dataset, >1:
twodimensional 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 metho
d call.
    hrErrorCode    : HRESULT;                    // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                      // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

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

- **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 [▶ 357].

```
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
```

- pDataIn: The assigned input buffer.
- 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 [▶ 357]. This output is identical to the return value of the method.

---

ℹ️ If a timeout occurs or no MultiArray buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

---

ℹ️ A copy action is executed if an error occurs.

---

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

---

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.11 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).

**Further comments**

4 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.

**Memory properties**

The sample quantity $N$, which is used to calculate the current result, automatically increases with each new incoming data set, 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 IEEE 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.

> **ℹ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link:

**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`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(nSubChannels = 0) | LREAL, 1,<br>nChannels | LREAL, 1,<br>nChannels |
| Standard version for several data sets<br>(nSubChannels = 0) | LREAL, 2,<br>nChannels x *not specified** | LREAL, 1,<br>nChannels |
| Multi-channel version<br>(nSubChannels > 0) | LREAL, 2,<br>nChannels x nSubChannels | LREAL, 2,<br>nChannels x nSubChannels |
| Multi-channel version for several data sets<br>(nSubChannels > 0) | LREAL, 3,<br>nChannels x nSubChannels x *not specified** | LREAL, 2,<br>nChannels x nSubChannels |

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

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- stInitPars: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;    // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ If a timeout occurs or no MultiArray 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 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ If a timeout occurs or no MultiArray buffer is available for the result, then neither the input data nor the result data are lost. They are forwarded on the next call.

```
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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

Alternatively the automatic reset in the method `CallEx()` can be used.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_EmpiricalMean [▶ 128] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalStandardDeviation [▶ 138] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalSkew [▶ 133] calculates the empirical skew of input values.

The function block FB_CMA_MomentCoefficients [▶ 179] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 201] function 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 [▶ 97] function 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.12    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

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

. 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 set, 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 IEEE 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.

> ⓘ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link:

**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`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nSubChannels = 0`) | LREAL, 1, nChannels | LREAL, 1, nChannels |
| Standard version for several data sets (`nSubChannels = 0`) | LREAL, 2, nChannels x *not specified\** | LREAL, 1, nChannels |
| Multi-channel version (`nSubChannels > 0`) | LREAL, 2, nChannels x nSubChannels | LREAL, 2, nChannels x nSubChannels |

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Multi-channel version for several data sets (`nSubChannels > 0`) | `LREAL`, 3, nChannels x nSubChannels x *not specified**   | `LREAL`, 2, nChannels x nSubChannels |

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

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;    // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

Alternatively the automatic reset in the method `CallEx()` can be used.

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_EmpiricalStandardDeviation [▶ 138] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalSkew [▶ 133] calculates the empirical skew of input values.

The function block FB_CMA_EmpiricalExcess [▶ 123] calculates the empirical excess of input values.

The function block FB_CMA_MomentCoefficients [▶ 179] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 201] function 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 [▶ 97] function 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.13 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).

**Further comments**

3 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.

**Memory properties**

The sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data set, 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 IEEE 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.

---

**ⓘ**     **●   Handling of NaN 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.

---

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link:

**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`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nSubChannels = 0`) | `LREAL, 1,`<br>nChannels | `LREAL, 1,`<br>nChannels |
| Standard version for several data sets (`nSubChannels = 0`) | `LREAL, 2,`<br>nChannels x *not specified** | `LREAL, 1,`<br>nChannels |
| Multi-channel version (`nSubChannels > 0`) | `LREAL, 2,`<br>nChannels x nSubChannels | `LREAL, 2,`<br>nChannels x nSubChannels |

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Multi-channel version for several data sets<br>(`nSubChannels > 0`) | `LREAL, 3,`<br>nChannels x nSubChannels x *not specified*\* | `LREAL, 2,`<br>nChannels x nSubChannels |

\*: The length of this dimension can be selected as desired and can thus adapt itself to the application or to the output buffer of the preceding algorithm.

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout  : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError           : BOOL;                     // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode      : HRESULT;                  // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults      : ULINT;                    // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ If a timeout occurs or no MultiArray 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 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ If a timeout occurs or no MultiArray 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

- `stInitPars`: Function block-specific structure with initialization parameters of type <u>ST_CM_EmpiricalMoments_InitPars [▶ 276]</u>. 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 MultiArray buffers. The default value is four.

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

Alternatively the automatic reset in the method `CallEx()` can be used.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus transferred to a target block, all further function blocks of the analysis chain have to be called cyclically as explained in the <u>API PLC Reference [▶ 82]</u>.
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the <u>communication ring [▶ 75]</u>. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_EmpiricalMean [▶ 128] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalStandardDeviation [▶ 138] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalExcess [▶ 123] calculates the empirical excess of input values.

The function block FB_CMA_MomentCoefficients [▶ 179] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 201] function 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 [▶ 97] function 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.14    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

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

. The Bessel's correction is always applied, in contrast to FB_CMA_MomentCoefficients [▶ 179]. 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**

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

**Memory properties**

The sample quantity $N$, which is used to calculate the current result, automatically increases with each new incoming data set, 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 IEEE 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.

> **ℹ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (nChannels > 1), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to eCM_InfRTime_AmbiguousChannelResults. If the results of all channels are impermissible, then the value on the function block corresponds to eCM_ErrRTime_ErrornousChannelResults.

A list of return values of all channels can be queried using the method GetChannelErrors().

When processing several subchannels (nSubChannels > 0), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of nChannels must be adopted; further configuration takes place in this case via the parameter nSubChannels.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of nChannels must be identical to the number of input signals; the number of subchannels nSubChannels corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link: Statistical methods [▶ 319]

**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.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(nSubChannels = 0) | LREAL, 1,<br>nChannels | LREAL, 1,<br>nChannels |
| Standard version for several data sets<br>(nSubChannels = 0) | LREAL, 2,<br>nChannels x *not specified** | LREAL, 1,<br>nChannels |
| Multi-channel version<br>(nSubChannels > 0) | LREAL, 2,<br>nChannels x nSubChannels | LREAL, 2,<br>nChannels x nSubChannels |
| Multi-channel version for several data sets<br>(nSubChannels > 0) | LREAL, 3,<br>nChannels x nSubChannels x *not specified** | LREAL, 2,<br>nChannels x nSubChannels |

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

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- stInitPars: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;    // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_EmpiricalMoments_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

Alternatively the automatic reset in the method `CallEx()` can be used.

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_EmpiricalMean [▶ 128] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalSkew [▶ 133] calculates the empirical skew of input values.

The function block FB_CMA_EmpiricalExcess [▶ 123] calculates the empirical excess of input values.

The function block FB_CMA_MomentCoefficients [▶ 179] calculates the empirical mean value, i.e. standard deviation, skew and excess, depending on the parameterization.

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 201] function 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 [▶ 97] function 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

> **i** Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.15    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 [▶ 87]. In the time-continuous display, the envelope $x_{env}(t)$ of the signal $x(t)$ is defined as:

$$x_{env}(t) = |x_{analytic}(t)| \quad , \quad x_{analytic}(t) = x(t) + i\mathcal{H}[x(t)]$$

The envelope can be interpreted as amplitude-modulated component of the signal x(t), for example

$$x(t) = x_{env}(t) \cos \varphi(t)$$

The phase-modulated component $\varphi(t)$ can also be calculated, see FB_CMA_InstantaneousPhase [▶ 160]. 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.

> **i** **Note the window length**
>
> The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**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 (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on <u>NaN values [▶ 74]</u>.

> ⓘ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels ($nChannels > 1$), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to $eCM\_InfRTime\_AmbiguousChannelResults$. If the results of all channels are impermissible, then the value on the function block corresponds to $eCM\_ErrRTime\_ErrornousChannelResults$.

A list of return values of all channels can be queried using the method $GetChannelErrors()$.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input $stInitPars$.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version ($nChannels = 1$) | LREAL, 1, nWindowLength/2 | LREAL, 1, nWindowLength/2 |
| Multi-channel version ($nChannels > 1$) | LREAL, 2, nChannels x nWindowLength/2 | LREAL, 2, nChannels x nWindowLength/2 |

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;           // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- $stInitPars$: Function-block-specific structure with initialization parameters of the type <u>ST_CM_Envelope_InitPars [▶ 276]</u>. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                        // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                     // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                       // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].
- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_Envelope_InitPars [▶ 276]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The FB_CMA_AnalyticSignal [▶ 87] block calculates the analytical signal of a time series.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 5.1.16  FB_CMA_EnvelopeSpectrum

**Calculates the envelope spectrum of a time signal.**

The envelope spectrum is a combined function block of FB_CMA_Envelope [▶ 143] and FB_CMA_PowerSpectrum [▶ 197]. 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].

The internally used window length and the length of the FFT for the calculation of the signal envelope are calculated depending on the selected window function and associated overlap (`nOverlap`), in relation to the calculation of power spectrum. The rule for the window length is `L = 2*(nWindowLength-nOverlap)`. The FFT length `N` corresponds to the next larger power of 2, to which the following applies: `N >= 4*L/3`.

ℹ️ **Note the window length**

The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**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.

This delay effect may also be enlarged when using some window functions with a wide overlap (`nOverlap`) when calculating the power spectrum.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

ℹ️ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: Envelope spectrum [▶ 336].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
| --- | --- | --- |
| Standard version<br>(`nChannels = 1`) | LREAL, 1,<br>nWindowLength-nOverlap | LREAL, 1,<br>nFFT_Length/2+1 |
| Multi-channel version<br>(`nChannels > 1`) | LREAL, 2,<br>nChannels x nWindowLength-nOverlap | LREAL, 2,<br>nChannels x nFFT_Length/2+1 |

```
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
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_EnvelopeSpectrum_InitPars [▶ 277]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                       // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                    // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
```

```
    nCntResults    : ULINT;                              // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_EnvelopeSpectrum_InitPars [▶ 277]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_Envelope [▶ 143] calculates the envelope of a time series.

The FB_CMA_PowerSpectrum [▶ 197] 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.17 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 (called histogram in the graphic display) of single-channel and also multi-channel real-valued input data. Each channel is considered independently. The frequency distribution of the cyclically incoming data buffer is calculated for every single channel, both individual values and arrays being allowed as input buffers.

The lower and upper limits as well as the number of classes (also called bins) are transferred for the parameterization. The individual class limits are then distributed in identical intervals across the so defined total range, see Histogram [▶ 28]. Values that lie below the lower limit and values that lie above the upper limit are each counted in two additional bins.

The return value is a two-dimensional array with unsigned 64-bit integer values. The first index is the number of the channel and the second index is the number of the histogram bin concerned. The counts of the elements with a value below the lower limit value or above the upper limit value are contained in the first or last bin respectively.

If a histogram counter exceeds a value of 2 to the power of 64, approx. 1.8E19, 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.

### Memory properties

The sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data set, 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`.

### 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. No NaN values are expected at the output.

### Behavior when processing multi-channel input data

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link: Histogram [▶ 317].

**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`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(`nSubChannels = 0`) | LREAL, 1,<br>nChannels | ULINT, 2,<br>nChannels x nBins+2 |
| Standard version for several data sets<br>(`nSubChannels = 0`) | LREAL, 2,<br>nChannels x *not specified** | ULINT, 2,<br>nChannels x nBins+2 |
| Multi-channel version<br>(`nSubChannels > 0`) | LREAL, 2,<br>nChannels x nSubChannels | ULINT, 3,<br>nChannels x nSubChannels x nBins+2 |
| Multi-channel version for several data sets<br>(`nSubChannels > 0`) | LREAL, 3,<br>nChannels x nSubChannels x *not specified** | ULINT, 3,<br>nChannels x nSubChannels x nBins+2 |

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

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;          // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_HistArray_InitPars [▶ 278]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                           // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                        // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
```

```
rors, warnings and more.
    nCntResults    : ULINT;                        // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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.

- **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.

```
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
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`. The two parameters to be configured per channel and subchannel are `[fMinBinned, fMaxBinned]`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | LREAL, 1, 2 |
| Channel-specific configuration (nSubChannels = 0) | LREAL, 2, nChannels x 2 |
| Subchannel-specific configuration (nSubChannels > 0) | LREAL, 2, nSubChannels x 2 |
| Channel and subchannel-specific configuration (nSubChannels > 0) | LREAL, 3, nChannels x nSubChannels x 2 |

```
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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_HistArray_InitPars [▶ 278]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

Alternatively the automatic reset in the method `CallEx()` can be used.

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The FB_CMA_Quantiles [▶ 201] function block calculates the quantiles of input value distributions.

The FB_CMA_MomentCoefficients [▶ 179] function block calculates the statistical moment coefficients: average value, standard deviation, skew and kurtosis.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.18    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 [▶ 160]. In the time-continuous representation, the instantaneous frequency $\omega(t)$ of the signal $x(t)$ is defined as:

$$\omega(t) = \frac{d}{dt}\varphi(t) = \frac{d}{dt}\arctan\frac{\mathcal{H}[x(t)]}{x(t)} \quad , \quad x_{\text{analytic}} = x(t) + i\mathcal{H}[x(t)]$$

The instantaneous frequency can be interpreted as frequency-modulated component of the signal x(t), for example

$$x(t) = x_{\text{env}}(t)\cos\varphi(t) = x_{\text{env}}(t)\cos\int\omega(t)dt$$

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 FB_CMA_Envelope [▶ 143] .

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.

ℹ️ **Note the window length**

The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**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 (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> **ⓘ**  **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(`nChannels = 1`) | `LREAL, 1,`<br>nWindowLength/2 | `LREAL, 1,`<br>nWindowLength/2 |
| Multi-channel version<br>(`nChannels > 1`) | `LREAL, 2,`<br>nChannels x nWindowLength/2 | `LREAL, 2,`<br>nChannels x nWindowLength/2 |

```
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 initialized for results (0 for no initialization)
    tTransferTimeout  : LTIME := LTIME#500US;                   // timeout checking off during access
 to inter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of type
  ST_CM_InstantaneousFrequency_InitPars [▶ 278]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                                 // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                              // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
```

```
rors, warnings and more.
    nCntResults     : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function block-specific structure with initialization parameters of type ST_CM_InstantaneousFrequency_InitPars [▶ 278]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_InstantaneousPhase [▶ 160] calculates the instantaneous phase of a time signal.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

> ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.19     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 [▶ 87]:

$$\varphi(t) = \arctan \frac{\mathcal{H}[x(t)]}{x(t)} \quad , \quad x_{\text{analytic}} = x(t) + i\mathcal{H}[x(t)]$$

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 [▶ 143].

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.

---

**ℹ** **Note the window length**

The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**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 (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

---

**ℹ** **Handling of NaN 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.

---

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

---

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (nChannels = 1) | LREAL, 1, nWindowLength/2 | LREAL, 1, nWindowLength/2 |
| Multi-channel version (nChannels > 1) | LREAL, 2, nChannels x nWindowLength/2 | LREAL, 2, nChannels x nWindowLength/2 |

```
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
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to
inter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_InstantaneousPhase_InitPars [▶ 279]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                      // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                   // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                     // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of type ST_CM_InstantaneousPhase_InitPars [▶ 279]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].

Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_InstantaneousFrequency [▶ 156] calculates the instantaneous frequency of a time signal.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ⓘ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.20   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 RMS values for the vibration acceleration, vibration velocity and the vibration amplitude, each in a defined frequency range.

The function block treats the input signal as a signal with several independent channels. For each channel the values for up to three different integration orders within the frequency range are integrated over a defined frequency interval, and the RMS values are then calculated. The function block is suitable for vibration assessment according to DIN ISO 10816 and DIN ISO 7919 or DIN ISO 20816, see Vibration assessment [▶ 34].

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 properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on <u>NaN values [▶ 74]</u>.

---

**ℹ**    **Handling of NaN 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.

---

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nMaxBands = 1`) | `LREAL, 2,` nChannels x nWindowLength-nOverlap | `LREAL, 2,` nChannels x nOrder+1 |
| Versions for several frequency bands (`nMaxBands > 1`) | `LREAL, 2,` nChannels x nWindowLength-nOverlap | `LREAL, 3,` nChannels x nOrder+1 x nMaxBands |

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout  : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_IntegratedRMS_InitPars [▶ 279]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError        : BOOL;                        // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode   : HRESULT;                     // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults   : ULINT;                       // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

---

ℹ If a timeout occurs or no MultiArray 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():**

On calling this method, the frequency bands must be configured at the beginning. 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.

- **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 [▶ 357].

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

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`. The two parameters to be configured per frequency band and channel are `[fLowerFrequencyLimit, fUpperFrequencyLimit]`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | `LREAL, 1,`<br>2 |
| Frequency band-specific configuration (`nMaxBands >= 1`) | `LREAL, 2,`<br>nMaxBands x 2 |
| Channel-specific configuration (`nMaxBands = 1`) | `LREAL, 2,`<br>nChannels x 2 |
| Channel and frequency band-specific configuration | `LREAL, 3,`<br>nChannels x nMaxBands x 2 |

```
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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_IntegratedRMS_InitPars [▶ 279]. 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 MultiArray buffers. The default value is four.

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

### PassInputs():

As long as an `FB_CMA_Source` instance is called and signal data are thus transferred to a target block, all further function blocks of the analysis chain have to be called cyclically as explained in the <u>API PLC Reference [▶ 82]</u>.

Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the <u>communication ring [▶ 75]</u>. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

### Similar function blocks

The function block <u>FB_CMA_MultiBandRMS [▶ 184]</u> calculates RMS values for single and multi-channel real-valued time series for configurable frequency bands.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section <u>Compatibility [▶ 64]</u>.

BECKHOFF

## 5.1.21    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 [▶ 18] and Frequency analysis [▶ 35]:

The input data buffer is first overlapped with the immediately preceding buffers 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 [▶ 26].

The FB_CMA_MagnitudeSpectrum function block behaves similar to FB_CMA_PowerSpectrum [▶ 197]. The difference is squaring of the results in FB_CMA_PowerSpectrum [▶ 197].

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.

**Scaling**

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 are eliminated. Tabulated parameters described in the section "Options for spectrum scaling [▶ 361]" can be used for the arithmetic scaling of absolute measurements.

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (nOverlap).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> **ⓘ  Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (nChannels > 1), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to eCM_InfRTime_AmbiguousChannelResults. If the results of all channels are impermissible, then the value on the function block corresponds to eCM_ErrRTime_ErrornousChannelResults.

A list of return values of all channels can be queried using the method GetChannelErrors().

**Sample implementation**

A sample implementation is available under the following link: Magnitude spectrum: [▶ 303].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(nChannels = 1) | LREAL, 1,<br>nWindowLength-nOverlap | LREAL, 1,<br>nFFT_Length/2+1 |
| Multi-channel version<br>(nChannels > 1) | LREAL, 2,<br>nChannels x nWindowLength-nOverlap | LREAL, 2,<br>nChannels x nFFT_Length/2+1 |

```
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
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_MagnitudeSpectrum_InitPars [▶ 281]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_MagnitudeSpectrum_InitPars [▶ 281]. 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 MultiArray buffers. The default value is four.

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes</u> [▶ 357].

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

### PassInputs():

As long as an `FB_CMA_Source` instance is called and signal data are thus transferred to a target block, all further function blocks of the analysis chain have to be called cyclically as explained in the <u>API PLC Reference</u> [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the <u>communication ring</u> [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes</u> [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

### Similar function blocks

The <u>FB_CM_PowerSpectrum</u> [▶ 197] block calculates the power spectrum by means of squaring of the values in the last step.

The <u>FB_CMA_PowerCepstrum</u> [▶ 193] function block calculates a transformation that emphasizes harmonics.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section <u>Compatibility</u> [▶ 64].

## 5.1.22 FB_CMA_MeanStressCorrection

**Calculates mean stress-free reversal points from classified reversal points.**

The function block FB_CMA_MeanStressCorrection implements the mean stress corrections according to Goodman and Gerber for the conversion of range-mean counts into mean value-free range-mean counts. The counts are processed in the form of multidimensional multi-arrays.

The fatigue damage of a component can be calculated with the function block FB_CMA_MinersRule [▶ 175]on the basis of the half loops counted by the function block FB_CMA_RainflowCounting [▶ 206]. In the case of damage monitoring, a mean stress correction of the counting result can be carried out with the help of the function block FB_CMA_MeanStressCorrection [▶ 171].

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: Fatigue Analysis [▶ 343]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) element type, number of dimensions, dimension sizes |
| --- | --- | --- |
| Standard variant (`nChannels = 1`) | UDINT, 2, (nBins+2) x (nBinsMean+2) | UDINT, 2, (nBins+2) x (nBinsMean+2) |
| Multichannel variant (`nChannels > 1`) | UDINT, 3, nChannels x (nBins+2) x (nBinsMean+2) | UDINT, 3, nChannels x (nBins+2) x (nBinsMean+2) |

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

```
VAR_INPUT
    stInitPars      : ST_CM_MeanStressCorrection_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)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access t
o inter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_MeanStressCorrection_InitPars [▶ 282]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError       : BOOL;                        // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode   : HRESULT;                     // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults   : ULINT;                       // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ If a timeout occurs or no MultiArray 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 method for mean stress correction can be configured by calling this method. The selection is made via values of the type E_CM_MeanStressCorrection [▶ 266]. Further information can be found in the section Fatigue life analysis and damage calculation [▶ 51].

- **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.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg      : POINTER TO UDINT;  // pointer to 1-dimensional array (UDINT) of arguments
    nArgSize  : UDINT;             // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | UDINT, 1,<br>1 |
| Channel-specific configuration | UDINT, 2,<br>nChannels x 1 |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MeanStressCorrection_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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_MeanStressCorrection_InitPars [▶ 282]. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_RainflowCounting [▶ 206] uses a counting matrix of load and mean values calculated on the basis of the three-point rule.

The algorithm of the function block FB_CMA_MinersRule [▶ 175] can be used to calculate the fatigue damage of components based on the counted half-cycles.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

## 5.1.23     FB_CMA_MinersRule

**Calculates the fatigue damage of a part.**

The function block FB_CMA_MinersRule calculates the fatigue damage of components on the basis of load and average values as well as the number of counted half loops.

The result is a one-dimensional array `aResult : ARRAY[0..1] OF LREAL,` which contains the calculated total damage (see value *D* in fatigue calculation Background [▶ 51]) and the underlying number of half loops, i.e. `[{fDamage},{fCycles}]`.

The fatigue damage of a component can be calculated with the function block FB_CMA_MinersRule [▶ 175] on the basis of the half loops counted by the function block FB_CMA_RainflowCounting [▶ 206]. In the case of damage monitoring, a mean stress correction of the counting result can be carried out with the help of the function block FB_CMA_MeanStressCorrection [▶ 171].

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: Fatigue Analysis [▶ 343]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard variant<br>(nChannels = 1) | UDINT, 2,<br>(nBins+2) x (nBinsMean+2) | LREAL, 1,<br>2 |

| Versions | Input buffer (MultiArray input stream)<br>element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>element type, number of dimensions, dimension sizes |
|---|---|---|
| Multichannel variant<br>(nChannels > 1) | LREAL, 2,<br>nChannels x (nBins+2) x (nBinsMean+2) | LREAL, 2,<br>nChannels x 2 |

```
VAR_INPUT
    stInitPars      : ST_CM_MinersRule_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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;           // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- stInitPars: Function block-specific structure with initialization parameters of the type ST_CM_MinersRule_InitPars [▶ 283]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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 number of half-loops until an error is identified can be configured per defined bin.

- **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.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg      : POINTER TO LREAL;  // pointer to 1-dimensional array (LREAL) of arguments
    nArgSize  : UDINT;             // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | `LREAL, 1,` nBins+2 |
| Channel-specific configuration | `LREAL, 2,` nChannels x (nBins+2) |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_RainflowCounting_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
```

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

- `stInitPars`: Function block-specific structure with initialization parameters of the type <u>ST_CM_MinersRule_InitPars [▶ 283]</u>. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_RainflowCounting [▶ 206] uses a counting matrix of load and mean values calculated on the basis of the three-point rule.

The function block FB_CMA_MeanStressCorrection [▶ 171] enables the execution of a mean stress correction with regard to the tensile strength of the monitored component.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
| --- | --- | --- |
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

## 5.1.24 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 [▶ 265]. 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 random sample scope $N$ (for all `nOrder`), the arithmetical 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 $E$)

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

$$E' = K' - 3 \qquad\qquad E = K - 3$$

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 set, 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 IEEE 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.

> **ⓘ  Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the 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`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nSubChannels = 0`) | LREAL, 1, nChannels | LREAL, 2, nChannels x nOrder+1 |
| Standard version for several data sets (`nSubChannels = 0`) | LREAL, 2, nChannels x *not specified*\* | LREAL, 2, nChannels x nOrder+1 |
| Multi-channel version (`nSubChannels > 0`) | LREAL, 2, nChannels x nSubChannels | LREAL, 3, nChannels x nSubChannels x nOrder+1 |
| Multi-channel version for several data sets (`nSubChannels > 0`) | LREAL, 3, nChannels x nSubChannels x *not specified*\* | LREAL, 3, nChannels x nSubChannels x nOrder+1 |

\*: The length of this dimension can be selected as desired and can thus adapt itself to the application or to the output buffer of the preceding algorithm.

```
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
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to
inter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_MomentCoefficients_InitPars [▶ 283]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**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
```

```
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated 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 <u>List of error codes [▶ 357]</u>.

- **ipErrorMessage**: Contains more detailed information on the current return value. Refer here to the section <u>Error description and information [▶ 84]</u>. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

> **ℹ** If a timeout occurs or no MultiArray 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 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 [▶ 357]. This output is identical to the return value of the method.

---

> **i** If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_MomentCoefficients_InitPars [▶ 283]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

Alternatively the automatic reset in the method CallEx() can be used.

**PassInputs():**

As long as an FB_CMA_Source instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is

sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

### Similar function blocks

The function block FB_CMA_EmpiricalMean [▶ 128] calculates the empirical average of input values.

The function block FB_CMA_EmpiricalStandardDeviation [▶ 138] calculates the empirical standard deviation of input values.

The function block FB_CMA_EmpiricalSkew [▶ 133] calculates the empirical skew of input values.

The function block FB_CMA_EmpiricalExcess [▶ 123] calculates the empirical excess of input values.

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_Quantiles [▶ 201] function 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 [▶ 97] function 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.25    FB_CMA_MultiBandRMS

**Calculated RMS value for single- and multi-channel real-valued time series for configurable frequency bands**

The function block FB_CMA_MultiBandRMS calculates the RMS values of the signals for single and multi-channel time series in relation to 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 [▶ 284]. The frequency bands are configured by calling the Configure() method.

The function block can be used for monitoring of bearing damage frequencies, for example.

Delimitation to FB_CMA_IntegratedRMS [▶ 163]:

The function block FB_CMA_IntegratedRMS 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. Thus, this function block can, for example, directly calculate the RMS value for vibration acceleration, vibration velocity and vibration displacement for a defined frequency band.

**Configuration**

A three-dimensional array with values is transferred as the configuration parameters to the Configure() method of the function block (optionally two-dimensional in the case of a single input channel, otherwise all channels are configured identically). The lower and upper limits of a frequency band are specified in this array. The function block then calculates the RMS values for these frequency bands of each channel on the basis of the input data.

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

| | |
|---|---|
| **i** | **Handling of NaN values**<br>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. |

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394505867/.zip.

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`. For the processing of any number of channels (`nChannels >= 1`) the rule is:

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(nMaxBands >= 1) | LREAL, 2,<br>nChannels x nWindowLength-<br>nOverlap | LREAL, 2,<br>nChannels x nMaxBands |

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_MultiBandRMS_InitPars [▶ 284]. 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                               // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                            // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;      // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                              // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

ℹ️ If a timeout occurs or no MultiArray 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():**

On calling this method, the frequency bands must be configured at the beginning. 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`. The two parameters to be configured per frequency band and channel are [`fLowerFrequencyLimit, fUpperFrequencyLimit`].

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | LREAL, 1, 2 |
| Frequency band-specific configuration (nMaxBands >= 1) | LREAL, 2, nMaxBands x 2 |
| Channel-specific configuration (nMaxBands = 1) | LREAL, 2, nChannels x 2 |
| Channel and frequency band-specific configuration | LREAL, 3, nChannels x nMaxBands x 2 |

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [</u>▶<u> 357]</u>.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type <u>ST_CM_MultiBandRMS_InitPars [</u>▶<u> 284]</u>. 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 MultiArray buffers. The default value is four.

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [</u>▶<u> 357]</u>.

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus transferred to a target block, all further function blocks of the analysis chain have to be called cyclically as explained in the <u>API PLC Reference</u> [▶<u> 82]</u>.
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the <u>communication ring [</u>▶<u> 75]</u>. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [</u>▶<u> 357]</u>.

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block <u>FB_CMA_IntegratedRMS [</u>▶<u> 163]</u> 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.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 5.1.26 FB_CMA_OrderPowerSpectrum

**Calculation of the order spectrum of real-valued input signals.**

The order analysis is used to analyze vibrations on rotating machines. A special feature is that the order analysis also provides reliably interpretable results if the rotary speed of the rotating component is not uniform during the analysis, e.g. wind turbines or ramping motors.

In frequency analysis, e.g. with the `FB_CMA_PowerSpectrum`, a time signal (e.g. a vibration signal) is transformed into the frequency range. The spectral values are then represented via the frequency. In the order analysis, conversely, the time signal before the transform is transformed and interpolated with the help of an encoder signal so that the vibration signal has spatially equidistant sampling points along a shaft revolution. The following consideration helps you to understand the basic idea. The vibration signal and the position signal are sampled equidistantly in time. If the wave velocity is constant, the vibration signal can simply be projected onto equidistant positions along a shaft revolution. If the speed changes during a revolution, the samplings along a revolution are no longer spaced equidistantly; if, for example, the shaft becomes faster, the distance between two samplings increases, since the shaft has turned further between two temporal samplings than at a lower shaft speed. By including an encoder signal, this effect can be compensated by appropriate interpolation. Subsequently, the power spectrum, cf. `FB_CMA_PowerSpectrum`, is calculated. The order spectrum is then not plotted via the frequency, but via the order.

The frequency axis is scaled with the maximum speed `fMaxRPM` (in revolutions per minute) of the observed shaft. For the order axis this results in:

The following applies to the frequency axis:

```
fSampleRate      := cOversamples * (1000.0 / fTaskCycleTime);
fResolutionFreq  := fSampleRate / cFFTLength;
fNyquistFreq     := fSampleRate / 2;
```

The following applies to the order axis:

```
fMaxOrder        := fNyquistFreq / (fMaxRPM / 60);
fResolutionOrder := fResolutionFreq / (fMaxRPM / 60);
```

The parameterization is done via the structure ST_CM_OrderPowerSpectrum_InitPars [▶ 285].

The function block `FB_CMA_OrderPowerSpectrum` expects one vibration signal and one position signal per channel. These are introduced as tuples into the analysis chain by the function block FB_CMA_SourcePaired [▶ 245].

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> **ℹ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link:

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

The following applies when processing a single channel (`nChannels = 1`):

| Multi-array in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **input stream A (Vibration)** | LREAL | 1 | `nWindowLength – nOverlap` |
| **input stream B (Position)** | LREAL | 1 | `(nWindowLength – nOverlap) / (fSampleRateSignal / fSampleRatePosition )` |
| **output stream** | LREAL | 1 | `nFFT_Length/2+1` |

The following applies when processing multiple channels (`nChannels > 1`):

| Multi-array in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **input stream A (Vibration)** | LREAL | 2 | `nChannels x (nWindowLength – nOverlap)` |
| **input stream B (Postion)** | LREAL | 1 | `(nWindowLength – nOverlap) / (fSampleRateSignal / fSampleRatePosition )` |
| **output stream** | LREAL | 2 | `nChannels x 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_OrderPowerSpectrum_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_OrderPowerSpectrum_InitPars [▶ 285]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError        : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode   : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults   : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

### Methods

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_OrderPowerSpectrum_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 in
itialized for results (0 for no initialization)
END_VAR
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_OrderPowerSpectrum_InitPars [▶ 285]. 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 MultiArray buffers. The default value is four.

### 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.

- **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 [▶ 357].

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

### PassInputs():

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

The function block FB_CMA_PowerSpectrum [▶ 197] calculates the power spectrum for real-valued input data.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 5.1.27    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:

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

Accordingly, it is defined as inverse Fourier transform of the logarithmized power spectrum (see FB_CMA_PowerSpectrum [▶ 197]). Forward and inverse transform bring the result back into the time range.

The function block is helpful for monitoring of gear units, see Gearbox monitoring [▶ 46].

In the numerical implementation the PowerSpectrum is calculated first. The input data buffer is overlapped with the immediately preceding buffers 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 transform. The result is an array with complex values.

---

**ℹ** **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.

---

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 bibliography [▶ 62].

Differentiation to the complex cepstrum:

The power cepstrum differs from the complex cepstrum, which is defined as logarithmized Fourier inverse transform 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.

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> ℹ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: Power cepstrum [▶ 338].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version<br>(`nChannels = 1`) | LREAL, 1,<br>nWindowLength-nOverlap | LCOMPLEX, 1,<br>nFFT_Length/2+1 |
| Multi-channel version<br>(`nChannels > 1`) | LREAL, 2,<br>nChannels x nWindowLength-nOverlap | LCOMPLEX, 2,<br>nChannels x nFFT_Length/2+1 |

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout  : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_PowerCepstrum_InitPars [▶ 287]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError        : BOOL;                           // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode   : HRESULT;                        // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults   : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

**ℹ** If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_PowerCepstrum_InitPars [▶ 287]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_Envelope [▶ 143] 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.28    FB_CMA_PowerSpectrum

**Calculation of the power spectrum of a real-valued input signal, and optional decibel scaling.**

The function 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 [▶ 18] and Frequency analysis [▶ 35]:

The input data buffer is first overlapped with the immediately preceding buffers 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 [▶ 26].

The FB_CMA_PowerSpectrum function block behaves similar to FB_CMA_MagnitudeSpectrum [▶ 168]. The difference is squaring of the results in FB_CMA_PowerSpectrum [▶ 197].

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.

**Scaling**

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 are eliminated. Tabulated parameters described in the section "Options for spectrum scaling [▶ 361]" can be used for the arithmetic scaling of absolute measurements.

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

ℹ️ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

### Inputs and outputs

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nChannels = 1`) | `LREAL, 1,`<br>nWindowLength-nOverlap | `LREAL, 1,`<br>nFFT_Length/2+1 |
| Multi-channel version (`nChannels > 1`) | `LREAL, 2,`<br>nChannels x nWindowLength-nOverlap | `LREAL, 2,`<br>nChannels x 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 in
itialized 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_PowerSpectrum_InitPars [▶ 288]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError        : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode   : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults   : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_PowerSpectrum_InitPars [▶ 288]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The FB_CMA_MagnitudeSpectrum [▶ 168] function block calculates the magnitude spectrum without squaring of the values in the last step.

The FB_CMA_PowerCepstrum [▶ 193] function block calculates a transformation that emphasizes harmonics.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.29    FB_CMA_Quantiles

**Calculates the quantile of the input value distribution for single- and multi-channel real-valued signals.**

The function block FB_CMA_Quantiles calculates p-quantiles of single or multi-channel real-valued input data. Each channel is considered independently.

The function block is initially based on the calculation of a frequency distribution, see FB_CMA_HistArray [▶ 151]. 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 Histogram [▶ 28]. The cumulative frequency distribution is then calculated, and from this the configured quantile, see Statistical analysis [▶ 28]. 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 sample quantity *N*, which is used to calculate the current result, automatically increases with each new incoming data set, 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`.

**Configuration**

A two-dimensional array with values is transferred to the `Configure()` method of the function block as configuration parameters. Each value represents the relative frequency for a channel and quantile to be calculated. The function 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. Alternatively, a one-dimensional array can be transferred with values for the quantile to be calculated, which are used for all channels.

**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.

---

> ● **Handling of NaN values**
>
> **i** 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link: Condition Monitoring with frequency analysis [▶ 327].

**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`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nSubChannels = 0`) | `LREAL, 1,` nChannels | `LREAL, 2,` nChannels x nMaxQuantiles |
| Standard version for several data sets (`nSubChannels = 0`) | `LREAL, 2,` nChannels x *not specified** | `LREAL, 2,` nChannels x nMaxQuantiles |
| Multi-channel version (`nSubChannels > 0`) | `LREAL, 2,` nChannels x nSubChannels | `LREAL, 3,` nChannels x nSubChannels x nMaxQuantiles |
| Multi-channel version for several data sets (`nSubChannels > 0`) | `LREAL, 3,` nChannels x nSubChannels x *not specified** | `LREAL, 3,` nChannels x nSubChannels x nMaxQuantiles |

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

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout  : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_Quantiles_InitPars [▶ 289]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;     // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ If a timeout occurs or no MultiArray 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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.

- **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.

```
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
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | LREAL, 1, nMaxQuantiles |
| Channel-specific configuration (`nSubChannels = 0`) | LREAL, 2, nChannels x nMaxQuantiles |
| Subchannel-specific configuration (`nSubChannels > 0`) | LREAL, 2, nSubChannels x nMaxQuantiles |
| Channel and subchannel-specific configuration (`nSubChannels > 0`) | LREAL, 3, nChannels x nSubChannels x nMaxQuantiles |

```
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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_Quantiles_InitPars [▶ 289]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

Alternatively the automatic reset in the method `CallEx()` can be used.

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- aChannelErrors: Error list of the type HRESULT of the length nChannels.

**Similar function blocks**

The FB_CMA_HistArray [▶ 151] function block calculates the histograms of input value distributions.

The FB_CMA_MomentCoefficients [▶ 179] function block calculates the statistical moment coefficients: average value, standard deviation, skew and kurtosis.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.30    FB_CMA_RainflowCounting

**Calculates classified reversal points in a stress-time sequence.**

The function block FB_CMA_RainflowCounting implements the three-point algorithm based on ASTM-E-1049-85, adapted to the processing of data streams. The counted half-cycles are returned in the form of a two-dimensional MultiArrays (*mean stress and stress range*).

The fatigue damage of a component can be calculated on the basis of the counted half-loops using the function block FB_CMA_MinersRule [▶ 175]. In the case of damage monitoring, a mean stress correction of the counting result can be carried out with the help of the function block FB_CMA_MeanStressCorrection [▶ 171].

**Memory properties**

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

**Behavior when processing multi-channel input data**

When processing several channels (nChannels > 1), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to eCM_InfRTime_AmbiguousChannelResults. If the results of all channels are impermissible, then the value on the function block corresponds to eCM_ErrRTime_ErrornousChannelResults.

A list of return values of all channels can be queried using the method GetChannelErrors().

**Sample implementation**

A sample implementation is available under the following link: Fatigue Analysis [▶ 343]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input stInitPars.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (`nChannels = 1`) | LREAL, 1, *not specified** | UDINT, 2, (nBins+2) x (nBinsMean+2) |
| Multi-channel version (`nChannels > 1`) | LREAL, 2, nChannels x *not specified** | UDINT, 3, nChannels x (nBins+2) x (nBinsMean+2) |

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

```
VAR_INPUT
    stInitPars       : ST_CM_RainflowCounting_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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

### 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.

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_RainflowCounting_InitPars [▶ 289]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError         : BOOL;                        // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                     // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                       // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

### Methods

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_RainflowCounting_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
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ℹ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function block-specific structure with initialization parameters of the type ST_CM_RainflowCounting_InitPars [▶ 289]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].

Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_MeanStressCorrection [▶ 171] enables the execution of a mean stress correction with regard to the tensile strength of the monitored component.

The algorithm of the function block FB_CMA_MinersRule [▶ 175] can be used to calculate the fatigue damage of components based on the counted half-cycles.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 5.1.31    FB_CMA_ReadCsvFile

**Reading of a CSV file and sequential output of the data into a PLC buffer (array).**

Data can be recorded, displayed and saved with the TwinCAT 3 Scope. One possible storage format is CSV. Thus, signal data can be stored in a CSV file and read in again later with the FB_CMA_ReadCsvFile. The signal data read in can be transferred as an alternative source of input data to an instance of FB_CMA_Source [▶ 238] and thus to the Condition Monitoring analysis chain.

**Inputs and outputs**

**Input parameters**

```
VAR_INPUT
    bExecute        : BOOL := TRUE;       // starts execution (on rising edge)
    bAbort          : BOOL;               // aborts execution (after finishing current ADS communica
tion)
    sNetId          : T_AmsNetId := '';   // TwinCAT system network address (could be kept empty for
 local host)
    tAdsTimeout     : TIME := T#1S;       // ADS timeout (only 1 second as condition for a fast data
 transmission)
    sFileName       : T_MaxString := '';  // CSV source file path and name (located on the system wi
th the specified Net Id)
    sSeparator      : STRING(2) := '$T';  // CSV field separator [e.g. tab '$T' or comma ',' or semi
colon ';' or colon ':' or blank ' ']
    nHeaderLines    : UDINT := 0;         // Number of lines in the CSV file which belongs to the he
ader. Data start after header lines.
    bColumn         : BOOL := TRUE;       // select whether the data is placed as column (e.g. TcSco
pe CSV file) or as line
END_VAR
VAR_IN_OUT
    aBuffer         : ARRAY[*] OF LREAL;  // buffer with individual length (do not switch the buffer
 during one execution)
END_VAR
```

- **bExecute**: The function block is called by a positive edge on the input `bExecute`, if the function block is not already active.

- **bAbort**: The execution is aborted if this input is set. The function block must still be called until the output `bBusy=FALSE`, so that an ADS communication that has already started internally can be terminated.

- **sNetId**: No specification of these input variables is required in order to perform the operation on the local device. Alternatively, an empty string can be specified. To direct the query to another computer, its AMS Net Id (of type T_AmsNetId) can be specified here.

- **tAdsTimeout**: Specifies a maximum time to execute the internal ADS communication.

- **sFileName**: Specifies the file path of the CSV file to be read.

- **sSeparator**: Specifies the CSV field separator used in the CSV file. Individual values are hereby separated from each other.

- **nHeaderLines**: Specifies the number of rows in which there is only header information. These are skipped when reading out. A CSV file saved with the TwinCAT 3 Scope View can contain different amounts of header information.

- **bColumn**: Indicates whether the data are located in a row one after the other (`bColumn=FALSE`) or underneath one another as a column in the CSV (`bColumn=TRUE`). The latter is the case with CSV files that have been saved with the TwinCAT 3 Scope View.

- **aBuffer**: Specifies the output data buffer into which the read values are written. This must be a PLC array of the type LREAL of any length. If the `FB_CMA_ReadCsvFile` is used as an alternative to the hardware sensor signal, the length typically corresponds to the oversampling factor. The specified buffer must be maintained over the duration of the operation. During the operation, the buffer `aBuffer` is filled several times with new data sets and this is signaled via the output variable `bBufferFull`.

**Output parameters**

```
VAR_OUTPUT
    bBufferFull     : BOOL;      // signals that aBuffer is fully filled
    nReadRecords    : UDINT;     // shows the number of records saved to aBuffer
    bBusy           : BOOL;      // TRUE if execution is active
    bError          : BOOL;      // TRUE if an error occurred
    hrErrorCode     : HRESULT;   // '< 0' = error; '> 0' = info; '0' = no error/info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;    // shows detailed information about occurred
 errors,warnings and more
END_VAR
```

- **bBufferFull**: The output is `TRUE` if the buffer `aBuffer` has been completely filled. This will often be the case over the duration of the operation until the CSV file has been read out completely. If a new data set is present in the buffer `aBuffer`, it should be processed immediately before the function block is called again. Typically, after each call to the function block, new data is available in the buffer. Nevertheless, the output `bBufferFull` should be checked to ensure a correct data stream.

- **nReadRecords**: The output indicates how many values the entire read data set contained.

- **bBusy**: The output is `TRUE` as long as the function block is active. The output becomes `FALSE` if the CSV file has been read in full or an error occurred.

- `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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

## 5.1.32    FB_CMA_RealFFT

**Calculation of the Fast Fourier Transform (FFT) for real-valued input signals.**

The function block FB_CMA_RealFFT calculates the Fourier transform of the real-valued input signal *x*[n] present at the function block. A high-performance FFT algorithm is used. It is possible to calculate the forward and inverse transform. The setting is made via the input `stInitPars` (see inputs and outputs).

Definition of the forward Fourier transform in DFT notation:

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

Definition of the inverse Fourier transform in DFT notation:

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

The highest frequency of a component in the input signal should be at the most around half of the sampling rate of the input signal so that aliasing effects are avoided.

The FFT is defined as the transform of a cyclically continued signal. This can lead to the determination of jumps as soon as the cyclically continued signal is not exactly continuous, i.e. the same at the beginning and end. The function blocks FB_CMA_PowerSpectrum [▶ 197] and FB_CMA_MagnitudeSpectrum [▶ 168] avoid these difficulties by an analysis in overlapping sections multiplied by a window function.

**Scaling**

For the quantitative evaluation of the spectrum by measurement means, the calculated spectrum is to be weighted with `1/nFFT_Length` for the DC component, i.e. the first array element of the output, and with `2/nFFT_Length` for all other elements of the output. The function block scales during the forward and inverse transformation in such a way that, in the case of successive forward and inverse transformations, the original input signal is directly calculated again at the output.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> ● **Handling of NaN values**
> **i** 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: FFT with real-value input signal [▶ 299].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (nChannels = 1) | LREAL, 1, nFFT_Length | LCOMPLEX, 1, nFFT_Length/2+1 |
| Version with full spectrum* (nChannels = 1) | LREAL, 1, nFFT_Length | LCOMPLEX, 1, nFFT_Length |
| Multi-channel version (nChannels > 1) | LREAL, 2, nChannels x nFFT_Length | LCOMPLEX, 2, nChannels x nFFT_Length/2+1 |
| Multi-channel version with full spectrum* (nChannels > 1) | LREAL, 2, nChannels x nFFT_Length | LCOMPLEX, 2, nChannels x nFFT_Length |

*: If the output buffer of the full spectrum is to be output, this can be achieved by negating the parameter `bHalfSpec (:=FALSE)`.

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;         // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block structure with initialization parameters of the type ST_CM_RealFFT_InitPars [▶ 290]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
```

```
    nCntResults    : ULINT;                                    // Counts outgoing results (MultiArrays were ca
lculated 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 <u>List of error codes [▶ 357]</u>.

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section <u>Error description and information [▶ 84]</u>. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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 init
ialized for results (0 for no initialization)
END_VAR
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
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 <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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.

- **Return value**: If an error occurs, a corresponding error code of the type HRESULT is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

- stInitPars: Function-block structure with initialization parameters of the type <u>ST_CM_RealFFT_InitPars [▶ 290]</u>. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_ComplexFFT [▶ 109] calculates the Fourier transformation of a complex-valued input signal.

The function block FB_CMA_PowerSpectrum [▶ 197] calculates the power spectrum of a real-valued input signal.

The function block FB_CMA_MagnitudeSpectrum [▶ 168] calculates the magnitude spectrum of a real-valued input signal.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 5.1.33 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-wise 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 MultiArray size set at the source function block (FB_CMA_Source [▶ 238]).

**Memory properties**

For the calculation of the RMS value, `nBufferLength` values of the time series are stored internally per channel/subchannel. 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. If the input buffer is larger than the internal buffer, the latter is filled with the latest values for the calculation.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

The output vector is filled with NaN until the internal buffer has been completely filled with new, valid values.

> **ⓘ** **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

**Sample implementation**

A sample implementation is available under the following link: Time-based RMS [▶ 313].

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version for several data sets (`nSubChannels = 0`) | `LREAL, 2,` nChannels x *not specified** | `LREAL, 1,` nChannels |
| Multi-channel version for several data sets (`nSubChannels > 0`) | `LREAL, 3,` nChannels x nSubChannels x *not specified** | `LREAL, 2,` nChannels x nSubChannels |

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

```
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 in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;         // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_RMS_InitPars [▶ 290]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
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
```

### Call():

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

---

ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_RMS_InitPars [▶ 290]. 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 MultiArray buffers. The default value is four.

### 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.

---

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes</u> [▶ 357].

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

### PassInputs():

As long as an `FB_CMA_Source` instance is called and signal data are thus transferred to a target block, all further function blocks of the analysis chain have to be called cyclically as explained in the <u>API PLC Reference</u> [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the <u>communication ring</u> [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes</u> [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

### Similar function blocks

The function block <u>FB_CMA_CrestFactor</u> [▶ 97] calculates the crest factor for each channel for single and multi-channel time series.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ Limited functional scope already available with CM 3.1. See section <u>Compatibility</u> [▶ 64].

## 5.1.34    FB_CMA_SlidingDFT

**Online calculation of spectral values for real-valued input signals**

The function block FB_CMA_SlidingDFT provides a configurable number of individual spectral values that are conform to the function block <u>FB_CMA_RealFFT</u> [▶ 211]. The DFT coefficients are calculated

$$X_n(k) = \sum_{m=0}^{N-1} x(q+m)e^{-i2\pi km/N}, \qquad q = n - N + 1$$

approximately with the attenuation parameter *r, 0<r<1,* by means of the recursion rule

$$\tilde{X}_n(k) = \sum_{m=0}^{N-1} x(q+m)r^{N-m}e^{-i2\pi km/N} = re^{i2\pi k/N}\left(\tilde{X}_{n-1}(k) - r^N x(n-N) + x(n)\right)$$

The calculation is done by using a sliding window, so that the spectral values are updated with each new sample. The advantages of the online calculation of selected spectral values compared to the FFT consist on the one hand of the constant utilization of the processor (no block processing) as well as the independence of the window length with respect to powers of two. The highest frequency of a component in the input signal should be at the most around half of the sampling rate of the input signal so that aliasing effects are avoided.

**Memory properties**

Due to the use of the last `nWindowLength` (`N`) samples for the calculation of the spectral values, jumps in the time series can lead in the frequency analysis to incorrect results in the spectral range. In a call with smaller input buffer size, fewer values can be transferred. In this case, the configured spectral values are updated for all newly transferred values.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> **ℹ Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation showing the possible uses of the function block and its configuration is available under the following link: Sliding DFT [▶ 351]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard variant (`nChannels = 1`) | `LREAL`, 1, *not specified** | `LCOMPLEX`, 1, nBins |
| Multichannel variant (`nChannels > 1`) | `LREAL`, 2, nChannels x *not specified** | `LCOMPLEX`, 2, nChannels x nBins |

```
VAR_INPUT
    stInitPars      : ST_CM_SlidingDFT_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 in
itialized for results (0 for no initialization)
```

```
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to i
nter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_SlidingDFT_InitPars [▶ 291]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;     // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

---

**i** If a timeout occurs or no MultiArray 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():**

On calling this method, the spectral values must be configured at the beginning. The DFT index is calculated for the frequency `f` to `k = f * fSampleRate / nWindowLength`. If `f` is not an integer multiple of the frequency resolution `fSampleRate / nWindowLength`, the spectral portions are distributed to two consecutive spectral values.

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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | `UDINT, 1,` <br> nBins |
| Channel-specific configuration (`nChannels > 1`) | `UDINT, 2,` <br> nChannels x nBins |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_SlidingDFT_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
```

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

- `stInitPars`: Function block-specific structure with initialization parameters of the type <u>ST_CM_SlidingDFT_InitPars [▶ 291]</u>. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_RealFFT [▶ 211] calculates the Fast Fourier Transform for real-valued input signals.

The function block ST_CM_SparseSpectrum_InitPars [▶ 291] calculates individual, configurable magnitude and spectral values.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 5.1.35 FB_CMA_SparseSpectrum

**Calculation of individual spectral values for real-valued input signals**

The function block FB_CMA_SparseSpectrum provides a configurable number of individual spectral values, which are scaled in compliance with the function blocks FB_CMA_RealFFT [▶ 211], FB_CMA_MagnitudeSpectrum [▶ 168] or FB_CMA_PowerSpectrum [▶ 197]. The DFT coefficients are calculated

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-i 2\pi n k/N}$$

with the Goertzel algorithm. This is more efficient than a radix-2 FFT if the following applies to the number *M* of the coefficients to be calculated with a window length *L*:

$$M \le \frac{5N}{6L} \log_2(N), \quad N = 2^m$$

Here, *N* is the next greater power of 2 with respect to the window length *L*. If only a few/individual spectral values are required, these can be calculated with the function block, if necessary directly in the (fast) sampling task, so that spontaneous changes in the spectrum can be reacted to more quickly.

The highest frequency of a component in the input signal should be at the most around half of the sampling rate of the input signal so that aliasing effects are avoided.

The function block performs several functions, see Analysis of data streams [▶ 18] and Frequency analysis [▶ 35].

The input data buffer is first overlapped with the immediately preceding buffers and multiplied with a window function. The DFT coefficients are then calculated using the Goertzel algorithm. The parameter eSpectrumType defines whether the absolute magnitude or its square is calculated from the resulting complex values. If the parameter bTransformToDecibel is TRUE, the values are transformed to decibel values if magnitude or power values have been calculated. These decibel values are the same for both spectral values, i.e. the influence of the squaring with the power values is taken into account in the calculation of decibel values by a factor of two for the magnitude values. Furthermore, scaling of the results is feasible via the parameter eScalingType, see Scaling of spectra [▶ 26].

**Scaling**

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 are eliminated. Tabulated parameters described in the section "Options for spectrum scaling [▶ 361]" can be used for the arithmetic scaling of absolute measurements.

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (nOverlap).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation showing the possible uses of the function block and its configuration is available under the following link: <u>Calculation of individual spectral values</u> [▶ 307]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard variant DFT (`nChannels = 1`, `eSpectrumType = eCM_DFT`) | `LREAL, 1,` nWindowLength-nOverlap | `LCOMPLEX, 1,` nBins |
| Standard variant Spectrum (`nChannels = 1`, `eSpectrumType <> eCM_DFT`) | `LREAL, 1,` nWindowLength-nOverlap | `LREAL, 1,` nBins |
| Multi-channel variant DFT (`nChannels > 1`, `eSpectrumType = eCM_DFT`) | `LREAL, 2,` nChannels x nWindowLength-nOverlap | `LCOMPLEX, 2,` nChannels x nBins |
| Multi-channel variant Spectrum (`nChannels > 1`, `eSpectrumType <> eCM_DFT`) | `LREAL, 2,` nChannels x nWindowLength-nOverlap | `LREAL, 2,` nChannels x nBins |

```
VAR_INPUT
    stInitPars      : ST_CM_SparseSpectrum_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
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of the type <u>ST_CM_SparseSpectrum_InitPars</u> [▶ 291]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section <u>Parallel processing</u> [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                       // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                    // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                      // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

ⓘ If a timeout occurs or no MultiArray 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():**

On calling this method, the spectral values must be configured at the beginning. The DFT index is calculated for the frequency `f` to `k = f * fSampleRate / nWindowLength`. If `f` is not an integer multiple of the frequency resolution `fSampleRate / nWindowLength`, the spectral portions are distributed to two consecutive spectral values.

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.

- **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 [▶ 357].

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | `UDINT, 1,`<br>nBins |
| Channel-specific configuration<br>(`nChannels > 1`) | `UDINT, 2,`<br>nChannels x nBins |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_SparseSpectrum_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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_SparseSpectrum_InitPars [▶ 291]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block <u>FB_CMA_RealFFT [▶ 211]</u> calculates the Fast Fourier Transform for real-valued input signals.

The function block <u>FB_CMA_MagnitudeSpectrum [▶ 168]</u> calculates the magnitude spectrum of a real-valued input signal.

The function block <u>FB_CMA_PowerSpectrum [▶ 197]</u> 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 5.1.36    FB_CMA_SpikeEnergySpectrum

**Analysis of peak energy of high-frequency signal components**

The function block FB_CMA_SpikeEnergySpectrum implements a widely used method for vibration analysis on machine elements. The method analyzes high-frequency signal components, in particular frequencies beyond the linear transmission range of an accelerometer, and is therefore very sensitive in the early detection of damage patterns.

The method is based on the evaluation of an accelerometer signal, wherein signal components are considered that are beyond the resonance frequency of the accelerometer. Accordingly, high-frequency pulse-shaped disturbances appear – e.g. arising when rolling over minor damage to the roller bearing - as modulation on a carrier signal. The carrier signal is determined by the resonance frequency of the mounted accelerometer. The spike energy method filters out the carrier signal and forms an envelope over the high-frequency pulse-shaped disturbances. Thus, they allow the damage frequencies of a component, e.g. a roller bearing, to be recognized more concisely in the spectrum.

Since the method strongly depends on the assembly, the location and the sensor type, the method is only suitable for trend monitoring, i.e. the transfer of threshold values to other measuring points is not possible.

The input data buffer of the function block FB_CMA_SpikeEnergySpectrum is first combined overlapping with the immediately preceding buffers and multiplied by a window function. The DFT coefficients are then calculated using the Goertzel algorithm. The parameter `eSpectrumType` defines whether the absolute magnitude or its square is calculated from the resulting complex values. If the parameter `bTransformToDecibel` is `TRUE`, the values are transformed to decibel values. These decibel values are the same for both spectral values (magnitude or its amplitude), i.e. the influence of the squaring on the

power values is taken into account by a factor of two for the magnitude values when calculating the decibel values. Furthermore, scaling of the results is feasible via the parameter `eScalingType`, see Scaling of spectra [▶ 26].

### Scaling

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 are eliminated. Tabulated parameters described in the section "Options for spectrum scaling [▶ 361]" can be used for the arithmetic scaling of absolute measurements.

### Memory properties

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

### NaN occurrence

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

> ℹ **Handling of NaN 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.

### Behavior when processing multi-channel input data

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

### Sample implementation

A sample implementation showing the possible uses of the function block and its configuration is available under the following link: Spike Energy Spectrum [▶ 346]

### Inputs and outputs

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard variant (`nChannels = 1`) | LREAL, 1, nWindowLength-nOverlap | LREAL, 1, nFFT_Length/2+1 |
| Multichannel variant (`nChannels > 1`) | LREAL, 2, nChannels x nWindowLength-nOverlap | LREAL, 2, nChannels x nFFT_Length/2+1 |

```
VAR_INPUT
    stInitPars      : ST_CM_SpikeEnergySpectrum_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
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_SpikeEnergySpectrum_InitPars [▶ 292]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                              // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                           // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;     // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                             // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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 decay time (`fDecayTime`) as well as the frequency band under consideration (via `fLowerFrequencyLimit`, `fUpperFrequencyLimit`) can be adjusted at runtime. The decay time should be selected in the optimal case so that the peak energy can completely decay, i.e. `fDecayTime > 1/f_fault`.

The corresponding PLC array must be defined as follows:
[`fDecayTime`, `fLowerFrequencyLimit`, `fUpperFrequencyLimit`]. The `Configure()` method can also be used for a new configuration with a different set of arguments.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>Error Codes Overview [▶ 357]</u>.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | `LREAL, 1,` 3 |
| Channel-specific configuration (`nChannels > 1`) | `LREAL, 2,` nChannels x 3 |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_SpikeEnergySpectrum_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)
END_VAR
```

**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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

- `stInitPars`: Function block-specific structure with initialization parameters of the type <u>ST_CM_SpikeEnergySpectrum_InitPars [▶ 292]</u>. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_PowerCepstrum [▶ 193] calculates the power cepstrum of a real-valued input signal.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

## 5.1.37    FB_CMA_Sink

**This function block writes data from a MultiArray buffer into an external PLC data buffer.**

It contains all the MultiArrays that are transferred to the specified analysis ID. Depending on the analysis chain the output results can contain NaN values.

---

*NOTICE*

**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

```
VAR_INPUT
    nOwnID          : UDINT;                    // ID for this FB instance
    tTransferTimeout : LTIME := LTIME#40US;  // timeout checking off during access to inter-
task FIFOs
END_VAR
```

### 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.

- `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`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

### Properties

The following properties provide information on the input values of the analysis chain associated with the last result output. See also the possible parameterization at FB_CMA_Source [▶ 238].

| Name | Type | Access | Description |
|---|---|---|---|
| nSourceTimestampNewest | ULINT [T_DCTIME64; 1ns steps since 1.1.2000] | Get | Associated with the last result output, this outputs the timestamp of the latest input value of the analysis chain. This is the upper limit (exclusive) of the timespan. |
| nSourceTimestampOldest | ULINT [T_DCTIME64; 1ns steps since 1.1.2000] | Get | Associated with the last result output, this outputs the timestamp of the oldest input value of the analysis chain. |

---

| |

| Name | Type | Access | Description |
|------|------|--------|-------------|
| | | | This is the lower limit (inclusive) of the timespan. |
| nSourceValues | ULINT | Get | Associated with the last result output, this shows the number of input values (signal values) of the analysis chain. |
| tSourceTimespan | LTIME | Get | Associated with the last result output, this outputs the timespan of the input values (signal values) of the analysis chain. tSourceTimespan := nSourceValues * tSamplePeriod; |

**Methods**

**Output1D():**

Writes data from a MultiArray 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.

- **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.

```
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;
    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 the MultiArray.
                                            If allocated it must point to a onedimensional array of UD
INT 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. The element type of the specified MultiArray buffer must correlate to the element type of the specified external data buffer.

- nWorkDim: The dimension of the MultiArray to be processed. These data are copied into the specified external data buffer. In general the MultiArray is also one-dimensional and nWorkDim:=0, but the MultiArray can also have additional dimensions, which may not then be copied, however.

- nElements: Specifies the number of elements to be copied from the MultiArray. nElements:=0 has to be set in order 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 MultiArray 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 MultiArray. If assigned, it must point to a one-dimensional array of UDINT that has as many elements as the MultiArray 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. 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.

**Output1DStd():**

Writes data from a MultiArray 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.

This method uses default values for the parameters `nWorkDim:=0`, `nElements:=0` and `pStartIndex:=0`.

- **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.

```
METHOD Output1DStd : HRESULT
VAR_INPUT
    pDataOut      : POINTER TO BYTE;      // address of data buffer
    nDataOutSize  : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    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. The element type of the specified MultiArray buffer must correlate to the element type of the specified external data buffer.

- 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. 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 MultiArray 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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the .

```
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;
    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_VAR
VAR_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 [▶ 267]. The element type of the specified MultiArray buffer must correlate to the element type of the specified external data buffer.

- `nWorkDim0`: Specifies the first dimension of the MultiArray to be processed. These data are copied into the first dimension of the specified external data buffer. In general the MultiArray is also two-dimensional and `nWorkDim0:=0`, but the MultiArray 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 MultiArray to be processed. These data are copied into the second dimension of the specified external data buffer. In general the MultiArray is also two-dimensional and `nWorkDim1:=1`, but the MultiArray 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 MultiArray. `nElementsDim0:=0` has to be set in order 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 MultiArray. `nElementsDim1:=0` has to be set in order 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 MultiArray 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 MultiArray. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray 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 [▶ 357]. 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.

**Output2DStd():**

Writes data from a MultiArray 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.

This method uses default values for the parameters `nWorkDim0:=0`, `nWorkDim1:=1`, `nElementsDim0:=0`, `nElementsDim1:=0` and `pStartIndex:=0`.

- **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 [▶ 357].

```
METHOD Output2DStd : HRESULT
VAR_INPUT
    pDataOut        : POINTER TO BYTE;  // address of data buffer
    nDataOutSize    : UDINT;                // size of data buffer in bytes
    eElementType    : E_MA_ElementTypeCode;
    nOptionPars     : DWORD;             // option mask
END_VAR
VAR_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 <u>E_MA_ElementTypeCode [▶ 267]</u>. The element type of the specified MultiArray buffer must correlate to the element type of the specified external data buffer.

- `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 <u>List of error codes [▶ 357]</u>. 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.

**Output3D():**

Writes data from a MultiArray into an external three-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.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
METHOD Output3D : HRESULT
VAR_INPUT
    pDataOut      : POINTER TO BYTE;  // address of data buffer
    nDataOutSize  : UDINT;            // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    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.
    nWorkDim2     : UDINT:=1;         // It designates the third dimension in the MultiArray being p
rocessed.
    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.
    nElementsDim2 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim2 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_VAR
VAR_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 <u>E_MA_ElementTypeCode [▶ 267]</u>. The element type of the specified MultiArray buffer must correlate to the element type of the specified external data buffer.

- `nWorkDim0`: Specifies the first dimension of the MultiArray to be processed. These data are copied into the first dimension of the specified external data buffer. In general the MultiArray is also two-dimensional and `nWorkDim0:=0`, but the MultiArray 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 MultiArray to be processed. These data are copied into the second dimension of the specified external data buffer. In general the MultiArray is also two-dimensional and `nWorkDim1:=1`, but the MultiArray 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`.

- `nWorkDim2`: Specifies the third dimension of the MultiArray to be processed. These data are copied into the third dimension of the specified external data buffer. In general the MultiArray is also three-dimensional and `nWorkDim2:=2`, but the MultiArray 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 `nWorkDim2:=0`.

- `nElementsDim0`: Specifies the number of elements to be copied from `nWorkDim0` of the MultiArray. `nElementsDim0:=0` has to be set in order 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 `nWorkDim1` of the MultiArray. `nElementsDim1:=0` has to be set in order 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.

- `nElementsDim2`: Specifies the number of elements to be copied from `nWorkDim2` of the MultiArray. `nElementsDim2:=0` has to be set in order 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 MultiArray 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 MultiArray. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray 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 . 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.

**Output3DStd():**

Writes data from a MultiArray into an external three-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.

This method uses default values for the parameters `nWorkDim0:=0`, `nWorkDim1:=1`, `nWorkDim2:=2`, `nElementsDim0:=0`, `nElementsDim1:=0`, `nElementsDim2:=0` and `pStartIndex:=0`.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the .

```
METHOD Output3DStd : HRESULT
VAR_INPUT
    pDataOut       : POINTER TO BYTE;  // address of data buffer
    nDataOutSize   : UDINT;            // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nOptionPars    : DWORD;            // option mask
END_VAR
VAR_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 . The element type of the specified MultiArray buffer must correlate to the element type of the specified external data buffer.

- `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 [▶ 357]. 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.

```
METHOD Init : HRESULT
VAR_INPUT
    nOwnID: UDINT; // ID for this FB instance
END_VAR
```

**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.

- **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 [▶ 357].
- `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 signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

## 5.1.38 FB_CMA_Source

**This function block writes data from an external PLC data buffer into a MultiArray buffer.**

It accumulates input data continuously, until the maximum size of the MultiArray is reached. Once the MultiArray 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 function 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`.

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **output stream** | eTypeCode | nDims | aDimSizes |

```
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#40US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

### 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.

- `stInitPars`: Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 296]. MultiArray 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError           : BOOL;                           // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode      : HRESULT;                        // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage   : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults      : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

## Properties

The following properties can optionally be used to specify the timestamp of the input values associated with the next input call. If the properties are not set, a current timestamp and a typical time difference between two input values are automatically determined internally.

At FB_CMA_Sink [▶ 231], the timespan and time stamp of the input values of the analysis chain are assigned to the result of the analysis.

| Name | Type | Access | Description |
|------|------|--------|-------------|
| nTimestamp | ULINT [T_DCTIME64; 1ns steps since 1.1.2000] | Set | Specification of the timestamp associated with the oldest value, which is transferred with the next input call. (example: StartTimeNextLatch -CycleTime) |
| tSamplePeriod | LTIME | Set | Specification of the time difference between two input values. (example: CycleTime / cOversamples). This specification is only necessary once, as it is a constant value. |

**Methods**

**Input1D():**

Writes data from an external one-dimensional data buffer into a MultiArray. This method must be called whenever new input samples are available, usually cyclically.

- **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 [▶ 357].

```
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;
    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 S
tartIndex is incremented by the number of copied elements. *)
    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
```

- pDataIn: The data buffer must contain the data from all channels.

- eElementType: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray 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 MultiArray is also one-dimensional and nWorkDim := 0, but the MultiArray can also have additional dimensions, which may not then be processed, however.

- `pStartIndex`: This is an optional parameter, which can be useful if the MultiArray has more than one dimension. Specifies the index of the first MultiArray element to be copied. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray has dimensions. After a successful copy process, the corresponding Start Index is incremented by the number of copied elements.

- `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 [▶ 357]. This output is identical to the return value of the method.

**Input1DStd():**

Writes data from an external one-dimensional data buffer into a MultiArray. This method must be called whenever new input samples are available, usually cyclically.

This method uses default values for the parameters `nWorkDim := 0` and `pStartIndex := 0`.

- **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 [▶ 357].

```
METHOD Input1DStd : 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;
    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
```

- `pDataIn`: The data buffer must contain the data from all channels.

- `eElementType`: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external 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 [▶ 357]. This output is identical to the return value of the method.

**Input2D():**

Writes data from an external two-dimensional data buffer into a MultiArray. This method must be called whenever new input samples are available, usually cyclically.

- **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 [▶ 357].

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataIn        : POINTER TO BYTE;       // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize    : UDINT;                 // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nWorkDim0      : UDINT:=0;              // It designates the first dimension in the MultiArray bei
ng processed. (e.g. 1..channels)
    nWorkDim1      : UDINT:=1;              // It designates the second dimension in the MultiArray be
ing 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 S
tartIndex is incremented by the number of copied elements. *)
    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
```

- `pDataIn`: The data buffer must contain the data from all channels.

- `eElementType`: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray 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 MultiArray is also two-dimensional and `nWorkDim0 := 0`, but the MultiArray 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 MultiArray counts the channels, then set `nWorkDim0:=1`.

- `nWorkDim1`: Defines the dimension in which the data are accumulated. In general the MultiArray is also two-dimensional and `nWorkDim1 := 1`, but the MultiArray 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 MultiArray collects the data, then set `nWorkDim1 := 0`.

- `pStartIndex`: This is an optional parameter, which can be useful if the MultiArray has more than two dimensions. Specifies the index of the first MultiArray element to be copied. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray has dimensions. After a successful copy process, the corresponding Start Index is incremented by the number of copied elements.

- `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 [▶ 357]. This output is identical to the return value of the method.

**Input2DStd():**

Writes data from an external two-dimensional data buffer into a MultiArray. This method must be called whenever new input samples are available, usually cyclically.

This method uses default values for the parameters `nWorkDim0 := 0`, `nWorkDim1 := 1` and `pStartIndex := 0`.

- **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 [▶ 357].

```
METHOD Input2DStd : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize   : UDINT;               // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    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
```

- `pDataIn`: The data buffer must contain the data from all channels.

- `eElementType`: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external 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 [▶ 357]. This output is identical to the return value of the method.

**Input3D():**

Writes data from an external three-dimensional data buffer into a MultiArray. This method must be called whenever new input samples are available, usually cyclically.

- **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 [▶ 357].

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;       // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize   : UDINT;                 // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nWorkDim0     : UDINT:=0;              // It designates the first dimension in the MultiArray bei
ng processed. (e.g. 1..channels)
    nWorkDim1     : UDINT:=1;              // It designates the second dimension in the MultiArray be
ing processed.
    nWorkDim2     : UDINT:=2;              // It designates the third dimension in the MultiArray bei
ng 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 S
tartIndex is incremented by the number of copied elements. *)
    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
```

- `pDataIn`: The data buffer must contain the data from all channels.

- `eElementType`: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray 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 MultiArray is also two-dimensional and `nWorkDim0 := 0`, but the MultiArray 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 MultiArray counts the channels, then set `nWorkDim0 := 1`.

- `nWorkDim1`: Defines the dimension that corresponds to the number of subchannels. In general, the MultiArray is also three-dimensional and `nWorkDim0 := 0` and `nWorkDim1 := 1`, but the MultiArray may 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 subchannels, while the third dimension of the MultiArray counts the subchannels, then set `nWorkDim1 := 2`.

- `nWorkDim2`: Defines the dimension in which the data are accumulated. In general the MultiArray is also two-dimensional and `nWorkDim1 := 1`, but the MultiArray 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 MultiArray collects the data, then set `nWorkDim1 := 0`.

- `pStartIndex`: This is an optional parameter, which can be useful if the MultiArray has more than two dimensions. Specifies the index of the first MultiArray element to be copied. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray has dimensions. After a successful copy process, the corresponding Start Index is incremented by the number of copied elements.

- `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 [▶ 357]. This output is identical to the return value of the method.

**Input3DStd():**

Writes data from an external three-dimensional data buffer into a MultiArray. This method must be called whenever new input samples are available, usually cyclically.

This method uses default values for the parameters `nWorkDim0 := 0, nWorkDim1 := 1, nWorkDim2 := 2` and `pStartIndex := 0`.

- **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 [▶ 357].

```
METHOD Input2DStd : HRESULT
VAR_INPUT
    pDataIn        : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize    : UDINT;                // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    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
```

- `pDataIn`: The data buffer must contain the data from all channels and subchannels.

- `eElementType`: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external 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 [▶ 357]. This output is identical to the return value of the 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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 296]. MultiArray 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 MultiArray buffers. The default value is four.

**ResetData():**

The method deletes all data sets that have already been added so that the current output buffer (MultiArray) can be filled from the start again. If external indices are to be used when filling (`pStartIndex` parameter), they must be explicitly reset.

- **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.

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

**ResetAnalysisChain()**:

Calling this method causes an automatic reset of all algorithms in the full analysis chain. Internally, a ResetData() is carried out each time before accepting the new data set. Consequently, this also implies a ResetData() on FB_CMA_Source.

If the analysis chain is only to be active for a certain period, this method offers the option to reset all algorithms before the next execution.

Errors can occur when calling an input method and cause interruptions in the time series collection. If the following algorithms in the analysis chain calculate spectra, then the ResetAnalysisChain() method can be called in the case of an error when calling an input method. Because it is not possible to calculate correct spectra on the basis of fragmented time series.

- **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.

```
METHOD ResetAnalysisChain : HRESULT
VAR_INPUT
END_VAR
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

# 5.1.39    FB_CMA_SourcePaired

**This function block writes data from external PLC data buffers to a pair of MultiArray buffers.**

It accumulates input data continuously, until the maximum size of the multi-array is reached. When these are completely filled, it is transferred to the target analysis ID.

---

**ⓘ** **Synchronization of the data**

The assignment of the data from the first stream to the second is direct, i.e. the first value from Stream A is related to the first value from Stream B. Measured data, e.g. position and vibration values must be synchronized in advance.

---

An instance of FB_CMA_Source must not be used as target for any other analysis function 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`.

| MultiArray in the | Element type | Dimensions | Dimensional variables |
|---|---|---|---|
| **output stream A** | eTypeCode | `nDims` | `aDimSizes` |
| **output stream B** | eTypeCode | `nDims` | `aDimSizes` |

```
VAR_INPUT
    stInitParsA    : ST_MA_MultiArray_InitPars;      // init parameters for first MultiArray
    stInitParsB    : ST_MA_MultiArray_InitPars;      // init parameters for second MultiArray
    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#40US;           // timeout checking off during access to int
er-task FIFOs
END_VAR
```

**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.

- stInitParsA: Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 296]. MultiArray buffers are specified for the result buffers. These parameters must correlate to the above definition of the output buffers.

- stInitParsB: Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 296]. MultiArray 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 MultiArray buffers. The default value is four.

- tTransferTimeout: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError         : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- ipErrorMessage: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Properties**

The following properties can optionally be used to specify the timestamp of the input values associated with the next input call. If the properties are not set, a current timestamp and a typical time difference between two input values are automatically determined internally.

At FB_CMA_Sink [▶ 231], the timespan and time stamp of the input values of the analysis chain are assigned to the result of the analysis.

| Name | Type | Access | Description |
|---|---|---|---|
| nTimestamp | ULINT [T_DCTIME64; 1ns steps since 1.1.2000] | Set | Specification of the timestamp associated with the oldest value, which is |

| Name | Type | Access | Description |
|------|------|--------|-------------|
| | | | transferred with the next input call. (example: StartTimeNextLatch -CycleTime) |
| tSamplePeriod | LTIME | Set | Specification of the time difference between two input values. (example: CycleTime / cOversamples). This specification is only necessary once, as it is a constant value. |

**Methods**

**Input1D():**

Writes data from external one-dimensional data buffers to a pair of MultiArrays. This method must be called whenever new input samples are available, usually cyclically.

- **Return value**: If an error occurs, a corresponding error code of the type $HRESULT$ is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
METHOD Input1D : HRESULT
VAR_INPUT
    pDataInA     : POINTER TO BYTE;       // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA : UDINT;                 // size of data buffer in bytes
    pDataInB     : POINTER TO BYTE;       // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB : UDINT;                 // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDimA    : UDINT;                  // It designates the dimension in the first multi array be
ing processed.
    nWorkDimB    : UDINT;                  // It designates the dimension in the second multi array b
eing processed.
    pStartIndexA  : 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. *)
    pStartIndexB  : POINTER TO UDINT;     // see pStartIndexA
    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
```

- `pDataInA`, `pDataInB`: The data buffer must contain the data from all channels.

- `eElementTypeA`, `eElementTypeB`: This input is of the type <u>E_MA_ElementTypeCode [▶ 267]</u>. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external data buffer.

- `nWorkDimA`, `nWorkDimB`: Defines the dimension in which the data are accumulated. In general the MultiArray is also one-dimensional and `nWorkDimA := 0` or `nWorkDimB := 0` , but the MultiArray can also have additional dimensions, which may not then be processed, however.

- `pStartIndexA`, `pStartIndexB`: This is an optional parameter, which can be useful if the MultiArray has more than one dimension. Specifies the index of the first MultiArray element to be copied. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray has dimensions. After a successful copy process, the corresponding Start Index is incremented by the number of copied elements.

- `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 defined in the List of error codes [▶ 357]. This output is identical to the return value of the method.

**Input1DStd():**

Writes data from external one-dimensional data buffers to a pair of MultiArrays. This method must be called whenever new input samples are available, usually cyclically.

This method uses default values for the parameters nWorkDimA := 0, nWorkDimB := 0 and pStartIndexA := 0, pStartIndexB := 0.

- **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 [▶ 357].

```
METHOD Input1DStd : HRESULT
VAR_INPUT
    pDataInA     : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA : UDINT;                // size of data buffer in bytes
    pDataInB     : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB : UDINT;                // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    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
```

- pDataInA, pDataInB: The data buffer must contain the data from all channels.

- eElementTypeA, eElementTypeB: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external 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 defined in the List of error codes [▶ 357]. This output is identical to the return value of the method.

**Input2D():**

Writes data from external two-dimensional data buffers to a pair of MultiArrays. This method must be called whenever new input samples are available, usually cyclically.

- **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 [▶ 357].

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataInA     : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA : UDINT;                // size of data buffer in bytes
    pDataInB     : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB : UDINT;                // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDimA0   : UDINT;                // It designates the first dimension in the first multi ar
ray being processed.
    nWorkDimA1   : UDINT;                // It designates the second dimension in the first multi a
rray being processed.
    nWorkDimB0   : UDINT;                // It designates the first dimension in the second multi a
rray being processed.
    nWorkDimB1   : UDINT;                // It designates the second dimension in the second multi
array being processed.
    pStartIndexA : 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. *)
    pStartIndexB  : POINTER TO UDINT;    // see pStartIndexA
    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
```

- `pDataInA, pDataInB`: The data buffer must contain the data from all channels.

- `eElementTypeA, eElementTypeB`: This input is of the type <u>E_MA_ElementTypeCode [▶ 267]</u>. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external data buffer.

- `nWorkDimA0, nWorkDimB0`: Defines the dimension that matches the number of channels. In general, the MultiArray is also two-dimensional and `nWorkDimA0 := 0` or `nWorkDimB0 := 0`, but the MultiArray can also have additional dimensions, which may then not be processed.
  E.g: If the first index of the specified data buffer stands for the channels, but the second dimension of the MultiArray enumerates the channels, then set `nWorkDimA0 := 1` or `nWorkDimB0 := 1`.

- `nWorkDimA1, nWorkDimB1`: Defines the dimension in which the data are accumulated. In general, the MultiArray is also two-dimensional and `nWorkDimA1 := 1` or `nWorkDimB1 := 1`, but the MultiArray can also have additional dimensions, which may then not be processed.
  E.g: If the second index of the specified data buffer stands for the collected data, but the first dimension of the MultiArray collects the data, then set `nWorkDimB1 := 0` or `nWorkDimB1 := 0`.

- `pStartIndexA, pStartIndexB`: This is an optional parameter, which can be useful if the MultiArray has more than one dimension. Specifies the index of the first MultiArray element to be copied. If assigned, it must point to a one-dimensional array of `UDINT` that has as many elements as the MultiArray has dimensions. After a successful copy process, the corresponding Start Index is incremented by the number of copied elements.

- `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 defined in the <u>List of error codes [▶ 357]</u>. This output is identical to the return value of the method.

**Input2DStd():**

Writes data from external two-dimensional data buffers to a pair of MultiArrays. This method must be called whenever new input samples are available, usually cyclically.

This method uses default values for the parameters `nWorkDimA0 := 0`, `nWorkDimA1 := 1`, `nWorkDimB0 := 0`, `nWorkDimB1 := 1` and `pStartIndexA := 0`, `pStartIndexB := 0`.

- **Return value**: If an error occurs, a corresponding error code of the type `HRESULT` is output. Possible values are described in the <u>List of error codes [▶ 357]</u>.

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataInA      : POINTER TO BYTE;       // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA  : UDINT;                 // size of data buffer in bytes
    pDataInB      : POINTER TO BYTE;       // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB  : UDINT;                 // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    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
```

- `pDataInA, pDataInB`: The data buffer must contain the data from all channels.

- `eElementTypeA, eElementTypeB`: This input is of the type E_MA_ElementTypeCode [▶ 267]. The element type of the specified MultiArray buffer (initialization parameters) must match the element type of the specified external 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 defined in the List of error codes [▶ 357]. This output is identical to the return value of the method.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitParsA    : ST_MA_MultiArray_InitPars;       // init parameters for first MultiArray
    stInitParsB    : ST_MA_MultiArray_InitPars;       // init parameters for second MultiArray
    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
```

**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.

- **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 [▶ 357].

- `stInitParsA`: Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 296]. MultiArray buffers are specified for the result buffers. These parameters must correlate to the above definition of the output buffers.

- `stInitParsB`: Function-block-specific structure of the initialization parameters of the type ST_MA_MultiArray_InitPars [▶ 296]. MultiArray 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 MultiArray buffers. The default value is four.

**ResetData():**

The method deletes all data sets that have already been added so that the current output buffer (MultiArray) can be filled from the start again. If external indices are to be used when filling (`pStartIndex` parameter), they must be explicitly reset.

- **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.

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

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

**Also see about this**

## 5.1.40    FB_CMA_VibrationAssessment

**Vibration assessment of real-valued input signals.**

The function block FB_CMA_VibrationAssessment performs a vibration assessment of real-valued input signals on the basis of ISO 10816-3. This is explained in more detail in the section Application concepts, see Vibration assessment [▶ 34]. The function block combines the calculation of integrated RMS values on configurable frequency bands with their classification for single and multi-channel input data.

The result is a one-dimensional array that contains three values for each frequency band: the highest calculated classification (in the range $-1..nMaxClasses$), the associated integration order (in the range $0..nOrder$) and the channel (in the range $1..nChannels$). It is recommended to save the result of the classification on the Sink in a two-dimensional array in the form `aResult : ARRAY[1..nMaxBands] OF ARRAY [1..3] OF DINT`. The data are then available for each band as follows: `[{class},{order},{channel}]`.

The data contained can be used to make a statement/evaluation with regard to the machine state on the basis of the vibration measurements. The possible configuration of several frequency bands facilitates the evaluation; the condition for various speeds of the machine can be determined simultaneously taking the example of the ISO 10816-3.

> ℹ️ **Note the window length**
>
> The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**Memory properties**

Due to the use of the Welch method, the current input data buffer, together with the last-transferred buffers, is used for the calculation. The number of buffers incorporated depends on the selected overlap (`nOverlap`).
The frequency analysis takes step changes in the time series into account. In order to achieve a correct result, therefore, the input data buffers used must be concatenated without gaps and without jumps.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: Vibration assessment according to ISO 10816-3 (compact) [▶ 323].

BECKHOFF

## Inputs and outputs

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (nChannels = 1) | LREAL, 1, nWindowLength-nOverlap | DINT, 1, nMaxBands*3 |
| Multi-channel version (nChannels > 1) | LREAL, 2, nChannels x nWindowLength-nOverlap | DINT, 1, nMaxBands*3 |

```
VAR_INPUT
    stInitPars      : ST_CM_VibrationAssessment_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
```

### 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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_VibrationAssessment_InitPars [▶ 294]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError          : BOOL;                                // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                             // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;       // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                               // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_VibrationAssessment_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)
END_VAR
```

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

---

> **i**  If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- stInitPars: Function-block-specific structure with initialization parameters of the type ST_CM_VibrationAssessment_InitPars [▶ 294]. 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 MultiArray buffers. The default value is four.

**Configure():**

On calling this method, the classification arguments and frequency bands must be configured at the beginning. The corresponding PLC arrays must be defined as follows. The Configure() method can also be used for a new configuration with a different set of arguments.

- **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 [▶ 357].

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg1     : POINTER TO LREAL; // pointer to 2-dimensional
array (LREAL) of arguments for classification
    nArgSize1 : UDINT;            // size of arguments buffer in bytes
    pArg2     : POINTER TO LREAL; // pointer to 2-dimensional
array (LREAL) of arguments for frequency bands
    nArgSize2 : UDINT;            // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`. The two parameters to be configured per frequency band are `[fLowerFrequencyLimit, fUpperFrequencyLimit]` (input buffer 2).

| Versions | Input buffer 1 (MultiArray input stream) Element type, number of dimensions, dimension sizes | Input buffer 2 (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Identical configuration of all channels (`nChannels >= 1`) | `LREAL, 2,` nOrder+1 x nMaxClasses | `LREAL, 2,` nMaxBands x 2 |

### 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.

- **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 [▶ 357].

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

### PassInputs():

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

### GetChannelErrors():

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_IntegratedRMS [▶ 163] calculates (optionally) integrated RMS values for single and multi-channel real-valued time series.

The function block FB_CMA_WatchUpperThresholds [▶ 255] performs the configurable threshold value monitoring of multi-channel data.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.41 FB_CMA_WatchUpperThresholds

**Configurable threshold value monitoring of multi-channel data**

Similar to the function block FB_CMA_DiscreteClassification [▶ 118], this function 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 function 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 function block can be configured at runtime by specifying the threshold value for each channel and each threshold value category.

**Memory properties**

Depending on the function block configuration with `bMemorize`, 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.

**NaN occurrence**

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

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

When processing several subchannels (`nSubChannels > 0`), particular attention must be paid to the formatting of the input and output data. If the input data consist of a multi-channel result of an upstream function block, the value of `nChannels` must be adopted; further configuration takes place in this case via the parameter `nSubChannels`.

Sample: In the statistical consideration (e.g. by FB_CMA_Quantiles) of the frequency channels of a multi-channel spectrum (e.g. FB_CMA_MagnitudeSpectrum), the value of `nChannels` must be identical to the number of input signals; the number of subchannels `nSubChannels` corresponds to the length of the spectrum.

### Inputs and outputs

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream) Element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard version (nSubChannels = 0) | LREAL, 1, nChannels | DINT (32bit), 1, 2 |
| Multi-channel version (nSubChannels > 0) | LREAL, 2, nChannels x nSubChannels | DINT (32bit), 2, nChannels x 2 |

```
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
```

### 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.

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_WatchUpperThresholds_InitPars [▶ 295]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section Parallel processing [▶ 75].

### Output parameters

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> ℹ️ If a timeout occurs or no MultiArray 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.

- **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 [▶ 357].

- `stInitPars`: Function-block-specific structure with initialization parameters of the type ST_CM_WatchUpperThresholds_InitPars [▶ 295]. 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 MultiArray buffers. The default value is four.

**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.

- **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 [▶ 357].

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

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) Element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels and subchannels | `LREAL, 1,` nMaxClasses |
| Channel-specific configuration (`nSubChannels = 0`) | `LREAL, 2,` nChannels x nMaxClasses |
| Subchannel-specific configuration (`nSubChannels > 0`) | `LREAL, 2,` nSubChannels x nMaxClasses |
| Channel and subchannel-specific configuration (`nSubChannels > 0`) | `LREAL, 3,` nChannels x nSubChannels x nMaxClasses |

**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.

- **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 [▶ 357].

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

Alternatively `bMemorize=FALSE` can be set in the initialization structure for an automatic reset.

**PassInputs()**:

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors()**:

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The FB_CMA_DiscreteClassification [▶ 118] function block classifies multi-channel input data.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

## 5.1.42    FB_CMA_ZoomFFT

**Efficient calculation of spectral values over a defined frequency band**

The function block FB_CMA_ZoomFFT calculates a section (frequency band) of the Fourier transforms of the real-valued input signal *x*[n] applied to the function block. The result of the function block is identical for the frequency band under examination to the result from FB_CMA_RealFFT, but with correspondingly lower computing effort.

The frequency band to be calculated (the "zoom") is defined by a centering frequency `fCenter`$f_c$ and a decimation factor `nDecimationFactor`$D$. The decimation factor must be a power of 2 (2, 4, 8, 16, ...). The frequency band is then defined as:

$$\left[ f_c - \frac{D}{2}\Delta f, f_c + \frac{D}{2}\Delta f \right]$$

The frequency resolution Δ*f* in the frequency band remains as one would expect with a normal FFT over the entire spectrum. The following applies:

$$\Delta f = \frac{f_s}{N} = \frac{f_s/D}{N/D} = \frac{\tilde{f}_s}{\tilde{N}} = \Delta\tilde{f}, \qquad \tilde{f}_s = \frac{f_s}{D}, \quad \tilde{N} = \frac{N}{D}$$

Sampling theorem: The highest frequency of a component in the input signal should be at the most around half of the sampling rate of the input signal so that aliasing effects are avoided.

**NaN occurrence**

If the input vector contains one or more NaN (Not a Number) values, the entire output sector is filled with NaN. See separate section for further information on NaN values [▶ 74].

ℹ **Handling of NaN 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.

**Behavior when processing multi-channel input data**

When processing several channels (`nChannels > 1`), there is a possibility of each channel having different return values. In this case, return values can be queried separately on the function block. If the results from one or more channels are impermissible, but not all channels, the value on the function block corresponds to `eCM_InfRTime_AmbiguousChannelResults`. If the results of all channels are impermissible, then the value on the function block corresponds to `eCM_ErrRTime_ErrornousChannelResults`.

A list of return values of all channels can be queried using the method `GetChannelErrors()`.

**Sample implementation**

A sample implementation is available under the following link: <u>Zoom FFT</u> [▶ 349]

**Inputs and outputs**

The input and output buffers correspond to one of the following definitions (input / output shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream)<br>element type, number of dimensions, dimension sizes | Output buffer (MultiArray output stream)<br>element type, number of dimensions, dimension sizes |
|---|---|---|
| Standard variant<br>(`nChannels = 1`) | `LREAL, 1,`<br>nFFT_Length | `LCOMPLEX, 1,`<br>nFFT_Length/<br>(2*nDecimationFactor+1) |
| Standard variant<br>(`nChannels = 1`) | `LCOMPLEX, 1,`<br>nFFT_Length | `LCOMPLEX, 1,`<br>nFFT_Length/<br>(2*nDecimationFactor+1) |
| Multichannel variant<br>(`nChannels > 1`) | `LREAL, 2,`<br>nChannels x `nFFT_Length` | `LCOMPLEX, 2,`<br>nChannels x nFFT_Length/<br>(2*nDecimationFactor+1) |
| Multichannel variant<br>(`nChannels > 1`) | `LCOMPLEX, 2,`<br>nChannels x `nFFT_Length` | `LCOMPLEX, 2,`<br>nChannels x nFFT_Length/<br>(2*nDecimationFactor+1) |

```
VAR_INPUT
    stInitPars        : ST_CM_ZoomFFT_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 in
itialized for results (0 for no initialization)
    tTransferTimeout  : LTIME := LTIME#500US;            // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

**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.

- `stInitPars`: Function block-specific structure with initialization parameters of the type <u>ST_CM_ZoomFFT_InitPars</u> [▶ 295]. 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 MultiArray buffers. The default value is four.

- `tTransferTimeout`: Setting of the synchronous timeout for internal MultiArray forwardings. See section <u>Parallel processing</u> [▶ 75].

**Output parameters**

```
VAR_OUTPUT
    bError          : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;    // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated 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 [▶ 357].

- `ipErrorMessage`: Contains more detailed information on the current return value. Refer here to the section Error description and information [▶ 84]. This special interface pointer is internally secured so that it is always valid/assigned.

**Methods**

**Call():**

The method is called each cycle in order to apply the algorithm to the current input data. 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.

- **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 [▶ 357].

```
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 [▶ 357]. This output is identical to the return value of the method.

> **i** If a timeout occurs or no MultiArray 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 parameter `fCenter` can be adjusted at runtime by calling this method. Here, the FFT index at the frequency `f` is calculated to `k = f * fSampleRate / nFFT_Length`. If `f` is not an integer multiple of the frequency resolution `fSampleRate / nFFT_Length`, the spectral components are divided into two consecutive spectral values.

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.

- **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 [▶ 357].

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

The input buffers correspond to one of the following definitions (input shape). The variable parameters are part of the function block input `stInitPars`.

| Versions | Input buffer (MultiArray input stream) element type, number of dimensions, dimension sizes |
|---|---|
| Identical configuration of all channels | `LREAL, 1,`<br>`1` |
| Channel-specific configuration<br>(`nChannels > 1`) | `LREAL, 1,`<br>`nChannels` |

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_ZoomFFT_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
```

**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.

- **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 [▶ 357].

- `stInitPars`: Function block-specific structure with initialization parameters of the type ST_CM_ZoomFFT_InitPars [▶ 295]. 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 MultiArray buffers. The default value is four.

**PassInputs():**

As long as an `FB_CMA_Source` instance is called and signal data are thus 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 [▶ 82].
Sometimes it is useful not to execute an algorithm for a certain time. For example, some algorithms should be executed only after prior training or configuration. The function block must be called cyclically, but it is sufficient for the data arriving at the function block to be forwarded in the communication ring [▶ 75]. This is done using the `PassInputs()` method in place of the `Call()` method. The algorithm itself is not called here, and accordingly no result is calculated and no output buffer generated.

- **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 [▶ 357].

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

**GetChannelErrors():**

The method enables the querying of a list of the channel-specific return values when processing several channels (`nChannels > 1`). A call is useful in the case that the return value of the function block corresponds to one of the values `eCM_InfRTime_AmbiguousChannelResults` or `eCM_ErrRTime_ErrornousChannelResults`.

- **Return value**: Information on the reading process of the list of error codes. The value is set to `TRUE` if the query was successful, otherwise to `FALSE`.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Error list of the type `HRESULT` of the length `nChannels`.

**Similar function blocks**

The function block FB_CMA_RealFFT [▶ 211] calculates the Fast Fourier Transform for real-valued input signals.

The function block ST_CM_SparseSpectrum_InitPars [▶ 291] calculates individual, configurable magnitude and spectral values.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | 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 [▶ 74].

> **ℹ 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 |

## 5.2.2 F_CM_CalculateRecommendedOverlap

This function calculates a recommended overlap based on the *amplitude flatness* and *overlap correlation* of the window function. See section: Calculation of the recommended overlap [▶ 22].

```
FUNCTION F_CM_CalculateRecommendedOverlap : UDINT
VAR_INPUT
    eWindowType        : E_CM_WindowType;        (* Window type. *)
    aWindowParameters  : T_CM_WindowParameters;  (* Window parameters for specific windows, e.g. F
latTop. *)
    nWindowLength      : UDINT;                  (* Length of analysis window. *)
END_VAR
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.2.3 F_CM_CalculateWoehlerCurve

This function calculates a Wöhler curve from material parameters as input.

```
FUNCTION F_CM_CalculateWoehlerCurve : HRESULT
VAR_INPUT
    fSRI                : LREAL;                  (* Stress Range Intercept. *)
    fUTS                : LREAL;                  (* Ultimate Tensile Strength OF the material.
 *)
    bUseUTSCorrection   : BOOL;                   (* Apply UTS correction. *)
```

```
    fK1                 : LREAL;                      (* Gradient of Woehler Curve for region N = 1
 to nNC1. *)
    fK2                 : LREAL;                      (* Gradient of Woehler Curve starting from nN
C2. *)
    nNC1                : ULINT;                      (* Transition Point for UTS correction. *)
    nNC2                : ULINT;                      (* Fatigue Transition Point between fK1 and f
K2. *)
    fRRatio             : LREAL;                      (* R-Ratio of the material test. *)
    fStepsize           : LREAL;                      (* Width of stress bins. *)
    nCurveLength        : UDINT;                      (* Length of curve. *)
    pCurve              : POINTER TO POINTER TO LREAL; (* Pointer to array data. *)
END_VAR
```
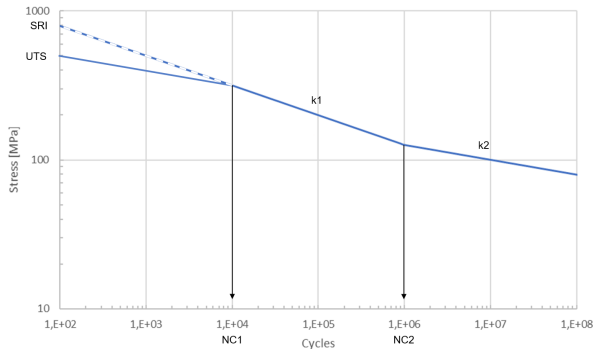


## Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

# 5.2.4 F_CM_CalculateWindow

This function calculates window functions to given input parameters. See section: Window functions [▶ 20].

```
FUNCTION F_CM_CalculateWindow : HRESULT
VAR_INPUT
    eWindowType       : E_CM_WindowType;           (* Window type. *)
    aWindowParameters : T_CM_WindowParameters;      (* Window parameters for specific windows, e.g
. FlatTop. *)
    nWindowLength     : UDINT;                      (* Length of analysis window. *)
    pWindow           : POINTER TO POINTER TO LREAL; (* Pointer to array data. *)
END_VAR
```

## Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

# 5.2.5 F_CM_ApplySpectralScaling

This function performs a scaling of spectral values. See section: Scaling of spectra [▶ 26]

```
FUNCTION F_CM_ApplySpectralScaling : HRESULT
VAR_INPUT
    eScalingType       : E_CM_ScalingType;         (* Scaling type. *)
    nFFT_Length        : UDINT;                     (* Length of FFT. *)
    nOrder             : UDINT;                     (* Order of spectrum: 1 = magnitude, 2 = powe
r. *)
    fSampleRate        : LREAL;                     (* Sample rate in Hertz. *)
    bTransformToDecibel : BOOL;                     (* Transform to decibel. *)
    fDecibelThreshold  : LREAL;                     (* Minimum argument of decadic logarithm for
64- bit IEEE 754 arithmetic. *)
    nWindowLength      : UDINT;                     (* Length of analysis window. *)
    pWindow            : POINTER TO POINTER TO LREAL; (* Pointer to window data. *)
    nResultLength      : UDINT;                     (* Length of spectrum, i.e. nFFT_Length or nF
```

```
FT_Length / 2 + 1. *)
    pSpectrum           : POINTER TO POINTER TO LREAL; (* Pointer to spectrum. *)
END_VAR
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

# 5.3 Data types

## 5.3.1 E_CM_ComplexDataHandling

```
TYPE E_CM_ComplexDataHandling : (
    eCM_CplxReal          := 0,   (* Real part of complex data. *)
    eCM_CplxImag          := 1,   (* Imaginary part of complex data. *)
    eCM_CplxAbs           := 2,   (* Absolute value of complex data. *)
    eCM_CplxPhase         := 3,   (* Phase/angle of complex data. *)
    eCM_CplxUnwrappedPhase := 4   (* Unwrapped Phase/angle of complex data. *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.2 E_CM_CorrelationMode

```
TYPE E_CM_CorrelationMode :
    eCM_Base             := 0,    (* The correlation is calculate as the convolution of the two sig
nals. *)
    eCM_Normed           := 1,    (* The correlation is calculate as the of the two signals divided
 by the number of elements added. *)
    eCM_Covariance       := 2,    (* The covariance with bias (divided by N instead N-1) is calcula
ted. *)
    eCM_CovarianceBessel := 3,    (* The covariance is calculated. *)
    eCM_Pearson          := 4     (* The correlation is calculate as the Pearson correlation coeffi
cient. *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.3 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 |

## 5.3.4      E_CM_MeanStressCorrection

```
TYPE E_CM_MeanStressCorrection : (
    eCM_NoCorrection := 0,     (* No correction of the stress matrix. *)
    eCM_Goodman      := 1,     (* Goodman correction of the stress matrix. *)
    eCM_Gerber       := 2,     (* Gerber correction of the stress matrix.  *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.5      E_CM_ScalingType

Further details can be found in the chapter Spectrum scaling [▶ 361].

```
TYPE E_CM_ScalingType :
(
    eCM_NoScaling           := 0,
    eCM_DiracScaling        := 1,
    eCM_PeakAmplitude       := 2,
    eCM_RootPowerSum        := 3,
    eCM_RMS                 := 4,
    eCM_GainCorrection      := 5,
    eCM_PowerSpectralDensity := 6,
    eCM_UnitaryScaling      := 7
) UDINT;
END_TYPE
```

**Requirements**

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

## 5.3.6      E_CM_SlidingDFTType

```
TYPE E_CM_SlidingDFTType : (
    eCM_SDFT           := 0,     (* Plain (damped) SDFT. *)
    eCM_mSDFT          := 1,     (* Modulated SDFT. *)
    eCM_SlidingGoertzel := 2,     (* Sliding Goertzel transform. *)
    eCM_DouglasAndSoh  := 3      (* Douglas and Soh algorithm. *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.7      E_CM_SpectrumType

```
TYPE E_CM_SpectrumType : (
    eCM_DFT       := 0,     (* Plain DFT. *)
    eCM_Magnitude := 1,     (* Magnitude spectrum. *)
    eCM_Power     := 2      (* Power spectrum. *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.8 E_CM_UnwrapMethod

```
TYPE E_CM_UnwrapMethod :
(
    eCM_NoUnwrapping        := 0,
    eCM_ThresholdUnwrapping := 1
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.9 E_CM_WindowMode

```
TYPE E_CM_WindowMode : (
    eCM_SlidingWindow := 0,   (* The analysis is done over the last cycles in each cycle. *)
    eCM_FixWindow     := 1,   (* The analysis is done over an interval of fix size. If one interval
 is passed another interval begins. *)
    eCM_Continuous    := 2    (* The analysis is done for the whole time since starting of the anal
ysis. *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.10 E_CM_WindowType

Further details can be found in the chapter Window functions [▶ 20].

```
TYPE E_CM_WindowType :
(
    eCM_HannWindow        := 16#05300901,
    eCM_RectangularWindow := 16#05300902,
    eCM_BartlettWindow    := 16#05300905,
    eCM_HammingWindow     := 16#05300906,
    eCM_KaiserWindow      := 16#05300907,
    eCM_FlatTopWindow     := 16#05300917
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.11 E_MA_ElementTypeCode

```
TYPE E_MA_ElementTypeCode :
(
    eMA_TypeCode_NONE     := 0,     (* Internally used. *)
    eMA_TypeCode_BYTE     := 2,
    eMA_TypeCode_CHAR     := 3,
    eMA_TypeCode_WCHAR    := 4,
    eMA_TypeCode_BOOL     := 5,     (* Boolean type. *)
    eMA_TypeCode_INT16    := 6,
    eMA_TypeCode_UINT16   := 7,
    eMA_TypeCode_INT32    := 8,     (* Used e.g. for classification results. *)
    eMA_TypeCode_UINT32   := 9,
    eMA_TypeCode_INT64    := 10,
    eMA_TypeCode_UINT64   := 11,    (* 64-bit long unsigned. Use  for statistical counters. *)
    eMA_TypeCode_REAL     := 12,    (* Unsupported: 32-bit floating point type. *)
    eMA_TypeCode_LREAL    := 13,    (* Standard floating-point type. *)
    eMA_TypeCode_LCOMPLEX := 15,    (* Standard 128-bit complex type (real part first). *)
```

```
    eMA_TypeCode_SPUNKNOWN := 22,    (* Used for TCOM Pointers. *)
) UDINT;
END_TYPE
```

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.12      T_CM_WindowParameters

```
TYPE T_CM_WindowParameters : ARRAY [0..4] OF LREAL;
END_TYPE
```

> ● **Use of `T_CM_WindowParameters`**
>
> **i**  Free parameters for the configuration of window functions are specified by a general array of the length 5. Only the first value of this array is used if `eCM_KaiserWindow` is selected; all values are used if `eCM_FlatTopWindow` is selected. The parameters are irrelevant for all other window types.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.13      Error codes

### 5.3.13.1      E_CM_ErrorCode

| Code (HRESULT) | Symbol | Description / solution |
|---|---|---|
| 0 | eCM_OK | No error, everything is OK. |
| **Logic error** | | |
| 16#9851_0100 | eCM_ErrLogic | General logic error |
| 16#9851_0102 | eCM_ErrLogic_NotImplemented | Function is not yet implemented. |
| 16#9851_0110 | eCM_ErrLogic_LackOfInitialization | Algorithm is not initialized correctly. |
| 16#9851_0125 | eCM_ErrLogic_InvalidObjectState | Operation is invalid for the object state. |
| **Configuration error** | | |
| 16#9851_1000 | eCM_ErrConfig | General configuration error |
| **Mappings to ADSERR_DEVICE_NOMEMORY** | | |
| 16#9851_1100 | eCM_ErrConfig_OutOfMemory | Memory allocation failed. => increase router memory (see chapter Memory management [▶ 71]) |
| **all errors below result in an HRESULT of ADS_E_INVALIDPARM** | | |
| 16#9851_1800 | eCM_ErrConfig_IllegalParameter | Configuration parameter is invalid. |
| 16#9851_1900 | eCM_ErrConfig_ParameterOutOfRange | Configuration parameter is out of range. |
| 16#9851_1901 | eCM_ErrConfig_ParameterOutOfRange_NoPowerOfTwo | Parameter is not a power of two as required. |
| 16#9851_1902 | eCM_ErrConfig_ParameterOutOfRange_FFT_length_Zero | The FFT length is zero. |
| 16#9851_1903 | eCM_ErrConfig_ParameterOutOfRange_DecibelThreshold_too_small | Decibel threshold is too low, which could cause underflow. |
| 16#9851_1904 | eCM_ErrConfig_ParameterOutOfRange_LogThreshold_too_small | Logarithm threshold is too low, which could cause underflow. |
| 16#9851_1905 | eCM_ErrConfig_ParameterOutOfRange_nInLength_Minimum_two | Input length is too small. Value must be at least two. |

| Code (HRESULT) | Symbol | Description / solution |
|---|---|---|
| 16#9851_190D | eCM_ErrConfig_ParameterOutOfRange_nChannels_smaller_one | Number of channels is zero. |
| 16#9851_190E | eCM_ErrConfig_ParameterOutOfRange_nBins_smaller_one | Number of bins is zero. |
| 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 is outside [0 .. 1]. |
| 16#9851_1912 | eCM_ErrConfig_ParameterOutOfRange_illegal_threshold_order | Threshold value order is invalid. The threshold values must be in ascending order. |
| 16#9851_1913 | eCM_ErrConfig_ParameterOutOfRange_threshold_number_toolarge | More threshold values specified 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 | Sampling rate is zero or negative. |
| 16#9851_191A | eCM_ErrConfig_ParameterOutOfRange_TimeConstant_too_small | Transferred time constant is too small. |
| 16#9851_192C | eCM_ErrConfig_ParameterOutOfRange_fScaleFactor_invalid | Scaling factor is invalid. |
| 16#9851_192D | eCM_ErrConfig_ParameterOutOfRange_DivThreshold_too_small | Divisional threshold is too low, which could cause underflow. |
| 16#9851_192E | eCM_ErrConfig_ParameterOutOfRange_nMaxBands_zero | Number of configurable bands is zero. |
| 16#9851_192F | eCM_ErrConfig_ParameterOutOfRange_nOrder_invalid | Order is impermissible. |
| 16#9851_1930 | eCM_ErrConfig_ParameterOutOfRange_fDecayTime_invalid | fDecayTime is impermissible. |
| 16#9851_1B00 | eCM_ErrConfig_ParameterMismatch | Parameter dependence is not met. |
| 16#9851_1B01 | eCM_ErrConfig_ParameterMismatch_WindowLength_larger_FFT_length | Window length is greater than FFT length. |
| 16#9851_1B06 | eCM_ErrConfig_ParameterMismatch_overlap_larger_BufferLength | Overlap is greater than buffer length. |
| **Runtime error (during data processing)** | | |
| **these errors result in an HRESULT of ADS_E_INVALIDPARM** | | |
| 16#9851_2000 | eCM_ErrRTime | General runtime error |
| 16#9851_2011 | eCM_ErrRTime_IllegalPointer | Impermissible (interface) pointer or memory address |
| 16#9851_2015 | eCM_ErrRTime_NonMonotonousInputData | Input data is not monotonous as required. |
| 16#9851_2016 | eCM_ErrRTime_ErrornousChannelResults | Results on at least one channel are incorrect. |
| 16#9851_2021 | eCM_ErrRTime_IllegalBuffer | Impermissible data buffer |
| **illegal input buffer parameter (can occur in ADS calls with fixed buffer)** | | |
| 16#9851_2023 | eCM_ErrRTime_IllegalInput | Impermissible input buffer parameters |
| 16#9851_2025 | eCM_ErrRTime_IllegalInputDimensionNumber | Input buffer has an impermissible number of dimensions. |

| Code (HRESULT) | Symbol | Description / solution |
|---|---|---|
| 16#9851_2026 | eCM_ErrRTime_IllegalInputShape | Input buffer has an impermissible form. |
| 16#9851_2028 | eCM_ErrRTime_IllegalInputDataType | Impermissible element type of input buffer |
| 16#9851_202A | eCM_ErrRTime_IllegalInputNoArray | No MultiArray transferred as input parameter. |
| **illegal output buffer parameter (can occur in ADS calls with fixed buffer)** | | |
| 16#9851_2030 | eCM_ErrRTime_IllegalOutput | Impermissible output buffer parameters |
| 16#9851_2032 | eCM_ErrRTime_IllegalOutputDimensionNumber | Output buffer has an impermissible number of dimensions. |
| 16#9851_2033 | eCM_ErrRTime_IllegalOutputShape | Output buffer has an impermissible form. |
| 16#9851_2034 | eCM_ErrRTime_IllegalOutputDataType | Impermissible element type of the output buffer |
| 16#9851_2035 | eCM_ErrRTime_IllegalOutputNoArray | No MultiArray transferred as the output parameter. |
| **Interpolation errors** | | |
| 16#9851_2060 | eCM_ErrRTime_Interpolation | General interpolation error |
| 16#9851_2063 | eCM_ErrRTime_Interpolation_OutOfBounds | Input data is outside of the limits. |
| 16#9851_2064 | eCM_ErrRTime_Interpolation_InvalidDimension | Array with input data has an impermissible size for method used. |
| 16#9851_2065 | eCM_ErrRTime_Interpolation_InvalidConstraints | Restrictions on the derivatives at the end points cannot be met. |
| **Info Codes** | | |
| 16#9851_B103 | eCM_InfRTime_InsufficientInputData | Insufficient input data for a representative result |
| 16#9851_B104 | eCM_InfRTime_InvalidInputData | Input data are impermissible. |
| 16#9851_B105 | eCM_InfRTime_AmbiguousChannelResults | Results on at least one channel are ambiguous. |
| 16#9851_B106 | eCM_InfRTime_nBufferLength_too_small | Configured buffer length is too small for input data. |

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.13.2    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** | | |

| code (HRESULT) | symbol | description / solution |
|---|---|---|
| 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 [▶ 71] ). Check installed TwinCAT version (see System requirements [▶ 64] ). |
| 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 |

### 5.3.13.3 E_MA_ErrorCode

| code (HRESULT) | symbol | description / solution |
|---|---|---|
| 0 | eMA_OK | No Error, everything is OK |
| **logic errors** | | |
| 16#9871_0100 | eMA_ErrLogic | General logic error |
| 16#9871_0102 | eMA_ErrLogic_NotImplemented | Function is not yet implemented. |
| 16#9871_0110 | eMA_ErrLogic_LackOfInitialization | Algorithm is not initialized correctly. |
| 16#9871_0126 | eMA_ErrLogic_ObjectCreationFailed | Object creation failed. Probably driver is missing or out-dated. |
| **configuration errors** | | |
| 16#9871_1000 | eMA_ErrConfig | General configuration error |
| **Maps to ADSERR_DEVICE_NOMEMORY** | | |

| code (HRESULT) | symbol | description / solution |
|---|---|---|
| 16#9871_1100 | eMA_ErrConfig_OutOfMemory | Memory allocation failed. => Increase router memory (see chapter Memory management [▶ 71]). Equivalent to ADSERR_DEVICE_NOMEMORY |
| **all of the following errors lead to an HRESULT of ADS_E_INVALIDPARM** | | |
| 16#9871_1800 | eMA_ErrConfig_IllegalParameter | Configuration parameter is not valid. |
| 16#9871_1900 | eMA_ErrConfig_ParameterOutOfRange | Configuration parameter is out of range. |
| **runtime errors (while data processing)** | | |
| 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_2013 | eMA_ErrRTime_InstanceExists | There is already an allocated instance. |
| 16#9871_2014 | eMA_ErrRTime_NoInstanceExists | There is no allocated or initialized instance. |
| 16#9871_2021 | eMA_ErrRTime_IllegalBuffer | Illegal data buffer |
| 16#9871_2022 | eMA_ErrRTime_IllegalSubarraySize | Illegal size of subarray |
| 16#9871_2029 | eMA_ErrRTime_IllegalPermutation | Requested permutation of dimensions is not valid. |
| **illegal input buffer parameters** | | |
| 16#9871_2023 | eMA_ErrRTime_IllegalInput | Illegal input signature |
| 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_2028 | eMA_ErrRTime_IllegalInputDataType | Illegal element type of input data stream |
| 16#9871_202A | eMA_ErrRTime_IllegalInputNoArray | No multiarray passed as input parameter. |
| **illegal output buffer parameters** | | |
| 16#9871_2030 | eMA_ErrRTime_IllegalOutput | Illegal output signature |
| 16#9871_2035 | eMA_ErrRTime_IllegalOutputNoArray | No multiarray passed as output parameter. |

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14 InitPars structures

### 5.3.14.1 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_CM_Object_InitPars :
STRUCT
    nFFT_Length    : UDINT := 512;     (* Length of FFT. *)
    nWindowLength  : UDINT := 400;     (* Length of analysis window. *)
    nChannels      : UDINT := 1;       (* Number of channels. *)
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.
- `nChannels` defines the number of independent channels. This must be greater than zero.

> ℹ **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 25 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.2    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_CM_Object_InitPars :
STRUCT
    nInLength        : UDINT := 256;      (* Length of input data array.  *)
    bSortDownward    : BOOL  := FALSE;    (* If true, sort in descending order (largest values first
). *)
    fScaleFactor     : LREAL := 1.0;      (* Scaling factor to transform index values, for example t
o frequency values. *)
    nChannels        : UDINT := 1;        (* Number of channels. *)
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.
- `fScaleFactor` can be used in order to directly display, for example, the amplitude with associated frequencies instead of the index position (fScaleFactor = 1).
- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.3    ST_CM_CrestFactor_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_CrestFactor_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels         : UDINT := 32;              (* Number of channels. *)
    nSubChannels      : UDINT := 0;               (* Number of subchannels. *)
    nBufferLength     : UDINT := 250;             (* Buffer length. *)
    fDecibelThreshold : LREAL := cCM_MinArgLog10; (* Minimum argument of decadic logarithm for 6
4-bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.
- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.
- `nBufferLength` is the number of input values per channel 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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.4    ST_CM_CrestFactorPlus_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_CrestFactorPlus_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels          : UDINT := 32;          (* Number of channels. *)
    nSubChannels       : UDINT := 0;           (* Number of subchannels. *)
    nBufferLength      : UDINT := 250;         (* Buffer length. *)
    bTransformToDecibel : BOOL := TRUE;        (* Transform to Decibel. *)
    fDecibelThreshold   : LREAL := cCM_MinArgLog10;(* Minimum argument of decadic logarithm for 64-
bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.

- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.

- `nBufferLength` is the number of input values per channel held in the internal buffer.

- `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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.5    ST_CM_ComplexDataHandling_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_ComplexDataHandling_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    eComplexDataHandling : E_CM_ComplexDataHandling := E_CM_ComplexDataHandling.eCM_CplxReal;   (*
Extraction of real or imaginary part or calculation of absolute value  or phase of complex data. *)
    nInLength          : UDINT := 256;                                                          (*
Length of input data array. *)
    fPhaseThreshold    : LREAL := 3.14159265358979;                                             (*
Threshold value for phase unwrapping. *)
    nChannels          : UDINT := 1;                                                            (*
Number of channels. *)
END_STRUCT
END_TYPE
```

- `eComplexDataHandling` defines the action that is performed using Call(), e.g. extracting real or imaginary parts, calculating the amount or phase of the complex input data.

- `nInLength` is the length of the input array.
- `fPhaseThreshold` defines the limit value for rolling out the phase. The value Pi is used as a rule.
- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.6      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_Object_InitPars :
STRUCT
    nFFT_Length  : UDINT := 256;     (* Length of FFT. *)
    bForward     : BOOL  := TRUE;    (* Flag indicating forward FFT. *)
    nChannels    : UDINT := 1;       (* Number of channels. *)
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 that specifies the direction of the FFT. If the value is `TRUE`, the normal FFT is calculated. Otherwise the inverse FFT is used.
- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.7      ST_CM_Correlation_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_Correlation_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nNegativeShift   : UDINT := 0;                                    (* Maximum negative
 time shift in samples. *)
    nPositiveShift   : UDINT := 0;                                    (* Maximum positive
 time shift in samples. *)
    nStepsize        : UDINT := 0;                                    (* Stepsize in Samp
les. *)
    eCorrelationMode : E_CM_CorrelationMode := E_CM_CorrelationMode.eCM_Normed;  (* Calculation mode
 of correlation. *)
    eWindowMode      : E_CM_WindowMode := E_CM_WindowMode.eCM_FixWindow;         (* Window handling,
 i.e. sliding, fixed, continuous. *)
    nWindowLength    : UDINT := 1000;                                 (* Length of analys
is window. *)
    nChannels        : UDINT := 1;                                    (* Number of channe
ls. *)
END_STRUCT
END_TYPE
```

- `nNegativeShift` specifies the maximum time shift in the negative direction in samples.
- `nPositiveShift` specifies the maximum time shift in the positive direction in samples.
- `nStepsize` defines the step size in samples in which the correlation coefficients are calculated.
- `eCorrelationMode` defines the calculation variant for the correlation parameters.
- `eWindowMode` defines the window on which the calculation is based. The length of the window is determined by the parameter `nWindowLength`.

- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.
- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.8 ST_CM_DiscreteClassification_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_DiscreteClassification_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels     : UDINT := 10;    (* Number of channels. *)
    nSubChannels  : UDINT := 0;     (* Number of subchannels. *)
    nMaxClasses   : UDINT := 3;     (* Number of configurable threshold classes. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.
- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.
- `nMaxClasses` is the maximum number of classes that are configured. This must be at least one.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.9 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_Object_InitPars :
STRUCT
    nChannels            : UDINT := 512;    (* Number of channels. *)
    nSubChannels         : UDINT := 0;      (* Number of subchannels. *)
    bPopulationEstimates : BOOL := TRUE;    (* Apply Bessel's correction to results. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.
- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.
- `bPopulationEstimates` is a Boolean parameter that specifies whether a Bessel correction is to be carried out.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.10 ST_CM_Envelope_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_Envelope_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length    : UDINT := 512;    (* Length of FFT. *)
```

```
    nWindowLength  : UDINT := 400;     (* Length of analysis window. *)
    nChannels      : UDINT := 1;       (* Number of channels. *)
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.

- `nChannels` defines the number of independent channels. This must be greater than zero.

> ℹ **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 25 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.11 ST_CM_EnvelopeSpectrum_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_EnvelopeSpectrum_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;                        (* Length of FFT. *)
    nWindowLength       : UDINT := 400;                        (* Length of analysis window. *)
    bTransformToDecibel : BOOL  := TRUE;                       (* Transform to decibel. *)
    fDecibelThreshold   : LREAL := cCM_MinArgLog10;            (* Minimum
argument of decadic logarithm for 64-bit IEEE 754 arithmetic. *)
    eWindowType         : E_CM_WindowType := eCM_HannWindow;   (* Window type. *)
    aWindowParameters   : T_CM_WindowParameters := [2.5,1,1,1,1]; (* Window parameters for specifi
c windows, e.g. FlatTop. *)
    nOverlap            : UDINT := -1;                         (* Size of overlap in samples. *
)
    eScalingType        : E_CM_ScalingType := eCM_DiracScaling; (* Scaling type. *)
    nChannels           : UDINT := 1;                          (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` is the length of the FFT of the spectrum. 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.

- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * \log10(x)$.

- `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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type `eCM_HannWindow`.

- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type `eCM_HannWindow`.

- `aWindowParameters` contains the free parameters of selected window functions. When using `eCM_KaiserWindow`, the first entry defines the parameter beta; if `eCM_FlatTopWindow` is used, all parameters are used. See section Window functions [▶ 20].

- `nOverlap` defines the number of overlapping samples. This must be greater than or equal to zero. If the value `cCM_OverlapRecommended` is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value `cCM_OverlapInactive` deactivates the internally used buffer and sets the value to zero.

- `eScalingType` enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is `eCM_DiracScaling`. 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.

- `nChannels` defines the number of independent channels. This must be greater than zero.

---

ℹ️ **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 25 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.12 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_Object_InitPars :
STRUCT
    nChannels    : UDINT :=  512;  (* Number of channels. *)
    nSubChannels : UDINT :=  0;    (* Number of subchannels. *)
    nBins        : UDINT :=  100;  (* Number of bins. *)
    fMinBinned   : LREAL := -120;  (* Minimum binned value. *)
    fMaxBinned   : LREAL :=  100;  (* Maximum binned value. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.

- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.

- `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.

- `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`.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.13 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_CM_Object_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;           (* Length of FFT. *)
    nWindowLength       : UDINT := 400;           (* Length of analysis window. *)
    fMagnitudeThreshold : LREAL := cCM_MinArgDiv; (* Minimum value for the numerical calculabili
```

```
ty. *)
    fSampleRate          : LREAL := 50000;            (* Sample rate in Hertz. *)
    nChannels            : UDINT := 1;                (* Number of channels. *)
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.

- **fMagnitudeThreshold** defines the limit value for the numerical calculability of the instantaneous frequency. The limit value relates to the value

$$\operatorname{conj}\{\mathcal{H}\{x[n]\}\}\mathcal{H}\{x[n+1]\} < \mathrm{cCM\_MinArgDiv}$$

- **fSampleRate** Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.

- **nChannels** defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.14    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_CM_Object_InitPars :
STRUCT
    nFFT_Length      : UDINT := 512;                             (* Length of FFT. *)
    nWindowLength    : UDINT := 400;                             (* Length of analysis window
. *)
    eUnwrapMethod    : E_CM_UnwrapMethod := eCM_ThresholdUnwrapping;   (* Unwrap method for phase v
alues. *)
    fPhaseThreshold  : LREAL := cCM_MinArgDiv;                   (* Minimum value for calcula
ting the instantaneous phase. *)
    nChannels        : UDINT := 1;                               (* Number of channels. *)
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.

- **eUnwrapMethod** defines the method used for *phase-unwrapping* with regard to the phase in multiples of 2 PI (see E_CM_UnwrapMethod [▶ 267]).

- **fPhaseThreshold** limit value for calculating the instantaneous phase. The value is related to the signal envelope. Interpretation: If the signal level is too low, the calculation of the phase is numerically too uncertain and cannot be evaluated reliably. 0 is then output as the phase.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.15    ST_CM_IntegratedRMS_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_IntegratedRMS_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                         (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                         (* Length of analysis window.
 *)
```

```
    fSampleRate         : LREAL := 20000;                    (* Sample rate in Hertz. *)
    fLowerFrequencyLimit : LREAL := 20.0;                    (* Lower limit of frequency b
and in Hertz. *)
    fUpperFrequencyLimit : LREAL := 1000.0;                  (* Upper limit of frequency b
and in Hertz. *)
    nMaxBands           : UDINT := 1;                        (* Maximum number of frequenc
y bands. *)
    nOrder              : UDINT := 2;                        (* Maximum order of integrati
on: 0 = acceleration, 1 = velocity, 2 = place. *)
    nChannels           : UDINT := 2;                        (* Number of channels. *)
    eWindowType         : E_CM_WindowType := eCM_HannWindow; (* Window type. *)
    aWindowParameters   : T_CM_WindowParameters := [2.5,1,1,1,1]; (* Window parameters for spec
ific windows, e.g. FlatTop. *)
    nOverlap            : UDINT := -1;                       (* Size of overlap in samples
. *)
    bTransformToDecibel  : BOOL := TRUE;                     (* Transform to decibel. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;         (* Minimum argument of decadi
c logarithm for 64- bit IEEE 754 arithmetic. *)
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 Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.

- 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.

- nMaxBands specifies the maximum number of frequency bands for which the RMS value is calculated.

- 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 defines the number of independent channels. This must be greater than zero.

- eWindowType defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type eCM_HannWindow.

- aWindowParameters contains the free parameters of selected window functions. When using eCM_KaiserWindow, the first entry defines the parameter beta; if eCM_FlatTopWindow is used, all parameters are used. See section Window functions [▶ 20].

- nOverlap  defines the number of overlapping samples. This must be greater than or equal to zero. If the value cCM_OverlapRecommended is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value cCM_OverlapInactive deactivates the internally used buffer and sets the value to zero.

- bTransformToDecibel is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * \log10(x)$.

- 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. (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 2.3e-308, which is equivalent to the constant cCM_MinArgLog10.)

**Note the window length**

The value of nWindowLength must be less or equal the value of nFFT_Length. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.16 ST_CM_MagnitudeSpectrum_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_MagnitudeSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length            : UDINT := 512;                    (* Length of FFT. *)
    nWindowLength          : UDINT := 400;                    (* Length of analysis window.
*)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;        (* Minimum argument of decadic
 logarithm for 64- bit IEEE 754 arithmetic. *)
    bTransformToDecibel  : BOOL := TRUE;                    (* Transform to decibel. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;   (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];   (* Window parameters for speci
fic windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                    (* Size of overlap in samples.
 *)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;   (* Scaling type. *)
    nChannels            : UDINT := 1;                    (* Number of channels. *)
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.

- `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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * \log 10(x)$.

- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type `eCM_HannWindow`.

- `aWindowParameters` contains the free parameters of selected window functions. When using `eCM_KaiserWindow`, the first entry defines the parameter beta; if `eCM_FlatTopWindow` is used, all parameters are used. See section Window functions [▶ 20].

- `nOverlap`  defines the number of overlapping samples. This must be greater than or equal to zero. If the value `cCM_OverlapRecommended` is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value `cCM_OverlapInactive` deactivates the internally used buffer and sets the value to zero.

- `eScalingType` enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is `eCM_DiracScaling`. 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.

- `nChannels` defines the number of independent channels. This must be greater than zero.

<table>
<tr><td></td></tr>
</table>

**Note the window length**

The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.17 ST_CM_MeanStressCorrection_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_MeanStressCorrection_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    eMeanStressCorrection : E_CM_MeanStressCorrection
                        := E_CM_MeanStressCorrection.eCM_Goodman;  (* Mean stress correction typ
e, e.g. Goodman or Gerber correction. *)
    nBins               : UDINT := 1;                             (* Number of bins for the str
ess levels. *)
    fBinWidth           : LREAL := 1.0;                           (* Width of the stress bins.
*)
    nBinsMean           : UDINT := 1;                             (* Number of bins for the mea
n levels. *)
    fBinWidthMean       : LREAL := 1.0;                           (* Width of the mean stress b
ins. *)
    fUTS                : LREAL := 700.0;                         (* Ultimate tensile strength
of material. *)
    nChannels           : UDINT := 1;                             (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `eMeanStressCorrection` defines the correction mode to be used. The correction according to Goodman and Gerber is available as well as the option not to make a correction.

- `nBins` defines the number of bins for the stress level. The total number of bins is `nBins + 2`; the value must be at least one. The first bin of each row contains the stress levels that are less than or equal to the defined minimum; analogously, all values greater than the defined maximum fall into the last bin of each row.

- `fBinWidth` defines the width of the bins for the stress level. The parameter must be chosen according to the dimension of the incoming counting matrix, i.e. in the case of correction of an output from the function block FB_CMA_RainflowCounting, the value `(fMaxStress – fMinStress)/nBins` must be selected here.

- `nBinsMean` defines the number of bins for the mean values. The value must be at least one. Two separate bins are added in the first/last column of the *Halfcycle Count Matrix*, where mean values are kept that are less than or equal to or, respectively, greater than the defined limits.

- `fBinWidthMean` defines the width of the bins for the mean values of the stress cycles. The parameter must be chosen according to the dimension of the incoming counting matrix, i.e. in the case of correction of an output from the function block FB_CMA_RainflowCounting, the value `(fMaxMean – fMinMean)/nBinsMean` must be selected here.

- `fUTS` defines the tensile strength of the monitored material

- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.18    ST_CM_MinersRule_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_MinersRule_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nBins       : UDINT := 64;    (* Number of bins for the stress levels. *)
    nBinsMean   : UDINT := 64;    (* Number of bins for the mean levels. *)
    fDamage     : LREAL := 0.0;   (* Additional constant damage *)
    nCycles     : ULINT := 0;     (* Initial number of cycles counted *)
    nChannels   : UDINT := 1;     (* Number of channels. *)
END_STRUCT
END_TYPE
```

- nBins defines the number of bins for the stress level. The total number of bins is nBins + 2; the value must be at least one. The first bin of each row contains the stress levels that are less than or equal to the defined minimum; analogously, all values greater than the defined maximum fall into the last bin of each row.

- nBinsMean defines the number of bins for the mean values. The value must be at least one. Two separate bins are added in the first/last column of the *Halfcycle Count Matrix*, where mean values are kept that are less than or equal to or, respectively, greater than the defined limits.

- fDamage defines the constant damage from the beginning. The total damage is thus calculated from the sum of fDamage and the accumulated damage with respect to the configured Wöhler curve.

- nCycles defines the initial value of the counted half-cycles at the beginning. The total number of cycles is calculated from the sum of nCycles and the current number of input data.

- nChannels defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.19    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_Object_InitPars :
STRUCT
    nChannels            : UDINT := 512;                              (* Number of channel
s. *)
    nSubChannels         : UDINT := 0;                                (* Number of subchan
nels. *)
    nOrder               : E_CM_MCoefOrder := E_CM_MCoefOrder.eCM_Kurtosis;   (* Maximum order of
the moment coefficients that are calculated. *)
    bPopulationEstimates : BOOL := FALSE;                            (* Apply Bessel's co
rrection to results. *)
    bKurtosisExcess      : BOOL := TRUE;                             (* Flag if kurtosis
is reduced by 3. *)
END_STRUCT
END_TYPE
```

- nChannels defines the number of independent channels. This must be greater than zero.

- nSubChannels defines the number of independent subchannels. This must be greater than or equal to zero.

- nOrder is the maximum order of the moment coefficients (E_CM_MCoefOrder [▶ 265]) 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.
- `bKurtosisExcess` is a Boolean value that specifies whether the excess is reduced by the value.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.20    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_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                        (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                        (* Length of analysis window.
 *)
    fSampleRate          : LREAL := 20000;                      (* Sample rate in Hertz. *)
    nMaxBands            : UDINT := 10;                         (* Maximum number of frequenc
y bands. *)
    nChannels            : UDINT := 10;                         (* Number of channels. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;   (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1]; (* Window parameters for spec
ific windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                         (* Size of overlap in samples
. *)
    bTransformToDecibel  : BOOL := TRUE;                        (* Transform to decibel. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;            (* Minimum argument of decadi
c logarithm for 64- bit IEEE 754 arithmetic. *)
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` Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.
- `nMaxBands` specifies the maximum number of frequency bands for which the RMS value is calculated.
- `nChannels` defines the number of independent channels. This must be greater than zero.
- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type `eCM_HannWindow`.
- `aWindowParameters` contains the free parameters of selected window functions. When using `eCM_KaiserWindow`, the first entry defines the parameter beta; if `eCM_FlatTopWindow` is used, all parameters are used. See section Window functions [▶ 20].
- `nOverlap`  defines the number of overlapping samples. This must be greater than or equal to zero. If the value `cCM_OverlapRecommended` is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value `cCM_OverlapInactive` deactivates the internally used buffer and sets the value to zero.
- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \to 20 * \log10(x)$.
- `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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

> **Note the window length**
>
> The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.21     ST_CM_Normalization_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_Normalization_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels        : UDINT := 1;    (* Number of channels. *)
    nSubChannels     : UDINT := 0;    (* Number of subchannels. *)
    nInLength        : UDINT := 256;  (* Length of input data array. *)
    fParameter_s     : LREAL := 1.0;  (* Parameter s for normalization. *)
    nParameter_p     : UDINT := 1;    (* Parameter p for normalization. *)
    nParameter_q     : UDINT := 1;    (* Parameter q for normalization. *)
    nWorkingDimension : UDINT := 0;   (* Dimension index of MultiArray for normalization. *)
    bAbsoluteValues  : BOOL := TRUE;  (* Use absolute values in calculation. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.
- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.
- `nInLength` is the length of the input array.
- `fParameter_s` defines the scaling factor of the normalization.
- `fParameter_p` defines the exponent of the summands in the normalization.
- `fParameter_q` defines the quotient of the outer exponent in the normalization.
- `nWorkingDimension` specifies the null-based dimension of the input multiarray over which summation/normalization is performed.
- `bAbsoluteValues` is a flag that can be used to select whether the individual values are added together in the amount.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

## 5.3.14.22     ST_CM_OrderPowerSpectrum_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_OrderPowerSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    fSampleRateSignal    : LREAL := 1000;                    (* Sample rate in Hertz of vib
ration signal. *)
    fSampleRatePosition  : LREAL := 1000;                    (* Sample rate in Hertz of pos
itions. *)
```

```
    fPositionScaling     : LREAL := 1;                        (* One rotation needs to be re
presented by 1. *)
    fMaxRPM              : LREAL := 1800;                     (* Max. rotations per minute.
*)
    nFFT_Length          : UDINT := 512;                     (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                     (* Length of analysis window.
*)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;         (* Minimum argument of decadic
 logarithm for 64- bit IEEE 754 arithmetic. *)
    bTransformToDecibel  : BOOL := TRUE;                     (* Transform to decibel. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;   (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];   (* Window parameters for speci
fic windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                      (* Size of overlap in samples.
 *)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;   (* Scaling type. *)
    nChannels            : UDINT := 1;                       (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `fSampleRateSignal` defines the sampling rate of the incoming vibration signal in Hertz.

- `fSampleRatePosition` defines the sampling rate of the incoming position signal in Hertz. This may be unequal to `fSampleRateSignal`.

- `fPositionScaling` is the scaling factor for the position signal. A full revolution must be represented by this value. If the position signal is scaled in degrees, `fPositionScaling = 360` must be set. If the position signal is scaled so that a full revolution of the shaft produces the value 2*pi, then `fPositionScaling = 2 * pi`.

- `fMaxRPM` is the maximum number of revolutions per minute. With the `fSampleRateSignal`, this parameter defines the maximum resolvable order of the order spectrum.

All other parameters are analogous to the parameterization of the power spectrum, see ST_CM_PowerSpectrum_InitPars [▶ 288].

- `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.

- `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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * \log10(x)$.

- `eWindowType` defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type `eCM_HannWindow`.

- `aWindowParameters` contains the free parameters of selected window functions. When using `eCM_KaiserWindow`, the first entry defines the parameter beta; if `eCM_FlatTopWindow` is used, all parameters are used. See section Window functions [▶ 20].

- `nOverlap` defines the number of overlapping samples. This must be greater than or equal to zero. If the value `cCM_OverlapRecommended` is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value `cCM_OverlapInactive` deactivates the internally used buffer and sets the value to zero.

- `eScalingType` enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is `eCM_DiracScaling`. 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.

- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.23 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_Object_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;                         (* Length of FFT. *)
    nWindowLength       : UDINT := 400;                         (* Length of analysis window. *)
    eWindowType         : E_CM_WindowType := eCM_HannWindow;    (* Window type. *)
    aWindowParameters   : T_CM_WindowParameters := [2.5,1,1,1,1]; (* Window parameters for specifi
c windows, e.g. FlatTop. *)
    nOverlap            : UDINT := -1;                          (* Size of overlap in samples. *
)
    fLogThreshold       : LREAL := cCM_MinArgLogN;             (* Minimum argument of decadic l
ogarithm for 64- bit IEEE 754 arithmetic
    eScalingType        : E_CM_ScalingType := eCM_DiracScaling;  (* Scaling type. *)
    nChannels           : UDINT := 1;                          (* Number of channels. *)
 *)
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 [▶ 267]). A good default value is the window type `eCM_HannWindow`.
- `aWindowParameters` contains the free parameters of selected window functions. When using `eCM_KaiserWindow`, the first entry defines the parameter beta; if `eCM_FlatTopWindow` is used, all parameters are used. See section Window functions [▶ 20].
- `nOverlap` defines the number of overlapping samples. This must be greater than or equal to zero. If the value `cCM_OverlapRecommended` is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value `cCM_OverlapInactive` deactivates the internally used buffer and sets the value to zero.
- `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` enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is `eCM_DiracScaling`. 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.
- `nChannels` defines the number of independent channels. This must be greater than zero.

> **ℹ 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 v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.24    ST_CM_PowerSpectrum_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_PowerSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                    (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                    (* Length of analysis window.
*)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;        (* Minimum argument of decadic
 logarithm for 64- bit IEEE 754 arithmetic. *)
    bTransformToDecibel  : BOOL := TRUE;                    (* Transform to decibel. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow; (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1]; (* Window parameters for speci
fic windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                     (* Size of overlap in samples.
 *)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;  (* Scaling type. *)
    nChannels            : UDINT := 1;                      (* Number of channels. *)
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.

- 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. (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 2.3e-308, which is equivalent to the constant cCM_MinArgLog10.)

- 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 the type E_CM_WindowType [▶ 267]). A good default value is the window type eCM_HannWindow.

- aWindowParameters contains the free parameters of selected window functions. When using eCM_KaiserWindow, the first entry defines the parameter beta; if eCM_FlatTopWindow is used, all parameters are used. See section Window functions [▶ 20].

- nOverlap  defines the number of overlapping samples. This must be greater than or equal to zero. If the value cCM_OverlapRecommended is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value cCM_OverlapInactive deactivates the internally used buffer and sets the value to zero.

- eScalingType enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is eCM_DiracScaling. 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.

- nChannels defines the number of independent channels. This must be greater than zero.

---

● **Note the window length**

ℹ The value of nWindowLength must be less or equal the value of nFFT_Length. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.25  ST_CM_Quantiles_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_Quantiles_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels      : UDINT := 512;    (* Number of channels. *)
    nSubChannels   : UDINT := 0;      (* Number of subchannels. *)
    fMinBinned     : LREAL := -120;   (* Minimum binned value. *)
    fMaxBinned     : LREAL := 100;    (* Maximum binned value. *)
    nBins          : UDINT := 100;    (* Number of bins. *)
    nMaxQuantiles  : UDINT := 10;     (* Maximum number of quantiles. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.

- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to 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.

- `nMaxQuantiles` is the number of quantiles to be calculated for each channel. This must be an integer greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.26  ST_CM_RainflowCounting_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_RainflowCounting_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nBins       : UDINT := 64;      (* Number of bins for the stress levels. *)
    fminStress  : LREAL := -40.0;   (* Lower limit of the stress bins. *)
    fmaxStress  : LREAL := 40.0;    (* Upper limit of the stress bins. *)
    nBinsMean   : UDINT := 64;      (* Number of bins for the mean levels. *)
    fminMean    : LREAL := -40.0;   (* Lower limit of bins for mean stress values. *)
    fmaxMean    : LREAL := 40.0;    (* Upper limit of bins for mean stress values. *)
    nChannels   : UDINT := 1;       (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nBins` defines the number of bins for the stress level. The total number of bins is `nBins + 2`; the value must be at least one. The first bin of each row contains the stress levels that are less than or equal to the defined minimum; analogously, all values greater than the defined maximum fall into the last bin of each row.

- `fminStress` defines the lower limit of the stress level. All values that are less than or equal to this value are accumulated in a bin.

- `fmaxStress` defines the upper limit of the stress level. Values that exceed the maximum are counted in the last bin.

- `nBinsMean` defines the number of bins for the mean values. The value must be at least one. Two separate bins are added in the first/last column of the *Halfcycle Count Matrix*, where mean values are kept that are less than or equal to or, respectively, greater than the defined limits.

- `fminMean` defines the lower limit of the mean values. All values that are less than or equal to this value are accumulated in a bin.

- `fmaxMean` defines the upper limit of the mean values. Values that exceed the maximum are counted in the last bin.

- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.27    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_Object_InitPars :
STRUCT
    nFFT_Length : UDINT := 512;    (* Length of FFT. *)
    bForward    : BOOL := TRUE;    (* Flag indicating forward FFT. *)
    bHalfSpec   : BOOL := TRUE;    (* Flag indicating length of output; if TRUE, the algorithm outp
uts half the spectrum (nFFT_Length/2 + 1). *)
    nChannels   : UDINT := 1;      (* Number of channels. *)
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 that specifies 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 size of the result buffer. If the value is `TRUE`, the algorithm outputs the half-spectrum (`nFFT_Length`/2 + 1).

- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.28    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_Object_InitPars :
STRUCT
    nChannels           : UDINT := 4;                (* Number of channels. *)
    nSubChannels        : UDINT := 0;                (* Number of subchannels. *)
    nBufferLength       : UDINT := 2000;             (* Buffer length. *)
    fDecibelThreshold   : LREAL := cCM_MinArgLog10;  (* Minimum argument of decadic logarithm for
 64-bit IEEE 754 arithmetic. *)
    bTransformToDecibel : BOOL := TRUE;              (* Transform to decibel. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.

- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.

- `nBufferLength` is the number of input values per channel 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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

- `bTransformToDecibel` is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * \log10(x)$.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.29 ST_CM_SlidingDFT_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_SlidingDFT_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nWindowLength  : UDINT := 1000;      (* Length of analysis window. *)
    nBins          : UDINT := 1;         (* Number of spectral bins. *)
    fDampingFactor : LREAL := 0.995;     (* Damping factor for SDFT. *)
    nChannels      : UDINT := 1;         (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nWindowLength` is the length of the analysis window in samples. The length must be greater than one and an even number.

- `nBins` is the number of spectral values to be calculated. The corresponding indices (`k := f / fSampleRate / nWindowLength`) must be configured after initialization using the `Configure()` method.

- `fDampingFactor` defines the value of the damping parameter of the recursion formula.

- `nChannels` defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.30 ST_CM_SparseSpectrum_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_SparseSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nWindowLength      : UDINT := 1000;                               (* Length of analy
sis window. *)
    fSampleRate        : LREAL := 10000;                              (* Sample rate in
Hertz. *)
    nBins              : UDINT := 1;                                  (* Number of spect
ral bins. *)
    eSpectrumType      : E_CM_SpectrumType := E_CM_SpectrumType.eCM_Magnitude; (* Type of spectru
m, i.e. plain DFT, magnitude, power. *)
    eWindowType        : E_CM_WindowType := eCM_HannWindow;           (* Window type. *)
    aWindowParameters  : T_CM_WindowParameters := [2.5,1,1,1,1];      (* Window paramete
rs for specific windows, e.g. FlatTop. *)
    nOverlap           : UDINT := -1;                                 (* Size of overlap
 in samples. *)
    eScalingType       : E_CM_ScalingType := eCM_DiracScaling;        (* Scaling type. *
```

```
)
    bTransformToDecibel  : BOOL := FALSE;                              (* Transform to de
cibel. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;                   (* Minimum argumen
t of decadic logarithm for 64- bit IEEE 754 arithmetic. *)
    nChannels            : UDINT := 1;                                 (* Number of chann
els. *)
END_STRUCT
END_TYPE
```

- nWindowLength is the length of the analysis window in samples. The length must be greater than one and an even number.

- fSampleRate Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.

- nBins is the number of spectral values to be calculated. The corresponding indices (k := f / fSampleRate / nWindowLength) must be configured after initialization using the Configure() method.

- eSpectrumType defines the type of spectral values (of the type E_CM_SpectrumType [▶ 266]) to be calculated. The scaling of the values is adapted to the respective type, so that the values correspond to the results of the function blocks FB_CMA_RealFFT [▶ 211], FB_CMA_MagnitudeSpectrum [▶ 168] or FB_CMA_PowerSpectrum [▶ 197].

- eWindowType defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type eCM_HannWindow.

- aWindowParameters contains the free parameters of selected window functions. When using eCM_KaiserWindow, the first entry defines the parameter beta; if eCM_FlatTopWindow is used, all parameters are used. See section Window functions [▶ 20].

- nOverlap defines the number of overlapping samples. This must be greater than or equal to zero. If the value cCM_OverlapRecommended is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value cCM_OverlapInactive deactivates the internally used buffer and sets the value to zero.

- eScalingType enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is eCM_DiracScaling. 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.

- bTransformToDecibel is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * log10(x)$.

- 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. (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 2.3e-308, which is equivalent to the constant cCM_MinArgLog10.)

- nChannels defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.31    ST_CM_SpikeEnergySpectrum_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_SpikeEnergySpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 1024;                             (* Length of FFT
. *)
    nWindowLength        : UDINT := 800;                              (* Length of ana
lysis window. *)
```

```
    fSampleRate          : LREAL := 10000.0;                          (* Sample rate i
n Hertz. *)
    fDecayTime           : LREAL := 0.08;                             (* Decay time co
nstant in seconds. *)
    fLowerFrequencyLimit : LREAL := 500.0;                            (* Lower limit o
f frequency band in Hertz. *)
    fUpperFrequencyLimit : LREAL := 4950.0;                           (* Upper limit o
f frequency band in Hertz. *)
    nOrder               : UDINT := 4;                                (* Order of band
pass filter in range [1 .. 10]. *)
    eWindowType          : E_CM_WindowType := E_CM_WindowType.eCM_HannWindow;   (* Window type.
*)
    aWindowParameters    : T_CM_WindowParameters := [2.5, 1.0, 1.0, 1.0, 1.0];  (* Window parame
ters for specific windows, e.g. FlatTop. *)
    nOverlap             : UDINT := 4294967295;                       (* Size of overl
ap in samples. *)
    eScalingType         : E_CM_ScalingType := E_CM_ScalingType.eCM_RootPowerSum;  (* Scaling type.
 *)
    bTransformToDecibel  : BOOL := FALSE;                             (* Transform to
decibel. *)
    fDecibelThreshold    : LREAL := 2.3e-308;                         (* Minimum argum
ent of decadic logarithm for 64- bit IEEE 754 arithmetic. *)
    nChannels            : UDINT := 1;                                (* Number of cha
nnels. *)
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 Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.

- fDecayTime defines the decay time of the time-wave form (spike function). The value can be configured using the Configure() method in order to improve the accuracy of the calculated spikes.

- fLowerFrequencyLimit is the lower limit of the internally used bandpass (bandwidth filter). The value should be higher than the expected machine vibration.

- fUpperFrequencyLimit is the upper limit of the internally used bandpass (bandwidth filter). The value should be chosen as high as possible in order to be able to identify high-frequency shocks.

- nOrder defines the order of the IIR filter used.

- eWindowType defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type eCM_HannWindow.

- aWindowParameters contains the free parameters of selected window functions. When using eCM_KaiserWindow, the first entry defines the parameter beta; if eCM_FlatTopWindow is used, all parameters are used. See section Window functions [▶ 20].

- nOverlap defines the number of overlapping samples. This must be greater than or equal to zero. If the value cCM_OverlapRecommended is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value cCM_OverlapInactive deactivates the internally used buffer and sets the value to zero.

- eScalingType enables the selection of the scaling to be used (of the type E_CM_ScalingType [▶ 266]), in case absolute scaling is required. The default value is eCM_DiracScaling. 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.

- bTransformToDecibel is a Boolean value that indicates whether the result of the FFT is to be transformed to the decibel scale, according to transformation $x \rightarrow 20 * \log10(x)$.

- 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. (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 2.3e-308, which is equivalent to the constant cCM_MinArgLog10.)

- nChannels defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.32    ST_CM_VibrationAssessment_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_VibrationAssessment_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                     (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                     (* Length of analysis window.
 *)
    fSampleRate          : LREAL := 20000;                   (* Sample rate in Hertz. *)
    nChannels            : UDINT := 2;                       (* Number of channels. *)
    nOrder               : UDINT := 2;                       (* Maximum order of integrati
on: 0 = acceleration, 1 = velocity, 2 = place. *)
    nMaxBands            : UDINT := 1;                       (* Maximum number of frequenc
y bands. *)
    nMaxClasses          : UDINT := 3;                       (* Number of configurable thr
eshold classes. *)
    bMemorize            : BOOL :=TRUE;                      (* Flag if results are memori
zed. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;    (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];  (* Window parameters for spec
ific windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                      (* Size of overlap in samples
. *)
    bTransformToDecibel  : BOOL := TRUE;                     (* Transform to decibel. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;         (* Minimum argument of decadi
c logarithm for 64- bit IEEE 754 arithmetic. *)
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 Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.

- nChannels defines the number of independent channels. This must be greater than zero.

- 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).

- nMaxBands specifies the maximum number of frequency bands for which the RMS value is calculated.

- nMaxClasses is the maximum number of classes that are configured. This must be at least one.

- bMemorize is a Boolean variable. If it is FALSE, the function block recalculates the number of the highest category and the corresponding channel for each step. If the value is TRUE, the result values are stored when a limit value is exceeded until the ResetData() method is executed or a channel reaches a higher category. The default value is TRUE.

- eWindowType defines the used window function (of the type E_CM_WindowType [▶ 267]). A good default value is the window type eCM_HannWindow.

- aWindowParameters contains the free parameters of selected window functions. When using eCM_KaiserWindow, the first entry defines the parameter beta; if eCM_FlatTopWindow is used, all parameters are used. See section Window functions [▶ 20].

- nOverlap  defines the number of overlapping samples. This must be greater than or equal to zero. If the value cCM_OverlapRecommended is selected, a recommended overlap is calculated internally (see F_CM_CalculateRecommendedOverlap [▶ 263]); the value cCM_OverlapInactive deactivates the internally used buffer and sets the value to zero.

- 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. (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 2.3e-308, which is equivalent to the constant `cCM_MinArgLog10`.)

---

> **ℹ️ Note the window length**
>
> The value of `nWindowLength` must be less or equal the value of `nFFT_Length`. The length of the FFT can orient itself to the required frequency resolution. Typically a value of about 4/5 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 for avoiding circular aliasing in calculations with inverse transformation in the time domain. Despite the higher frequency resolution, however, the result contains no more information.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.33    ST_CM_WatchUpperThresholds_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_WatchUpperThresholds_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels       : UDINT := 10;    (* Number of channels. *)
    nSubChannels    : UDINT := 0;     (* Number of subchannels. *)
    nMaxClasses     : UDINT := 3;     (* Number of configurable threshold classes. *)
    bMemorize       : BOOL :=TRUE;    (* Flag if results are memorized. *)
END_STRUCT
END_TYPE
```

- `nChannels` defines the number of independent channels. This must be greater than zero.

- `nSubChannels` defines the number of independent subchannels. This must be greater than or equal to zero.

- `nMaxClasses` is the maximum number of classes that are configured. This must be at least one.

- `bMemorize` is a Boolean variable. If it is `FALSE`, the function block recalculates the number of the highest category and the corresponding channel for each step. If the value is `TRUE`, the result values are stored when a limit value is exceeded until the ResetData() method is executed or a channel reaches a higher category. The default value is `TRUE`.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

## 5.3.14.34    ST_CM_ZoomFFT_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_CM_ZoomFFT_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length       : UDINT := 2048;      (* Length of FFT. *)
    fSampleRate       : LREAL := 10000.0;   (* Sample rate in Hertz. *)
    fCenter           : LREAL := 200.0;     (* Center frequency of band in Hertz. *)
    nDecimationFactor : UDINT := 16;        (* FFT decimation factor. *)
    nOrder            : UDINT := 4;         (* Order of lowpass filter in range [1 .. 20]. *)
```

```
    nChannels        : UDINT := 1;          (* Number of channels. *)
END_STRUCT
END_TYPE
```

- nFFT_Length is the length of the FFT. It must be greater than one and an integral power of two.
- fSampleRate Sampling rate of the incoming time signal. The value is used for the scaling of the result in Hz.
- fCenter is the center of the frequency band to be used. The bandwidth depends on the decimation factor.
- nDecimationFactor is the decimation factor of the applied low-pass filter after centering the input signal. The value must be a power of two to ensure integer division when downsampling.
- nOrder defines the order of the IIR filter used.
- nChannels defines the number of independent channels. This must be greater than zero.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

### 5.3.14.35    ST_MA_MultiArray_InitPars

Function block-specific structure with initialization parameters, which are analyzed when the function block is initialized.

```
TYPE ST_MA_MultiArray_InitPars :
STRUCT
    eTypeCode  : E_MA_ElementTypeCode := eMA_TypeCode_LREAL;               (* Used datatype. *
)
    nDims      : UDINT := 1;                                               (* Number of dimens
ions the array will have. *)
    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 di
mension. *)
END_STRUCT
END_TYPE
```

- eTypeCode The parameter specifies the element type (E_MA_ElementTypeCode [▶ 267]) of the MultiArray buffer element.
- nDims The parameter specifies the number of dimensions of the MultiArray buffer.
- aDimSizes The size of each dimension is specified by this array. If the shape of the input buffer of a following algorithm is defined with 'm x n' (in the input stream), the MultiArray buffer must be created with aDimSizes := [m,n].

**Requirements**

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

# 5.4    Global constants

## 5.4.1    GVL_CM

**Analysis function block constants:**

```
cCMA_MaxDest  : UDINT := 20;
cCMA_MaxID    : UDINT := 1000;
```

- cCMA_MaxDest: Maximum number of destinations for an analysis function block.
- cCMA_MaxID: Maximum ID that can be issued for an analysis function block (corresponds to the maximum number of possible analysis function blocks).

**Transfer Tray parameter:**

The internal transfer tray is initialized with the following constants for the transfer of data between the analysis function blocks.

```
cCMA_InitParsTransferTray : ST_MA_TransferTray_InitPars := (
                            nStreams     := 2048,
                            nMaxEntries  := 10,
                            nQueueSize   := 64,
                            bLockFree    := TRUE,
                            nUpdatePeriod := 2 );
```

- `nStreams` This parameter specifies how many independently functioning queues are provided. There should be a separate queue for each task-spanning data stream. Additional queues 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.

- `nQueueSize` The reserved length of the queues. This value must be larger than `nMaxEntries` and in addition must be an integral power of two.

- `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 blocks 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.

- `nUpdatePeriod` This parameter specifies how often internal interim 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 platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

# 5.4.2 GVL_CM_Base

**Limit value 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 of these constants is to avoid value range errors. The logarithm of real numbers is not defined in the case of negative numbers and zero. For this reason, constants are defined with values close to zero. The same applies to division by zero.

**Constants for recommended overlapping:**

```
cCM_OverlapRecommended  : UDINT := TO_UDINT(-1);    (* setting for recommended overlap (e.g. HannWin
dow: 50% overlapping) *)
cCM_OverlapInactive     : UDINT := TO_UDINT(-2);    (* setting in order to deactivate overlap *)
```

These constants are for setting the size of the internal data buffer when using the window functions. They serve in the InitPars structures [▶ 272] of the respective function blocks as values for the parameter nOverlap [▶ 278].

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | 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.

## 5.4.4 Param

**List of the parameters than can be set before runtime**

```
VAR_GLOBAL CONSTANT
    eEventTraceLevel  : TcEventSeverity := TcEventSeverity.Critical;
END_VAR
```

- `eEventTraceLevel`: The parameter enables the central setting of the TcEventSeverity of the events for the use of the TcEventLogger (see Overview) inside the Condition Monitoring library.

**Setting**

Parameters are changed via the Library Manager: in the Tc3_CM_Base library in the GVLs folder under Param in the column "Value (editable)".
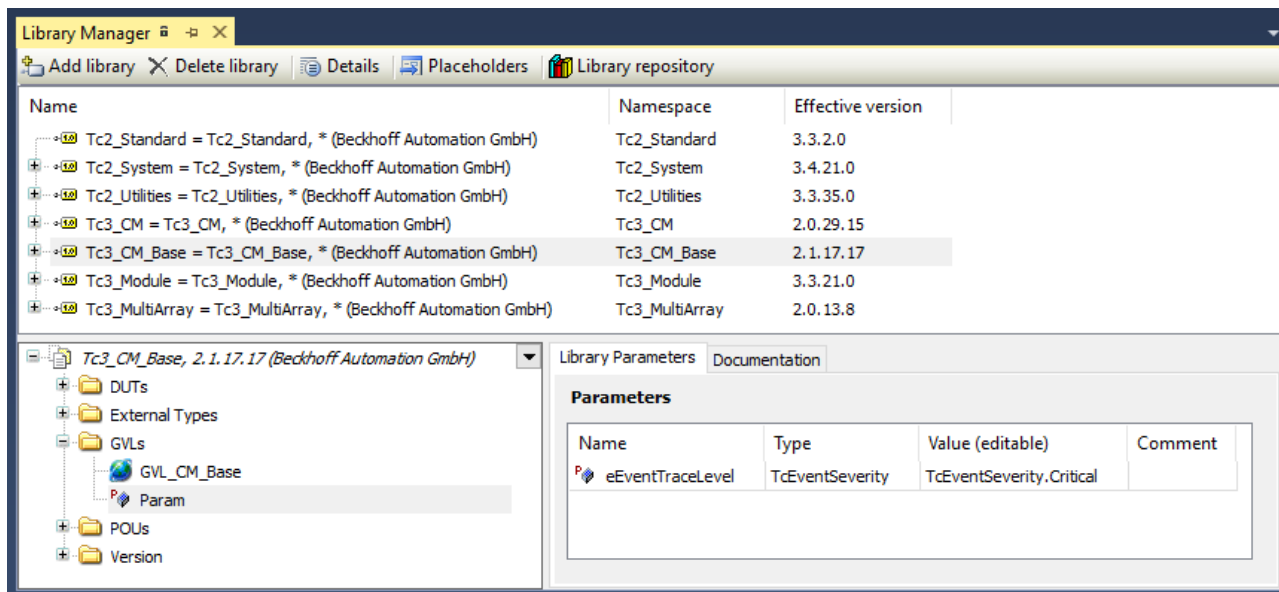


**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM_Base |

# 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 [▶ 211].

The sample is available for download from here:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394507531/.zip

**Block diagram**
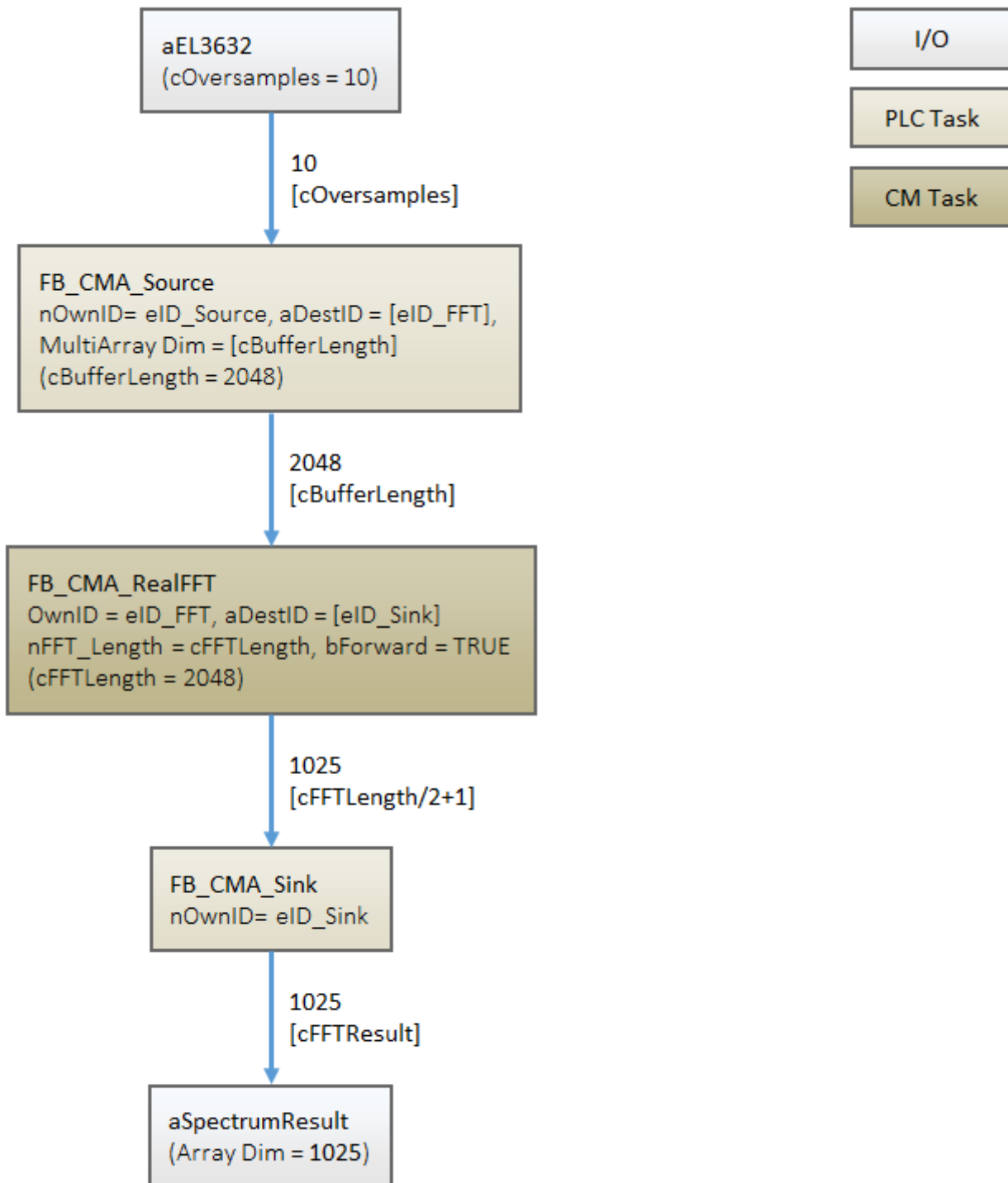


**Program parameters**

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 [▶ 109]. In contrast to the function block FB_CMA_RealFFT [▶ 211], the data type
LCOMPLEX is used for the required multi-array.

The sample is available for download from here:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394479755/.zip

**Block diagram**
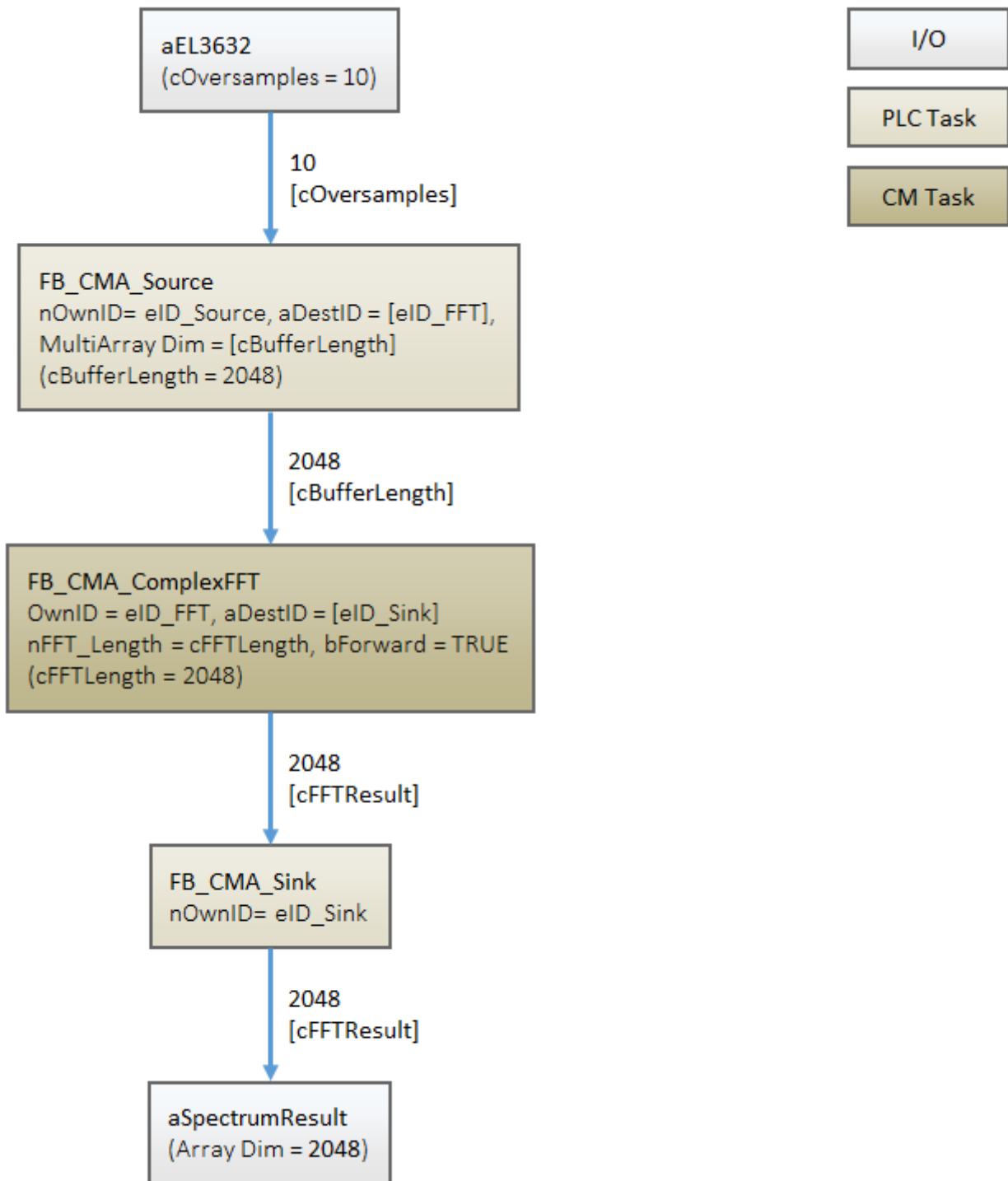


**Program parameters**

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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394574859/.zip

**BECKHOFF**
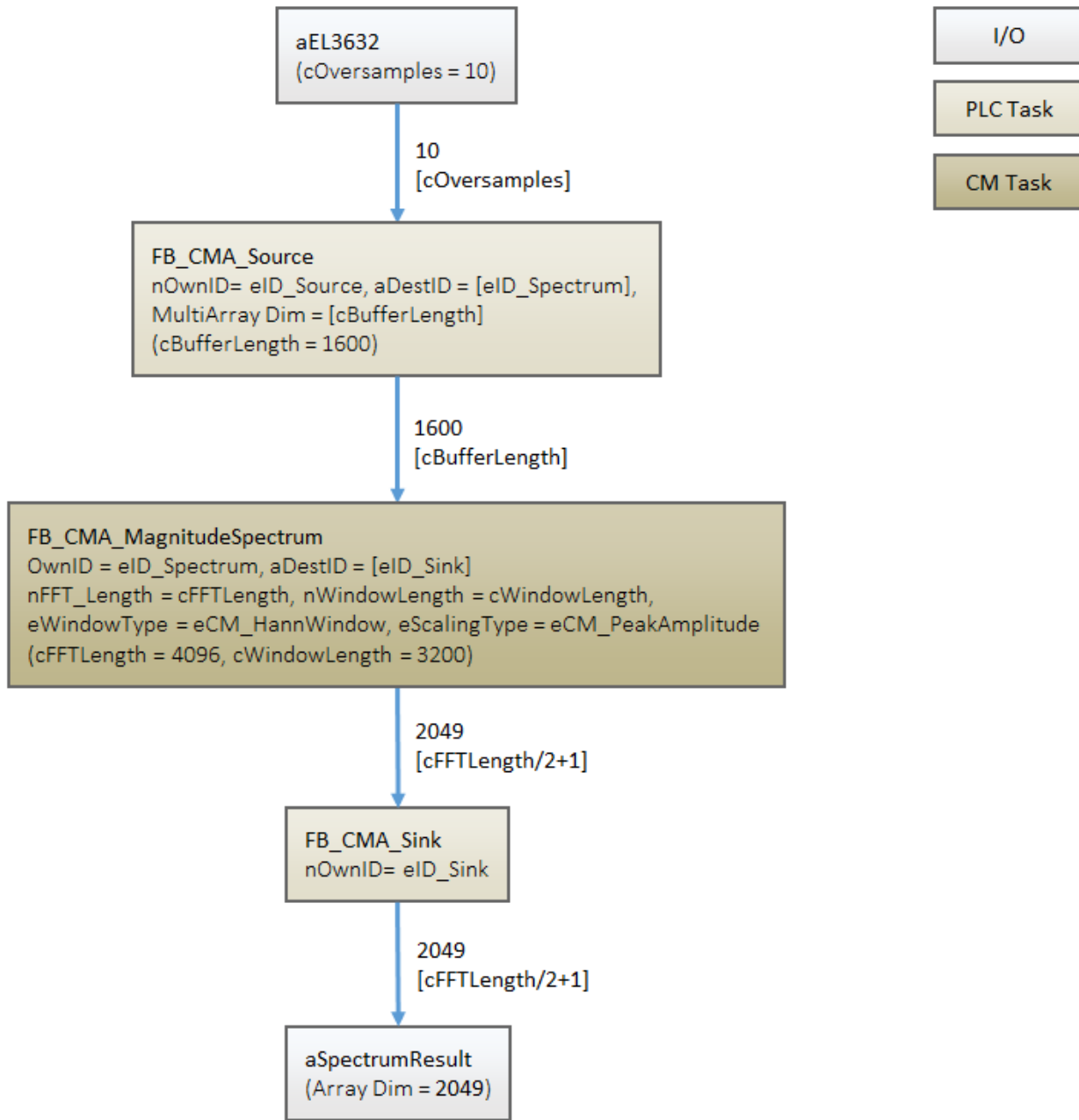
**Block Diagram**



**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

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 := 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 MAIN program:

```
fbSource.Input1D(pDataIn := ADR(aBuffer),
            nDataInSize := SIZEOF(aBuffer),
            eElementType := eMA_TypeCode_LREAL,
            nWorkDim := 0,
            pStartIndex := 0,
            nOptionPars := 0);

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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394576523/.zip

**Block diagram**
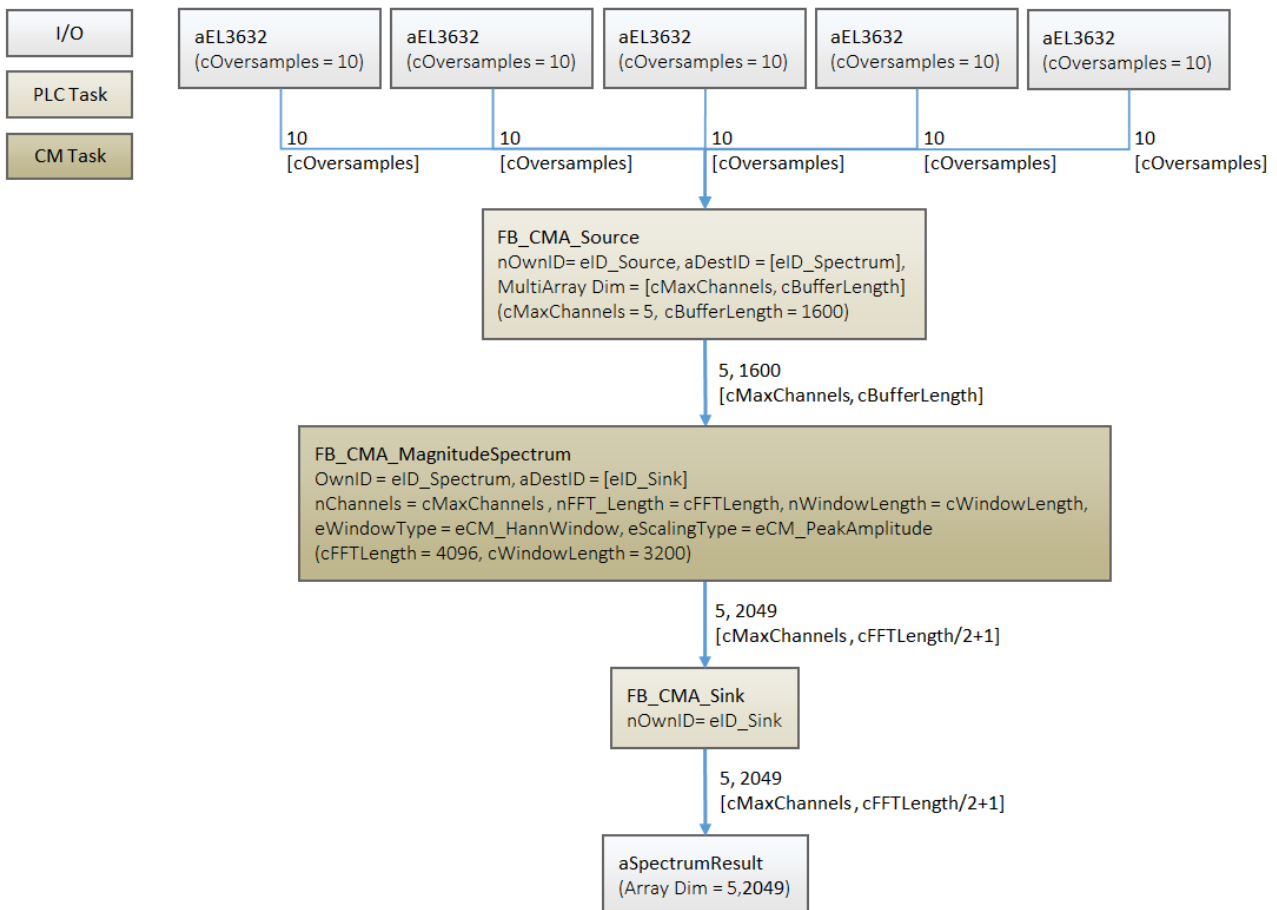


**Program parameters**

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_RootPowerSum |
| Conversion to decibels | FALSE |

### Global constants

These parameters are defined as constants in the list of global variables.

```
VAR_GLOBAL CONSTANT
    cOversamples  : UDINT := 20;                                  // number of oversamples
    cMaxChannels  : UDINT := 5;                                   // Number of data channels
    cWindowType   : E_CM_WindowType := E_CM_WindowType.eCM_HannWindow; // window type for analysis
    cWindowLength : UDINT := 3200;                                // length of signal window.
    cOverlap      : UDINT := 1600;                                // recommended buffer overlap
    cBufferLength : UDINT := cWindowLength -
 cOverlap;             // internal buffer size with 50% overlapping
    cFFTLength    : UDINT := 4096;                                // length of FFT for mag.
 spectrum
    cFFTResult    : UDINT := 2049;                               // result of mag. spectrum
(cFFTLength/2+1)
END_VAR
```

### Global variables

The parameters are defined in the list of global variables.

```
VAR_GLOBAL
    bInvalidateCh4 : BOOL := FALSE; // Invalidate input signal on channel 4
END_VAR
```

### Explanations

The result of the sample code can be tested for a sinusoidal signal of any amplitude and frequency. The RMS values are saved in the array `aRmsValue` in accordance with the respective channel number. The result must correspond exactly to the peak amplitude of every 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.

The setting of the global variables `bInvalidateCh4 := TRUE` demonstrates a possible error handling with impermissible input data.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

## 6.5    Calculation of individual spectral values

This sample implements as an example the possible uses of the function block FB_CMA_SparseSpectrum. Various window functions and scalings are presented for the possible calculations of DFT, magnitude and power values. Numerical effects in the detection are shown in two frequency portions of a generated signal: The first frequency corresponds to a multiple of the numerical resolution in the spectral range, while the second lies between two such values.

The source code for the sample is available for download from here: https://infosys.beckhoff.com/content/ 1033/TF3600_TC3_Condition_Monitoring/Resources/9066662027/.zip

## Block diagram
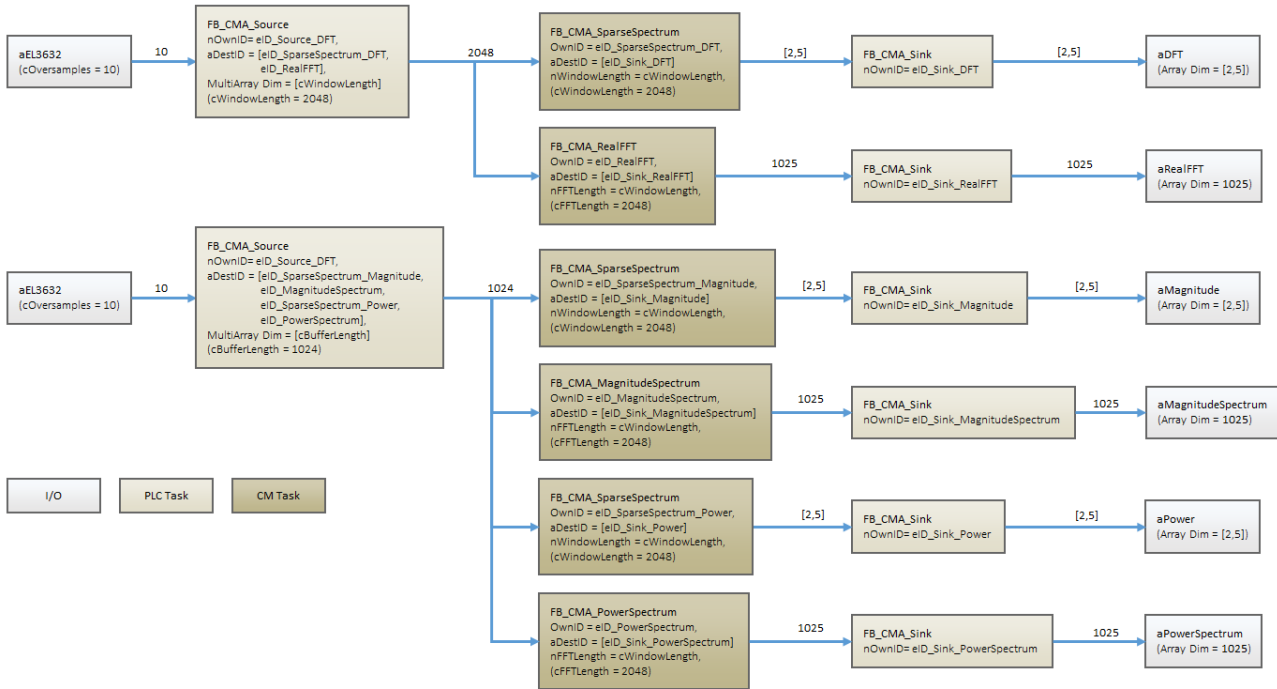


## Program parameters

The table below shows a list of the important parameters for configuring the function blocks.

| FFT-length | 2048 / 2048 / 2048 |
|---|---|
| Window size | 2048 / 2048 / 2048 |
| Buffer size | 2048 / 1024 / 1024 |
| Window type | eCM_RectangularWindow / eCM_HannWindow / eCM_HannWindow |
| Scaling method | eCM_NoScaling / eCM_PeakAmplitude / eCM_PeakAmplitude |
| Type of spectral values | eCM_DFT / eCM_Magnitude / eCM_Power |

## Configuration of the frequency bands

The GVL_Constants define the central parameters for initializing the algorithm and the properties of the generated signal.

```
VAR_GLOBAL CONSTANT
    cSampleRate  : LREAL := 10000;                              // Sample rate of input s
ignal.
    cWindowLength : UDINT := 2048;                              // Internal buffer size w
ith 50% overlapping
    cResolution  : LREAL := cSamplerate / cWindowLength;        // Frequency resolution
    cBands       : UDINT := 2;                                  // Number of bands
    cSetFrequency : ARRAY[1..cBands] OF LREAL := [ 41*cResolution, 413 ];  // Frequency in Hz; [ exa
ct, intermediate ]
    cSetAmplitude : ARRAY[1..cBands] OF LREAL := [ 1.0, 2.0 ];  // Peak amplitudes of sin
e signals
    cBandWidth   : UDINT := 5;                                  // Computed bins per
frequency.
    cDFTBins      : UDINT := cBandWidth * cBands;               // Number of spectral bin
s, cBandWidth bins per frequency
END_VAR
```

The generated signal (MAIN.aBuffer) consists of two frequency components. One is chosen with respect to the numerical resolution, i.e. it is a multiple of f = 10000 Hz/2048 = 4.8828125 Hz. The second is chosen such that the peak lies between two spectral values and thus cannot be precisely represented. In order to be able to illustrate the numerical effects, a further four values are calculated around the respectively sought spectral values. The configuration takes place in der MAIN_CM.

```
// Compute parameters, adjust if cDFTBins is changed.
FOR i := 1 TO cBands DO
    k := LREAL_TO_DINT(cSetFrequency[i] / cResolution);
    aDFTBins[(i-1) * cBandWidth + 1] := DINT_TO_UDINT(MAX(k-2,1));
    aDFTBins[(i-1) * cBandWidth + 2] := DINT_TO_UDINT(MAX(k-1,1));
    aDFTBins[(i-1) * cBandWidth + 3] := DINT_TO_UDINT(MIN(k+0,nyquist));
    aDFTBins[(i-1) * cBandWidth + 4] := DINT_TO_UDINT(MIN(k+1,nyquist));
    aDFTBins[(i-1) * cBandWidth + 5] := DINT_TO_UDINT(MIN(k+2,nyquist));
END_FOR
```

### Visualization of the results

The sample includes an extensive TwinCAT measurement project in which the calculated spectral values from the function block FB_CMA_SparseSpectrum are compared with the reference algorithms (FB_CMA_RealFFT, FB_CMA_MagnitudeSpectrum, FB_CMA_PowerSpectrum). The difference between an excitation as a multiple of the numeric frequency resolution and a value between two such values is illustrated. The visualized frequency bands can be considered as a "zoom" of the corresponding area.

The following illustrations show the results from the comparison of the function blocks FB_CMA_RealFFT and FB_CMA_MagnitudeSpectrum.
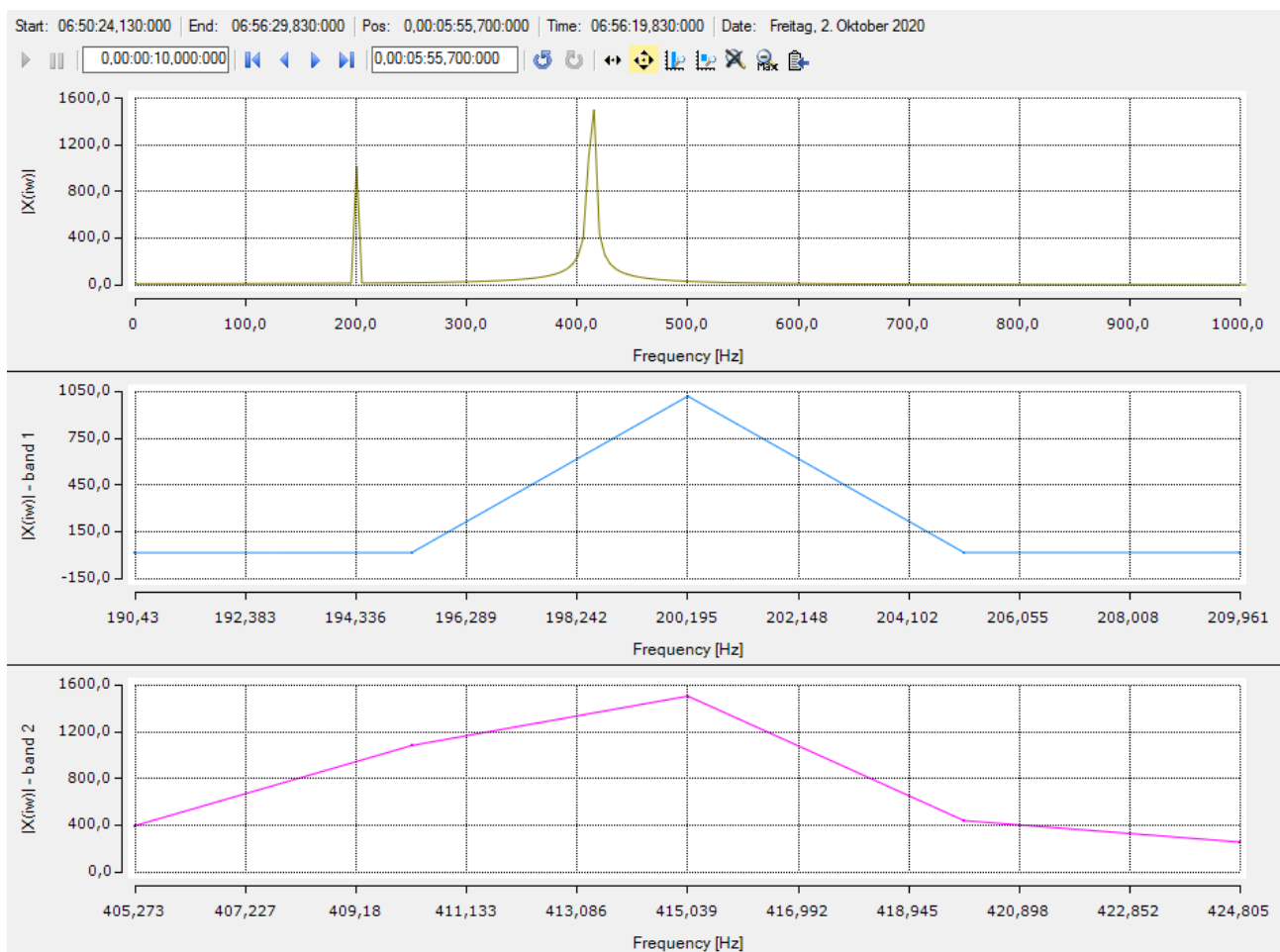


Spectrum of the FB_RealFFT (top) and the spectral values of bands 1 (center) and 2 (bottom).
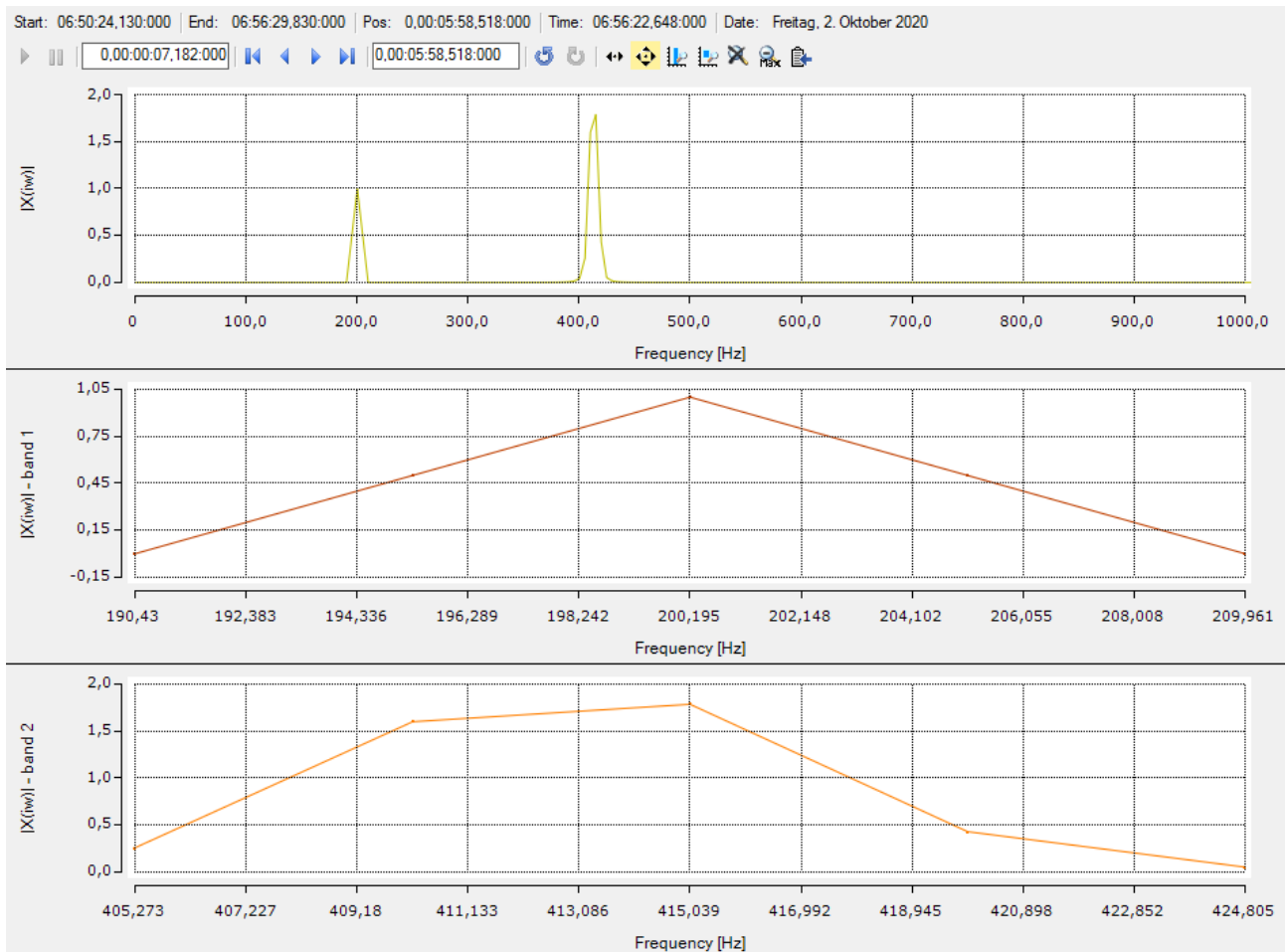
Magnitude spectrum of the function block FB_MagnitudeSpectrum (top) as well as the spectral values of bands 1 (center) and 2 (bottom).

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

# 6.6 Window functions

This sample implements several single-channel magnitude spectra and compares the use of different window functions. Three variants are presented: One with the overlap mechanism deactivated, an automatic calculation of the recommended overlap and a manual configuration using a suitable firmware function.

The source code for the sample is available for download from here: https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/5261536139/.zip

## Block diagram
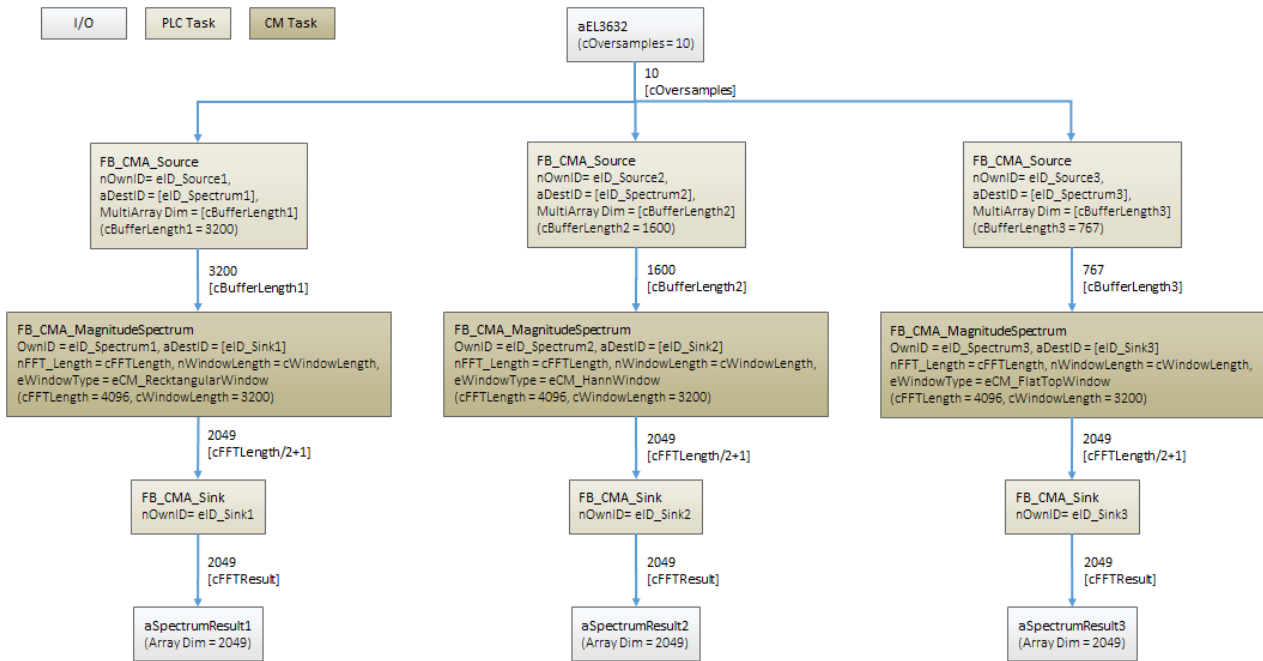


## Program parameters

The table below shows a list of important parameters for the configuration of the magnitude spectrum function blocks.

| | |
|---|---|
| FFT length | 4096 |
| Window size | 3200 |
| Buffer size | 3200 / 1600 / 767 |
| Window type | `eCM_RectangularWindow` / `eCM_HannWindow` / `eCM_FlatTopWindow` |
| Decibel transformation | `False` |

## Calculation/specification of the overlap

### Version 1: Use of the function F_CM_CalculateRecommendedOverlap [▶ 263]

By means of the named function the recommended overlap can be calculated at runtime. After download/ login, the necessary values are called in the `MAIN_CM` in the first cycle.

```
IF bCalculateOverlap THEN
    // recommended overlap for window 1
    nOverlap1 := F_CM_CalculateRecommendedOverlap(eWindowType := cWindowType1, aWindowParameters :=
cWindowParameters1, nWindowLength := cWindowLength);

    // recommended overlap for window  2
    nOverlap2 := F_CM_CalculateRecommendedOverlap(eWindowType := cWindowType2, aWindowParameters :=
cWindowParameters2, nWindowLength := cWindowLength);

    // recommended overlap for window  3
    nOverlap3 := F_CM_CalculateRecommendedOverlap(eWindowType := cWindowType3, aWindowParameters :=
cWindowParameters3, nWindowLength := cWindowLength);

    bCalculateOverlap := FALSE;
END_IF
```

The lengths obtained in this way can subsequently be entered in the configuration of the analysis chains (parameters `cOverlap1, cOverlap2, cOverlap3` in the `GVL_Constants`.). The dependencies in the buffer length must be observed here! After downloading/logging in again, the application is configured with regard to the overlap.

### Version 2: Reading the initial parameters in the Online view [▶ 83]

If the value `nOverlap := cCM_OverlapRecommended` is used with the initialization structure for the overlap, then the optimum overlap is calculated in the initialization phase of the function block, depending on the window parameters. This value can be read in the Online View of the function block in the node `stInitPars` after the download and login. With this variant it is not necessary to start the application!

| Expression | Type | Value |
|---|---|---|
| ⊟ 🔧 stInitPars | ST_CM_MagnitudeS... | |
| 🔹 eObjState | TCOM_STATE | TCOM_STATE_OP |
| 🔹 sObjName | STRING(127) | 'CMA_MagnitudeSpectrum' |
| 🔹 eTraceLevel | TCEVENTSEVERITY | Critical |
| 🔹 nFFT_Length | UDINT | 4096 |
| 🔹 nWindowLength | UDINT | 3200 |
| 🔹 eWindowType | E_CM_WINDOWTYPE | eCM_HannWindow |
| ⊞ 🔹 aWindowParameters | T_CM_WindowPara... | |
| 🔹 nOverlap | UDINT | 1600 |
| 🔹 eScalingType | E_CM_SCALINGTYPE | eCM_DiracScaling |
| 🔹 bTransformToDecibel | BOOL | FALSE |
| 🔹 fDecibelThreshold | LREAL | 2.3E-308 |
| 🔹 nChannels | UDINT | 1 |

Online View for Spectrum 2 (Hann window)

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

# 6.7    Scaling of spectra

As described under Scaling of spectra [▶ 26], 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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394510859/.zip

**Block diagram**
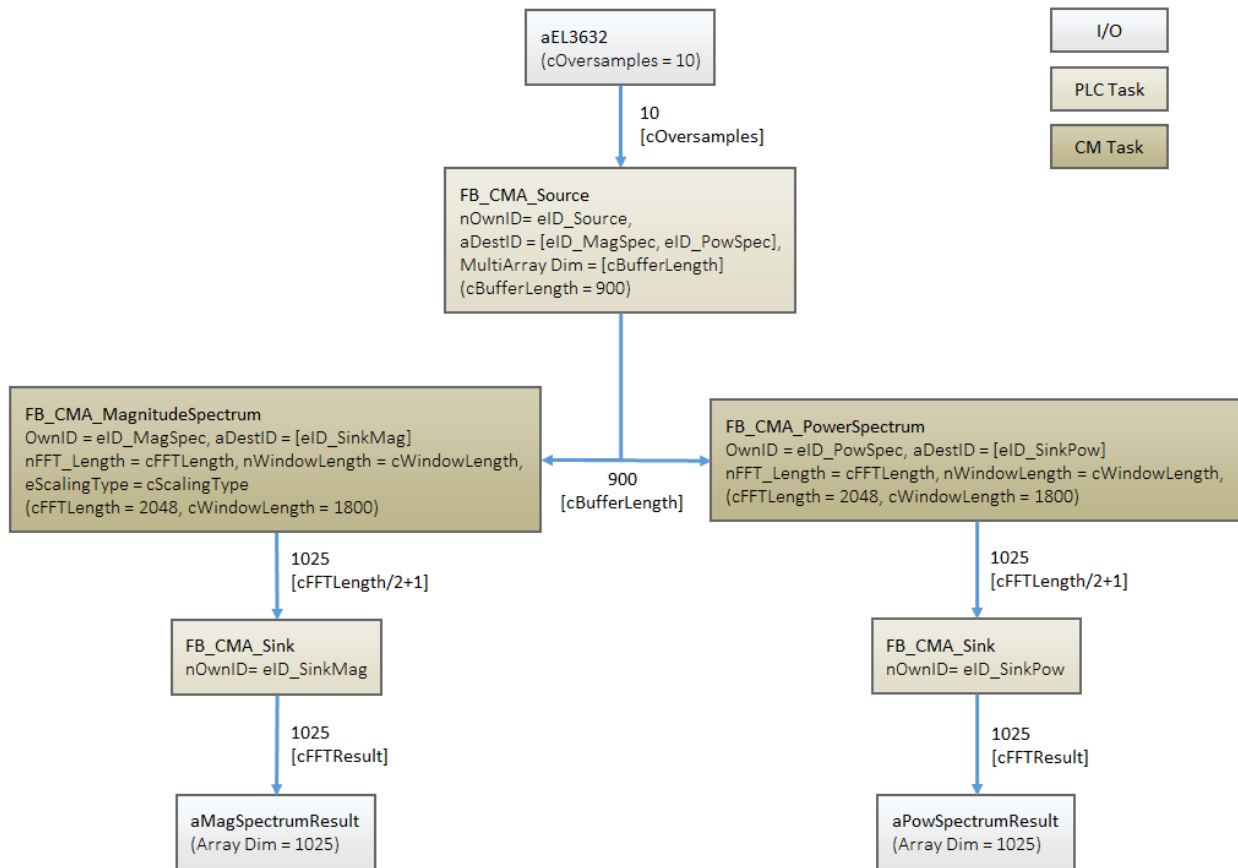


**Program parameters**

The table below shows a list of important parameters for the configuration of the used function blocks FB_CMA_MagnitudeSpectrum [▶ 168] and FB_CMA_PowerSpectrum [▶ 197].

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 |

# 6.8    Time-based RMS

The sample illustrates the implementation of a time-based RMS calculation for a signal with the function block FB_CMA_RMS [▶ 215].

The sample is available for download from here:

https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394509195/.zip

**Block diagram**
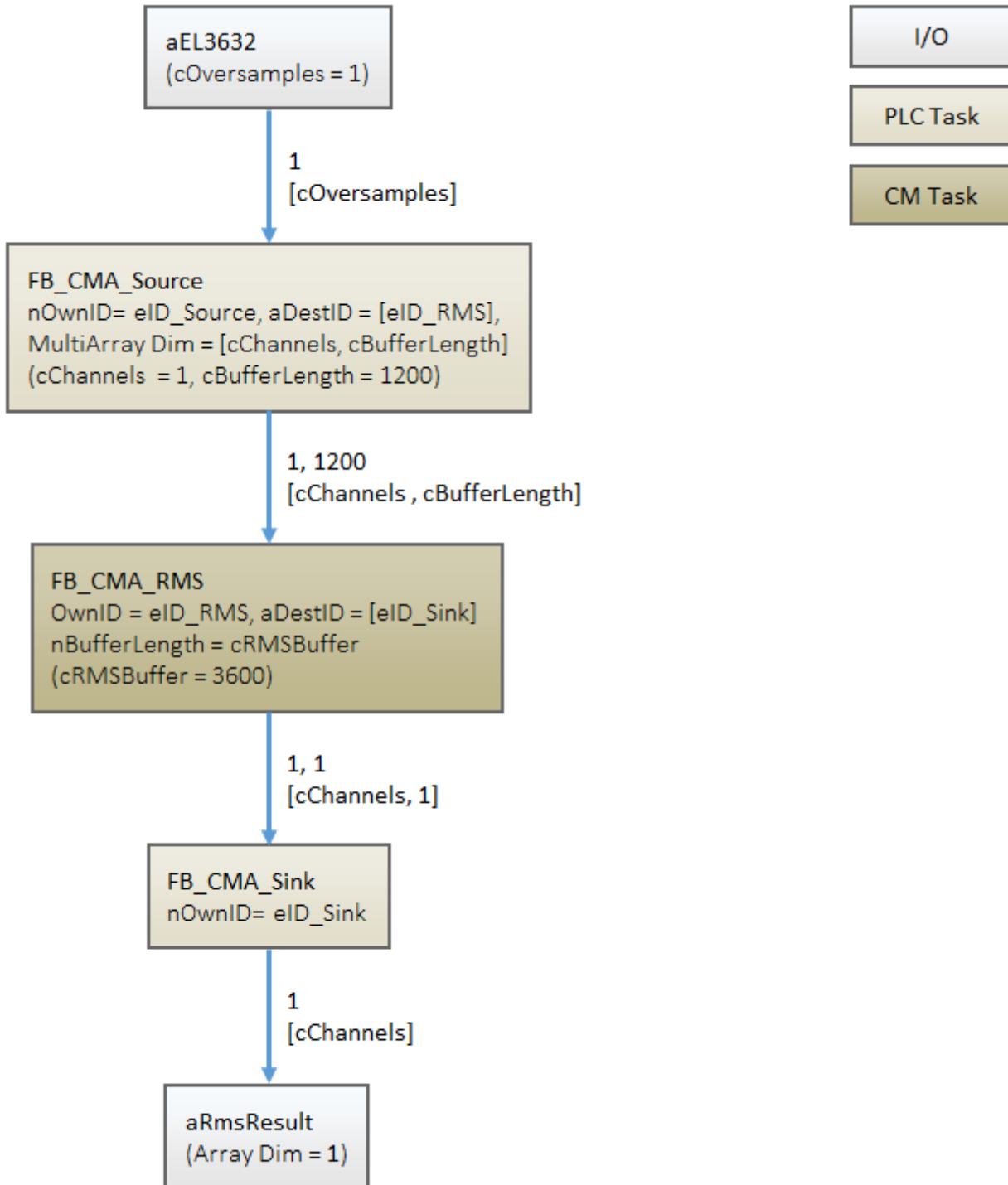


**Program parameters**

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 |

# 6.9 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 [▶ 184].

The sample is available for download from here:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394505867/.zip

**BECKHOFF**
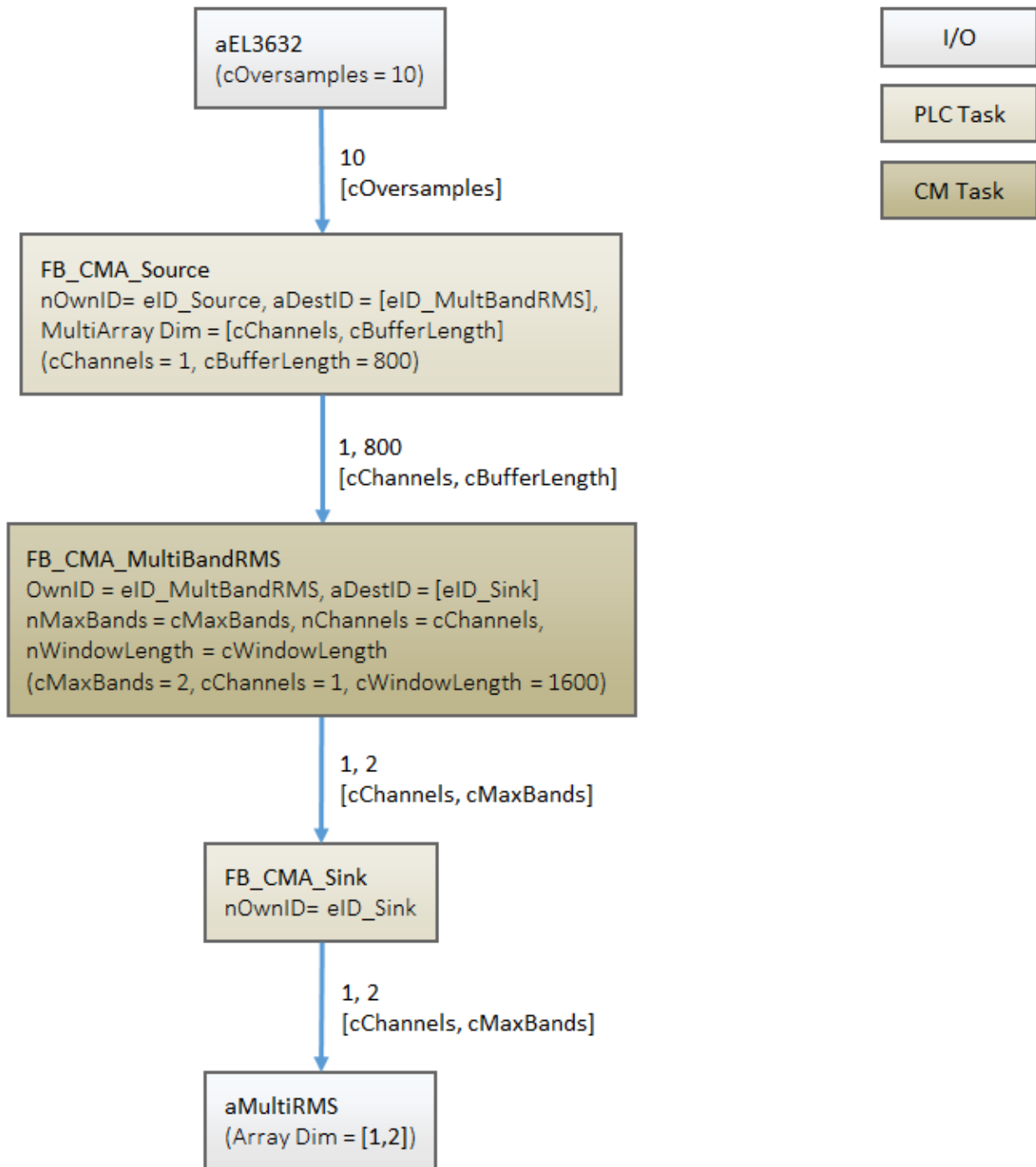
**Block diagram**



**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 |

# 6.10 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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394573195/.zip

**Block diagram**
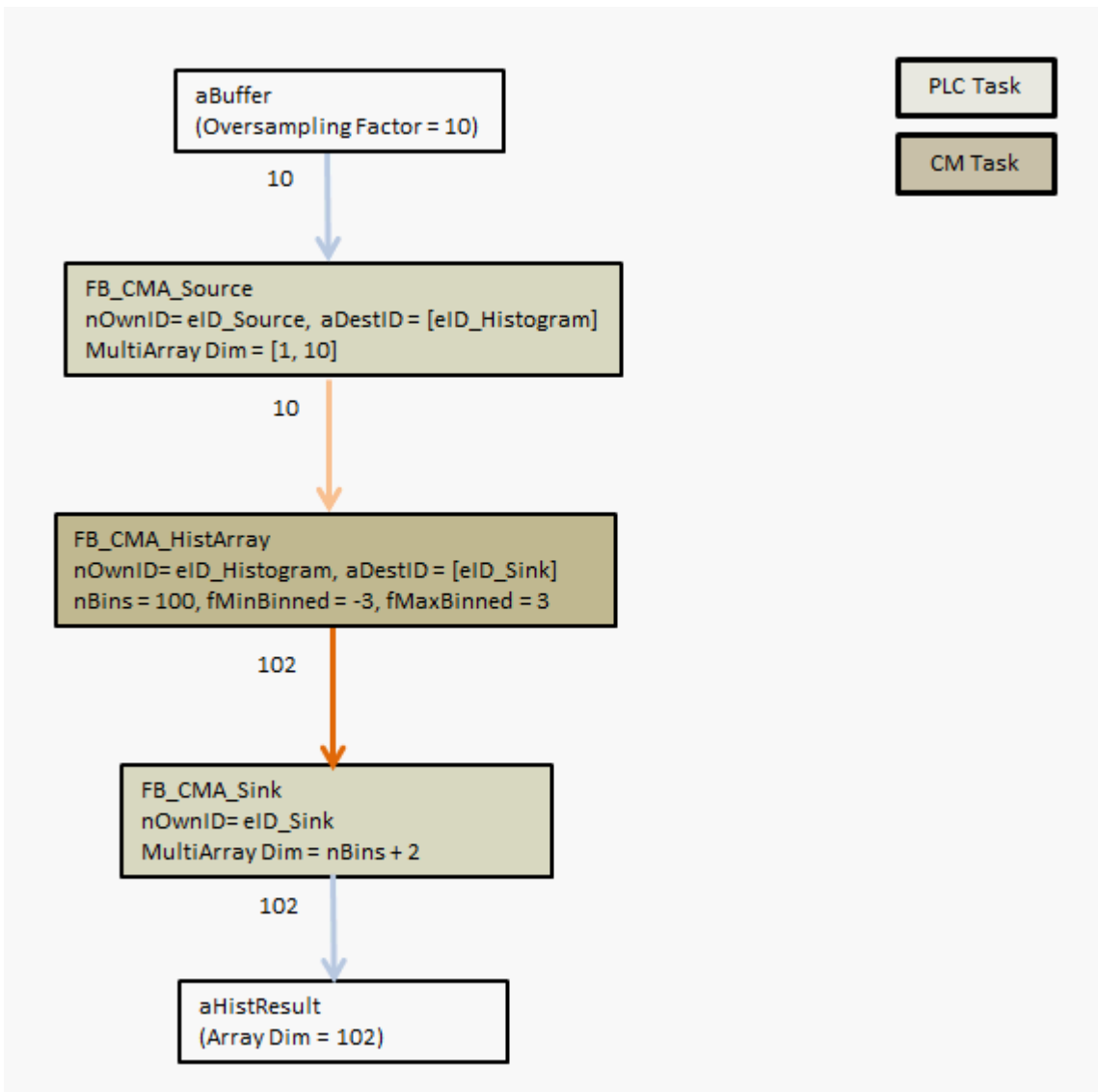


**Program parameters**

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  := 0);

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,
fMinBinnded := -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.

> **i** 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 |

# 6.11  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 [▶ 151], FB_CMA_EmpiricalMean [▶ 128], FB_CMA_EmpiricalStandardDeviation [▶ 138], FB_CMA_EmpiricalSkew [▶ 133] and FB_CMA_EmpiricalExcess [▶ 123] are used.

The sample is available for download from here:

https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/5261532811/.zip

**Block diagram**
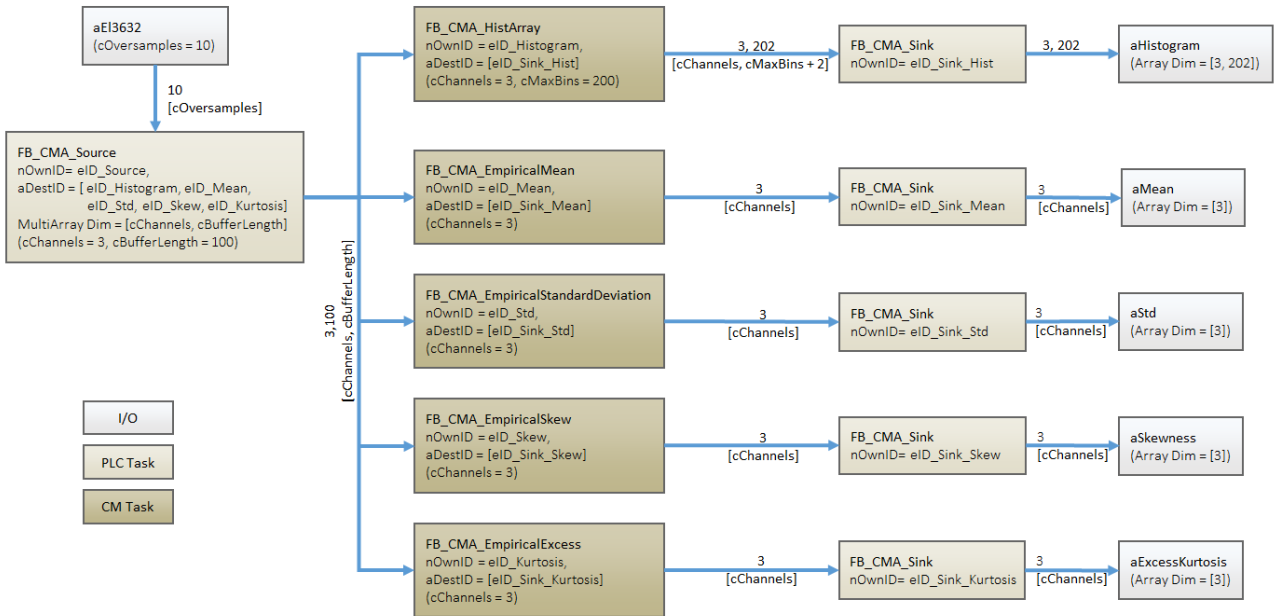


**Program parameters**

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 |

# 6.12 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 [▶ 34]. The classification based on the calculated RMS values is done directly in the MAIN program. Alternatively, the function blocks FB_CMA_WatchUpperThresholds [▶ 255] or FB_CMA_DiscreteClassification [▶ 118] could be used.

The sample is available for download from here:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394512523/.zip

An alternative implementation can be found in the sample Vibration assessment according to ISO 10816-3 (compact) [▶ 323] and in the sample Vibration assessment according to ISO 10816-3 (extended) [▶ 325].

**Block diagram**

```
aEL3632[1]                    aEL3632[2]                    ┌──────────┐
(cOversamples = 10)           (cOversamples = 10)           │   I/O    │
                                                            └──────────┘
     │ 10                          │ 10                      ┌──────────┐
     │ [cOversamples]              │ [cOversamples]          │ PLC Task │
     ▼                             ▼                         └──────────┘
                                                            ┌──────────┐
FB_CMA_Source                                               │ CM Task  │
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])
```
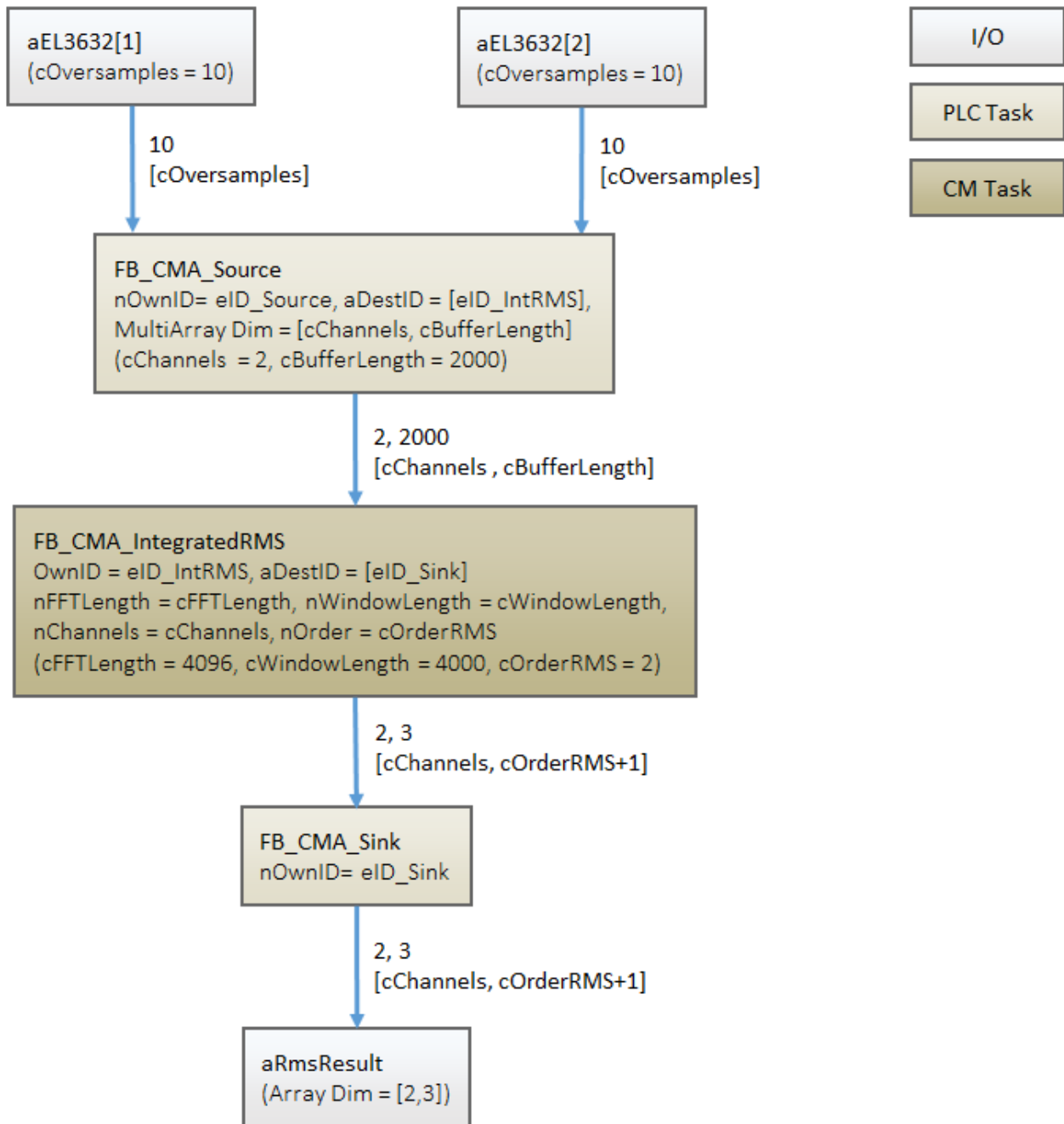
**Program parameters**

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 |

### 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 Digitization [▶ 14]).

### 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
```

Accordingly, as indicated in the diagram above, an array of the size [2,2000] is obtained for the transfer to the function block FB_CMA_IntegratedRMS.

### 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. The output of the function block is accordingly a 2-dimensional array with [2,3] (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 [▶ 72]. 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 vibration 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.

**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 Vibration assessment according to ISO 10816-3 (compact)

Vibration assessment based on ISO 10816-3 is explained in more detail in section Application concepts, see Vibration assessment [▶ 34]. The classification takes place via the function block FB_CMA_VibrationAssessment [▶ 251].

The sample is available for download from here: https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/8236478987/.zip

An alternative implementation can be found in the sample Vibration assessment according to ISO 10816-3 [▶ 320] and in the sample Vibration assessment according to ISO 10816-3 (extended) [▶ 325].

**Block diagram**
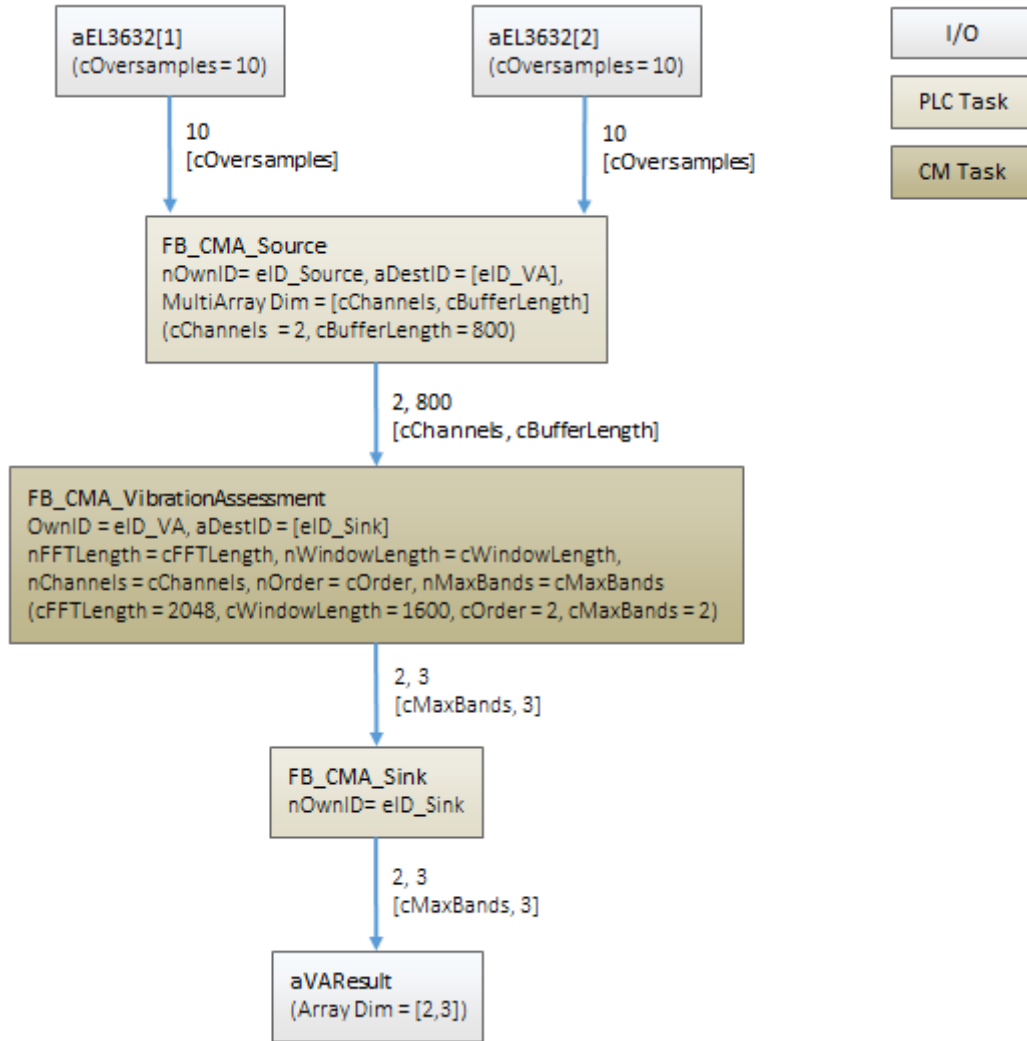


**Program parameters**

The table below shows a list of important parameters for the configuration of the function blocks that are used.

| | |
|---|---|
| Buffer size | 800 |
| Channels | 2 |
| FFT length | 2048 |
| Window size | 1600 |
| Sampling rate | 10000 |
| Number of frequency bands | 2 |
| Lower frequency bound | [10, 200] |
| Upper frequency bound | [1000, 2000] |
| Order (RMS) | 2 |
| Window type | eCM_HannWindow |
| Conversion to decibels | FALSE |

**Configuration**

The speed and deflection of the measured data are used for the machine monitoring according to ISO 10816-3. A classification on the basis of the acceleration data does not take place. For this reason the classification is implicitly deactivated by setting the limit values to a sufficiently high value (see `GVL_Constants`):

```
cISOClassDef_Vibration : ARRAY[1..cMaxClasses] OF LREAL := [1E6, 1E6, 1E6];
cISOClassDef_Velocity  : ARRAY[1..cMaxClasses] OF LREAL := [2.3E-3, 4.5E-3, 7.1E-3];
cISOClassDef_Displ     : ARRAY[1..cMaxClasses] OF LREAL := [29E-6, 57E-6, 90E-6];
```

**Evaluation**

The definition of the classification on the basis of ISO 10816-3, based on the speed and deflection for all channels, takes place in the control task. If a new result is available, it is evaluated as follows:

```
IF bCalculate THEN
    // Highscore in classification according to ISO 10816-3.
    ISO_10816_HighscoreClass   := aVAResult[ISO_10816_nSelectedBand][1];  // class
    ISO_10816_HighscoreOrder   := aVAResult[ISO_10816_nSelectedBand][2];  // order
    ISO_10816_HighscoreChannel := aVAResult[ISO_10816_nSelectedBand][3];  // channel

    IF NOT (ISO_10816_HighscoreClass = E_IsoClass.Error) THEN
        nCountResults := fbSink.nCntResults;
        // ToDo: if succeeded
    ELSE
        // ToDo: if error; RMS result is NaN. Code here what to do.
    END_IF
END_IF
```

The result data `aVAResult` contain the following information for all configured frequency bands:

- `ISO_10816_HighscoreClass`: The classification (`A-D`) of the machine state on the basis of the configured limit values.
- `ISO_10816_HighscoreOrder`: The integration order for the calculated evaluation, i.e. `0` for the acceleration, `1` for the speed and `2` for the deflection.
- `ISO_10816_HighscoreChannel`: The basic channel for the calculated evaluation.

In this sample, the frequency band for the evaluation is selected via the variable `ISO_10816_nSelectedBand`.

**Memory property**

The memory property of the algorithm can be changed via the initial parameter `bMemorize`. If the parameter is set to `TRUE`, the highest classification is not overwritten again by a lower evaluation until the method `ResetData()` is called. Compare the behavior of the function block FB_CMA_WatchUpperThresholds [▶ 255] here.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ⓘ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].
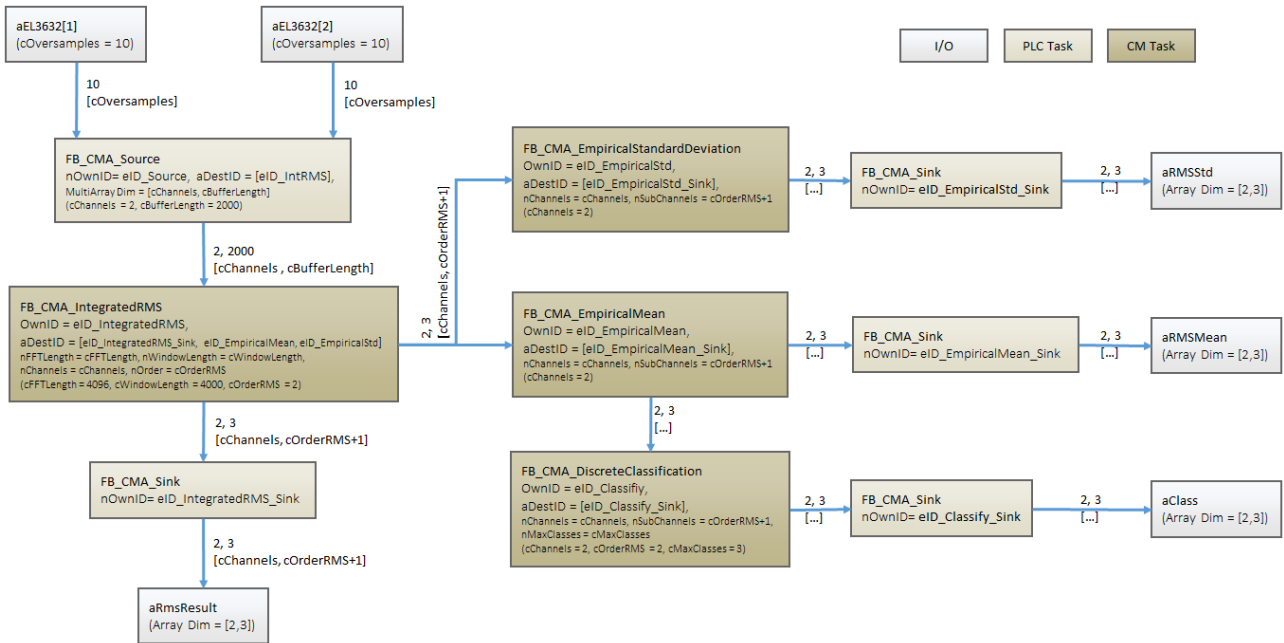
# 6.14 Vibration assessment according to ISO 10816-3 (extended)

Vibration assessment based on ISO 10816-3 is explained in more detail in section Application concepts, see Vibration assessment [▶ 34].

In comparison with the sample <u>Vibration assessment according to ISO 10816-3 [▶ 320]</u> the calculated (integrated) RMS values are first averaged with the help of the function block <u>FB_CMA_EmpiricalMean</u> <u>[▶ 128]</u>. The classification takes place here on the basis of the mean values by an instance of <u>FB_CMA_DiscreteClassification [▶ 118]</u>. By using statistical data, the machine evaluation is more stable than with the direct processing of the RMS values. The same effect could also be achieved in the above sample by adjusting the window length accordingly, but with a significantly higher computational effort.

The sample is available for download from here: https://infosys.beckhoff.com/content/1033/ TF3600_TC3_Condition_Monitoring/Resources/5261534475/.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 | 2000 |
|---|---|
| Channels | 2 |
| FFT length | 4096 |
| Window size | 4000 |
| Sampling rate | 10000 |
| Lower frequency bound | 10 |
| Upper frequency bound | 1000 |
| Order (RMS) | 2 |
| Classes (classification) | 3 |
| Window type | eCM_HannWindow |
| Data sets (statistics) | 100 |
| Reset after calculation (statistics) | `TRUE` |

**Configuration**

The speed and deflection of the measured data are used for the machine monitoring according to ISO 10816-3. A classification based on the acceleration data does not take place, but is used in related standards, e.g. ISO 10816-21 for wind turbines. The limit values used here are defined in the `GVL_Constants`:

```
cISOClassDef_Acc        : ARRAY[1..3] OF LREAL := [1E6, 1E6, 1E6];
cISOClassDef_Velocity   : ARRAY[1..3] OF LREAL := [2.3E-3, 4.5E-3, 7.1E-3];
cISOClassDef_Displ      : ARRAY[1..3] OF LREAL := [29E-6, 57E-6, 90E-6];
```

**Evaluation**

The calculated RMS values for velocity and deflection as well as the associated mean values and variance (standard deviations) are stored channel-wise in instances of `ST_Channel`. Exclusively the mean values are used for the classification. The variance can be used as a measure of the reliability of the mean value. The result of the classification is illustrated on the basis of the configured limit values by means of `E_IsoClass`. The result data are visualized via a Scope project.



**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

ℹ️ Limited functional scope already available with CM 3.1. See section Compatibility [▶ 64].

# 6.15  Condition Monitoring with frequency analysis

This tutorial configures a complete monitoring application, based on the TwinCAT 3 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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394571531/.zip

It can be modified and extended as required.

**Block diagram**
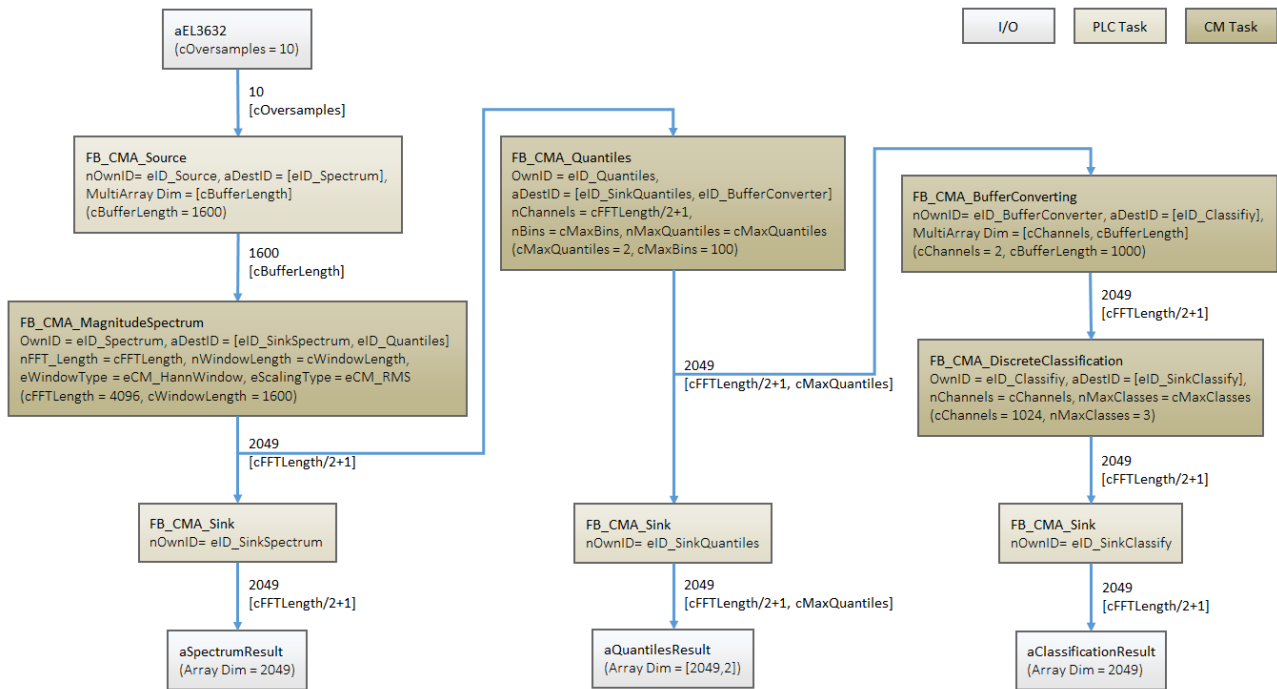


## Step 1: Application specification

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 <u>bearing analysis [▶ 39]</u>, <u>gear unit analysis [▶ 46]</u> or <u>frequency analysis [▶ 35]</u>, 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 4096 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. You can find additional information on this topic at the <u>task setting [▶ 72]</u>. The aim of this tutorial is to calculate the magnitude spectrum of 4096 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 in the <u>Task cycle time section [▶ 72]</u>. Make sure that adequate <u>router memory capacity [▶ 71]</u> 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 4096 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 <u>scaling options [▶ 361]</u>, 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 $Q_{0.5}$ is a value for which almost 50% of the values of a distribution are less than $Q_{0.5}$. It is also referred to as median value. Similarly, a 90% quantile $Q_{0.9}$ indicates a value for which 90% of the values of a distribution are less 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 you can find a more detailed description of the function block FB_CMA_Quantiles [▶ 201]. 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
```

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 [▶ 94]. 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.
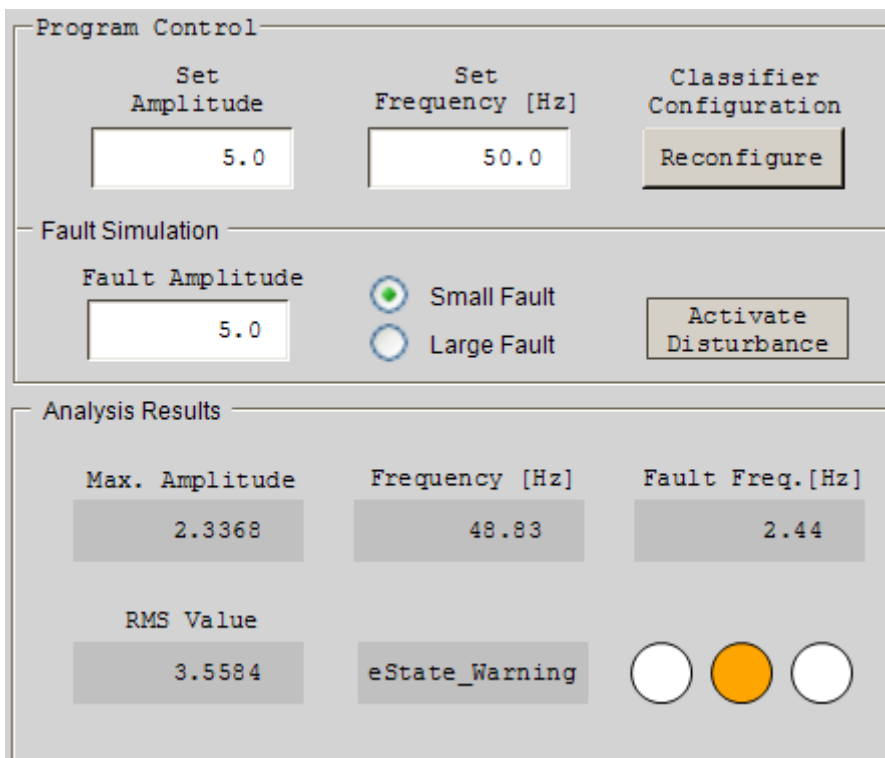


The diagram illustrates that the state of the application is shown in the corresponding display field. A small-sized error can be simulated by pressing the Add Fault button. You can see how the RMS value of the input signal slowly increases beyond the threshold value and how the state changes accordingly. 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 |

# 6.16 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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394495883/.zip

**Block diagram for the analysis chain:**
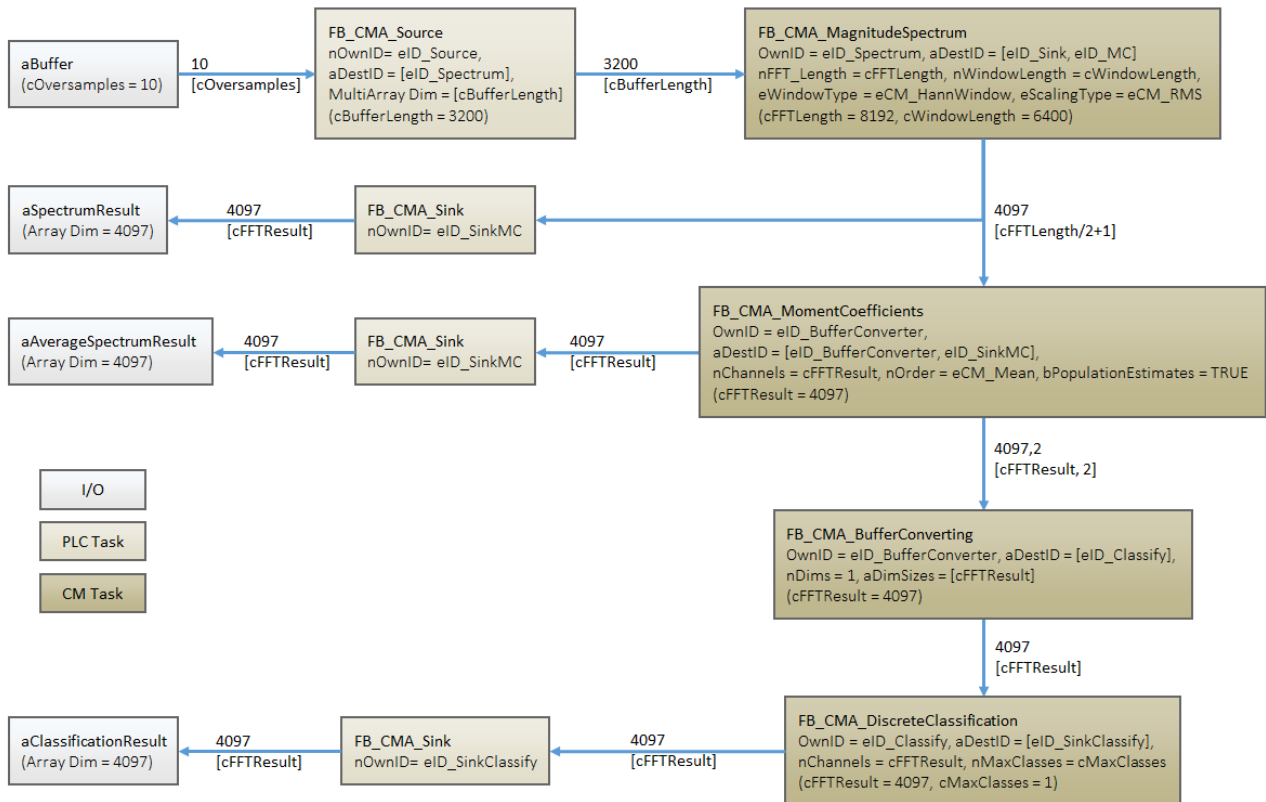


**Program parameters**

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 |

# 6.17 Crest factor

This sample calculates the crest factor for an input signal. Although the function block FB_CMA_CrestFactor [▶ 97] 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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394544267/.zip

**Block Diagram**
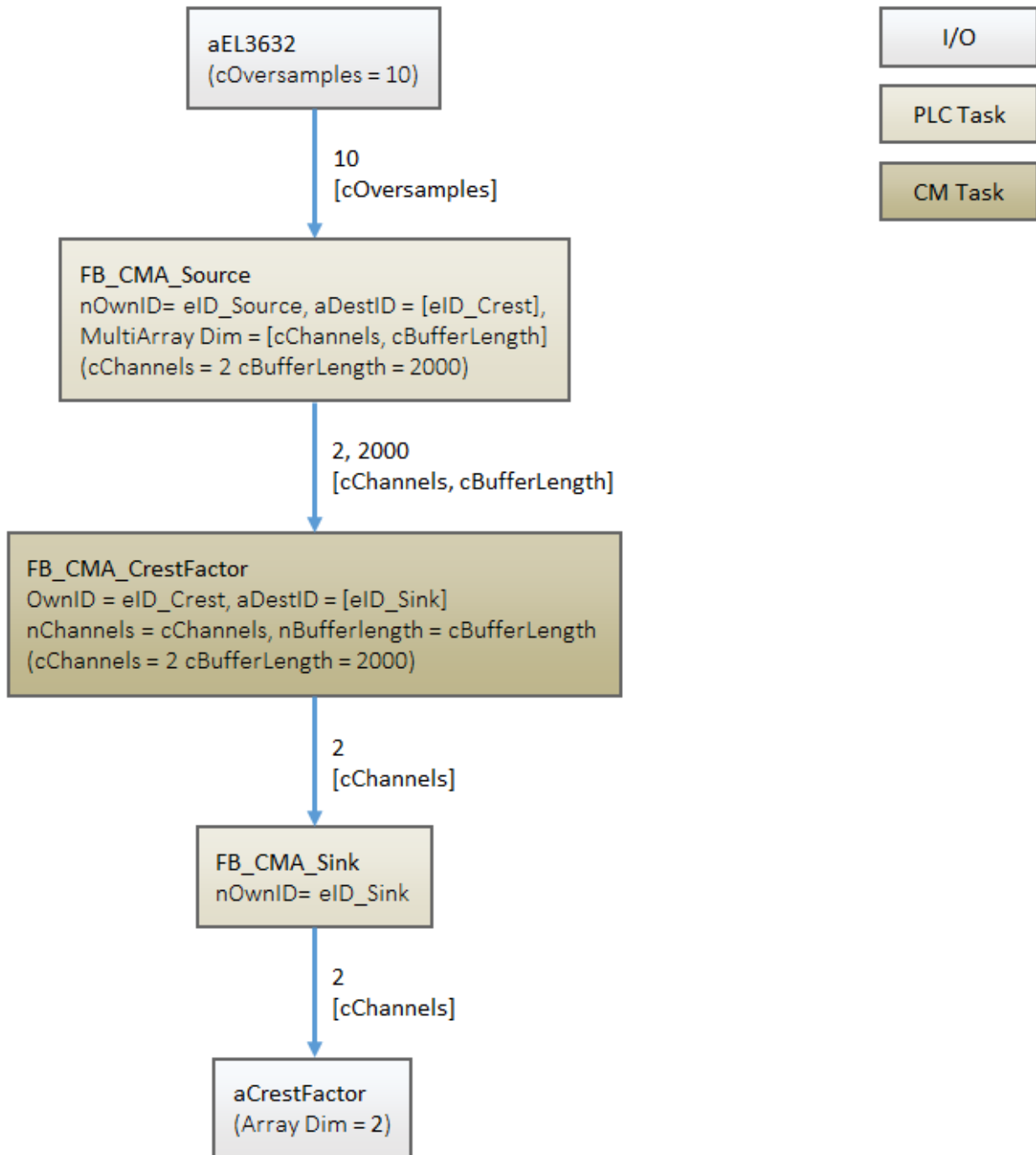


**Program parameters**

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 |

# 6.18 Envelope spectrum

The sample illustrates a calculation implementation for an envelope spectrum with the function block FB_CMA_EnvelopeSpectrum [▶ 147]. 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:
https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/3394494219/.zip

**Block diagram**
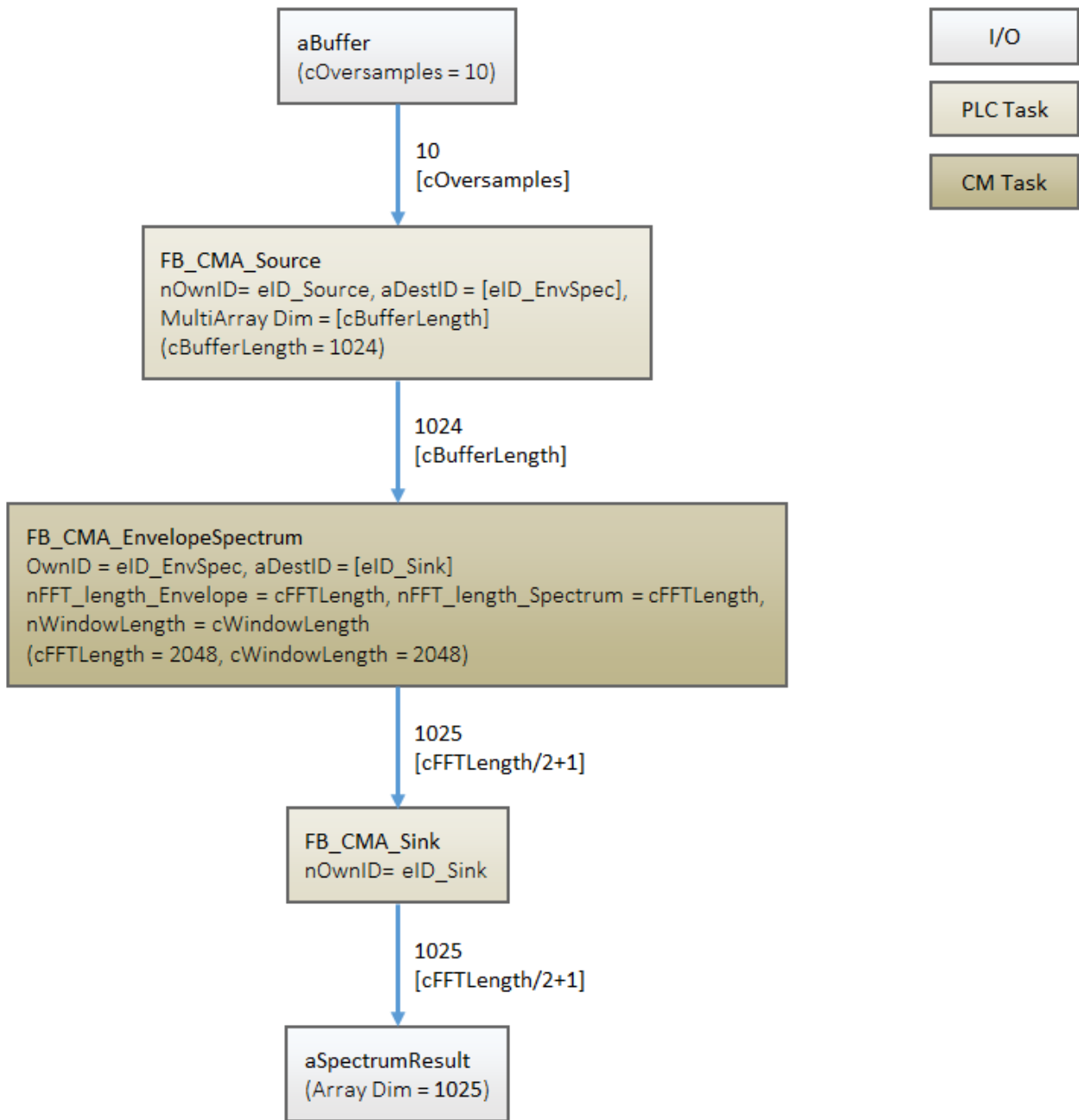


**Program parameters**

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 |

# 6.19   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:

https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/5261531147/.zip

**Block diagram**
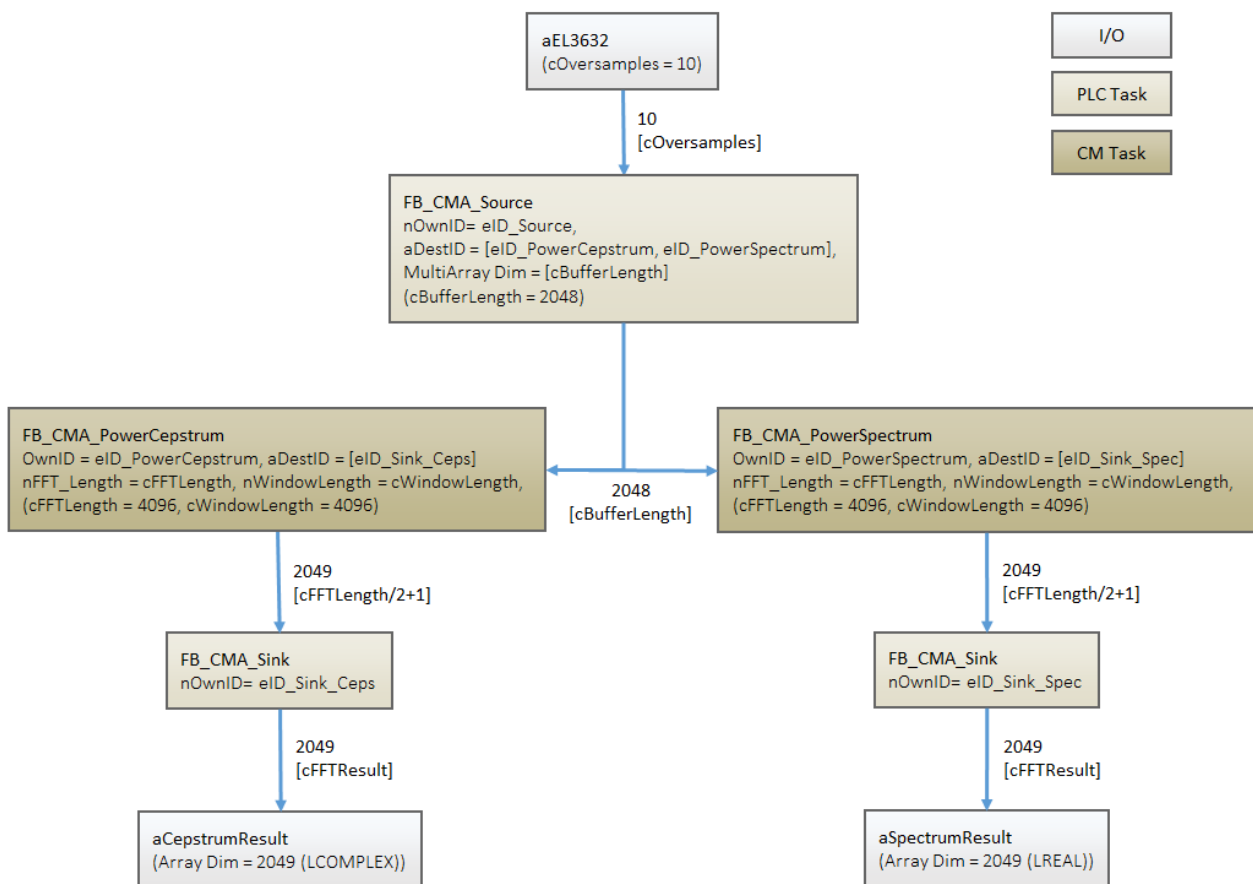


**Program parameters**

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 |

# 6.20 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 [▶ 238] to the function block FB_CMA_MagnitudeSpectrum [▶ 168].

The source code for the sample is available for download from here:

https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/5261425419/.zip

**Block diagram**
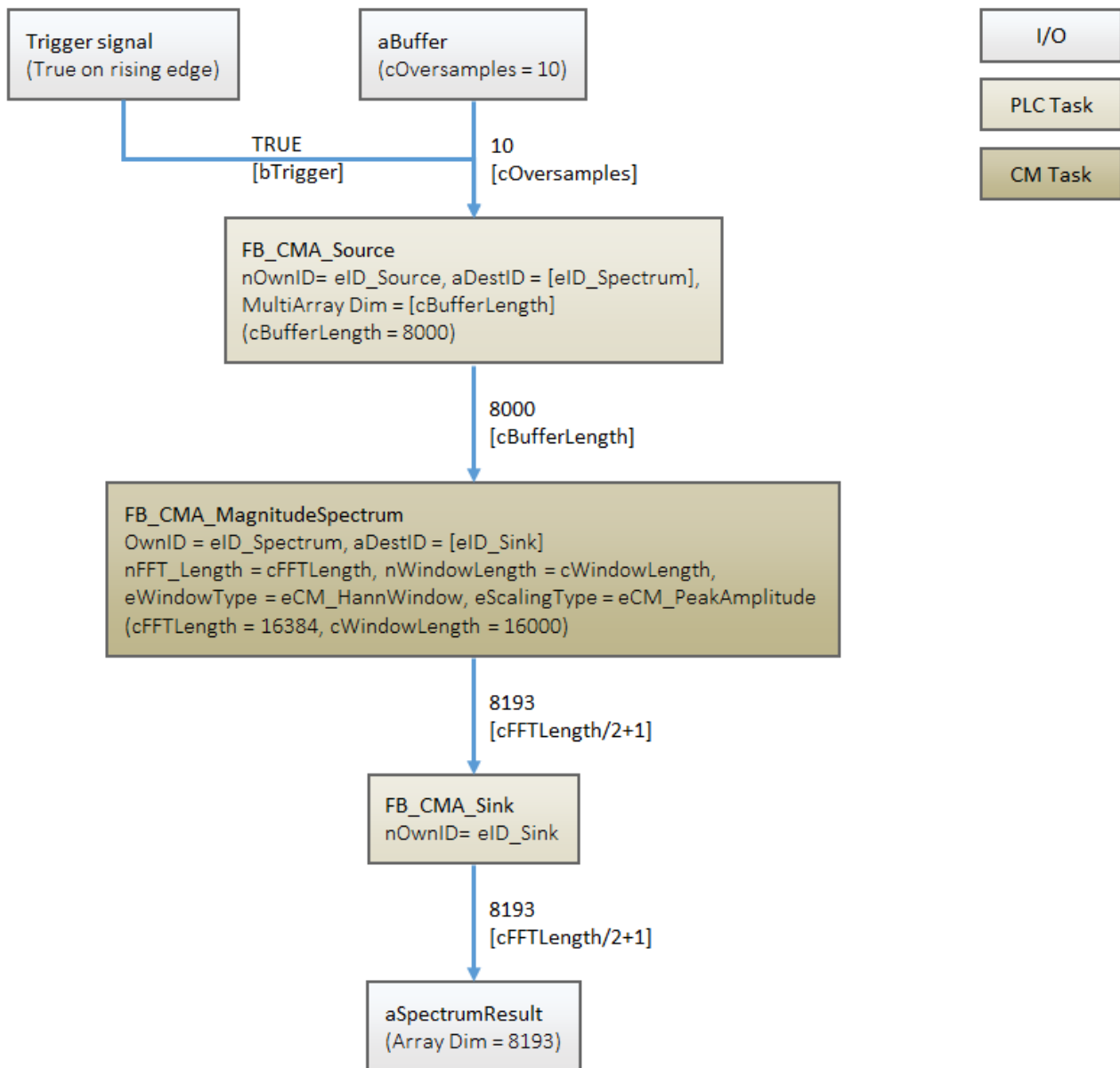
**Program parameters**

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  := 0);

        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: [▶ 303] 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 |

# 6.21  Order analysis

In this sample implementation of the FB_CMA_OrderPowerSpectrum [▶ 189] a simulated NC axis and a synthetic vibration signal are used to illustrate the parameterization of the OrderPowerSpectrum as an example and to show the difference between it and the ordinary PowerSpectrum.

The TwinCAT Solution uses a 500 µs cycle time for the NC. It is therefore preferable to use a Beckhoff IPC or Embedded PC as a target system. It is better to turn off Autostart Boot Project, especially on third-party PCs.

The code can be downloaded here: https://infosys.beckhoff.com/content/1033/ TF3600_TC3_Condition_Monitoring/Resources/9846479115/.zip

**Block diagram**
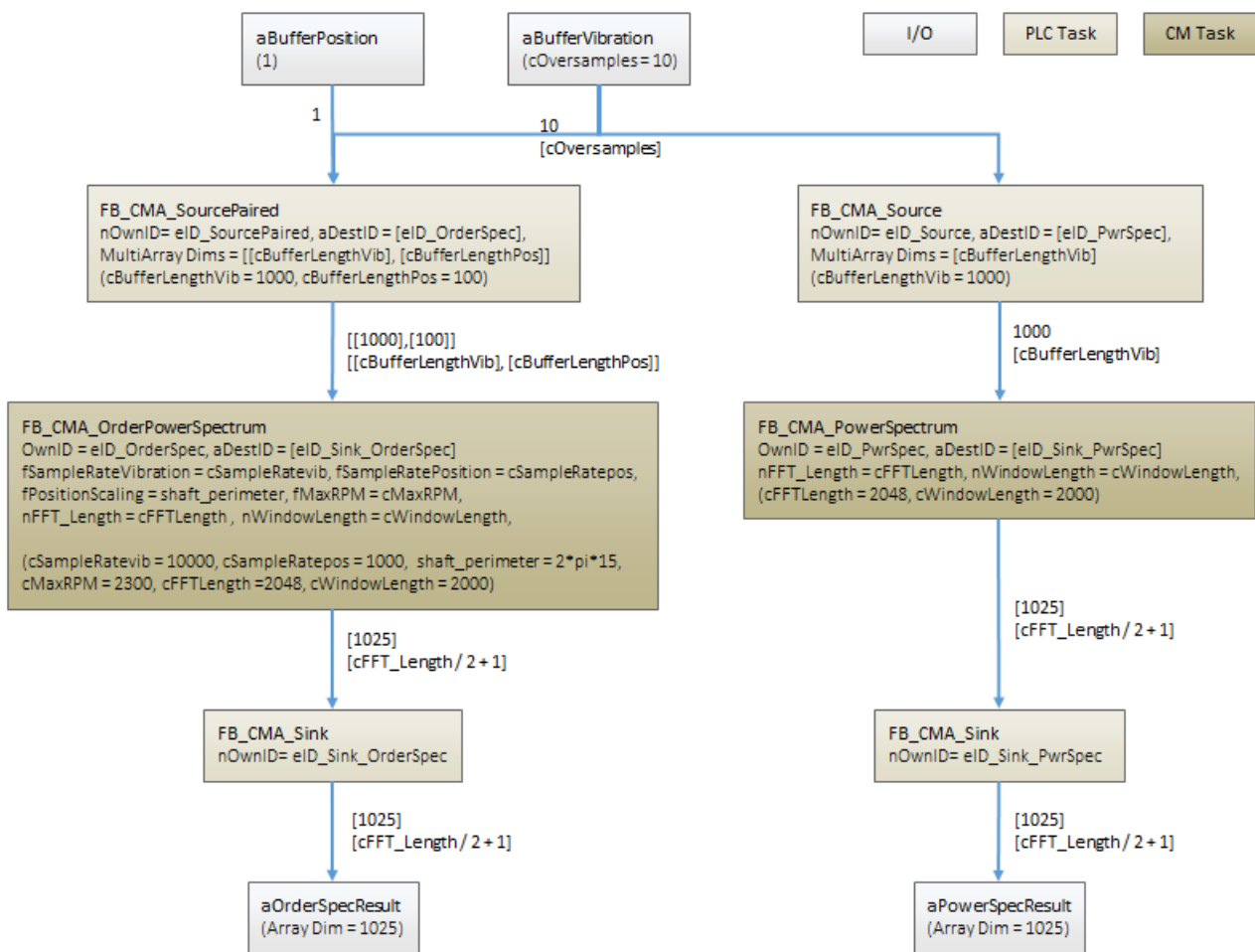


**Program parameters**

The table below shows a list of the important parameters for the configuration of the order spectrum, cf. GVL_constants.

The artificial vibration signal is generated in the PLC task (1 ms cycle time) with an oversampling factor of 10. This corresponds to a sampling rate of 10 kHz, cf. `cSampleRateVib`. The position signal is queried from the PLC task at 1 ms from the NC via the axis interface. This corresponds to a sampling rate of 1 kHz, cf. `cSampleRatePos`. The maximum speed of rotation is `cMaxRPM`.

This results in a maximally resolvable order 130.4.

The resolution of the order axis is determined by the FFT length, in this case 2048, resulting in a resolution of 0.1274. This value can be found in the sample program in the TwinCAT Scope on the symbol aOrderSpecResult under Scalefactor (i), in order to scale the representation of the array of bins to the order.

The window length of the vibration signal `cWindowLength` describes the signal length, which is used for transformation to the frequency range. 2000 samples have been chosen. The FFT length is to be set larger than the window length and as a power of 2. For the transformation, a Hanning window is configured with standard overlap, so that the input buffer is defined as `cWindowLength/2`, i.e. 1000. In the time in which the vibration buffer has 1000 elements, the position buffer contains 100 elements.

| `cSampleRateVib` | 10000 |
|---|---|
| `cSampleRatePos` | 1000 |
| `cShaftPerimeter` | 2 * PI * 15 |
| `cFFTLength` | 2048 |
| `cWindowLength` | 2000 |
| `cBufferLengthVib` | 1000 |
| `cBufferLengthPos` | 100 |
| `cMaxRPM` | 2300 |

**Configuration**

The source function block is only called if the speed of the shaft lies between a minimum and a maximum value. Below and above, the analysis chain is reset and there is a pause until the shaft velocity moves back into a valid range.

```
IF ABS(motorspeed) >= cMinRPM / 60 AND ABS(motorspeed) <= cMaxRPM / 60 THEN
```

A synthetic signal is generated, which has a defined amplitude at defined orders. The orders from the array are displayed in the scope with markers for orientation.

The virtual axis automatically changes its speed of rotation, cf. `nmovestate` in the MAIN (PRG). In the TwinCAT Scope, the results of the OrderPowerSpectrum and the ordinary PowerSpectrum are shown on top of each other. Below that the time curve of the speed of rotation is plotted for orientation. As shown in the graph below, the synthetically generated amplitudes are seen precisely at the intended orders (cf. MAIN. Orders) in the order spectrum as sharp peaks. In the PowerSpectrum, the amplitudes can only be recognized as fuzzy peaks depending on the speed of rotation of the shaft at a different frequency and in the range of changing speed.
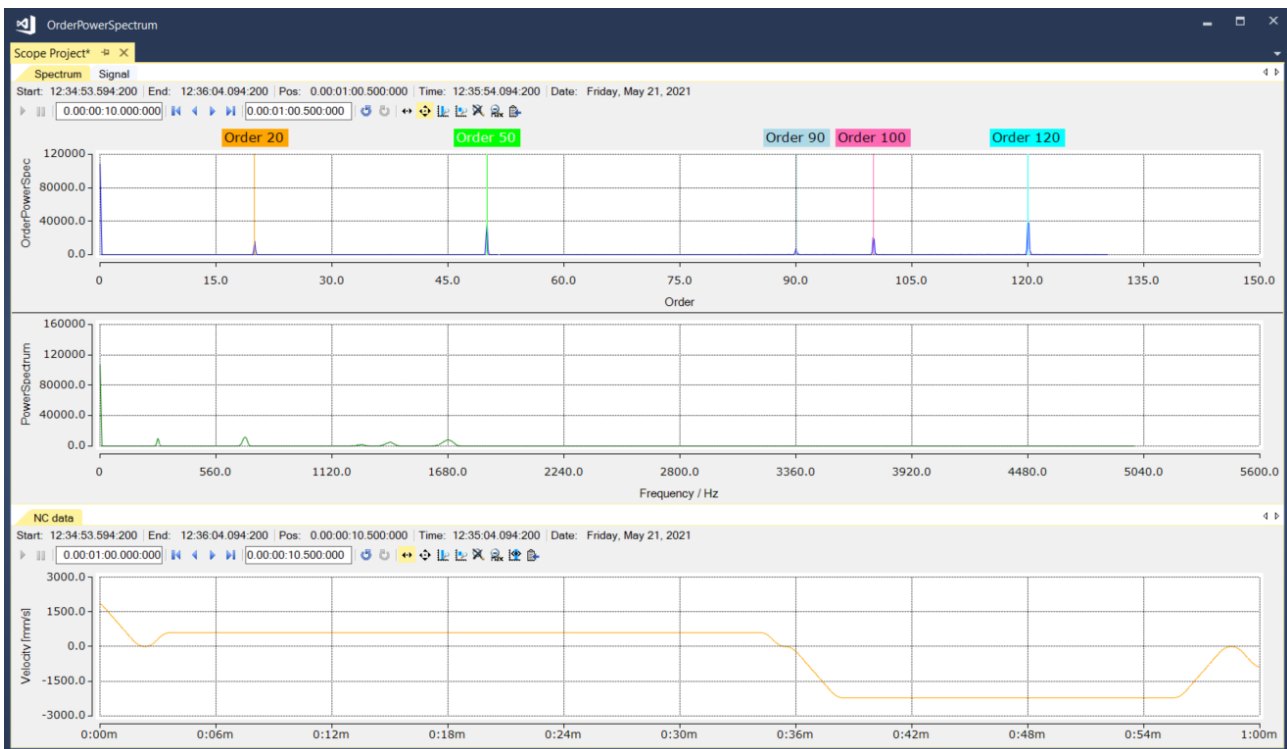
**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base, Tc3_MultiArray |

# 6.22  Fatigue Analysis

This sample explains the functional mode and application possibilities of the function blocks FB_CMA_RainflowCounting [▶ 206] FB_CMA_MeanStressCorrection [▶ 171] and FB_CMA_MinersRule [▶ 175] for estimating the fatigue process of monitored components.

For the calculation of the Wöhler curve, the function F_CM_CalculateWoehlerCurve [▶ 263] is executed on the basis of fictitious material parameters.

Further explanations can be found under the item Application concepts [▶ 33] in the section Fatigue life analysis and damage calculation [▶ 51].

The source code for this sample can be downloaded here: https://infosys.beckhoff.com/content/1033/ TF3600_TC3_Condition_Monitoring/Resources/9785658251/.zip

## Block diagram
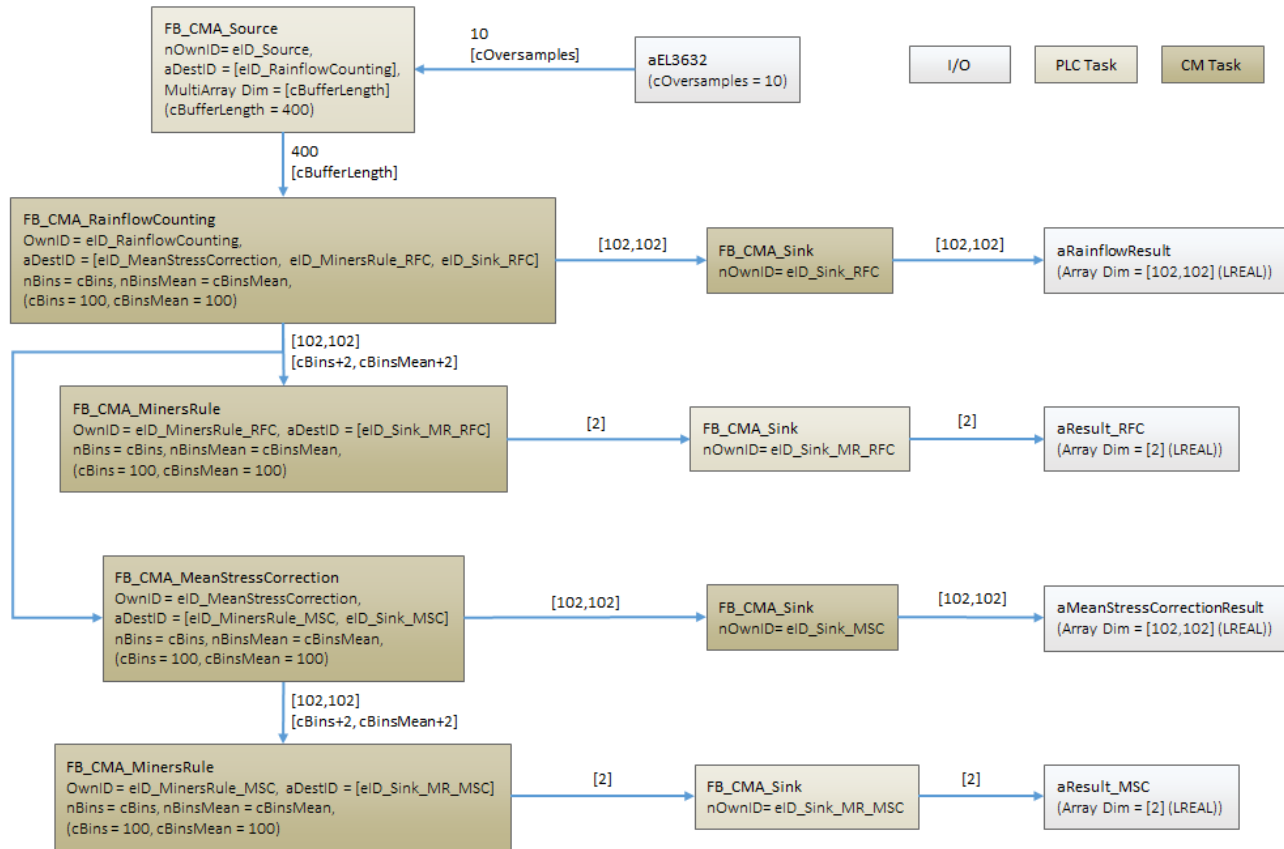


## Program parameters

The table below shows a list with the important parameters (GVL_Constants) for the configuration of the function blocks.

First, the expected measuring range for mechanical stresses is defined by `cStressMin` and `cStressMax`. It is important that all stresses occurring during monitoring lie within this interval. The resolution with which the stress ranges of the stress cycles are then counted is defined by `cRfRangeBins` and `cRfMeanBins`. By default, a minimum of 64 x 64 would be set here, usually 128 x 128. Further constants for parameterizing the function blocks are then defined from these four values.

In the sample code, the influence of mean stress correction is taken into account and can be compared with the calculation without mean stress correction in the TwinCAT Scope. The sample uses the correction according to Goodman, but can be switched to the correction according to Gerber or disabled via `cMSCType`.

Furthermore, material parameters are specified for the definition of the material-specific Wöhler curve used. The parameters used here are fictitious and selected in such a way that a visual effect can be observed in the TwinCAT Scope after a short time.

| `cStressMin` | -50 MPa | Minimum occurring stress |
|---|---|---|
| `cStressMax` | 50 MPa | Maximum occurring stress |
| `cRfRangeBins` | 100 | Number of bins in the stress range axis |
| `cRfMeanBins` | 100 | Number of bins in the mean stress axis |
| `cMSCType` | eCM_Goodman | Mean stress correction type |
| `cSRI` | 350 | Stress Range Intercept (stress intersection with cycle number 1) |
| `cUTS` | 700 | Ultimate Tensile Strength |
| `cK1` | 3 | Slope of the Wöhler curve between N = 1 and cNC1 |
| `cK2` | 5 | Slope of the Wöhler curve starting from point cNC2 |
| `cNC1` | 100 | Transition point for UTS correction |
| `cNC2` | 100000 | Fatigue Transition Point |

**Explanations**

After activating the TwinCAT configuration and starting the TwinCAT Scope, several representations can be seen.

At the bottom on the right-hand side you can see a static graphic, which represents the Wöhler curve. This representation takes place from the parameterization in the GVL_Constant.

The synthetically generated stress signal is plotted over time in black on the left side. In this example, the stress is always given in megapascals (MPa). In the graphic below that, the progress of the sum damage is displayed, where green stands for the damage sum with activated mean stress correction and blue for the damage sum with deactivated mean stress correction. The progress of the damage grows from zero, with one representing the maximum sum damage (point of computational component failure) using the material parameters provided. Accordingly, 3 horizontal lines are drawn at 70%, 90% and 100%, which could be used as warning and alarm thresholds. The bottom graph indicates the estimated remaining fatigue life in seconds. Accordingly, these two curves fall as the simulation continues.
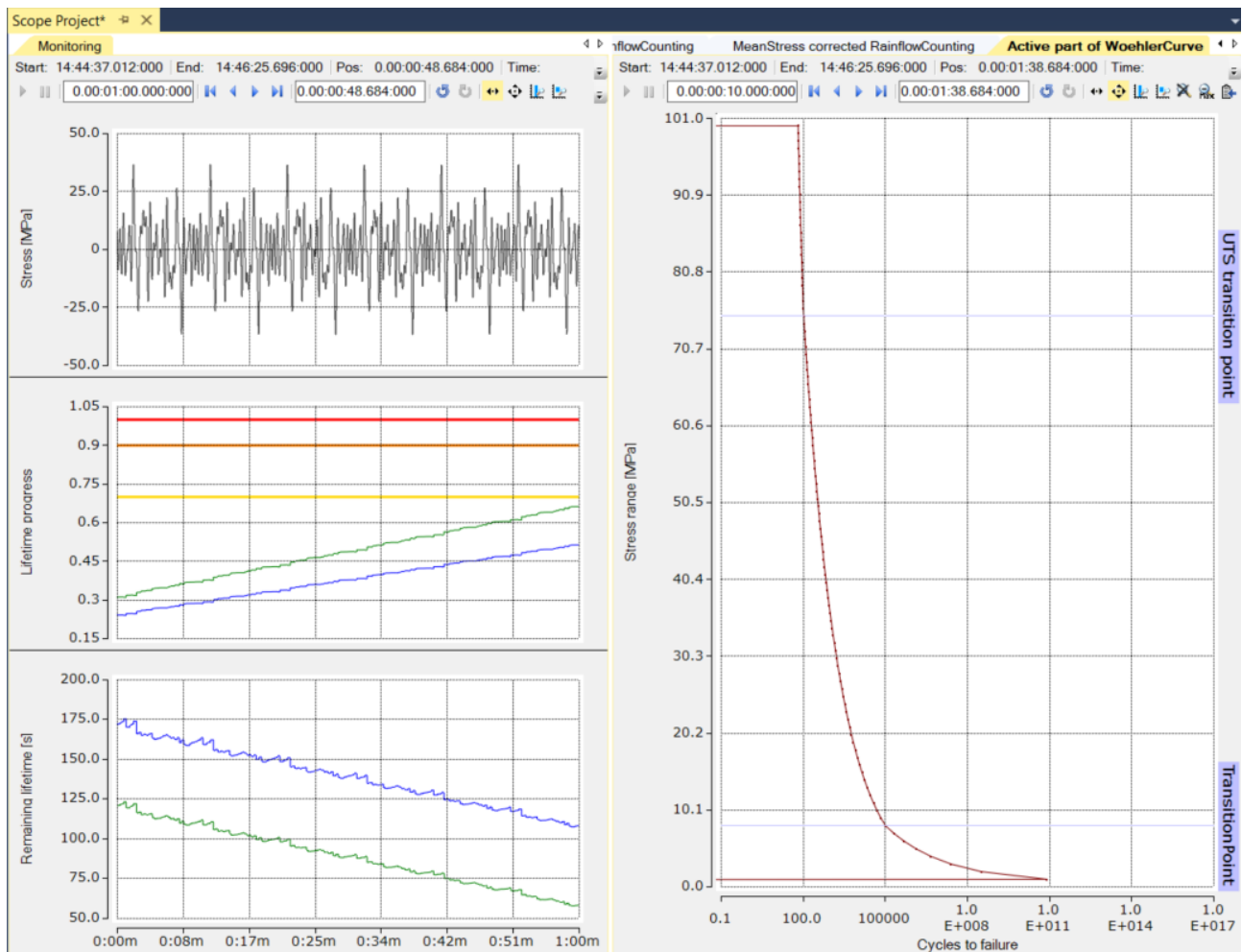


Likewise, the already counted half-cycles of the Rainflow Counting are shown in array bar charts, again in green with mean stress correction and in blue without mean stress correction. The higher damage caused by the correction of the stress ranges with activated mean stress correction can be clearly observed.

In the left part of the graphic you can see that the mean stress correction makes the mean stress axis obsolete (sets it to zero) and impinges on the stress range as a correction. For this reason, half-cycle counts on the left are offset slightly to the right on the stress range axis.
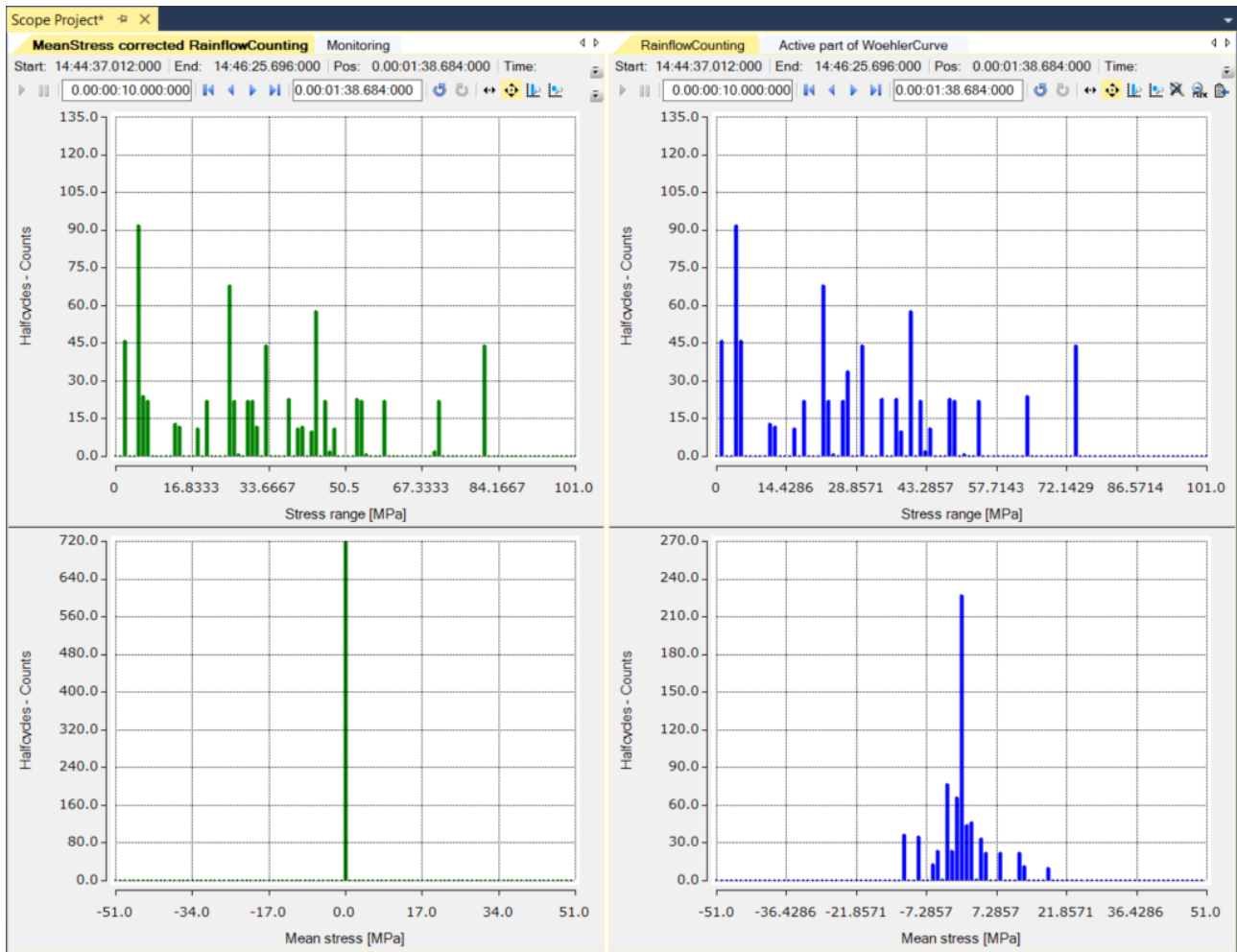
**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 6.23   Spike Energy Spectrum

This sample implements a way to analyze bearing damage by looking at high-frequency components in the vibration data. Due to the sensitivity of the peak energy in comparison with a real setup, the energy of the simulated shocks was chosen to be sufficiently large. In use, trends in the peak value spectrum over time should be considered.

The source code for this sample can be downloaded here: https://infosys.beckhoff.com/content/1033/ TF3600_TC3_Condition_Monitoring/Resources/9783190923/.zip

**Block diagram**

```
                    ┌──────────────────────────┐          ┌──────────┐
                    │ aEL3632                  │          │   I/O    │
                    │ (cOversamples = 10)      │          └──────────┘
                    └──────────────────────────┘
                                │                         ┌──────────┐
                              10                          │ PLC Task │
                        [cOversamples]                    └──────────┘
                                │
                                ▼                         ┌──────────┐
        ┌────────────────────────────────────────┐       │ CM Task  │
        │ FB_CMA_Source                          │       └──────────┘
        │ nOwnID= eID_Source, aDestID = [eID_SESpectrum],
        │ MultiArray Dim = [cBufferLength]       │
        │ (cBufferLength = 400)                  │
        └────────────────────────────────────────┘
                                │
                              400
                        [cBufferLength]
                                │
                                ▼
    ┌──────────────────────────────────────────────────────┐
    │ FB_CMA_SpikeEnergySpectrum                           │
    │ OwnID = eID_Spectrum, aDestID = [eID_Sink]           │
    │ nFFT_Length = cFFTLength, nWindowLength = cWindowLength,
    │ fDecayTime = 0,0093, fLowerFrequencyLimit = 1000,    │
    │ fUpperFrequencyLimit = 4950                          │
    │ eWindowType = eCM_HannWindow, eScalingType = eCM_RootPowerSum
    │ (cFFTLength = 1024, cWindowLength = 800)             │
    └──────────────────────────────────────────────────────┘
                                │
                              513
                        [cFFTLength/2+1]
                                │
                                ▼
                    ┌──────────────────────────┐
                    │ FB_CMA_Sink              │
                    │ nOwnID= eID_Sink         │
                    └──────────────────────────┘
                                │
                              513
                        [cFFTLength/2+1]
                                │
                                ▼
                    ┌──────────────────────────┐
                    │ aSESpectrumResult        │
                    │ (Array Dim = 513)        │
                    └──────────────────────────┘
```

**Program parameters**

The table below shows a list of the important parameters for configuring the function blocks.

| | |
|---|---|
| FFT-length | 1024 |
| Window size | 800 |
| Buffer size | 400 |
| Length of the peak value spectrum | 513 |
| Lower limit of the bandpass filter [Hz] | 1000 |
| Cut-off frequency of the bandpass filter [Hz] | 4950 |

## Explanations

The generated amplitude-modulated input signal is composed as follows: A shock (yellow) is modulated with a frequency of 200 Hz onto a fundamental oscillation (blue). The input signal of the function block thus corresponds to the curve below (green).
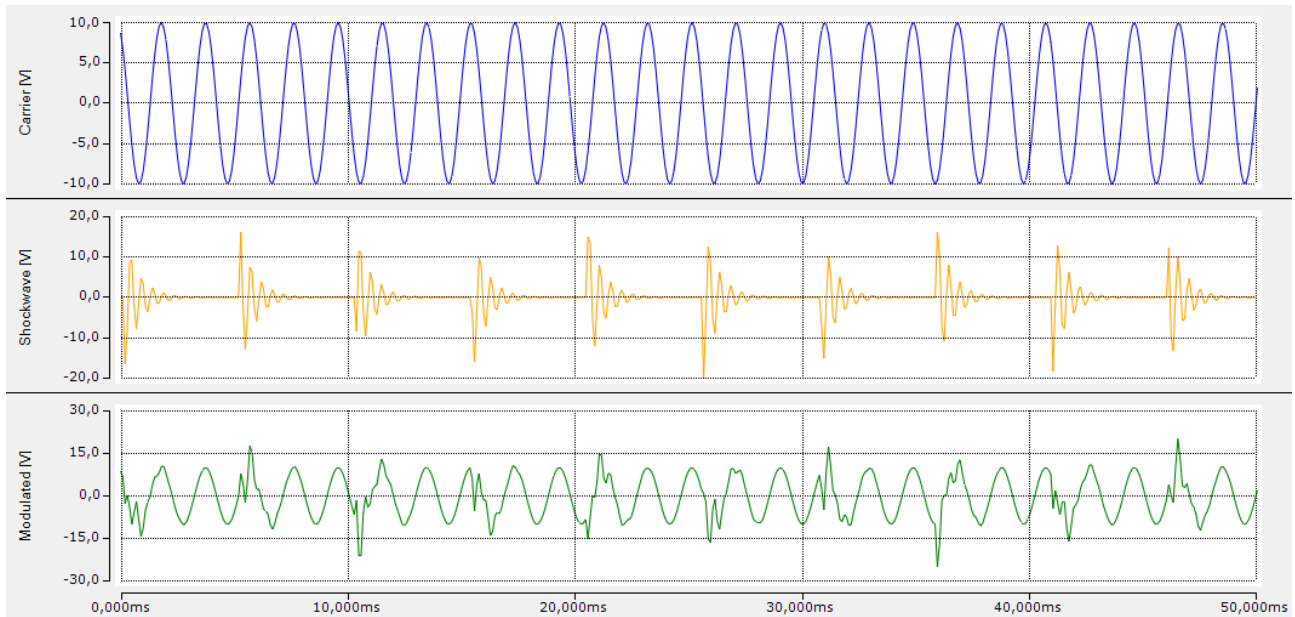


The input data are processed as follows: The signal components that are not relevant for the analysis (below 1 kHz) are filtered out by means of an IIR bandpass filter. In the case of exact arithmetic, this would correspond to the shock (yellow). The filtered signal is now transformed into a time-wave form in which the energy peaks are displayed in relation to the configured decay time (`fDecayTime`) ("peak to peak"). The following image shows the time-wave shape for the input data described above.
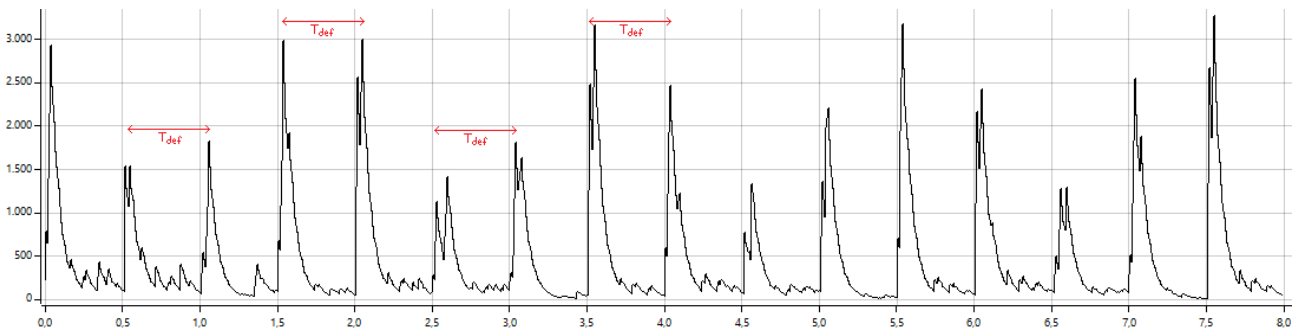


The frequency of the occurring peak values, i.e. the repetition rate of high-frequency components in the form of shocks in the original signal, is already recognizable in this representation. The distance between two consecutive peaks is 0.5 ms, i.e. 1 / 0.005 s = 200 Hz. This periodicity is represented by the calculation of a magnitude spectrum. This leads to the following visualization in the scope project of the sample:
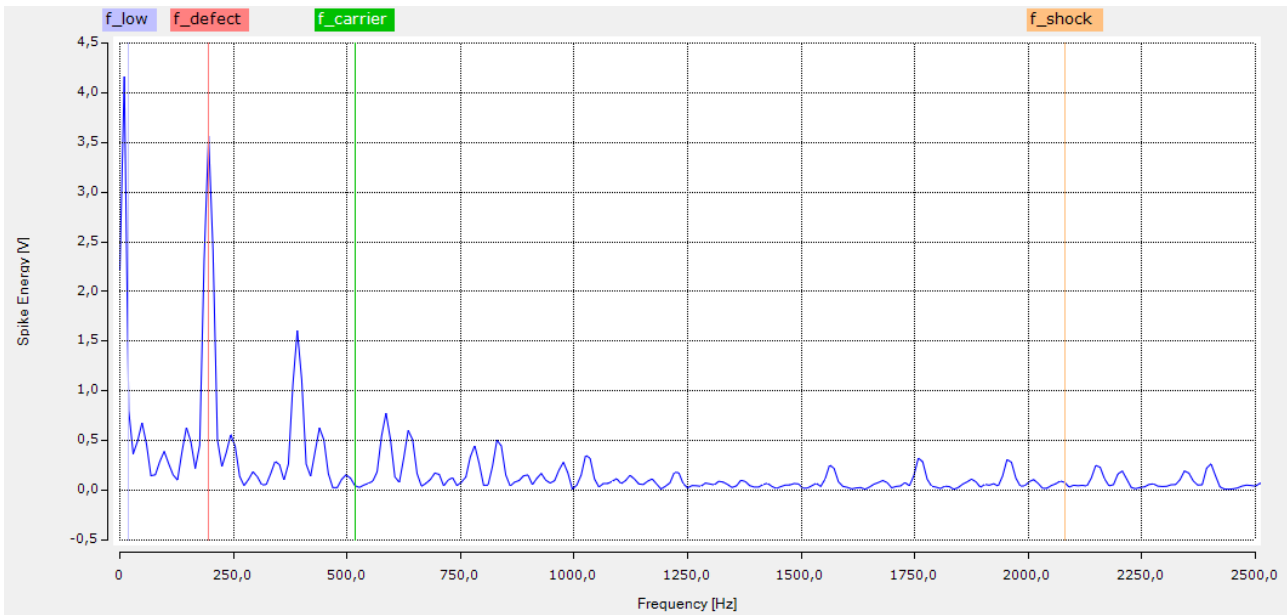
In the resulting peak value spectrum, the component of the error frequency (`f_defect`) at 200 Hz and the associated harmonics is clearly recognizable. Furthermore, the frequency components of the fundamental vibration (`f_carrier`) and those of the shocks (`f_shock`) are eliminated by the bandpass filtering and transformation into the time-wave form.

The peak value below `f_low` is not meaningful because it lies below the resolution limit of `cMinFrequency = cSampleRate/cWindowLength = 12,5` Hz.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 6.24  Zoom FFT

This sample shows examples of the possible uses of the function block FB_CMA_ZoomFFT. Using this function block, the section of a spectrum also calculated with the function block FB_CMA_RealFFT is calculated.

The source code for this sample can be downloaded here: https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/9783192587/.zip

**Block diagram**
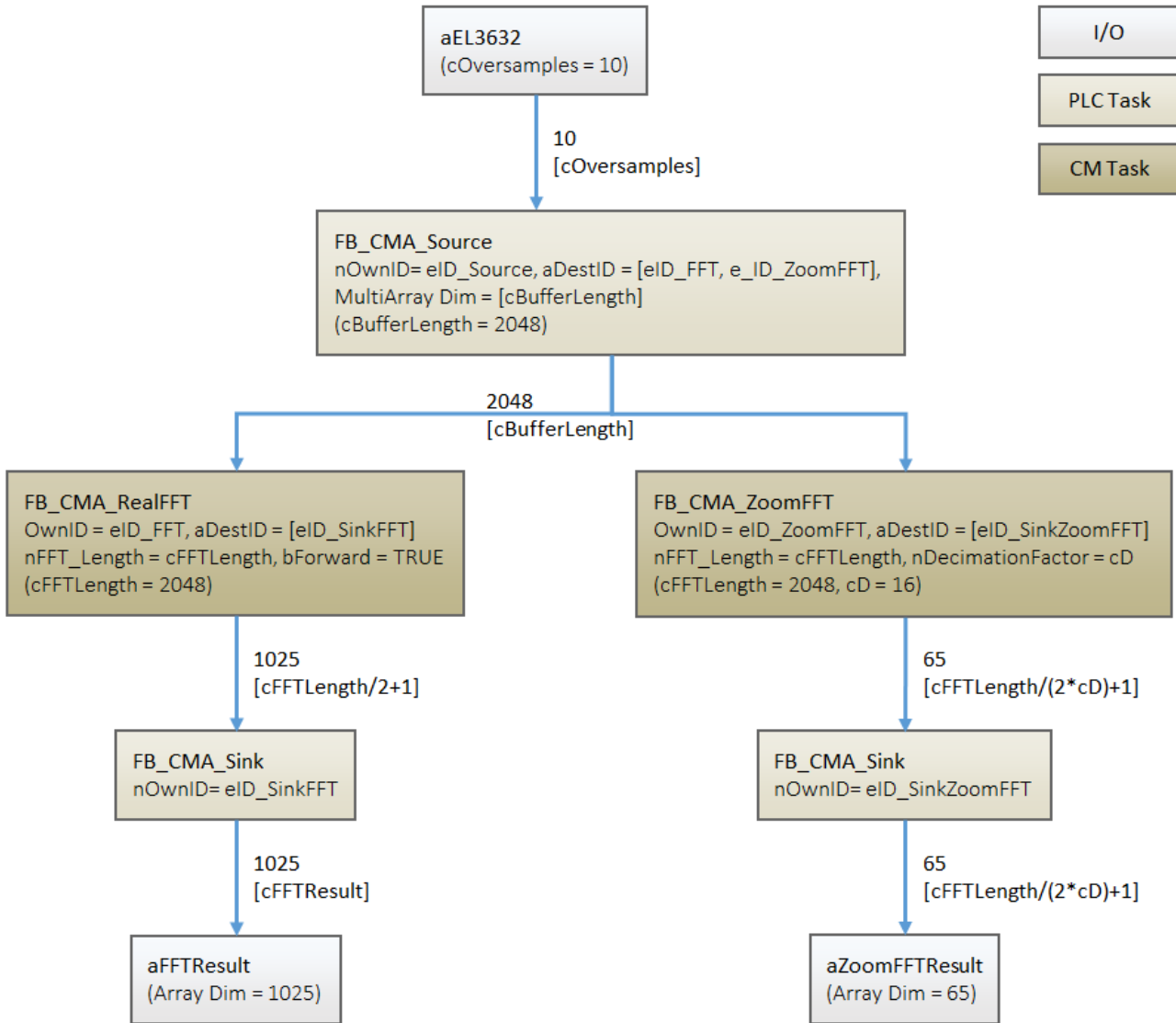


**Program parameters**

The table below shows a list of the important parameters for configuring the function blocks.

| | |
|---|---|
| FFT-length | 2048 / 2048 |
| Buffer size | 2048 / 2048 |
| Decimation factor | 16 / - |
| Length of the calculated spectrum | 65 / 1025 |

**Explanations**

The scope project contained in the sample code visualizes a spectrum calculated by means of the function block FB_CMA_RealFFT in the range of 0 to 1 kHz as well as a section in the range 48.83 Hz and 361.33 Hz, which was calculated with the function block FB_CMA_ZoomFFT. Here, the resolution in the frequency range is identical (4.8828125 Hz), where for the entire spectrum an FFT of length 2048 and in the section via the zoom FFT an FFT of length 128 is calculated on the decimated time signal.
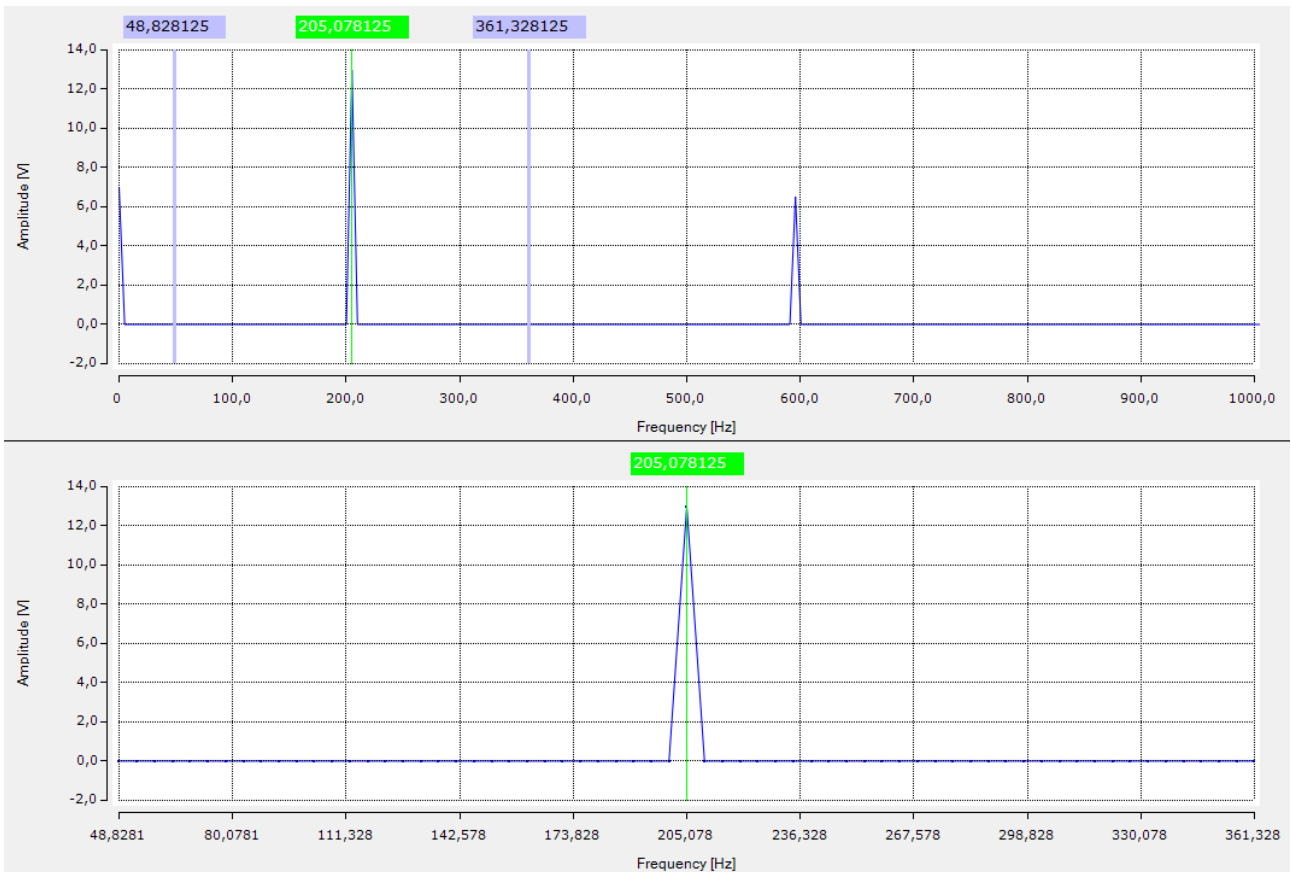
**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 6.25  Sliding DFT

This sample implements examples of the possible uses of the function block FB_CMA_SlidingDFT, where the function block is configured in such a way that all spectral values are calculated and the results are compared to those from the calculation using the function block FB_CMA_RealFFT. Furthermore, to improve the calculated spectrum, a Hanning window is applied in the spectral range.

The source code for this sample can be downloaded here: https://infosys.beckhoff.com/content/1033/ TF3600_TC3_Condition_Monitoring/Resources/9783189259/.zip

**Block diagram**
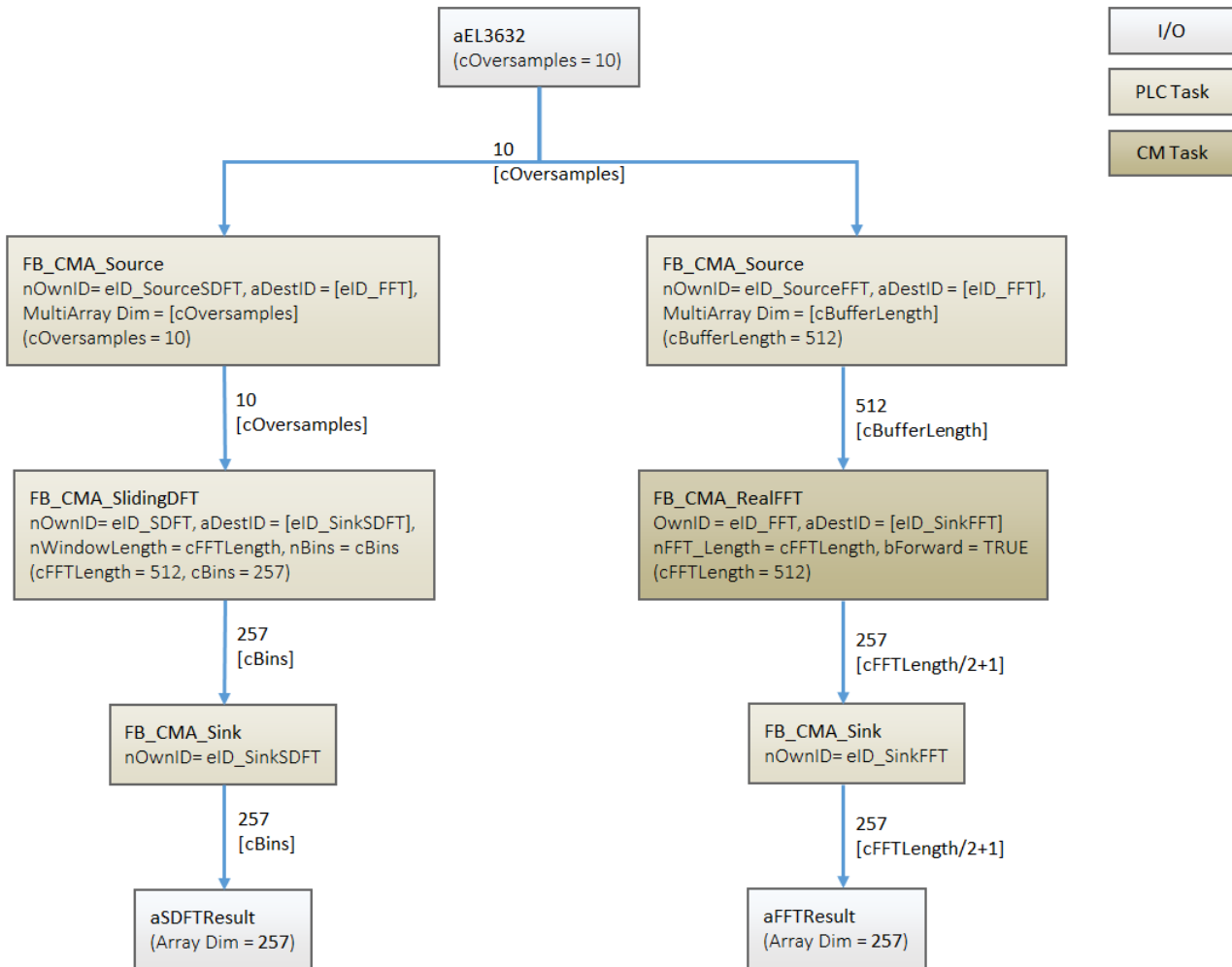


**Program parameters**

The table below shows a list of the important parameters for configuring the function blocks.

| FFT-length | 512 / 512 |
|---|---|
| Buffer size | 10 / 512 |
| FFT result length | 257 / 257 |

**Explanations**

The scope project included in the sample visualizes the calculated spectra over their magnitude in the range of 0 to 1 kHz. On the one hand, the influence of the side lobes in the spectrum of the function block FB_CMA_SlidingDFT (blue line) is noticeable. The deviations essentially depend on the chosen damping parameter `fDampingFactor`. The deviations become smaller with values above `0.995`, but the calculation becomes unstable with too little attenuation `(>0.999)`. Due to the definition of the recursion rule for the spectral line `k = 0`, the calculation for the DC component, on the other hand, is more accurate. An improvement of the calculated values can be achieved by applying the Hanning window in the spectral range (fig. below).
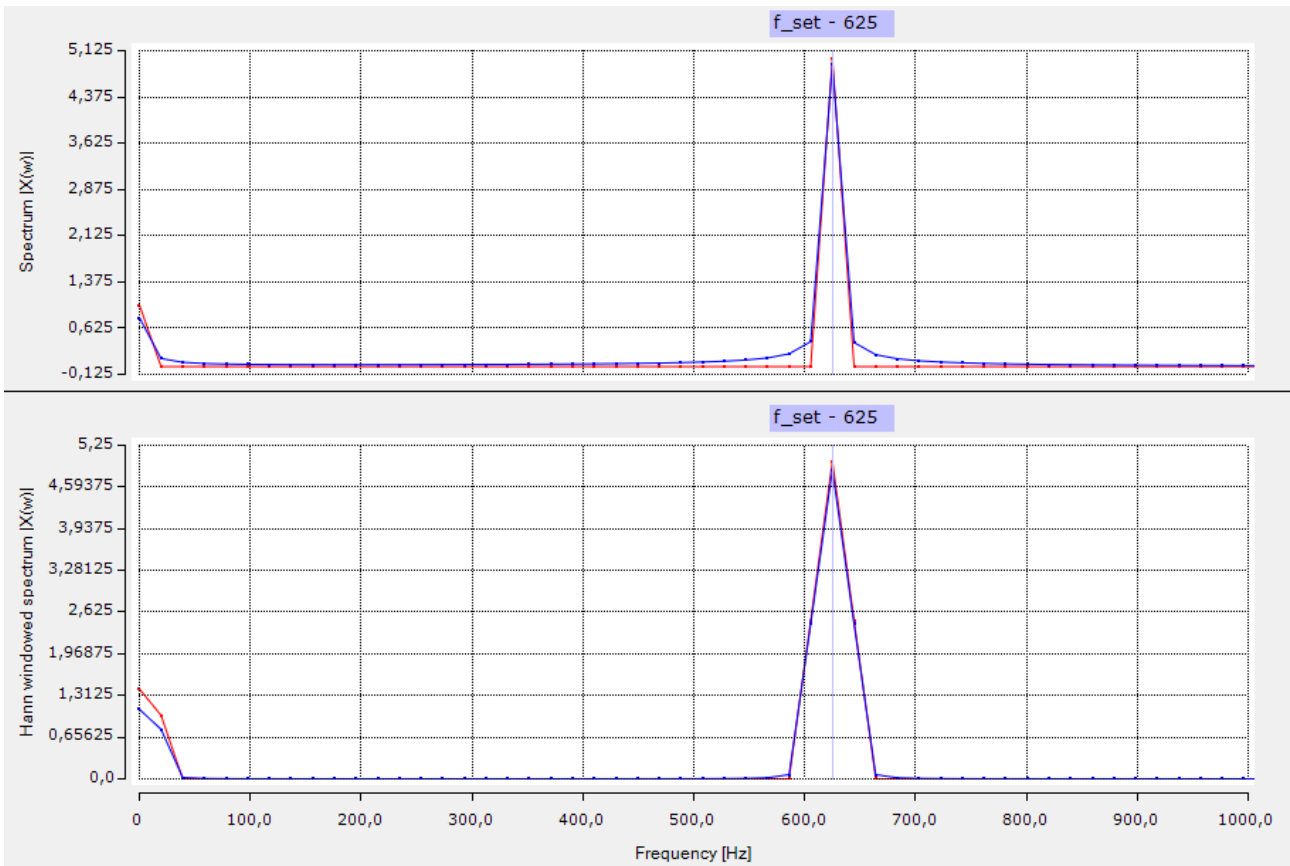
A significant difference between the two function blocks is the utilization in the respective task. This is illustrated in the following graphic.
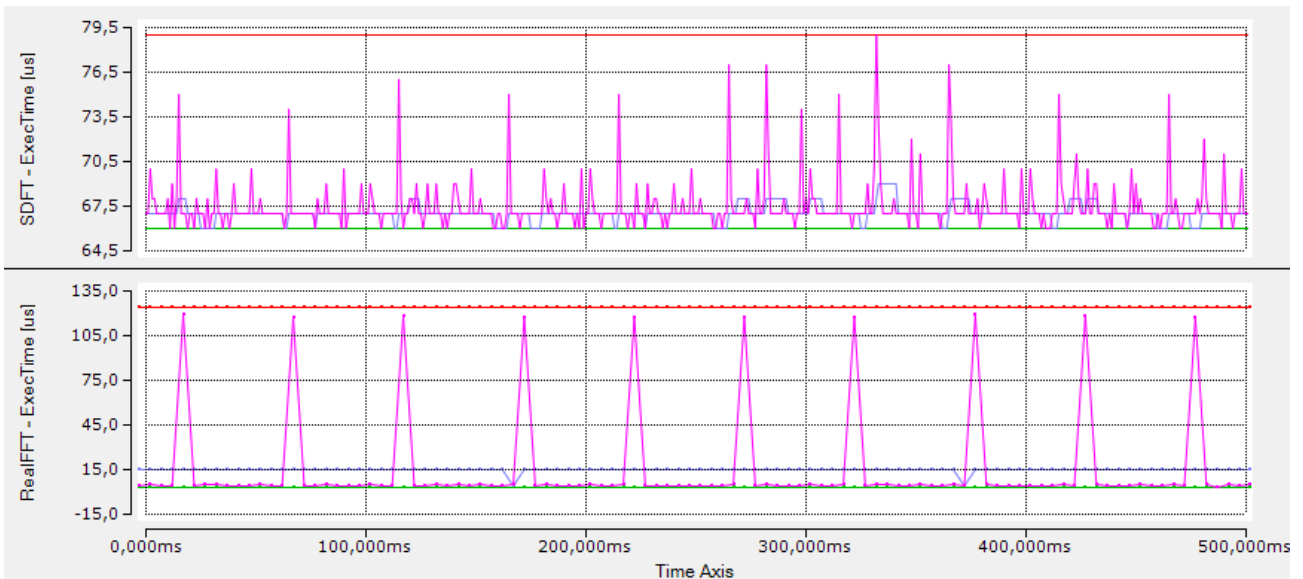


In contrast to the utilization peaks of the RealFFT (fig. below), the utilization of the SDFT (fig. above) is more constant. The latter value is significantly higher on average, but is subject to fewer fluctuations.

**Requirements**

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

## 6.26 Correlation function

This sample implements examples of the possible uses of the function block FB_CMA_Correlation. Here, the correlation coefficients are calculated for a temporal offset of +/- 100 samples with respect to a reference signal.

The source code for this sample can be downloaded here: https://infosys.beckhoff.com/content/1033/TF3600_TC3_Condition_Monitoring/Resources/9833229195/.zip

**Block diagram**
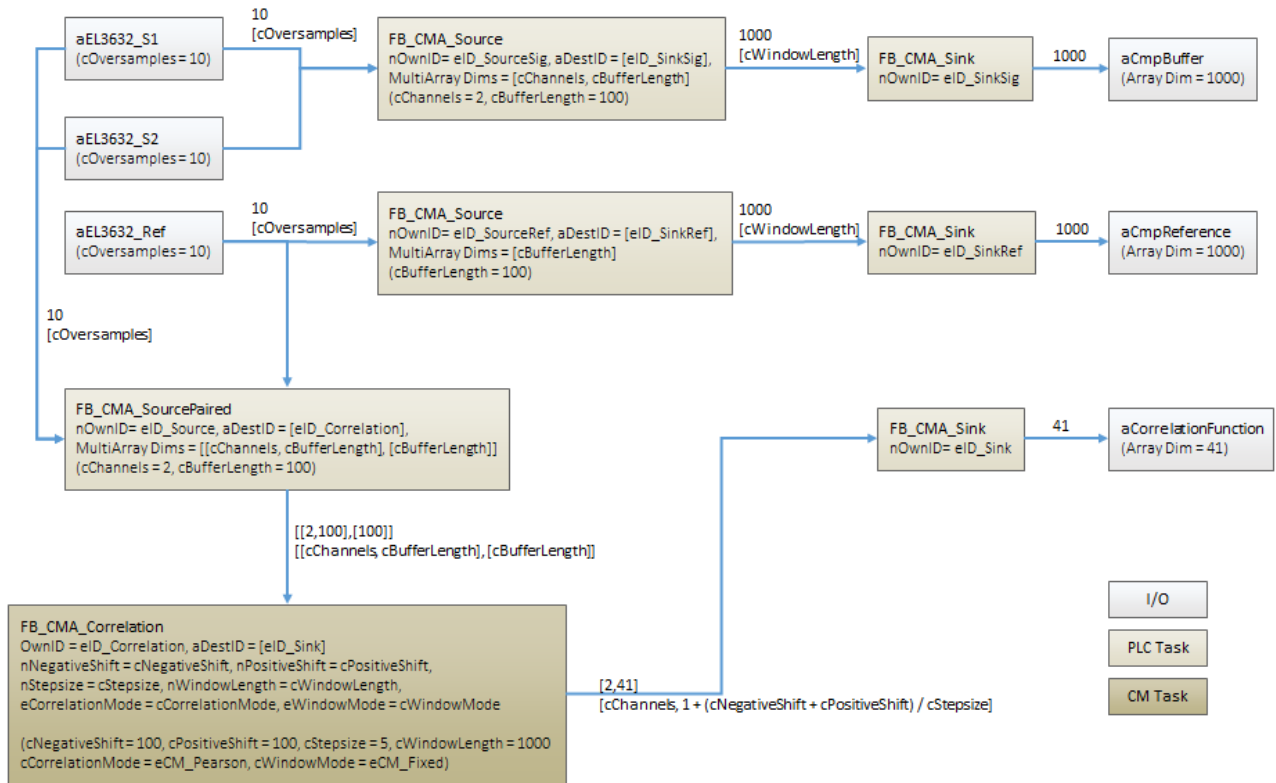


**Program parameters**

The table below shows a list of the important parameters for configuring the function blocks.

| nWindowLength | 1000 |
|---|---|
| nNegativeShift | 100 |
| nPositiveShift | 100 |
| nStepsize | 5 |
| eCorrelationMode | eCM_Pearson |
| eWindowMode | eCM_Fixed |

**Explanations**

In this sample, the correlation parameters are calculated between a sine wave with amplitude 1 (red) and a sine wave amplified by +5 ms (50 samples at 10 kHz sampling rate) (blue) and a triangular pulse (green) with an identical frequency (50 Hz) and a delay of -5 ms. The test signals are superimposed with normal noise. The following graphic shows the output signals (above) and the noisy signals (below)
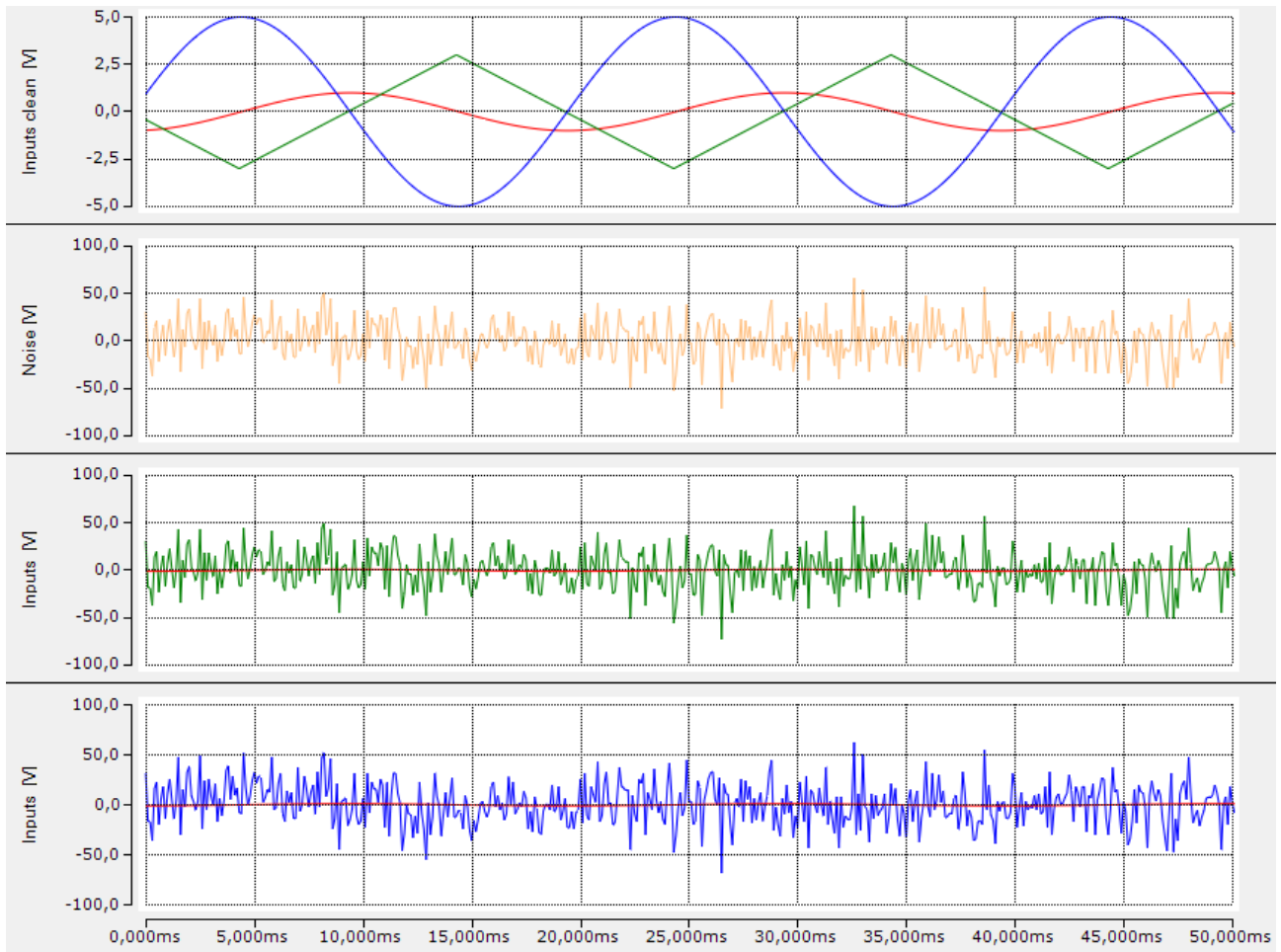
The noise is made so high that the output in the time domain can no longer be identified. However, the calculation of the correlation coefficients allows the classification in relation to the reference signal:
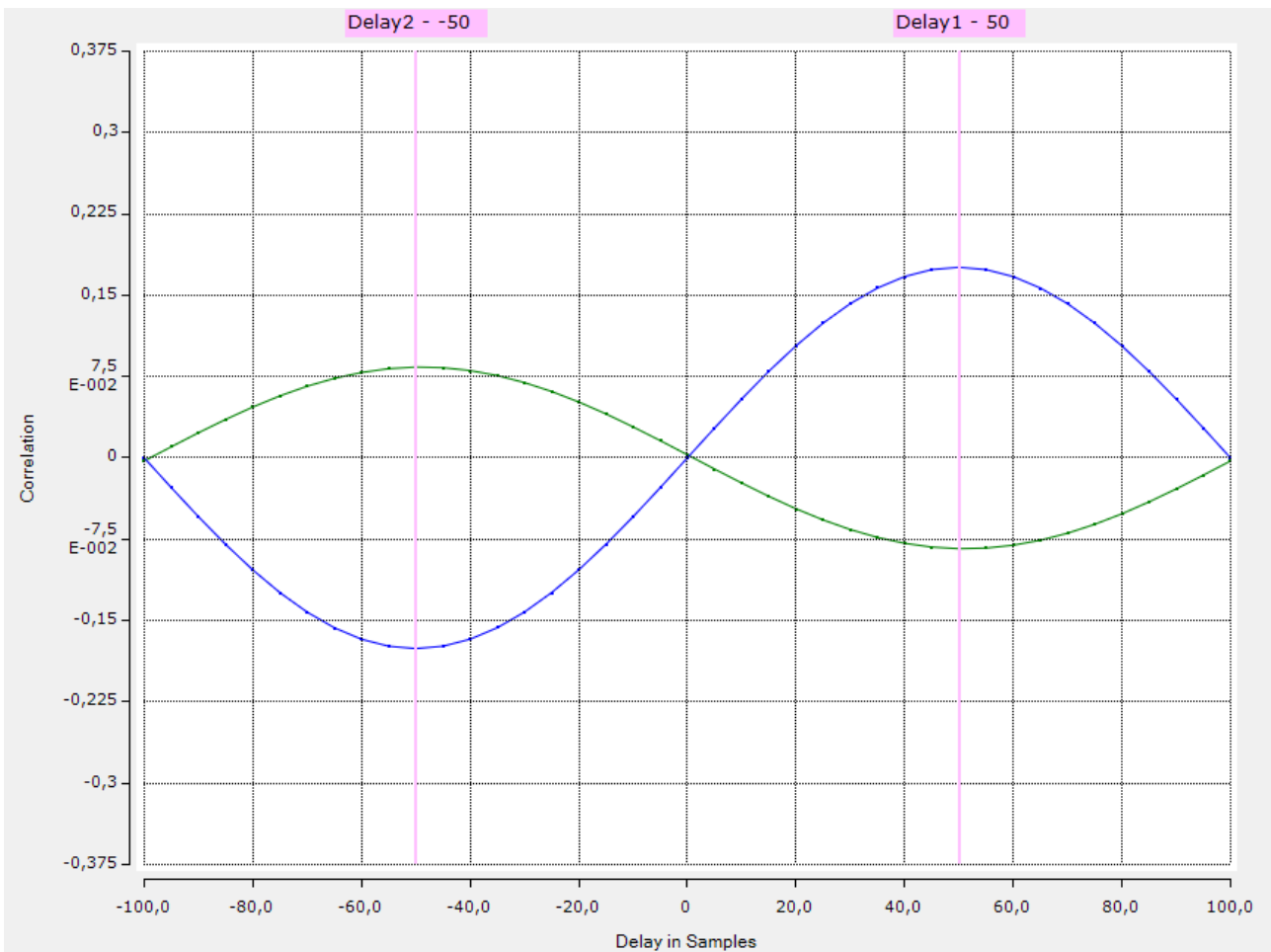
The highest correlation of the amplified sine wave (blue) is at a shift of 50 samples to the reference, i.e. the signal is delayed by 5 ms. Analogously, the triangular pulse (green) is shifted by 50 samples, i.e. it leads the reference by 5 ms.

### Requirements

| Development environment | Target platform | PLC libraries to include |
|---|---|---|
| TwinCAT v3.1.4022.25 | PC or CX (x86, x64) | Tc3_CM, Tc3_CM_Base |

# 7 Appendix

## 7.1 Error Codes Overview

Error codes are returned by type HRESULT. A test for non-zero value is insufficient for values of type HRESULT.

| declaration | error | ok | ok but with info | check functions |
|---|---|---|---|---|
| hrErrorCode :HRESULT; | < 0 | >= 0 | > 0 | SUCCEEDED(), FAILED() |

The following error codes may occur.

| 16#9811_0000 - 16#9811_FFFF | listed in TwinCAT (ADS) Error Codes [▶ 357] (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 [▶ 268] |
| 16#9852_0000 - 16#9852_0FFF | Condition Monitoring Analysis Error Codes are listed in E_CMA_ErrorCodes [▶ 270] |
| 16#9871_0000 - 16#9871_FFFF | MultiArray Error Codes can be found in E_MA_ErrorCode [▶ 271] |

**i** 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 [▶ 71]) |
| 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 [▶ 75]. |

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 [▶ 74]). 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

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 358]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 358]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 359]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 360]... (0x9811_1000 ...)

**BECKHOFF**

## Global error codes

| Hex | Dec | HRESULT | Name | Description |
|-----|-----|---------|------|-------------|
| 0x0 | 0 | 0x98110000 | ERR_NOERROR | No error. |
| 0x1 | 1 | 0x98110001 | ERR_INTERNAL | Internal error. |
| 0x2 | 2 | 0x98110002 | ERR_NORTIME | No real time. |
| 0x3 | 3 | 0x98110003 | ERR_ALLOCLOCKEDMEM | Allocation locked – memory error. |
| 0x4 | 4 | 0x98110004 | ERR_INSERTMAILBOX | Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help. |
| 0x5 | 5 | 0x98110005 | ERR_WRONGRECEIVEHMSG | Wrong HMSG. |
| 0x6 | 6 | 0x98110006 | ERR_TARGETPORTNOTFOUND | Target port not found – ADS server is not started or is not reachable. |
| 0x7 | 7 | 0x98110007 | ERR_TARGETMACHINENOTFOUND | Target computer not found – AMS route was not found. |
| 0x8 | 8 | 0x98110008 | ERR_UNKNOWNCMDID | Unknown command ID. |
| 0x9 | 9 | 0x98110009 | ERR_BADTASKID | Invalid task ID. |
| 0xA | 10 | 0x9811000A | ERR_NOIO | No IO. |
| 0xB | 11 | 0x9811000B | ERR_UNKNOWNAMSCMD | Unknown AMS command. |
| 0xC | 12 | 0x9811000C | ERR_WIN32ERROR | Win32 error. |
| 0xD | 13 | 0x9811000D | ERR_PORTNOTCONNECTED | Port not connected. |
| 0xE | 14 | 0x9811000E | ERR_INVALIDAMSLENGTH | Invalid AMS length. |
| 0xF | 15 | 0x9811000F | ERR_INVALIDAMSNETID | Invalid AMS Net ID. |
| 0x10 | 16 | 0x98110010 | ERR_LOWINSTLEVEL | Installation level is too low –TwinCAT 2 license error. |
| 0x11 | 17 | 0x98110011 | ERR_NODEBUGINTAVAILABLE | No debugging available. |
| 0x12 | 18 | 0x98110012 | ERR_PORTDISABLED | Port disabled – TwinCAT system service not started. |
| 0x13 | 19 | 0x98110013 | ERR_PORTALREADYCONNECTED | Port already connected. |
| 0x14 | 20 | 0x98110014 | ERR_AMSSYNC_W32ERROR | AMS Sync Win32 error. |
| 0x15 | 21 | 0x98110015 | ERR_AMSSYNC_TIMEOUT | AMS Sync Timeout. |
| 0x16 | 22 | 0x98110016 | ERR_AMSSYNC_AMSERROR | AMS Sync error. |
| 0x17 | 23 | 0x98110017 | ERR_AMSSYNC_NOINDEXINMAP | No index map for AMS Sync available. |
| 0x18 | 24 | 0x98110018 | ERR_INVALIDAMSPORT | Invalid AMS port. |
| 0x19 | 25 | 0x98110019 | ERR_NOMEMORY | No memory. |
| 0x1A | 26 | 0x9811001A | ERR_TCPSEND | TCP send error. |
| 0x1B | 27 | 0x9811001B | ERR_HOSTUNREACHABLE | Host unreachable. |
| 0x1C | 28 | 0x9811001C | ERR_INVALIDAMSFRAGMENT | Invalid AMS fragment. |
| 0x1D | 29 | 0x9811001D | ERR_TLSSEND | TLS send error – secure ADS connection failed. |
| 0x1E | 30 | 0x9811001E | ERR_ACCESSDENIED | Access denied – secure ADS access denied. |

## Router error codes

| Hex | Dec | HRESULT | Name | Description |
|-----|-----|---------|------|-------------|
| 0x500 | 1280 | 0x98110500 | ROUTERERR_NOLOCKEDMEMORY | Locked memory cannot be allocated. |
| 0x501 | 1281 | 0x98110501 | ROUTERERR_RESIZEMEMORY | The router memory size could not be changed. |
| 0x502 | 1282 | 0x98110502 | ROUTERERR_MAILBOXFULL | The mailbox has reached the maximum number of possible messages. |
| 0x503 | 1283 | 0x98110503 | ROUTERERR_DEBUGBOXFULL | The Debug mailbox has reached the maximum number of possible messages. |
| 0x504 | 1284 | 0x98110504 | ROUTERERR_UNKNOWNPORTTYPE | The port type is unknown. |
| 0x505 | 1285 | 0x98110505 | ROUTERERR_NOTINITIALIZED | The router is not initialized. |
| 0x506 | 1286 | 0x98110506 | ROUTERERR_PORTALREADYINUSE | The port number is already assigned. |
| 0x507 | 1287 | 0x98110507 | ROUTERERR_NOTREGISTERED | The port is not registered. |
| 0x508 | 1288 | 0x98110508 | ROUTERERR_NOMOREQUEUES | The maximum number of ports has been reached. |
| 0x509 | 1289 | 0x98110509 | ROUTERERR_INVALIDPORT | The port is invalid. |
| 0x50A | 1290 | 0x9811050A | ROUTERERR_NOTACTIVATED | The router is not active. |
| 0x50B | 1291 | 0x9811050B | ROUTERERR_FRAGMENTBOXFULL | The mailbox has reached the maximum number for fragmented messages. |
| 0x50C | 1292 | 0x9811050C | ROUTERERR_FRAGMENTTIMEOUT | A fragment timeout has occurred. |
| 0x50D | 1293 | 0x9811050D | ROUTERERR_TOBEREMOVED | The port is removed. |

## General ADS error codes

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x700 | 1792 | 0x98110700 | ADSERR_DEVICE_ERROR | General device error. |
| 0x701 | 1793 | 0x98110701 | ADSERR_DEVICE_SRVNOTSUPP | Service is not supported by the server. |
| 0x702 | 1794 | 0x98110702 | ADSERR_DEVICE_INVALIDGRP | Invalid index group. |
| 0x703 | 1795 | 0x98110703 | ADSERR_DEVICE_INVALIDOFFSET | Invalid index offset. |
| 0x704 | 1796 | 0x98110704 | ADSERR_DEVICE_INVALIDACCESS | Reading or writing not permitted. |
| 0x705 | 1797 | 0x98110705 | ADSERR_DEVICE_INVALIDSIZE | Parameter size not correct. |
| 0x706 | 1798 | 0x98110706 | ADSERR_DEVICE_INVALIDDATA | Invalid data values. |
| 0x707 | 1799 | 0x98110707 | ADSERR_DEVICE_NOTREADY | Device is not ready to operate. |
| 0x708 | 1800 | 0x98110708 | ADSERR_DEVICE_BUSY | Device is busy. |
| 0x709 | 1801 | 0x98110709 | ADSERR_DEVICE_INVALIDCONTEXT | Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC. |
| 0x70A | 1802 | 0x9811070A | ADSERR_DEVICE_NOMEMORY | Insufficient memory. |
| 0x70B | 1803 | 0x9811070B | ADSERR_DEVICE_INVALIDPARM | Invalid parameter values. |
| 0x70C | 1804 | 0x9811070C | ADSERR_DEVICE_NOTFOUND | Not found (files, ...). |
| 0x70D | 1805 | 0x9811070D | ADSERR_DEVICE_SYNTAX | Syntax error in file or command. |
| 0x70E | 1806 | 0x9811070E | ADSERR_DEVICE_INCOMPATIBLE | Objects do not match. |
| 0x70F | 1807 | 0x9811070F | ADSERR_DEVICE_EXISTS | Object already exists. |
| 0x710 | 1808 | 0x98110710 | ADSERR_DEVICE_SYMBOLNOTFOUND | Symbol not found. |
| 0x711 | 1809 | 0x98110711 | ADSERR_DEVICE_SYMBOLVERSIONINVALID | Invalid symbol version. This can occur due to an online change. Create a new handle. |
| 0x712 | 1810 | 0x98110712 | ADSERR_DEVICE_INVALIDSTATE | Device (server) is in invalid state. |
| 0x713 | 1811 | 0x98110713 | ADSERR_DEVICE_TRANSMODENOTSUPP | AdsTransMode not supported. |
| 0x714 | 1812 | 0x98110714 | ADSERR_DEVICE_NOTIFYHNDINVALID | Notification handle is invalid. |
| 0x715 | 1813 | 0x98110715 | ADSERR_DEVICE_CLIENTUNKNOWN | Notification client not registered. |
| 0x716 | 1814 | 0x98110716 | ADSERR_DEVICE_NOMOREHDLS | No further handle available. |
| 0x717 | 1815 | 0x98110717 | ADSERR_DEVICE_INVALIDWATCHSIZE | Notification size too large. |
| 0x718 | 1816 | 0x98110718 | ADSERR_DEVICE_NOTINIT | Device not initialized. |
| 0x719 | 1817 | 0x98110719 | ADSERR_DEVICE_TIMEOUT | Device has a timeout. |
| 0x71A | 1818 | 0x9811071A | ADSERR_DEVICE_NOINTERFACE | Interface query failed. |
| 0x71B | 1819 | 0x9811071B | ADSERR_DEVICE_INVALIDINTERFACE | Wrong interface requested. |
| 0x71C | 1820 | 0x9811071C | ADSERR_DEVICE_INVALIDCLSID | Class ID is invalid. |
| 0x71D | 1821 | 0x9811071D | ADSERR_DEVICE_INVALIDOBJID | Object ID is invalid. |
| 0x71E | 1822 | 0x9811071E | ADSERR_DEVICE_PENDING | Request pending. |
| 0x71F | 1823 | 0x9811071F | ADSERR_DEVICE_ABORTED | Request is aborted. |
| 0x720 | 1824 | 0x98110720 | ADSERR_DEVICE_WARNING | Signal warning. |
| 0x721 | 1825 | 0x98110721 | ADSERR_DEVICE_INVALIDARRAYIDX | Invalid array index. |
| 0x722 | 1826 | 0x98110722 | ADSERR_DEVICE_SYMBOLNOTACTIVE | Symbol not active. |
| 0x723 | 1827 | 0x98110723 | ADSERR_DEVICE_ACCESSDENIED | Access denied. |
| 0x724 | 1828 | 0x98110724 | ADSERR_DEVICE_LICENSENOTFOUND | Missing license. |
| 0x725 | 1829 | 0x98110725 | ADSERR_DEVICE_LICENSEEXPIRED | License expired. |
| 0x726 | 1830 | 0x98110726 | ADSERR_DEVICE_LICENSEEXCEEDED | License exceeded. |
| 0x727 | 1831 | 0x98110727 | ADSERR_DEVICE_LICENSEINVALID | Invalid license. |
| 0x728 | 1832 | 0x98110728 | ADSERR_DEVICE_LICENSESYSTEMID | License problem: System ID is invalid. |
| 0x729 | 1833 | 0x98110729 | ADSERR_DEVICE_LICENSENOTIMELIMIT | License not limited in time. |
| 0x72A | 1834 | 0x9811072A | ADSERR_DEVICE_LICENSEFUTUREISSUE | Licensing problem: time in the future. |
| 0x72B | 1835 | 0x9811072B | ADSERR_DEVICE_LICENSETIMETOLONG | License period too long. |
| 0x72C | 1836 | 0x9811072C | ADSERR_DEVICE_EXCEPTION | Exception at system startup. |
| 0x72D | 1837 | 0x9811072D | ADSERR_DEVICE_LICENSEDUPLICATED | License file read twice. |
| 0x72E | 1838 | 0x9811072E | ADSERR_DEVICE_SIGNATUREINVALID | Invalid signature. |
| 0x72F | 1839 | 0x9811072F | ADSERR_DEVICE_CERTIFICATEINVALID | Invalid certificate. |
| 0x730 | 1840 | 0x98110730 | ADSERR_DEVICE_LICENSEOEMNOTFOUND | Public key not known from OEM. |
| 0x731 | 1841 | 0x98110731 | ADSERR_DEVICE_LICENSERESTRICTED | License not valid for this system ID. |
| 0x732 | 1842 | 0x98110732 | ADSERR_DEVICE_LICENSEDEMODENIED | Demo license prohibited. |
| 0x733 | 1843 | 0x98110733 | ADSERR_DEVICE_INVALIDFNCID | Invalid function ID. |
| 0x734 | 1844 | 0x98110734 | ADSERR_DEVICE_OUTOFRANGE | Outside the valid range. |

| Hex | Dec | HRESULT | Name | Description |
|------|------|------------|-------------------------------|-------------|
| 0x735 | 1845 | 0x98110735 | ADSERR_DEVICE_INVALIDALIGNMENT | Invalid alignment. |
| 0x736 | 1846 | 0x98110736 | ADSERR_DEVICE_LICENSEPLATFORM | Invalid platform level. |
| 0x737 | 1847 | 0x98110737 | ADSERR_DEVICE_FORWARD_PL | Context – forward to passive level. |
| 0x738 | 1848 | 0x98110738 | ADSERR_DEVICE_FORWARD_DL | Context – forward to dispatch level. |
| 0x739 | 1849 | 0x98110739 | ADSERR_DEVICE_FORWARD_RT | Context – forward to real time. |
| 0x740 | 1856 | 0x98110740 | ADSERR_CLIENT_ERROR | Client error. |
| 0x741 | 1857 | 0x98110741 | ADSERR_CLIENT_INVALIDPARM | Service contains an invalid parameter. |
| 0x742 | 1858 | 0x98110742 | ADSERR_CLIENT_LISTEMPTY | Polling list is empty. |
| 0x743 | 1859 | 0x98110743 | ADSERR_CLIENT_VARUSED | Var connection already in use. |
| 0x744 | 1860 | 0x98110744 | ADSERR_CLIENT_DUPLINVOKEID | The called ID is already in use. |
| 0x745 | 1861 | 0x98110745 | ADSERR_CLIENT_SYNCTIMEOUT | Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly. |
| 0x746 | 1862 | 0x98110746 | ADSERR_CLIENT_W32ERROR | Error in Win32 subsystem. |
| 0x747 | 1863 | 0x98110747 | ADSERR_CLIENT_TIMEOUTINVALID | Invalid client timeout value. |
| 0x748 | 1864 | 0x98110748 | ADSERR_CLIENT_PORTNOTOPEN | Port not open. |
| 0x749 | 1865 | 0x98110749 | ADSERR_CLIENT_NOAMSADDR | No AMS address. |
| 0x750 | 1872 | 0x98110750 | ADSERR_CLIENT_SYNCINTERNAL | Internal error in Ads sync. |
| 0x751 | 1873 | 0x98110751 | ADSERR_CLIENT_ADDHASH | Hash table overflow. |
| 0x752 | 1874 | 0x98110752 | ADSERR_CLIENT_REMOVEHASH | Key not found in the table. |
| 0x753 | 1875 | 0x98110753 | ADSERR_CLIENT_NOMORESYM | No symbols in the cache. |
| 0x754 | 1876 | 0x98110754 | ADSERR_CLIENT_SYNCRESINVALID | Invalid response received. |
| 0x755 | 1877 | 0x98110755 | ADSERR_CLIENT_SYNCPORTLOCKED | Sync Port is locked. |
| 0x756 | 1878 | 0x98110756 | ADSERR_CLIENT_REQUESTCANCELLED | The request was cancelled. |

**RTime error codes**

| Hex | Dec | HRESULT | Name | Description |
|------|------|------------|------------------------|-------------|
| 0x1000 | 4096 | 0x98111000 | RTERR_INTERNAL | Internal error in the real-time system. |
| 0x1001 | 4097 | 0x98111001 | RTERR_BADTIMERPERIODS | Timer value is not valid. |
| 0x1002 | 4098 | 0x98111002 | RTERR_INVALIDTASKPTR | Task pointer has the invalid value 0 (zero). |
| 0x1003 | 4099 | 0x98111003 | RTERR_INVALIDSTACKPTR | Stack pointer has the invalid value 0 (zero). |
| 0x1004 | 4100 | 0x98111004 | RTERR_PRIOEXISTS | The request task priority is already assigned. |
| 0x1005 | 4101 | 0x98111005 | RTERR_NOMORETCB | No free TCB (Task Control Block) available. The maximum number of TCBs is 64. |
| 0x1006 | 4102 | 0x98111006 | RTERR_NOMORESEMAS | No free semaphores available. The maximum number of semaphores is 64. |
| 0x1007 | 4103 | 0x98111007 | RTERR_NOMOREQUEUES | No free space available in the queue. The maximum number of positions in the queue is 64. |
| 0x100D | 4109 | 0x9811100D | RTERR_EXTIRQALREADYDEF | An external synchronization interrupt is already applied. |
| 0x100E | 4110 | 0x9811100E | RTERR_EXTIRQNOTDEF | No external sync interrupt applied. |
| 0x100F | 4111 | 0x9811100F | RTERR_EXTIRQINSTALLFAILED | Application of the external synchronization interrupt has failed. |
| 0x1010 | 4112 | 0x98111010 | RTERR_IRQLNOTLESSOREQUAL | Call of a service function in the wrong context |
| 0x1017 | 4119 | 0x98111017 | RTERR_VMXNOTSUPPORTED | Intel VT-x extension is not supported. |
| 0x1018 | 4120 | 0x98111018 | RTERR_VMXDISABLED | Intel VT-x extension is not enabled in the BIOS. |
| 0x1019 | 4121 | 0x98111019 | RTERR_VMXCONTROLSMISSING | Missing function in Intel VT-x extension. |
| 0x101A | 4122 | 0x9811101A | RTERR_VMXENABLEFAILS | Activation of Intel VT-x fails. |

**Specific positive HRESULT Return Codes:**

| HRESULT | Name | Description |
|-----------|-------------------|-------------|
| 0x0000_0000 | S_OK | No error. |
| 0x0000_0001 | S_FALSE | No error. Example: successful processing, but with a negative or incomplete result. |
| 0x0000_0203 | S_PENDING | No error. Example: successful processing, but no result is available yet. |
| 0x0000_0256 | S_WATCHDOG_TIMEOUT | No error. Example: successful processing, but a timeout occurred |

**TCP Winsock error codes**

| Hex | Dec | Name | Description |
|---|---|---|---|
| 0x274C | 10060 | WSAETIMEDOUT | A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond. |
| 0x274D | 10061 | WSAECONNREFUSED | Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running. |
| 0x2751 | 10065 | WSAEHOSTUNREACH | No route to host - a socket operation referred to an unavailable host. |
| | | | More 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 rate |
| $\Sigma w_n$ | eWindowFunction, nWindowLength | Sum of the values of the window function |
| $\Sigma w_n^2$ | 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([$\Sigma (x_n)^2$] / N) | | Root Mean Square value of a signal |
| 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 [▶ 266]), which can be selected by the function blocks FB_CMA_PowerSpectrum [▶ 197] and FB_CMA_MagnitudeSpectrum [▶ 168] and function 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 / $\Sigma w_n$ | 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.<br><br>MAX($|X_k|$) = A<br>However, the maximum values of the spectrum do not enable a robust estimation of the amplitude, since so-called Scalloping Losses may occur. |
| eCM_RootPowerSum | 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 |

| Scaling option | Contained factor | Description |
|---|---|---|
| | | 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 sinusoidal 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** | | |
| `eCM_PowerSpectralDensity` | SQRT(2 / $\Sigma w_n^2$) | 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($X_k$) = $|X_k|^2/F_s$ |
| | | 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($X_k$) = $|X_k|$/ SQRT($F_s$) |
| **Elementary** | | |
| `eCM_DiracScaling` | sqrt(N / $\Sigma w_n^2$ ) | 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. |

# 7.4    Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

**Download finder**

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

**Beckhoff's branch offices and representatives**

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

**Beckhoff Support**

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:            +49 5246 963-157
e-mail:            support@beckhoff.com

**Beckhoff Service**

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:            +49 5246 963-460
e-mail:            service@beckhoff.com

**Beckhoff Headquarters**

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:            +49 5246 963-0
e-mail:            info@beckhoff.com
web:            www.beckhoff.com

BECKHOFF

# 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) rep-

resent 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 IEEE 754 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 stan-

dard 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.

More Information:
**www.beckhoff.com/tf3600**