

Namespace TwinSharp

Classes

[AmsRoute](#)

Represents a route to a TwinCAT target system, including its name, address, AmsNetId, protocol, and flags. Provides functionality to retrieve the state information of the route.

[Constants](#)

TwinCAT constants made available to the entire project.

[EtherCatDevice](#)

Describes an EtherCAT device, using all standard objects as defined from the EtherCAT standard. This class can be derived and extended to create a specific EtherCAT device.

[EtherCatMaster](#)

The EtherCatMaster class provides methods to interact with an EtherCAT master device. It allows for reading the current state, device type, and name of the master, as well as retrieving information about connected slaves, such as their configuration, state, and topology. Additionally, it can read unexpected state changes and convert device status to a string representation.

[Extensions](#)

Contains extension methods for various classes.

[FileFinder](#)

This class can search for files and enumerate them on a remote TwinCAT target.

[FileSystem](#)

The FileSystem class provides methods for interacting with the file system on a target device using the TwinCAT ADS protocol. It allows opening, closing, reading, writing, seeking, deleting files, as well as creating and deleting directories.

[License](#)

Represents a license manager for a TwinCAT system.

[Realtime](#)

The Realtime class provides methods to interact with the TwinCAT systems real-time settings. It allows setting shared cores configuration, reading CPU settings, reading CPU latency, and getting the current CPU usage. It uses the AdsClient to communicate with the TwinCAT system.

[Scope](#)

Class to interact with a TwinCAT 2 scope. Note: not compatible with TwinCAT 3.

[SumReader](#)

Using the ADS Sum Command it is possible to read or write several variables in one command.

[TcSystem](#)

The TcSystem class represents a TwinCAT system and provides methods to interact with it. It allows switching the system to different modes (Config, Restart, Stop), listing EtherCAT masters, and listing local static routes. It also provides access to the system's real-time properties, license information, and file system through the Realtime, License, and FileSystem properties respectively.

Structs

[RTIMECPULATENCY](#)

Struct that describes the realtime latency of the CPU.

[RTIMECPUSETTINGS](#)

Struct that describes the amount of Windows (shared) cores and isolated cores for TwinCAT.

[ST_CheckLicense](#)

Structure with license information.

[ST_EcSlaveConfigData](#)

The structure ST_EcSlaveConfigData contains the EtherCAT configuration data for an EtherCAT slave device.

[ST_EcSlaveldentity](#)

The structure ST_EcSlaveldentity contains the EtherCAT identity data for an EtherCAT slave device.

[ST_EcSlaveState](#)

The structure ST_EcSlaveState contains the EtherCAT state and the link state of an EtherCAT slave device.

[ST_FileAttributes](#)

Describes the different attributes that a file can have. Such as readonly, hidden, system etc.

[ST_FindFileEntry](#)

The structure ST_FindFileEntry contains information about a file or directory found by the FindFirstFile and FindNextFile functions.

[ST_PortAddr](#)

The structure ST_PortAddr contains EtherCAT topology information for EtherCAT slave device. EtherCAT slave devices typically have 2 to 4 ports.

[ST_TopoLOGYDATAEX](#)

The structure ST_TopoLOGYDATAEX contains information on EtherCAT topology and hot-connect groups.

Enums

[E_EnumCmdType](#)

Control parameter for enumeration blocks. Not all parameters are used by each enumeration block!

[E_LicenseHResult](#)

The E_LicenseHResult enumeration contains the possible results of the license check.

[ExtendedAdsErrorCodes](#)

Beckhoff does not describe all of its existing error codes in its enum in ADS Abstractions. So we "extend" it here. Mostly error codes received from MDP/IPC is missing. More info:

https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclib_tc2_mdp/178768395.html



[FileOpenModeFlags](#)

The FileOpenModeFlags enum defines various modes for opening files, each represented by a unique flag. These flags can be combined using bitwise operations to specify multiple modes simultaneously.

Class AmsRoute

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Represents a route to a TwinCAT target system, including its name, address, AmsNetId, protocol, and flags. Provides functionality to retrieve the state information of the route.

```
public class AmsRoute
```

Inheritance

[object](#) ← AmsRoute

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Properties

Adress

Address of the TwinCAT target system The address depends on the transport protocol being used. In addition to TCP/IP addresses, addresses of Profibus devices are possible, which in turn must support the ADS protocol in order to be addressed as "target system" or "remote system".

```
public string Adress { get; }
```

Property Value

[string](#)

AmsNetId

AmsNetId of the target system

```
public AmsNetId AmsNetId { get; }
```

Property Value

AmsNetId

Flags

Bitmask of flags that describe the route.

```
public int Flags { get; }
```

Property Value

[int](#)

Name

Name of the possible target system logged on to the current TwinCAT router

```
public string Name { get; }
```

Property Value

[string](#)

Protocol

Protocol used for communication with the target system.

```
public string Protocol { get; }
```

Property Value

[string](#)

Methods

GetStateInfo()

Gets the state information for this ams route.

```
public StateInfo GetStateInfo()
```

Returns

StateInfo

ToString()

Returns a string representation of the route.

```
public override string ToString()
```

Returns

[string](#)

Class Constants

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

TwinCAT constants made available to the entire project.

```
public static class Constants
```

Inheritance

[object](#) ← Constants

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

MAX_STRING_LENGTH

Maximum length of a string. Used by some read operations.

```
public const int MAX_STRING_LENGTH = 255
```

Field Value

[int](#)

Enum E_EnumCmdType

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Control parameter for enumeration blocks. Not all parameters are used by each enumeration block!

```
public enum E_EnumCmdType
```

Fields

Abort = 2

Cancels the listing (closes open handles)

First = 0

Lists the first element

Next = 1

Lists the next element

Enum E_LicenseHResult

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The E_LicenseHResult enumeration contains the possible results of the license check.

```
public enum E_LicenseHResult : long
```

Fields

E_LHR_CertificateInvalid = 2551252783

Certificate is invalid

E_LHR_DeviceException = 2551252780

Exception at system startup

E_LHR_LicenseDemoDenied = 2551252786

Trial license not allowed

E_LHR_LicenseDuplicated = 2551252781

License data read multiple times

E_LHR_LicenseExceeded = 2551252774

License has too few instances

E_LHR_LicenseExpired = 2551252773

License expired

E_LHR_LicenseInvalid = 2551252775

License is invalid

E_LHR_LicenseNoFound = 2551252772

Missing license

E_LHR_LicenseNoTimeLimit = 2551252777

License not limited in time

E_LHR_LicenseOK = 0

License is valid

E_LHR_LicenseOK_Demo = 596

Trial license is valid

E_LHR_LicenseOK_OEM = 597

OEM license is valid

E_LHR_LicenseOK_Pending = 515

Validation of the licensing device (e.g. License Key Terminal) required, license is however still valid.

E_LHR_LicenseOemNotFound = 2551252784

OEM license for unknown OEM

E_LHR_LicensePlatformLevelInv = 2551252787

Invalid platform level for the license

E_LHR_LicenseRestricted = 2551252785

License invalid for the system

E_LHR_LicenseSystemIdInvalid = 2551252776

Incorrect system ID for the license

E_LHR_LicenseTimeInFuture = 2551252778

License problem: Time of issue is in the future

E_LHR_LicenseTimePeriodToLong = 2551252779

License period too long

E_LHR_SignatureInvalid = 2551252782

Invalid signature

Class EtherCatDevice

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Describes an EtherCAT device, using all standard objects as defined from the EtherCAT standard. This class can be derived and extended to create a specific EtherCAT device.

```
public class EtherCatDevice
```

Inheritance

[object](#) ← EtherCatDevice

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

EtherCatDevice(AdsClient)

Creates a representation of an EtherCAT device. The client should typically be connected to the AmsNetId of the EtherCAT master and the port number is the slaves adress (example 1025).

```
public EtherCatDevice(AdsClient client)
```

Parameters

client AdsClient

Properties

DeviceType

The device type of the EtherCAT device.

```
public uint DeviceType { get; }
```

Property Value

[uint](#)

ErrorRegister

Returns the error register of the EtherCAT device.

```
public byte ErrorRegister { get; }
```

Property Value

[byte](#)

ManufacturerDeviceName

This parameter specifies the manufacturers device name of the device.

```
public string ManufacturerDeviceName { get; }
```

Property Value

[string](#)

ManufacturerHardwareVersion

This parameter specifies the hardware version of the device.

```
public string ManufacturerHardwareVersion { get; }
```

Property Value

[string](#)

ManufacturerSoftwareVersion

The Software Version object has the following parameter:

```
public string ManufacturerSoftwareVersion { get; }
```

Property Value

[string](#)

ProductCode

This parameter specifies the product code of the device.

```
public uint ProductCode { get; }
```

Property Value

[uint](#)

RevisionNumber

The revision number of the EtherCAT device.

```
public uint RevisionNumber { get; }
```

Property Value

[uint](#)

SerialNumber

The serial number of the EtherCAT device.

```
public uint SerialNumber { get; }
```

Property Value

[uint](#)

VendorID

This parameter specifies the vendor ID of the device.

```
public uint VendorID { get; }
```

Property Value

[uint](#)

Methods

CoeReadAny<T>(uint, uint)

Read any object from the CoE object dictionary.

```
public T CoeReadAny<T>(uint index, uint subIndex)
```

Parameters

index [uint](#)

subIndex [uint](#)

Returns

T

Type Parameters

T

CoeWriteAny(uint, uint, object)

Write any object to the CoE object dictionary.

```
public void CoeWriteAny(uint index, uint subIndex, object value)
```

Parameters

index [uint](#)

subIndex [uint](#)

value [object](#)

Class EtherCatMaster

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The EtherCatMaster class provides methods to interact with an EtherCAT master device. It allows for reading the current state, device type, and name of the master, as well as retrieving information about connected slaves, such as their configuration, state, and topology. Additionally, it can read unexpected state changes and convert device status to a string representation.

```
public class EtherCatMaster
```

Inheritance

[object](#) ← EtherCatMaster

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Properties

AmsNetId

The AmsNetId of the EtherCAT master.

```
public AmsNetId AmsNetId { get; }
```

Property Value

AmsNetId

DeviceType

The device type of the EtherCAT master.

```
public ushort DeviceType { get; }
```

Property Value

[ushort](#)

MasterDevState

Can be used to read the current state of the EtherCAT master. Corresponds to the PLC FB: FB_EcGetMasterDevState

```
public ushort MasterDevState { get; }
```

Property Value

[ushort](#)

Name

The name of the EtherCAT master.

```
public string Name { get; }
```

Property Value

[string](#)

SlaveCount

Can be used to determine the number of slaves that are connected to the master. Corresponds to the PLC FB: FB_EcGetSlaveCount.

```
public uint SlaveCount { get; }
```

Property Value

[uint](#)

SlaveCountConfigured

Count of number of slaves configured in TwinCAT.

```
public ushort SlaveCountConfigured { get; }
```

Property Value

[ushort](#)

Methods

GetAllSlaveAbnormalStateChanges()

Can be used to read the unexpected EtherCAT state changes of all the slaves connected to the master. It returns the number of unexpected state changes of all slaves as an array of UDINTs. EtherCAT state changes are unexpected if they were not requested by the EtherCAT master, e.g. if an EtherCAT slave spontaneously switches from OP state to SAFEOP state. Corresponds to the function block FB_EcGetAllSlaveAbnormalStateChanges.

```
public uint[] GetAllSlaveAbnormalStateChanges()
```

Returns

[uint](#)[]

GetAllSlaveStates()

Reads the EtherCAT status and the Link status of all the slaves connected to the master. Corresponds to the function block FB_EcGetAllSlaveStates.

```
public ST_EcSlaveState[] GetAllSlaveStates()
```

Returns

[ST_EcSlaveState](#)[]

GetConfiguredSlaves()

Generates an array of all configured Slaves from the Master object directory. Corresponds to the function block FB_EcGetConfSlaves

```
public ST_EcSlaveConfigData[] GetConfiguredSlaves()
```

Returns

[ST_EcSlaveConfigData\[\]](#)

GetSlaveTopologyInfo()

Can be used to determine topology information. Equivavalent to the function block FB_EcGetSlaveTopolgyInfo.

```
public ST_TopoogyDataEx[] GetSlaveTopologyInfo()
```

Returns

[ST_TopoogyDataEx\[\]](#)

An array of structures of type ST_TopoogyDataEx, which contains the topology data.

MasterDevStateToString(ushort)

Converts the device status of the EtherCAT master to a string. For masterDevState == 0 'OK' is returned, otherwise, 'Not OK – Link error', e.g. for masterDevState == 1. If several errors are pending, they are separated by hyphens.

```
public string MasterDevStateToString(ushort masterDevState)
```

Parameters

[masterDevState ushort](#)

Returns

[string](#) ↗

ToString()

Returns a string representation of the EtherCAT master.

```
public override string ToString()
```

Returns

[string](#) ↗

Enum ExtendedAdsErrorCodes

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Beckhoff does not describe all of its existing error codes in its enum in ADS Abstractions. So we "extend" it here. Mostly error codes received from MDP/IPC is missing. More info:

https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclib_tc2_mdp/178768395.html

```
public enum ExtendedAdsErrorCodes : uint
```

Fields

NotSupportedException = 3970306048

Operation not supported.

Class Extensions

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Contains extension methods for various classes.

```
public static class Extensions
```

Inheritance

[object](#) ← Extensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

ReadDateTime(AdsClient, uint)

Reads a uint from the specified index group and offset and converts it to a DateTime object.

```
public static DateTime ReadDateTime(this AdsClient client, uint handle)
```

Parameters

client AdsClient

handle [uint](#)

Returns

[DateTime](#)

ReadString(AdsClient, uint, uint, int)

Reads a string from the specified index group and offset using the UTF8 encoding.

```
public static string ReadString(this AdsClient client, uint indexGroup, uint indexOffset,  
int len)
```

Parameters

client AdsClient

indexGroup [uint](#)

indexOffset [uint](#)

len [int](#)

Returns

[string](#)

ToUint(byte[])

Converts a byte array to a uint.

```
public static uint ToUint(this byte[] buffer)
```

Parameters

buffer [byte](#)[]

Returns

[uint](#)

Class FileFinder

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

This class can search for files and enumerate them on a remote TwinCAT target.

```
public class FileFinder
```

Inheritance

[object](#) ← FileFinder

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

EndOfEnumeration

End of enumeration was reached. During the first attempt to read a non-existing entry this output is set to TRUE.

```
public bool EndOfEnumeration { get; }
```

Property Value

[bool](#)

Methods

Abort()

Call this when you dont want to continue enumeration of files. Releases resources on TwinCAT side.

```
public void Abort()
```

GetNextFileOrNull()

Returns the next file found or null if no more files are found.

```
public ST_FindFileEntry? GetNextFileOrNull()
```

Returns

[ST_FindFileEntry?](#)

Enum FileOpenModeFlags

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The FileOpenModeFlags enum defines various modes for opening files, each represented by a unique flag. These flags can be combined using bitwise operations to specify multiple modes simultaneously.

```
public enum FileOpenModeFlags
```

Fields

FOPEN_MODEAPPEND = 4

"a": Opens for writing at the end of the file (appending) without removing the EOF marker before writing new data to the file; creates the file first if it doesnot exist.

FOPEN_MODEBINARY = 16

"b": Open in binary (untranslated) mode.

FOPEN_MODEPLUS = 8

"+": Opens for both reading and writing. (The file must exist.)

FOPEN_MODEREAD = 1

"r": Opens for reading. If the file does not exist or cannot be found, the call fails.

FOPEN_MODETEXT = 32

"t": Open in text (translated) mode.

FOPEN_MODEWRITE = 2

"w": Opens an empty file for writing. If the given file exists, its contents are destroyed.

Class FileSystem

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The FileSystem class provides methods for interacting with the file system on a target device using the TwinCAT ADS protocol. It allows opening, closing, reading, writing, seeking, deleting files, as well as creating and deleting directories.

```
public class FileSystem : IDisposable
```

Inheritance

[object](#) ← FileSystem

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

CreateDirectory(string)

Create a new folder on the target. Note: does not create folders recursively.

```
public void CreateDirectory(string path)
```

Parameters

path [string](#)

CreateFileFinder(string)

Use a file finder to search for files on target device.

```
public FileFinder CreateFileFinder(string searchQuery)
```

Parameters

searchQuery [string](#)

A valid directory name or directory with file name as string. The string can contain (*) and (?) as wildcards. If the path ends with a wildcard, dot or the directory name, the user must have access rights to this path and its subdirectories.

Returns

[FileFinder](#)

DeleteDirectory(string)

Can be used to delete a directory from the data storage device. A directory containing files cannot be deleted. Equivalent to the function block FB_RemoveDir.

```
public void DeleteDirectory(string path)
```

Parameters

path [string](#)

Dispose()

Disposes the File System object. Disposes the ADS client used.

```
public void Dispose()
```

FileClose(ushort)

The function block FB_FileClose closes the file, thereby putting it in a defined state for further processing by other programs. Equivalent to the TwinCAT function block FB_FileClose.

```
public void FileClose(ushort handle)
```

Parameters

handle [ushort](#)

FileDelete(string)

Deletes a file from the data storage device. Equivavalent to the function block FB_FileDelete.

```
public void FileDelete(string pathName)
```

Parameters

pathName [string](#)

File name, including the full path.

FileGetString(ushort, out bool)

Reads strings from a file. The string is read up to and including the line feed character, or up to the end of the file or the maximum permitted length of sLine. The null termination is appended automatically. The file must have been opened in text mode. Equivavelent to the function block FB_FileGets.

```
public string FileGetString(ushort handle, out bool endOfFile)
```

Parameters

handle [ushort](#)

endOfFile [bool](#)

True if end of file is reached.

Returns

[string](#)

FileOpen(string, FileModeFlags)

Creates a new file or opens an existing file for editing. Equivalent to the TwinCAT function block FB_FileOpen

```
public ushort FileOpen(string path, FileModeFlags mode)
```

Parameters

path [string](#)

mode [FileModeFlags](#)

Returns

[ushort](#)

FilePutString(ushort, string)

Writes strings into a file. The string is written to the file up to the null termination, but without the null character. The file must have been opened in text mode. Equivalent to the function block FB_FilePuts

```
public void FilePutString(ushort fileHandle, string str)
```

Parameters

fileHandle [ushort](#)

str [string](#)

FileRead(ushort, int)

The contents of an already opened file can be read. Before a read access, the file must have been opened in the corresponding mode. In addition to the FOPEN_MODEREAD, the appropriate format (FOPEN_MODEBINARY or FOPEN_MODETEXT) is also important to achieve the desired result. Equivalent to the function block FB_FileRead.

```
public byte[] FileRead(ushort handle, int byteCountToRead)
```

Parameters

`handle` [ushort](#)

`byteCountToRead` [int](#)

Number of bytes to be read.

Returns

[byte](#)[]

FileRename(string, string)

Can be used to rename a file. Equivavalent to the function block FB_FileRename.

```
public void FileRename(string oldPath, string newPath)
```

Parameters

`oldPath` [string](#)

`newPath` [string](#)

FileSeek(ushort, int, SeekOrigin)

Sets the file pointer of an open file to a definable position. Equivavalent to the function block FB_FileSeek.

```
public void FileSeek(ushort handle, int position, SeekOrigin origin)
```

Parameters

`handle` [ushort](#)

`position` [int](#)

`origin` [SeekOrigin](#)

FileTell(uint)

Determines the current position of the file pointer. The position indicates the relative offset from the start of the file. Equivalent to the function block FB_FileTell. Note that for files opened in "Append at end of file" mode, the current position is determined by the last I/O operation, not by the position of the next write access. After a read operation, for example, the file pointer is at the position where the next read access will take place, not at the position where the next write access will take place. In append mode, the file pointer is always moved to the end before the write operation. If no previous I/O operation was performed and the file was opened in append mode, the file pointer is at the start of the file.

```
public int FileTell(uint handle)
```

Parameters

handle [uint](#)

Returns

[int](#)

The current position of the file pointer.

FileWrite(ushort, byte[])

Writes data into a file. For write access the file must have been opened in the corresponding mode, and it must be closed again for further processing by external programs. Equivalent to the function block FB_FileWrite.

```
public uint FileWrite(ushort handle, byte[] data)
```

Parameters

handle [ushort](#)

data [byte](#)[]

Returns

[uint](#)

The number of bytes that were successfully written.

GetFileProperties(string)

```
public ST_FindFileEntry GetFileProperties(string path)
```

Parameters

path [string](#)

Returns

[ST_FindFileEntry](#)

Class License

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Represents a license manager for a TwinCAT system.

```
public class License
```

Inheritance

[object](#) ← License

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

GetValidLicenses()

Gets the valid licenses for the target.

```
public ST_CheckLicense[] GetValidLicenses()
```

Returns

[ST_CheckLicense\[\]](#)

An array of valid licenses.

Struct RTimeCpuLatency

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Struct that describes the realtime latency of the CPU.

```
public struct RTimeCpuLatency
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Current

The current latency time of a TwinCAT system in μs .

```
public uint Current
```

Field Value

[uint](#)

Limit

Limit.

```
public uint Limit
```

Field Value

[uint](#)

Maximum

The maximum latency time of a TwinCAT system in μs (maximum latency time since the TwinCAT system was last started).

```
public uint Maximum
```

Field Value

[uint](#) ↗

Struct RTimeCpuSettings

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Struct that describes the amount of Windows (shared) cores and isolated cores for TwinCAT.

```
public struct RTimeCpuSettings
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

AffinityMask

Affinity mask.

```
public ulong AffinityMask
```

Field Value

[ulong](#)

CpuFamily

CPU family.

```
public uint CpuFamily
```

Field Value

[uint](#)

CpuFreq

CPU frequency.

```
public uint CpuFreq
```

Field Value

[uint](#)

CpuType

CPU type.

```
public uint CpuType
```

Field Value

[uint](#)

NonWinCPUs

Number of non windows cores.

```
public uint NonWinCPUs
```

Field Value

[uint](#)

RtCpus

Number of real time cores.

```
public uint RtCpus
```

Field Value

[uint](#) ↗

WinCPUs

Number of Windows (shared) cores.

```
public uint WinCPUs
```

Field Value

[uint](#) ↗

Class Realtime

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The Realtime class provides methods to interact with the TwinCAT systems real-time settings. It allows setting shared cores configuration, reading CPU settings, reading CPU latency, and getting the current CPU usage. It uses the AdsClient to communicate with the TwinCAT system.

```
public class Realtime
```

Inheritance

[object](#) ← Realtime

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

CpuUsage

Gets the current CPU usage of a TwinCAT system. Corresponds to the function block TC_CpuUsage. This function corresponds to the display of CPU usage in the TwinCAT system menu under the real-time settings.

```
public uint CpuUsage { get; }
```

Property Value

[uint](#)

Methods

ReadCpuSettings()

Reads the CPU settings of the TwinCAT system.

```
public RTimeCpuSettings ReadCpuSettings()
```

Returns

[RTimeCpuSettings](#)

ReadLatency()

Reads the CPU latency of the TwinCAT system.

```
public RTimeCpuLatency ReadLatency()
```

Returns

[RTimeCpuLatency](#)

SetSharedCores(uint)

Sets the shared cores configuration for the TwinCAT system.

```
public AdsErrorCode SetSharedCores(uint sharedCores)
```

Parameters

sharedCores [uint](#)

Returns

AdsErrorCode

Struct ST_CheckLicense

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Structure with license information.

```
public struct ST_CheckLicense
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [object.Equals\(object, object\)](#) ,
[object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ST_CheckLicense(byte[], string)

Constructor for ST_CheckLicense from a byte array of length 48.

```
public ST_CheckLicense(byte[] bytes, string descriptionText)
```

Parameters

bytes [byte](#)[]

descriptionText [string](#)

Exceptions

[Exception](#)

Fields

ExpirationTime

Expiration time of the license

```
public readonly DateTime ExpirationTime
```

Field Value

[DateTime](#)

ExpirationTimeString

Expiration time of the license as a string

```
public readonly string ExpirationTimeString
```

Field Value

[string](#)

LicensesId

License ID

```
public readonly Guid LicenseId
```

Field Value

[Guid](#)

LicenseName

Name of the license

```
public string LicenseName
```

Field Value

[string](#)

eResult

License status

```
public readonly E_LicenseHResult eResult
```

Field Value

[E_LicenseHResult](#)

nCount

Number of instances for this license (0=unlimited)

```
public readonly uint nCount
```

Field Value

[uint](#)

Methods

ToString()

Returns a string representation of the license.

```
public override readonly string ToString()
```

Returns

[string](#)

Struct ST_EcSlaveConfigData

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The structure ST_EcSlaveConfigData contains the EtherCAT configuration data for an EtherCAT slave device.

```
public struct ST_EcSlaveConfigData
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [object.Equals\(object, object\)](#) ,
[object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ST_EcSlaveConfigData(byte[])

Constructor for ST_EcSlaveConfigData from a byte array of length 80.

```
public ST_EcSlaveConfigData(byte[] bytes)
```

Parameters

bytes [byte](#)[]

Exceptions

[Exception](#)

Fields

Addr

Address of an EtherCAT slave.

```
public ushort Addr
```

Field Value

[ushort](#)

DevType

EtherCAT device type of a slave

```
public uint DevType
```

Field Value

[uint](#)

Entries

Used internally.

```
public ushort Entries
```

Field Value

[ushort](#)

LinkStatus

Link status of an EtherCAT slave

```
public byte LinkStatus
```

Field Value

[byte](#)

MailboxInSize

Mailbox InSize of an EtherCAT slave

```
public ushort MailboxInSize
```

Field Value

[ushort](#)

MailboxOutSize

Mailbox OutSize of an EtherCAT slave

```
public ushort MailboxOutSize
```

Field Value

[ushort](#)

Name

Name of an EtherCAT slave.

```
public string Name
```

Field Value

[string](#)

Slaveldentity

EtherCAT identity data of a slave

```
public ST_EcSlaveIdentity SlaveIdentity
```

Field Value

[ST_EcSlaveIdentity](#)

Type

EtherCAT type of a slave.

```
public string Type
```

Field Value

[string](#)

Methods

ToString()

Returns a string representation of the slave configuration data.

```
public override readonly string ToString()
```

Returns

[string](#)

Struct ST_EcSlaveIdentity

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The structure ST_EcSlaveIdentity contains the EtherCAT identity data for an EtherCAT slave device.

```
public struct ST_EcSlaveIdentity
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ST_EcSlaveIdentity(byte[])

Constructor for ST_EcSlaveIdentity from a byte array of length 16.

```
public ST_EcSlaveIdentity(byte[] bytes)
```

Parameters

bytes [byte](#)[]

Exceptions

[Exception](#)

Fields

ProductCode

Product code of the slave device

```
public uint ProductCode
```

Field Value

[uint](#) ↗

RevisionNo

Indicates the revision number of the slave device.

```
public uint RevisionNo
```

Field Value

[uint](#) ↗

SerialNo

Indicates the serial number of the slave device.

```
public uint SerialNo
```

Field Value

[uint](#) ↗

VendorId

Vendor-ID of the slave device

```
public uint VendorId
```

Field Value

[uint](#) ↗

Struct ST_EcSlaveState

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The structure ST_EcSlaveState contains the EtherCAT state and the link state of an EtherCAT slave device.

```
public struct ST_EcSlaveState
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ST_EcSlaveState(byte[])

Constructor for ST_EcSlaveState from a byte array of length 2.

```
public ST_EcSlaveState(byte[] bytes)
```

Parameters

bytes [byte](#)[]

Exceptions

[Exception](#)

Fields

DeviceState

EtherCAT state of a slave

```
public byte DeviceState
```

Field Value

[byte](#) ↗

LinkState

Link state of a slave

```
public byte LinkState
```

Field Value

[byte](#) ↗

Struct ST_FileAttributes

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Describes the different attributes that a file can have. Such as readonly, hidden, system etc.

```
public struct ST_FileAttributes
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Archive

FILE_ATTRIBUTE_ARCHIVE

```
public bool Archive
```

Field Value

[bool](#)

Compressed

FILE_ATTRIBUTE_COMPRESSED

```
public bool Compressed
```

Field Value

[bool](#)

Device

FILE_ATTRIBUTE_DEVICE. Under CE: FILE_ATTRIBUTE_INROM or FILE_ATTRIBUTE_ENCRYPTED

```
public bool Device
```

Field Value

[bool](#) ↗

Directory

FILE_ATTRIBUTE_DIRECTORY

```
public bool Directory
```

Field Value

[bool](#) ↗

Encrypted

FILE_ATTRIBUTE_ENCRYPTED

```
public bool Encrypted
```

Field Value

[bool](#) ↗

Hidden

FILE_ATTRIBUTE_HIDDEN

```
public bool Hidden
```

Field Value

[bool](#) ↗

Normal

FILE_ATTRIBUTE_NORMAL

`public bool Normal`

Field Value

[bool](#) ↗

NotContentIndexed

FILE_ATTRIBUTE_NOT_CONTENT_INDEXED. Under CE: FILE_ATTRIBUTE_ROMMODULE

`public bool NotContentIndexed`

Field Value

[bool](#) ↗

Offline

FILE_ATTRIBUTE_OFFLINE. Under CE: FILE_ATTRIBUTE_ROMSTATICREF

`public bool Offline`

Field Value

[bool](#) ↗

ReadOnly

FILE_ATTRIBUTE_READONLY

```
public bool ReadOnly
```

Field Value

[bool](#) ↗

ReparsePoint

FILE_ATTRIBUTE_REPARSE_POINT

```
public bool ReparsePoint
```

Field Value

[bool](#) ↗

SparseFile

FILE_ATTRIBUTE_SPARSE_FILE

```
public bool SparseFile
```

Field Value

[bool](#) ↗

System

FILE_ATTRIBUTE_SYSTEM

```
public bool System
```

Field Value

[bool](#) ↗

Temporary

FILE_ATTRIBUTE_TEMPORARY

`public bool Temporary`

Field Value

[bool](#) ↗

Struct ST_FindFileEntry

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The structure ST_FindFileEntry contains information about a file or directory found by the FindFirstFile and FindNextFile functions.

```
public struct ST_FindFileEntry
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

AlternateFileName

Zero-terminated string with the alternative name of the file or directory in conventional 8.3 format(filename.ext).

```
public string AlternateFileName
```

Field Value

[string](#)

CreationTime

Creation time of the file.

```
public DateTime CreationTime
```

Field Value

[DateTime](#)

FileAttributes

File attributes.

```
public ST_FileAttributes FileAttributes
```

Field Value

[ST_FileAttributes](#)

FileName

Zero-terminated string with the name of the file or directory (type: T_MaxString).

```
public string FileName
```

Field Value

[string](#)

FileSize

File size in bytes.

```
public ulong FileSize
```

Field Value

[ulong](#)

LastAccessTime

For a file the structure indicates when it was last accessed (read or write). For a directory the structure indicates when it was created.

```
public DateTime LastAccessTime
```

Field Value

[DateTime](#)

LastWriteTime

Last write time of the file.

```
public DateTime LastWriteTime
```

Field Value

[DateTime](#)

Struct ST_PortAddr

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The structure ST_PortAddr contains EtherCAT topology information for EtherCAT slave device. EtherCAT slave devices typically have 2 to 4 ports.

```
public struct ST_PortAddr
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ST_PortAddr(byte[])

Constructor for ST_PortAddr

```
public ST_PortAddr(byte[] bytes)
```

Parameters

bytes [byte](#)[]

Exceptions

[Exception](#)

Fields

PortA

Address of the previous EtherCAT slave at port A of the current EtherCAT slave

```
public ushort PortA
```

Field Value

[ushort](#)

PortB

Address of the optional subsequent EtherCAT slave at port B of the current EtherCAT slave

```
public ushort PortB
```

Field Value

[ushort](#)

PortC

Address of the optional subsequent EtherCAT slave at port C of the current EtherCAT slave

```
public ushort PortC
```

Field Value

[ushort](#)

PortD

Address of the optional subsequent EtherCAT slave at port D of the current EtherCAT slave

```
public ushort PortD
```

Field Value

[ushort](#)

Struct ST_Topo

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The structure ST_Topo

```
public struct ST_Topo
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ST_Topo

Constructor for ST_Topo from a byte array of length 64.

```
public ST_Topo( byte[] bytes )
```

Parameters

bytes [byte](#)[]

Exceptions

[Exception](#)

Fields

aReserved1

Reserved for future use

```
public uint[] aReserved1
```

Field Value

[uint](#)[]

aReserved2

Reserved for future use

```
public uint[] aReserved2
```

Field Value

[uint](#)[]

nHCSlaveCountAct

Found number of Hot Connect group devices

```
public ushort nHCSlaveCountAct
```

Field Value

[ushort](#)[]

nHCSlaveCountCfg

Configured number of Hot Connect group devices

```
public ushort nHCSlaveCountCfg
```

Field Value

[ushort](#)[]

nOwnAutoIncAddr

Dedicated auto-increment EtherCAT address of the EtherCAT slave device

```
public ushort nOwnAutoIncAddr
```

Field Value

[ushort](#)

nOwnPhysicalAddr

Dedicated physical EtherCAT address of the EtherCAT slave device

```
public ushort nOwnPhysicalAddr
```

Field Value

[ushort](#)

nStatusBits

Status bits of the EtherCAT slave device

```
public uint nStatusBits
```

Field Value

[uint](#)

stAutoIncAddr

Auto-increment address information of the EtherCAT slave devices at port A...D

```
public ST_PortAddr stAutoIncAddr
```

Field Value

[ST_PortAddr](#)

stPhysicalAddr

Physical and auto-increment address information of the EtherCAT slave devices at port A..D

```
public ST_PortAddr stPhysicalAddr
```

Field Value

[ST_PortAddr](#)

Class Scope

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Class to interact with a TwinCAT 2 scope. Note: not compatible with TwinCAT 3.

```
public class Scope
```

Inheritance

[object](#) ← Scope

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

Scope(AmsNetId)

Creates a new instance of the Scope class. The AmsNetId should typically point to a TwinCAT 2 runtime.

```
public Scope(AmsNetId amsNetId)
```

Parameters

amsNetId AmsNetId

Properties

OnlineMode

Gets or sets the online mode of the scope. If true, the scope is online and can be triggered. If false, the scope is offline.

```
public bool OnlineMode { get; set; }
```

Property Value

bool ↗

Methods

LoadConfigurationFile(string)

Load *.scp File (Scope Configuration Project)

```
public void LoadConfigurationFile(string filename)
```

Parameters

filename string ↗

E.g. D:\TwinCAT\scope\achse2.scp

ManualTrigger()

Issuing this command triggers the Scope. It must, however, be online.

```
public void ManualTrigger()
```

Class SumReader

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

Using the ADS Sum Command it is possible to read or write several variables in one command.

```
public class SumReader
```

Inheritance

[object](#) ← SumReader

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

SumReader(AdsClient)

Creates a new sum reader. Used for reading multiple variables in one ADS command.

```
public SumReader(AdsClient client)
```

Parameters

client AdsClient

Properties

AdsErrorCodes

The ADS return codes for the variables that were read.

```
public AdsErrorCode[] AdsErrorCodes { get; }
```

Property Value

AdsErrorCode[]

Methods

AddVariable(string, Type)

Add a variable based on symbol name to this sum reader.

```
public void AddVariable(string symbolName, Type type)
```

Parameters

symbolName [string](#)

type [Type](#)

Exceptions

[NotImplementedException](#)

AddVariable(uint, uint, Type)

Add a variable based on index group and index offset to this sum reader.

```
public void AddVariable(uint indexGroup, uint indexOffset, Type type)
```

Parameters

indexGroup [uint](#)

indexOffset [uint](#)

type [Type](#)

ReadVariables(out byte[])

Reads the added variables. The values are stored in the `readValues` array in the order they were added. If the operation is not successful, `false` is returned and the field "AdsErrorCodes" will contain the error codes for the respective variable.

```
public bool ReadVariables(out byte[] returnedValues)
```

Parameters

`returnedValues` `byte[]`

Returns

`bool`

If the operation was successful without ADS errors. If not successful, user should examine the [AdsError Codes](#)

Class TcSystem

Namespace: [TwinSharp](#)

Assembly: TwinSharp.dll

The TcSystem class represents a TwinCAT system and provides methods to interact with it. It allows switching the system to different modes (Config, Restart, Stop), listing EtherCAT masters, and listing local static routes. It also provides access to the system's real-time properties, license information, and file system through the Realtime, License, and FileSystem properties respectively.

```
public class TcSystem
```

Inheritance

[object](#) ← TcSystem

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

TcSystem(AmsNetId)

Creates a representation of a TwinCAT system.

```
public TcSystem(AmsNetId target)
```

Parameters

target AmsNetId

Use AmsNetId.Local for a local system.

Properties

FileSystem

Gets the FileSystem object which provides access to the TwinCAT system's file system. Can be used to read and delete files on the remote system.

```
public FileSystem FileSystem { get; }
```

Property Value

[FileSystem](#)

License

Gets the License object which provides access to the TwinCAT system's license information.

```
public License License { get; }
```

Property Value

[License](#)

Realtime

Gets the Realtime object which provides access to the TwinCAT system's real-time properties.

```
public Realtime Realtime { get; }
```

Property Value

[Realtime](#)

Methods

ListEtherCatMasters()

Lists all the existing EtherCAT masters on the TwinCAT system.

```
public EtherCatMaster[] ListEtherCatMasters()
```

Returns

[EtherCatMaster\[\]](#)

ListLocalStaticRoutes()

List all the existing static routes on the local TwinCAT system.

```
public static AmsRoute[] ListLocalStaticRoutes()
```

Returns

[AmsRoute\[\]](#)

Exceptions

[FileNotFoundException](#)

Restart()

Can be used to restart the TwinCAT system. Corresponds to the Restart command on the TwinCAT system menu (on the right of the Windows taskbar). Restarting the TwinCAT system involves the TwinCAT system first being stopped, and then immediately started again

```
public void Restart()
```

Stop()

Can be used to stop the TwinCAT system. The function corresponds to the Stop command on the TwinCAT system menu (on the right of the Windows taskbar).

```
public void Stop()
```

SwitchToConfigMode()

A TwinCAT system in RUN mode (green TwinCAT system icon) can be switched to CONFIG mode (blue TwinCAT system icon) via the function block "TC_Config". If the system is already in CONFIG mode, it is first switched to STOP mode (red TwinCAT system icon) and then to CONFIG mode.

```
public void SwitchToConfigMode()
```

Namespace TwinSharp.CNC

Classes

[AxisStatus](#)

Axis status as taken from HLI.

https://infosys.beckhoff.com/content/1033/tf5200_hli_interface/174749195.html?id=8719845080216701702

[AxisStatusInChannel](#)

If an axis belongs to a channel, this class describes the status of that axis in that channel.

[AxisStatusInChannelAdresses](#)

[BackwardsOnPath](#)

The function for forward/backward motion on the path permits backward motion along the originally programmed path by means of a real-time signal when the NC program is active. Backward motion is terminated by resetting the real-time signal. Forward motion is then resumed.

[BlockSearch](#)

The operator can start machining at what is called the continuation position at any point in the program. After a program is interrupted (e.g. tool breakage), this is a quick method to reactivate machining at the point of interruption. The continuation position can be defined using a number of different block search types(file offset, block counter, block number, etc.).

[CNC](#)

Represents the TwinCAT CNC system. This class provides access to the CNC's platform, axes, and channels,

[CncAxis](#)

Represents a CNC axis and provides access to its status, external commanding, and dynamic position limitation. This class allows interaction with the axis to read its state, command movements, and set position limits.

[CncChannel](#)

The CncChannel class represents a CNC channel and provides various functionalities to manage and control the CNC operations. It includes properties and methods to interact with different aspects of the CNC, such as operation modes, the interpolator, control commands, contour visualization, feed override, path motion, single block mode, block search, distance to go, data streaming, technology processes, error management, zero offsets, and manual operation. The class also provides properties to read the covered block motion percentage, covered path distance, line count of the NC program, and path feed rates.

[CncPlatform](#)

Platform data is data which cannot be assigned to a specific axis or a channel but has an effect on the entire NC control. It allows reading various properties of the CNC platform such as the tick counter, cycle time, version, number of axes, number of channels, and their respective maximum allowable counts.

[Constants](#)

Constants used in the TwinCAT CNC system.

[ContourVisualization](#)

The controller can supply the axis positions for the graphic display of machine movements and visualise them by means of a user program or in the graphic user interface.

[ControlCommands](#)

The ControlCommands class provides methods to interact with and control various aspects of the CNC. It allows for the activation and deactivation of skip modes, reading and setting the current and commanded channel modes, and enabling or disabling optional stops during NC program execution.

[DataStreaming](#)

With the incremental specification of motion commands (streaming), the CAD/CAM system or the PLC stipulates the next path segment to be travelled (or even several segments). In this way, motion information not previously specified can still be modified until shortly before entering the command.

[DeleteDistanceToGo](#)

The "Delete distance to go" function interrupts the actual path motion and starts a short cut by straight line to the target position of next block. The distance to go of the current (interrupted) block is then deleted.

[DynamicPositionLimitation](#)

Additionally needs to be activated in the startup parameters by using the keyword FCT_DYN_POS_LIMIT of the parameter P-STUP-00070

[ErrorCodes](#)

Lists all the known error codes and descriptions that can be returned by the CNC system.

[ErrorManagement](#)

The ErrorManagement class is responsible for handling error messages from the CNC. It subscribes to error notifications, reads error messages, and provides functions to acknowledge them. When an error is received, it triggers the ErrorReceived event with detailed error information.

[ErrorRecievedEventArgs](#)

Event arguments for when an error is received. Contains the error information and a description of the error.

[Extensions](#)

Contains extension methods for various classes.

[ExternalAxisCommanding](#)

Specifying velocity or position command values by the PLC effective in addition to the interpolator. No monitoring takes place of transferred values for compliance with the dynamic axis limits. To activate this interface, set the parameter P-AXIS-00732 to 1.

[FeedOverride](#)

Provides properties to read and control the feed override of a channel.

[FeedOverrideAdresses](#)

Contains the addresses of the feed override variables. Can be used to add device notifications, sum read commands etc.

[HandWheelInc](#)

Control unit to manage the count of handwheel increments for a handwheel index, including flow control of user data.

[HcodeNeedsAcknowledgeEventArgs](#)

Event args supplied when a H code needs to be acknowledged.

[HrParameters](#)

Control unit to manage data for parameterising handwheel mode in manual mode, including flow control of user data

[Interpolator](#)

The Interpolator class provides an interface to interact with a CNC interpolator via an AdsClient. It allows reading and writing various properties related to the interpolator's state, such as axis count, moved path, velocity, tool life, and more. The class also provides addresses for device notifications and methods to retrieve interpolator addresses and properties.

[InterpolatorAdresses](#)

Addresses for the interpolator class.

[JogParameters](#)

Control unit to manage data for parameterising incremental jog mode in manual mode, including flow control of user data

[Key](#)

Control unit to manage data to enforce a button press in manual mode, including flow control of user data.

[ManualOperation](#)

Class that contains settings and functions for the manual operation of a CNC machine. Such as keys, rapid keys and hand wheels.

[McodeNeedsAcknowledgeEventArgs](#)

Event args supplied when a M code needs to be acknowledged.

[ObjectDescription](#)

Represents the description of a TASK COM object.

[OperationModeAdresses](#)

This class contains the addresses of objects used by the operation mode manager.

[OperationModeManager](#)

The OperationModeManager class provides an interface to manage and control the operation modes and states of the CNC via an AdsClient. It allows setting and retrieving the current operation mode and state, resetting the operation mode, and managing the interface existence.

[RapidKey](#)

During continuous jog mode it is possible to switch between the normal velocity and rapid traverse velocity. Here the rapid traverse is a button-specific feature and only becomes effective when the corresponding button is pushed and linked to an axis.

[SingleBlock](#)

When single-step mode is active, the machine operator has the option to execute an NC program step by step. The operator releases every NC line one by one. Comment lines or comment blocks and skipped blocks are skipped.

[TechnologyProcesses](#)

The TechnologyProcesses class manages the interaction with CNC technology units via an ADS client. It handles the acknowledgment of M and H codes, processes notifications from the CNC, and provides methods to read and acknowledge technology units.

[TipParameters](#)

Control unit to manage data for parameterising continuous jog mode in manual mode, including flow control of user data.

[ZeroOffsets](#)

The ZeroOffsets class is responsible for handling zero offsets (G54 etc) in the CNC system.

Structs

[HLI_ADD_CMD_VALUE](#)

Specifying velocity or position command values by the SPS effective in addition to the interpolator. No monitoring takes place of transferred values for compliance with the dynamic axis limits. To activate

this interface, set the parameter P- AXIS-00732 to 1.

[HLI_ERROR_MASKE](#)

Error mask.

[HLI_ERROR_SATZ](#)

This structure contains user data of an error message.

[HLI_ERROR_SATZ_KOPF](#)

This structure contains user data of an error message.

[HLI_ERROR_SATZ_RUMPF](#)

Body portion of that an error message.

[HLI_FB_ZEITANGABE](#)

Time specification on output of an error message.

[HLI_HB_ACTIVATION](#)

Control unit to manage data to activate a control element and assign it to an axis in manual mode.

[HLI_HB_HR_PARAMETER](#)

Control unit to manage data for parameterising handwheel mode in manual mode.

[HLI_HB_JOG_PARAMETER](#)

Control unit to manage data for parameterising incremental jog mode in manual mode.

[HLI_HB_KEY](#)

Control unit to manage data to enforce a button press in manual mode,

[HLI_HB_RAPID_KEY](#)

Control unit to activate/deactivate rapid traverse mode by a normal button press in manual mode.

[HLI_HB_TIP_PARAMETER](#)

Control unit to manage data for parameterising continuous jog mode in manual mode

[HLI_IMCM_MODE_STATE](#)

Struct for operation mode management that combines mode and state into one struct.

[HLI_INTF_VERSION_NAME](#)

Structure that contains the CNC version name.

[HLI_MODUL_NAME](#)

Structure that contains the name of one module.

[HLI_M_H_PROZESS](#)

Further describes the TECHNO_UNIT_CH struct when it is a M or H function.

HLI PROC TRANS TO MODE STATE

Control unit to switch over the operation mode and poll the current state of operation mode management, including flow control of user data.

HLI RUMPF NC PROG

Error information in relation to NC program.

HLI_WERT

Error value.

HLI_WERT_B

Additional error information.

HLI_WERT_B_DATA

Contains the actual data of an error message.

HLI_WERT_DATA

Error value data.

SOLLKONT VISU ACHS DATA STD

Struct that contains axis-specific visualisation data.

SOLLKONT VISU CH DATA STD

Struct that contains standard visualisation data.

SOLLKONT VISU DATA V0

Version 0 format of visualisation data.

SOLLKONT VISU PDU

Container that hold visualization data in any of the 11 existing versions.

TECHNO_UNIT_CH

A technology control unit contains elements for commanding, acknowledging and transferring any required parameters

Enums

AxisState

Enumeration that lists the possible states of an axis.

BlockSearchType

The block search type defines the method used to define the continuation position in the NC program.

ChannelMode

Enum for selection of a special channel mode such as syntax check or machining time calculation

HLI_DYNPL_STATE

Enumeration of the states of the dynamic position limitation.

OperationMode

The CNC distinguishes between five operating modes. It is possible to switch over between these operating modes via the operator-control and/or PLC interface, whereby only one operating mode may be active at any one time. This enumeration lists the operating modes that are defined.

OperationState

Enumeration of the possible states of an operation mode. Depending on the actually selected operation mode, these states may contain a further meaning.

SkipModes

Enumeration that describes the skip mode at interpreter level. Skip levels active simultaneously are enabled by bitwise ORing.

TechnologyFunction

Enum for the selection of a technology function (M/H/S/T)

Enum AxisState

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Enumeration that lists the possible states of an axis.

```
public enum AxisState : uint
```

Fields

HLI_AXIS_ACTIVE = 3

The axis is currently moved by the CNC due to an NC command or manual mode.

HLI_AXIS_ERROR = 7

After an error (in the drive or CNC, e.g. a software limit switch violation) the axis is in error state.

Commanding a new motion is only possible after a CNC reset.

HLI_AXIS_HOLD = 5

The CNC cannot move the axis because an external signal is set, such as feedhold or tracking mode, or the required drive enables are missing.

HLI_AXIS_READY = 1

The axis is ready and moves according to the specified command values after a command.

Class AxisStatus

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Axis status as taken from HLI.

https://infosys.beckhoff.com/content/1033/tf5200_hli_interface/174749195.html?id=8719845080216701702

```
public class AxisStatus
```

Inheritance

[object](#) ← AxisStatus

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActiveFeedrate

Active feedrate of the axis in mm/s.

```
public double ActiveFeedrate { get; }
```

Property Value

[double](#)

ActualPositionACS

Actual position of the current cycle in the axis coordinate system

```
public int ActualPositionACS { get; }
```

Property Value

[int](#)

AxisName

Name of the logical axis used for the current reference in the current automatic program / manual block(e.g.X, Y, Z). This may be changed by default when the channel(SDA-MDS list) is programmed or dynamically in the NC program by means of a swap command.

```
public string AxisName { get; }
```

Property Value

[string](#)

ChannelNumber

Return the number of the channel to which the axis is assigned, or 0 if unassigned.

```
public ushort ChannelNumber { get; }
```

Property Value

[ushort](#)

CommandedPositionACS

Command position of current cycle in the axis coordinate system.

```
public int CommandedPositionACS { get; }
```

Property Value

[int](#)

CurrentFeedrate

Current feedrate of the axis in mm/s.

```
public double CurrentFeedrate { get; }
```

Property Value

[double](#)

Direction

Positive sign if the last output setpoint generated led to a positive direction of motion. Negative sign if the last output setpoint generated led to a negative direction of motion. 0 if stationary.

```
public sbyte Direction { get; }
```

Property Value

[sbyte](#)

EndPositionACS

Target position in the current NC block, ACS. This represents the target position of the program coordinate system referred to the axes. It is valid only as long as no transformation is active. Currently, the target position is not transformed back onto the axes.

```
public int EndPositionACS { get; }
```

Property Value

[int](#)

HomingDone

The axis completed homing successfully and is now referenced.

```
public bool HomingDone { get; }
```

Property Value

[bool](#) ↗

InPosition

The axis is located in position, i.e. the control window is reached and the axis is not interpolated

```
public bool InPosition { get; }
```

Property Value

[bool](#) ↗

Mode

The axis mode configured in the axis parameter list (P-AXIS-00015) is indicated.

```
public uint Mode { get; }
```

Property Value

[uint](#) ↗

State

Even if an axis is not moved in the PCS, a corresponding Cartesian or kinematic transformation may nevertheless execute a motion of the physical axis. Example: 90° rotation about Z; Y is moved if X is programmed.

```
public AxisState State { get; }
```

Property Value

StatusInChannel

If the axis belongs to a channel, this class can be used to get the status of the axis in that channel.

```
public AxisStatusInChannel? StatusInChannel { get; }
```

Property Value

[AxisStatusInChannel](#)

Type

Type of axis. 1 = Translator. 2 = Rotator. 4 = Spindle.

```
public ushort Type { get; }
```

Property Value

[ushort](#) ↗

Class AxisStatusInChannel

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

If an axis belongs to a channel, this class describes the status of that axis in that channel.

```
public class AxisStatusInChannel
```

Inheritance

[object](#) ← AxisStatusInChannel

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualPositionPCS

Actual ACS position converted in the PCS.

```
public int ActualPositionPCS { get; }
```

Property Value

[int](#)

Adresses

Contains the index group and index offsets of Axis Channel Status objects. Can be used if you want to add ADS Device notifications, or SumRead commands.

```
public AxisStatusInChannelAdresses Adresses { get; }
```

Property Value

[AxisStatusInChannelAdresses](#)

AxisStatePCS

Axis state in the PCS.

```
public AxisState AxisStatePCS { get; }
```

Property Value

[AxisState](#)

CommandedPositionPCS

Position preset in the current cycle as setpoint.

```
public int CommandedPositionPCS { get; }
```

Property Value

[int](#)

ContinuousSpeed

Continuous speed.

```
public uint ContinuousSpeed { get; }
```

Property Value

[uint](#)

ControlElement

Control element.

```
public ushort ControlElement { get; }
```

Property Value

[ushort](#)

DistanceToGoPCS

Distance to go in the current NC block, difference between target position and command position.

```
public int DistanceToGoPCS { get; }
```

Property Value

[int](#)

EndPositionPCS

Target position of the current NC block.

```
public int EndPositionPCS { get; }
```

Property Value

[int](#)

HandwheelResolution

Handwheel resolution.

```
public double HandwheelResolution { get; }
```

Property Value

[double](#)

HomingDone

The axis completed homing successfully and is now referenced.

```
public bool HomingDone { get; }
```

Property Value

[bool](#)

IncrementalDistance

Incremental distance.

```
public uint IncrementalDistance { get; }
```

Property Value

[uint](#)

IncrementalSpeed

Incremental speed.

```
public uint IncrementalSpeed { get; }
```

Property Value

[uint](#)

LogicalNumber

Logical number of the axis.

```
public ushort LogicalNumber { get; }
```

Property Value

[ushort](#) ↗

ManualState

Manual state.

```
public ushort ManualState { get; }
```

Property Value

[ushort](#) ↗

Name

Name of the axis.

```
public string Name { get; }
```

Property Value

[string](#) ↗

OperationMode

Operation mode.

```
public ushort OperationMode { get; }
```

Property Value

[ushort](#) ↗

Type

Type of axis. 1 = Translator. 2 = Rotator. 4 = Spindle.

```
public ushort Type { get; }
```

Property Value

[ushort](#) ↗

Class AxisStatusInChannelAdresses

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

```
public class AxisStatusInChannelAdresses
```

Inheritance

[object](#) ← AxisStatusInChannelAdresses

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualPositionPCS

Sub index group of the actual position in the PCS.

```
public uint ActualPositionPCS { get; }
```

Property Value

[uint](#)

IndexGroup

Index group for all Axis Channel Status objects.

```
public uint IndexGroup { get; }
```

Property Value

[uint](#)

Class BackwardsOnPath

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The function for forward/backward motion on the path permits backward motion along the originally programmed path by means of a real-time signal when the NC program is active. Backward motion is terminated by resetting the real-time signal. Forward motion is then resumed.

```
public class BackwardsOnPath
```

Inheritance

[object](#) ← BackwardsOnPath

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActivateExternalPathVelocity

Activate the velocity commanded in the ext_command_speed control unit. To reach the commanded velocity, all axes involved in the motion are accelerated or decelerated. If this value is TRUE, the sign is considered in the current path feed (active_feed_r control unit).

```
public bool ActivateExternalPathVelocity { get; set; }
```

Property Value

[bool](#)

EnableInterface

Signal to CNC that we want to use this interface.

```
public bool EnableInterface { set; }
```

Property Value

[bool](#) ↗

ExternalPathVelocity

External path velocity specified. The path velocity setting is activated by the control unit ext_command_speed_valid. If the velocity specified in negative, the tool moves backwards along the path
Unit: 1 μm/s

```
public uint ExternalPathVelocity { get; set; }
```

Property Value

[uint](#) ↗

ResetBackwardMotionMemory

Deselects backward motion memory No further NC block is saved in the memory. The memory is deleted. The backward motion memory can only be cleared if no NC program is active.

```
public bool ResetBackwardMotionMemory { get; set; }
```

Property Value

[bool](#) ↗

SelectBackwardMotion

Select/deselect backward motion on the path In basic setting, M/H functions are executed without synchronisation(MOS) in this mode.

```
public bool SelectBackwardMotion { get; set; }
```

Property Value

[bool](#) ↗

SelectSimulatedForwardMotion

Select/deselect simulated forward motion on the path In basic setting, M/H functions are executed without synchronisation (MOS) in this mode. Sections in the NC program can be skipped during program runtime in combination with the NC command #OPTIONAL EXECUTION.

```
public bool SelectSimulatedForwardMotion { get; set; }
```

Property Value

[bool](#) ↗

Class BlockSearch

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The operator can start machining at what is called the continuation position at any point in the program. After a program is interrupted (e.g. tool breakage), this is a quick method to reactivate machining at the point of interruption. The continuation position can be defined using a number of different block search types(file offset, block counter, block number, etc.).

```
public class BlockSearch
```

Inheritance

[object](#) ← BlockSearch

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AutomaticResumption

This object defines that resumption of motion on the contour occurs automatically.

```
public bool AutomaticResumption { get; set; }
```

Property Value

[bool](#)

BlockNumberToFind

Defines the block number at which point actual machining is to continue.

```
public uint BlockNumberToFind { get; set; }
```

Property Value

[uint](#)

BlockSearchType

The continuation position can be defined using a number of different block search types (file offset, block counter, block number, etc.)

```
public BlockSearchType BlockSearchType { get; set; }
```

Property Value

[BlockSearchType](#)

BreakPoint

This object defines an automatic breakpoint by specifying the distance from program start. Unit [0.1 µm]

```
public double BreakPoint { get; set; }
```

Property Value

[double](#)

CoveredDistanceFromProgramStart

This object defines the distance from program start or #DISTANCE PROG START CLEAR at which machining is actually supposed to start. Unit [0.1 µm]

```
public double CoveredDistanceFromProgramStart { get; set; }
```

Property Value

[double](#)

CoveredDistanceMotionBlockInPerMill

This object defines the distance covered in the NC block in per mil at which machining is actually supposed to continue. The first part of the block in the block search is then executed without motion and only the remaining part is executed with moved axes. Value range: [0.0 to 1000.0]; default value= 0.0 Unit [0.1%]

```
public double CoveredDistanceMotionBlockInPerMill { get; set; }
```

Property Value

[double](#) ↗

MaxPathDeviation

This object defines the maximum deviation of the axes between actual position and continuation position when machining is resumed after block search. If resumption of motion on the contour is automatic, the maximum path deviation is not considered since the exact continuation position has already been reached. (Default= 0) Unit [0.1 µm]

```
public uint MaxPathDeviation { get; set; }
```

Property Value

[uint](#) ↗

NoStopOnResumption

This object defines whether resumption of motion on the contour should occur directly without any operator input.

```
public bool NoStopOnResumption { get; set; }
```

Property Value

[bool](#) ↗

ProgramStartAsOfFileOffset

The file offset defines a jump to a known position in the NC program. The program part before the jump point is not evaluated. Processing starts at the jump point as for a program shortened by file offset. This object defines the file offset. (Default value= 0)

```
public int ProgramStartAsOfFileOffset { set; }
```

Property Value

[int](#)

Enum BlockSearchType

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The block search type defines the method used to define the continuation position in the NC program.

```
public enum BlockSearchType
```

Fields

BlockCounter = 3

Continuation position by block counter.

BlockNumber = 4

Continuation position by block number and program name.

FileOffset = 1

Continuation position and end position by file offset.

NoBlockSearch = 0

No block search. Normal operation.

ProgramEnd = 5

Continuation position at program end This special block search type is used in particular in job planning on a simulation system for a rapid test of NC programs.

Class CNC

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Represents the TwinCAT CNC system. This class provides access to the CNC's platform, axes, and channels,

```
public class CNC : IDisposable
```

Inheritance

[object](#) ← CNC

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

CNC(AmsNetId)

Creates a new representation of a existing TwinCAT CNC system at the specified target AmsNetId.

```
public CNC(AmsNetId target)
```

Parameters

target AmsNetId

Properties

Axes

Array of all existing CNC axes.

```
public CncAxis[] Axes { get; }
```

Property Value

[CncAxis\[\]](#)

Channels

Array of all existing CNC channels.

```
public CncChannel[] Channels { get; }
```

Property Value

[CncChannel\[\]](#)

ClientCom

ADS client connected to the "COM CNC task" deals with the communication integration of the CNC kernel

```
public AdsClient ClientCom { get; }
```

Property Value

AdsClient

ClientGeo

ADS client connected to the GEO CNC. Operates in the interpolation cycle of the CNC. Among other tasks, it calculates the respective set values for the axes and controls the axes.

```
public AdsClient ClientGeo { get; }
```

Property Value

AdsClient

ClientPlc

ADS Client connected to the PLC. Used by HLI interface.

```
public AdsClient ClientPlc { get; }
```

Property Value

AdsClient

ClientSda

ADS client connected to the "SDA CNC task" deals with decoding and processing of the NC programs and should therefore have a lower priority than the GEO task.

```
public AdsClient ClientSda { get; }
```

Property Value

AdsClient

Platform

Platform data is data which cannot be assigned to a specific axis or a channel but has an effect on the entire NC control.

```
public CncPlatform Platform { get; }
```

Property Value

[CncPlatform](#)

Methods

Dispose()

Dispose the CNC object. Releases ADS handles internally.

```
public void Dispose()
```

Enum ChannelMode

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Enum for selection of a special channel mode such as syntax check or machining time calculation

```
public enum ChannelMode
```

Fields

ADD_MDI_BLOCK = 128

Extended manual block mode: the end of a manual block is not evaluated as a program end. It permits the commanding of further manual blocks.

BEARB_MODE_SCENE = 4096

When SCENE mode is enabled, the output of #SCENE commands is activated on the interface (see also [FCT-C17// Scene contour visualisation]). An additional client is linked to this output via DataFactory / CORBA.

ISG_STANDARD = 0

Normal mode

KIN_TRAFO_OFF = 256

Overwrites automatic enable for kinematic transformations by a characteristic parameter defined in the channel parameters (sda_mds*.lis).

MACHINE_LOCK = 64

Dry run without axis motion

ONLINE_PROD_TIME = 32

Simulation online machining time calculation

ON_LINE = 4

Online visualisation simulation

PROD_TIME = 16

Simulation machining time calculation (in TwinCAT without function)

SOLLKON_BlockSearch = 1

Block search

SOLLKON_NominalContourVisualisation = 2

Nominal contour visualisation simulation with output of visualisation data

SOLLKON_SuppressOutput = 2050

Nominal contour visualisation simulation without output of visualisation data

SUPPRESS_TECHNO_OUTPUT = 8192

Without output of technology functions (M/H/T). Set implicitly in connection with syntax check.

SYNCHK = 8

Syntax check simulation

Class CncAxis

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Represents a CNC axis and provides access to its status, external commanding, and dynamic position limitation. This class allows interaction with the axis to read its state, command movements, and set position limits.

```
public class CncAxis
```

Inheritance

[object](#) ← CncAxis

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Properties

DynamicPositionLimitation

Gets the DynamicPositionLimitation instance which allows setting and monitoring dynamic position limits for the axis.

```
public DynamicPositionLimitation DynamicPositionLimitation { get; }
```

Property Value

[DynamicPositionLimitation](#)

ExternalAxisCommanding

Gets the ExternalAxisCommanding instance which allows specifying additive velocity or position command values.

```
public ExternalAxisCommanding ExternalAxisCommanding { get; }
```

Property Value

[ExternalAxisCommanding](#)

Status

Gets the AxisStatus instance which provides access to various properties and methods to read and manipulate the state of the axis, including position, velocity, acceleration, torque, and error codes.

```
public AxisStatus Status { get; }
```

Property Value

[AxisStatus](#)

Methods

ToString()

Returns a string representation of the axis.

```
public override string ToString()
```

Returns

[string](#) ↗

Class CncChannel

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The CncChannel class represents a CNC channel and provides various functionalities to manage and control the CNC operations. It includes properties and methods to interact with different aspects of the CNC, such as operation modes, the interpolator, control commands, contour visualization, feed override, path motion, single block mode, block search, distance to go, data streaming, technology processes, error management, zero offsets, and manual operation. The class also provides properties to read the covered block motion percentage, covered path distance, line count of the NC program, and path feed rates.

```
public class CncChannel : IDisposable
```

Inheritance

[object](#) ← CncChannel

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

Descriptions

Gets a dictionary with object descriptions from the COM task.

```
public readonly Dictionary<string, ObjectDescription> Descriptions
```

Field Value

[Dictionary](#)<[string](#), [ObjectDescription](#)>

Properties

BackwardsOnPath

Provides functions for forward/backward motion on the path. Permits backward motion along the originally programmed path by means of a real-time signal when the NC program is active.

```
public BackwardsOnPath BackwardsOnPath { get; }
```

Property Value

[BackwardsOnPath](#)

BlockSearch

Provides a tool so the operator can start machining at what is called the continuation position at any point in the program. After a program is interrupted (e.g. tool breakage), this is a quick method to reactivate machining at the point of interruption. The continuation position can be defined using a number of different block search types(file offset, block counter, block number, etc.).

```
public BlockSearch BlockSearch { get; }
```

Property Value

[BlockSearch](#)

ChannelNumber

The unique channel number of this CNC channel.

```
public int ChannelNumber { get; }
```

Property Value

[int↗](#)

ContourVisualization

Provides tools to visualize the contour of NC programs.

```
public ContourVisualization ContourVisualization { get; }
```

Property Value

[ContourVisualization](#)

ControlCommands

Provides functions to control the CNC channel. Such as enabling and disabling skip modes. Control of optional stop.

```
public ControlCommands ControlCommands { get; }
```

Property Value

[ControlCommands](#)

CoveredBlockMotionPercent

Part of the path motion traversed in the current block in relation to the total path. If a main axis participates in the motion, the covered path motion is in relation to the block path of the first three axes. If no main axis participates in the motion, the covered path motion is the position lag with the longest motion time in relation to the block path.

```
public double CoveredBlockMotionPercent { get; }
```

Property Value

[double](#)

CoveredPathDistance

Reads the current distance covered in the NC program since program start or since the last # DISTANCE PROG START CLEAR NC command. The calculation is based on the current position in the current NC block. Unit: 0.1 µm

```
public double CoveredPathDistance { get; }
```

Property Value

[double](#)

DataStreaming

Provides a tool to use Data Streaming. With the incremental specification of motion commands (streaming), the CAD/CAM system or the PLC stipulates the next path segment to be travelled (or even several segments). In this way, motion information not previously specified can still be modified until shortly before entering the command.

```
public DataStreaming DataStreaming { get; }
```

Property Value

[DataStreaming](#)

DeleteDistanceToGo

Provides function to control the “Delete distance to go” function. It interrupts the actual path motion and starts a short cut by straight line to the target position of next block. The distance to go of the current (interrupted) block is then deleted.

```
public DeleteDistanceToGo DeleteDistanceToGo { get; }
```

Property Value

[DeleteDistanceToGo](#)

ErrorManager

Gets an object useful for error management. You can subscribe to receive error notifications, reads error messages, and provides functions to acknowledge error. When an error is received, it triggers the ErrorReceived event with detailed error information.

```
public ErrorManagement ErrorManager { get; }
```

Property Value

[ErrorManagement](#)

FeedOverride

Provides tools to read and control the feed override of this channel.

```
public FeedOverride FeedOverride { get; }
```

Property Value

[FeedOverride](#)

Interpolator

Provides functions to control the interpolator of this channel. Such as the interpolator's state, axis count, moved path, velocity, tool life, and more.

```
public Interpolator Interpolator { get; }
```

Property Value

[Interpolator](#)

LineCountNcProgram

The datum indicates the NC program line which is the source of the command just processed by the interpolator. The value is derived from the number of NC program lines which the decoder has read since the NC program started. All the lines read the decoder are counted, i.e.repeatedly read lines, empty

and comment lines. All commands to the interpolator resulting from decoding a NC program line are assigned to the associated line counter.

```
public uint LineCountNcProgram { get; }
```

Property Value

[uint](#)

ManualOperation

Provides a class that contains settings and functions for the manual operation of the CNC. Such as keys, rapid keys and hand wheels.

```
public ManualOperation ManualOperation { get; }
```

Property Value

[ManualOperation](#)

OperationModeManager

Gets an object that provides methods and properties to manage and control the operation modes and states

```
public OperationModeManager OperationModeManager { get; }
```

Property Value

[OperationModeManager](#)

PathFeedProgrammed

Path feed that was programmed by the F word in the NC program.. Unit: 1 µm/s

```
public double PathFeedProgrammed { get; }
```

Property Value

[double](#)

PathFeedProgrammedWeighted

Path feed was programmed in the NC program Weighted by the current real-time influences such as override. Unit: 1 µm/s

```
public double PathFeedProgrammedWeighted { get; }
```

Property Value

[double](#)

SingleBlock

Object that controls single-step mode. When it is active, the machine operator has the option to execute an NC program step by step. The operator releases every NC line one by one. Comment lines or comment blocks and skipped blocks are skipped

```
public SingleBlock SingleBlock { get; }
```

Property Value

[SingleBlock](#)

TechnologyProcesses

Gets an object that provides functions to subscribe to and acknowledge technology processes of this channel. Such as M, H, T and S functions/codes.

```
public TechnologyProcesses TechnologyProcesses { get; }
```

Property Value

ZeroOffsets

Provides functions to control the zero offsets of this channel. Such as setting, reading, and resetting zero offsets.

```
public ZeroOffsets ZeroOffsets { get; }
```

Property Value

[ZeroOffsets](#)

Methods

Dispose()

Dispose the CNC channel.

```
public void Dispose()
```

Class CncPlatform

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Platform data is data which cannot be assigned to a specific axis or a channel but has an effect on the entire NC control. It allows reading various properties of the CNC platform such as the tick counter, cycle time, version, number of axes, number of channels, and their respective maximum allowable counts.

```
public class CncPlatform
```

Inheritance

[object](#) ← CncPlatform

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AxisCount

Number of axes configured in the CNC platform.

```
public uint AxisCount { get; }
```

Property Value

[uint](#)

ChannelCount

Number of channels configured in the CNC platform.

```
public ushort ChannelCount { get; }
```

Property Value

[ushort](#) ↗

CycleTime

Cycle time of the CNC platform. Constant.

```
public uint CycleTime { get; }
```

Property Value

[uint](#) ↗

MaxAxisCount

Maximum allowable number of axes that can be configured in the CNC platform.

```
public uint MaxAxisCount { get; }
```

Property Value

[uint](#) ↗

MaxChannelCount

Maximum allowable number of channels that can be configured in the CNC platform.

```
public ushort MaxChannelCount { get; }
```

Property Value

[ushort](#) ↗

MaxSpindleCount

Maximum allowable number of spindles that can be configured in the CNC platform.

```
public uint MaxSpindleCount { get; }
```

Property Value

[uint](#)

TickCounter

Tick counter of the CNC platform. Always increasing when CNC is running.

```
public uint TickCounter { get; }
```

Property Value

[uint](#)

Version

Version of the CNC platform.

```
public string Version { get; }
```

Property Value

[string](#)

Class Constants

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Constants used in the TwinCAT CNC system.

```
public static class Constants
```

Inheritance

[object](#) ← Constants

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

ANZ_SIMU_KOORD

Number of axis to get coordinates for in graphic generation.

```
public const int ANZ_SIMU_KOORD = 32
```

Field Value

[int](#)

HLI_ERR_MASK_MAXIDX

Maximum index of HLI error mask.

```
public const int HLI_ERR_MASK_MAXIDX = 583
```

Field Value

[int](#)

HLI_HW_MAXIDX

Maximum count of number of hand wheels for manual jogging.

```
public const short HLI_HW_MAXIDX = 5
```

Field Value

[short](#)

HLI_KEY_MAXIDX

Maximum count of number of keys for manual jogging.

```
public const short HLI_KEY_MAXIDX = 11
```

Field Value

[short](#)

HLI_LAENGE_NAME

Maximum string length of HLI objects.

```
public const int HLI_LAENGE_NAME = 259
```

Field Value

[int](#)

HLI_MODUL_NAME_LAENGE

Maximum length of string of type module name.

```
public const int HLI_MODUL_NAME_LAENGE = 15
```

Field Value

[int ↗](#)

HLI_WERT_B_DATA_MAXIDX

Maximum index of HLI error value B data.

```
public const int HLI_WERT_B_DATA_MAXIDX = 23
```

Field Value

[int ↗](#)

HLI_WERT_DATA_MAXIDX

Maximum index of HLI error value data.

```
public const int HLI_WERT_DATA_MAXIDX = 7
```

Field Value

[int ↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V0

Maximum count of structure when V0 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V0 = 15
```

Field Value

[int ↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V1

Maximum count of structure when V1 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V1 = 10
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V10

Maximum count of structure when V10 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V10 = 4
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V11

Maximum count of structure when V11 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V11 = 3
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V2

Maximum count of structure when V2 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V2 = 5
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V3

Maximum count of structure when V3 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V3 = 10
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V4

Maximum count of structure when V4 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V4 = 7
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V5

Maximum count of structure when V5 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V5 = 4
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V6

Maximum count of structure when V6 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V6 = 7
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V7

Maximum count of structure when V7 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V7 = 6
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V8

Maximum count of structure when V8 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V8 = 4
```

Field Value

[int↗](#)

MAX_SOLLKONT_VISU_DATA_COUNT_V9

Maximum count of structure when V9 of visualization data is used.

```
public const int MAX_SOLLKONT_VISU_DATA_COUNT_V9 = 5
```

Field Value

int ↗

Class ContourVisualization

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The controller can supply the axis positions for the graphic display of machine movements and visualise them by means of a user program or in the graphic user interface.

```
public class ContourVisualization
```

Inheritance

[object](#) ← ContourVisualization

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

DataRecordChannelFIFO

Data record from channel-specific output buffer (FIFO).

```
public byte[] DataRecordChannelFIFO { get; }
```

Property Value

[byte](#)[]

DataRecordCountChannelFIFO

Number of data records in the channel-specific output FIFO

```
public uint DataRecordCountChannelFIFO { get; }
```

Property Value

[uint](#)

DataRecordCountGlobalFIFO

Number of data records in the global output FIFO

```
public uint DataRecordCountGlobalFIFO { get; }
```

Property Value

[uint](#)

ExecutionMode

Select nominal contour visualisation 0x0000 ISG_STANDARD Normal mode 0x0002 SOLLKON Nominal contour visualisation 0x0004 ON_LINE Online-Visu 0x0008 SYNCHK Syntax check

```
public ChannelMode ExecutionMode { get; set; }
```

Property Value

[ChannelMode](#)

MaxAbsolutePathError

Maximum absolute path error in [0.1 µm] for nominal contour visualisation of circles and polynomials

```
public double MaxAbsolutePathError { set; }
```

Property Value

[double](#)

MaxRelativePathError

Maximum relative path error in [0.1%] for nominal contour visualisation of circles or polynomials

```
public double MaxRelativePathError { set; }
```

Property Value

[double](#)

OutputGridSize

Output grid for nominal contour visualisation for linear blocks(G00/G01) in [0.1 µm]

```
public uint OutputGridSize { get; set; }
```

Property Value

[uint](#)

Methods

GetDataRecordFromGlobalFIFO(Memory<byte>)

Get data record from global output FIFO.

```
public int GetDataRecordFromGlobalFIFO(Memory<byte> bufferToFill)
```

Parameters

bufferToFill [Memory](#)<[byte](#)>

Returns

[int](#)

Class ControlCommands

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The ControlCommands class provides methods to interact with and control various aspects of the CNC. It allows for the activation and deactivation of skip modes, reading and setting the current and commanded channel modes, and enabling or disabling optional stops during NC program execution.

```
public class ControlCommands
```

Inheritance

[object](#) ← ControlCommands

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ChannelModeActive

Current special channel mode such as syntax check or machining time calculation

```
public ChannelMode ChannelModeActive { get; }
```

Property Value

[ChannelMode](#)

ChannelModeCommanded

Selection of a special channel mode such as syntax check or machining time calculation

```
public ChannelMode ChannelModeCommanded { get; set; }
```

Property Value

[ChannelMode](#)

OptionalStop

Activating/deactivating optional stop. If the function M01(optional stop) is programmed in the current block of the NC program, set this element to the value TRUE to stop at block end (ramped-down deceleration complying with the permissible accelerations). The following block can be enabled by activating the element "continue machining" if the NC kernel indicates that all axes are located within the control window by resetting the status flag wait_axes_in_position_r.

```
public bool OptionalStop { get; set; }
```

Property Value

[bool](#)

SkipMode

Activates/deactivates skip mode at interpreter level for the NC program. The status of skip mode is only evaluated at the start of the NC program. Switchover during execution of an NC program has no effect. Skip levels active simultaneously are enabled by bitwise ORing. Example: Enable all skip levels by setting 0x3FF.

```
public SkipModes SkipMode { get; set; }
```

Property Value

[SkipModes](#)

Class DataStreaming

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

With the incremental specification of motion commands (streaming), the CAD/CAM system or the PLC stipulates the next path segment to be travelled (or even several segments). In this way, motion information not previously specified can still be modified until shortly before entering the command.

```
public class DataStreaming
```

Inheritance

[object](#) ← DataStreaming

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Write(string)

This COM interface object can write the data stream with incremental NC commands. One complete NC line must always be written. Several NC lines may also be written jointly in one write access. Each NC line must be terminated by a carriage return (ASCII value = 13) and line feed (ASCII value = 10).

```
public void Write(string ncLines)
```

Parameters

ncLines [string](#)

Class DeleteDistanceToGo

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The "Delete distance to go" function interrupts the actual path motion and starts a short cut by straight line to the target position of next block. The distance to go of the current (interrupted) block is then deleted.

```
public class DeleteDistanceToGo
```

Inheritance

[object](#) ← DeleteDistanceToGo

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Delete

The rising edge of the commanded value has the effect that the CNC channel is decelerated to feed velocity 0. Then a linear motion is executed to the target position of the next motion block (short cut). The command only affects motion blocks.

```
public bool Delete { get; set; }
```

Property Value

[bool](#)

InterfaceExists

Signal to CNC that the interface exists and we want to use it.

```
public bool InterfaceExists { set; }
```

Property Value

[bool](#) ↗

Class DynamicPositionLimitation

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Additionally needs to be activated in the startup parameters by using the keyword FCT_DYN_POS_LIMIT of the parameter P-STUP-00070

```
public class DynamicPositionLimitation
```

Inheritance

[object](#) ← DynamicPositionLimitation

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

LowerPositionLimitMonitoringState

Gets the current state of the lower position limit monitoring.

```
public HLI_DYNPL_STATE LowerPositionLimitMonitoringState { get; }
```

Property Value

[HLI_DYNPL_STATE](#)

Methods

LowerPositionLimitInterfaceExists(bool)

Signal to CNC that the lower position limit interface exists and we want to use it.

```
public void LowerPositionLimitInterfaceExists(bool enabled)
```

Parameters

enabled [bool](#)

SetLowerPositionLimit(int)

Sets a position value and describes the lower limit of the position range which the axis should not exceed.

```
public void SetLowerPositionLimit(int position)
```

Parameters

position [int](#)

Unit: 0.1 μm or 10-4 °

Class ErrorCodes

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Lists all the known error codes and descriptions that can be returned by the CNC system.

```
public static class ErrorCodes
```

Inheritance

[object](#) ← ErrorCodes

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

CodesAndDescriptions

Dictionary with the error codes as keys and the error description as values.

```
public static readonly Dictionary<uint, string> CodesAndDescriptions
```

Field Value

[Dictionary](#) <[uint](#), [string](#)>

Class ErrorManagement

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The ErrorManagement class is responsible for handling error messages from the CNC. It subscribes to error notifications, reads error messages, and provides functions to acknowledge them. When an error is received, it triggers the ErrorRecieved event with detailed error information.

```
public class ErrorManagement
```

Inheritance

[object](#) ← ErrorManagement

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Events

ErrorRecieved

When an error is received, it triggers the ErrorRecieved event with detailed error information

```
public event EventHandler<ErrorRecievedEventArgs>? ErrorRecieved
```

Event Type

[EventHandler](#) <[ErrorRecievedEventArgs](#)>

Class ErrorRecievedEventArgs

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Event arguments for when an error is received. Contains the error information and a description of the error.

```
public class ErrorRecievedEventArgs
```

Inheritance

[object](#) ← ErrorRecievedEventArgs

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

Description

String description of the occured error.

```
public readonly string Description
```

Field Value

[string](#)

Error

Information about the occured error.

```
public readonly HLI_ERROR_SATZ_KOPF Error
```

Field Value

HLI ERROR SATZ KOPF

Class Extensions

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Contains extension methods for various classes.

```
public static class Extensions
```

Inheritance

[object](#) ← Extensions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

ByteArrayToStructure<T>(byte[])

Converts a byte array to a structure of the specified type.

```
public static T ByteArrayToStructure<T>(byte[] bytes) where T : struct
```

Parameters

bytes [byte](#)[]

Returns

T

Type Parameters

T

Exceptions

[InvalidOperationException](#)

Class ExternalAxisCommanding

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Specifying velocity or position command values by the PLC effective in addition to the interpolator. No monitoring takes place of transferred values for compliance with the dynamic axis limits. To activate this interface, set the parameter P-AXIS-00732 to 1.

```
public class ExternalAxisCommanding : IDisposable
```

Inheritance

[object](#) ← ExternalAxisCommanding

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AdditivePosition

Unit 0.1 μm

```
public int AdditivePosition { set; }
```

Property Value

[int](#)

AdditiveVelocity

Unit 1 μm/s

```
public int AdditiveVelocity { set; }
```

Property Value

[int ↗](#)

InterfaceExists

Signal to CNC that the interface exists and we want to use it.

```
public bool InterfaceExists { set; }
```

Property Value

[bool ↗](#)

Methods

Dispose()

Disposes of the ADS variable handles.

```
public void Dispose()
```

Class FeedOverride

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Provides properties to read and control the feed override of a channel.

```
public class FeedOverride
```

Inheritance

[object](#) ← FeedOverride

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualFeedOverride

Gets the actual feed override.

```
public ushort ActualFeedOverride { get; }
```

Property Value

[ushort](#)

Adresses

Provides the adresses of the feed override variables. So users can them to for example add ADS device notifications.

```
public FeedOverrideAdresses Adresses { get; }
```

Property Value

CommandedFeedOverride

Gets or sets the commanded feed override.

```
public ushort CommandedFeedOverride { get; set; }
```

Property Value

[ushort](#)

Class FeedOverrideAdresses

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Contains the addresses of the feed override variables. Can be used to add device notifications, sum read commands etc.

```
public class FeedOverrideAdresses
```

Inheritance

[object](#) ← FeedOverrideAdresses

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualFeedOverride

Actual feed override.

```
public uint ActualFeedOverride { get; }
```

Property Value

[uint](#)

IndexGroup

Index group for all properties.

```
public uint IndexGroup { get; }
```

Property Value

uint ↗

Struct HLI_ADD_CMD_VALUE

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Specifying velocity or position command values by the SPS effective in addition to the interpolator. No monitoring takes place of transferred values for compliance with the dynamic axis limits. To activate this interface, set the parameter P- AXIS-00732 to 1.

```
public struct HLI_ADD_CMD_VALUE
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

m_add_pos_value

Absolute value for additive position. Unit: 0,1 μm

```
public int m_add_pos_value
```

Field Value

[int](#)

m_add_speed_value

Value for additive velocity. 1 μm/s

```
public int m_add_speed_value
```

Field Value

[int](#)

sgn32_free_1

Reserved for future use.

```
public int sgn32_free_1
```

Field Value

[int ↗](#)

sgn32_free_2

Reserved for future use.

```
public int sgn32_free_2
```

Field Value

[int ↗](#)

sgn32_free_3

Reserved for future use.

```
public int sgn32_free_3
```

Field Value

[int ↗](#)

sgn32_free_4

Reserved for future use.

```
public int sgn32_free_4
```

Field Value

[int ↗](#)

sgn32_free_5

Reserved for future use.

```
public int sgn32_free_5
```

Field Value

[int ↗](#)

sgn32_free_6

Reserved for future use.

```
public int sgn32_free_6
```

Field Value

[int ↗](#)

Enum HLI_DYNPL_STATE

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Enumeration of the states of the dynamic position limitation.

```
public enum HLI_DYNPL_STATE
```

Fields

HLI_DYNPL_STATE_ACTIVATION = 1

This is the transition state after commanding the control unit until monitoring of axis position to the limit is activated.

HLI_DYNPL_STATE_ACTIVE = 2

The position limit is active and the axis position limit is monitored.

HLI_DYNPL_STATE_ACTIVE_BRAKE = 4

Deceleration process to maintain the position limit completed, axis is at standstill.

HLI_DYNPL_STATE_ACTIVE_BRACING = 3

A braking operation was initiated down to standstill to prevent the axis from exceeding the position limit.

HLI_DYNPL_STATE_INACTIVE = 0

The position limit is not active.

Struct HLI_ERROR_MASKE

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Error mask.

```
public struct HLI_ERROR_MASKE
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

ErrorMask

Data of error mask.

```
public byte[] ErrorMask
```

Field Value

[byte](#)[]

Struct HLI_ERROR_SATZ

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

This structure contains user data of an error message.

```
public struct HLI_ERROR_SATZ
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Body

Body portion of error message.

```
public HLI_ERROR_SATZ_RUMPF Body
```

Field Value

[HLI_ERROR_SATZ_RUMPF](#)

Head

Head portion of error message.

```
public HLI_ERROR_SATZ_KOPF Head
```

Field Value

[HLI_ERROR_SATZ_KOPF](#)

Struct HLI_ERROR_SATZ_KOPF

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

This structure contains user data of an error message.

```
public struct HLI_ERROR_SATZ_KOPF
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

BfType

Type of commandable function in which an error occurred.

```
public ushort BfType
```

Field Value

[ushort](#)

BodyType

Body type of an error. Depending on the error type, the error set body contains further information on the error which occurred. 1: HLI_RUMPF_TYP_NC_PROG 2: HLI_RUMPF_TYP_MDS 3: HLI_RUMPF_TYP_KOMMU 4: HLI_RUMPF_TYP_RAMDISK 5: HLI_RUMPF_TYP_FILE 6: HLI_RUMPF_TYP_INTPR_FILE 7: HLI_RUMPF_TYP_LISTE_BINAER 8: HLI_RUMPF_TYP_GCM 9: HLI_RUMPF_TYP_LEER 10: HLI_RUMPF_TYP_HLI 11: HLI_RUMPF_TYP_NC_PROG_LR

```
public uint BodyType
```

Field Value

[uint](#)

CncChannel

Channel number of the channel in which the signalled error occurred.

```
public ushort CncChannel
```

Field Value

[ushort](#)

ErrorId

Unique error number.

```
public uint ErrorCode
```

Field Value

[uint](#)

FillUp1

Reserved for future use.

```
public int FillUp1
```

Field Value

[int](#)

FillUp2

Reserved for future use.

```
public bool FillUp2
```

Field Value

[bool](#)

FillUp3

Reserved for future use.

```
public short FillUp3
```

Field Value

[short](#)

FillUp4

Undocumented, found through trial and error.

```
public int FillUp4
```

Field Value

[int](#)

FillUp5

Undocumented, found through trial and error.

```
public int FillUp5
```

Field Value

[int](#)

KommuId

Communication ID of the BF signalling an error in the CNC.

```
public ushort KommuId
```

Field Value

[ushort](#)

Line

Line number in the module on which the error occurred.

```
public int Line
```

Field Value

[int](#)

ModulName

Module name of the module signalling an error.

```
public HLI_MODUL_NAME ModulName
```

Field Value

[HLI_MODUL_NAME](#)

MultipleId

Error messages may be issued at several different points in the NC kernel. A unique multiple error number is issued to distinguish multiple usage.

```
public ushort MultipleId
```

Field Value

[ushort](#) ↗

ReactionType

Reaction class of an error. Value range [0, 8]

`public ushort ReactionType`

Field Value

[ushort](#) ↗

RectificationType

Recovery class of an error. Value range [0, 8]

`public ushort RectificationType`

Field Value

[ushort](#) ↗

SuppressTc2EventLogOutput

No log error message by event logger.

`public bool SuppressTc2EventLogOutput`

Field Value

[bool](#) ↗

TimeStamp

Time specification on output of an error message.

```
public HLI_FB_ZEITANGABE TimeStamp
```

Field Value

[HLI_FB_ZEITANGABE](#)

UtilErrorId

Error number when a utility function is used.

```
public uint UtilErrorId
```

Field Value

[uint](#)

UtilLine

Line number of the line in which the error occurred in a module with utility function.

```
public int UtilLine
```

Field Value

[int](#)

UtilModulName

Module name of the module with utility functions signalling an error.

```
public HLI_MODUL_NAME UtilModulName
```

Field Value

HLI_MODUL_NAME

VersionName

Version name of the CNC specified in the error message.

```
public HLI_INTF_VERSION_NAME VersionName
```

Field Value

[HLI_INTF_VERSION_NAME](#)

Struct HLI_ERROR_SATZ_RUMPF

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Body portion of that an error message.

```
public struct HLI_ERROR_SATZ_RUMPF
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Identifier1

Identifier 1.

```
public HLI_WERT Identifier1
```

Field Value

[HLI_WERT](#)

Identifier2

Identifier 2.

```
public HLI_WERT Identifier2
```

Field Value

[HLI_WERT](#)

Identifier3

Identifier 3.

```
public HLI_WERT Identifier3
```

Field Value

[HLI_WERT](#)

Identifier4

Identifier 4.

```
public HLI_WERT Identifier4
```

Field Value

[HLI_WERT](#)

Mask

Error mask.

```
public HLI_ERROR_MASKE Mask
```

Field Value

[HLI_ERROR_MASKE](#)

Value1

Individual error information 1.

```
public HLI_WERT_B Value1
```

Field Value

[HLI_WERT_B](#)

Value2

Individual error information 2.

`public HLI_WERT_B Value2`

Field Value

[HLI_WERT_B](#)

Value3

Individual error information 3.

`public HLI_WERT_B Value3`

Field Value

[HLI_WERT_B](#)

Value4

Individual error information 4.

`public HLI_WERT_B Value4`

Field Value

[HLI_WERT_B](#)

Value5

Individual error information 5.

```
public HLI_WERT_B Value5
```

Field Value

[HLI_WERT_B](#)

Struct HLI_FB_ZEITANGABE

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Time specification on output of an error message.

```
public struct HLI_FB_ZEITANGABE
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

CycleCounter

Number of interrupt cycles since system start at the instant of an error message

```
public uint CycleCounter
```

Field Value

[uint](#)

DateCounter

Date of the instant of the error message.

```
public uint DateCounter
```

Field Value

[uint](#)

Struct HLI_HB_ACTIVATION

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data to activate a control element and assign it to an axis in manual mode.

```
public struct HLI_HB_ACTIVATION
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

ControlElement

Number of the logical control element to be assigned to the logical axis.

```
public ushort ControlElement
```

Field Value

[ushort](#)

Remarks

When continuous and incremental jog mode is activated: one of the values which are defined as logical button pair numbers in the configuration list hand_mds.lis for the characteristics tasten_data[X].log_tasten_nr. When handwheel mode is activated: one of the values which are defined as logical handwheel numbers in the configuration list hand_mds.lis for the characteristics hr_data[0].log_hr_nr. If 0 is specified as the control element, the current operation mode of an axis is deselected

FillUp1

Reserved for future use.

```
public ushort FillUp1
```

Field Value

[ushort](#) ↗

FillUp2

Reserved for future use.

```
public ushort FillUp2
```

Field Value

[ushort](#) ↗

LogicalAxisNumber

Unique system-wide number of a logical axis. A control element is assigned to the specified logical axis with which the axis is to be moved in manual mode.

```
public ushort LogicalAxisNumber
```

Field Value

[ushort](#) ↗

OperationMode

Manual operation mode to be assigned to the logical axis.

```
public ushort OperationMode
```

Field Value

[ushort](#)

Remarks

0: no operation mode, current operation mode selected
1: Handwheel mode
2: Continuous jog mode
3: jog mode

ParameterIndex

Specifies the index of the parameter set to be used for the manual mode. Value range [0; 2]

```
public ushort ParameterIndex
```

Field Value

[ushort](#)

Remarks

The first value set in the parameter table (index = 0) is overwritten by the PLC interface when individual parameters are specified. The remaining parameter sets are not changed. This means, they correspond to the values specified in the axis-specific parameter lists.

Struct HLI_HB_HR_PARAMETER

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data for parameterising handwheel mode in manual mode.

```
public struct HLI_HB_HR_PARAMETER
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

FillUp1

Reserved for future use.

```
public ushort FillUp1
```

Field Value

[ushort](#)

LogicalAxisNumber

Unique system-wide number of a logical axis. The specified logical axis is assigned the handwheel resolution which is the basis for moving the axis in manual mode handwheel mode.

```
public ushort LogicalAxisNumber
```

Field Value

[ushort](#)

Resolution

Resolution of axis motion path for one handwheel revolution. The internally used total resolution of the axis in 0.1 µm per applied handwheel increment results from the current handwheel resolution in 0.1 µm/increment divided by the physical handwheel resolution in increment/revolution of the handwheel specified Unit: 0.1 µm / handwheel revolution

```
public int Resolution
```

Field Value

[int↗](#)

Struct HLI_HB_JOG_PARAMETER

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data for parameterising incremental jog mode in manual mode.

```
public struct HLI_HB_JOG_PARAMETER
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Distance

Path traversed by the logical axis in incremental jog mode each time the incremental jog button is pressed. Unit: 0.1 μm

```
public uint Distance
```

Field Value

[uint](#)

FillUp1

Reserved for future use.

```
public ushort FillUp1
```

Field Value

[ushort](#)

FillUp2

Reserved for future use.

```
public int FillUp2
```

Field Value

[int](#)

LogicalAxisNumber

Unique system-wide number of a logical axis. The specified logical axis is assigned the velocity and incremental step width for each button press to move the axis in manual mode in incremental jog mode.

```
public ushort LogicalAxisNumber
```

Field Value

[ushort](#)

Speed

Velocity to be assigned to the logical axis in incremental jog mode. Unit: 1 μm/s

```
public uint Speed
```

Field Value

[uint](#)

Struct HLI_HB_KEY

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data to enforce a button press in manual mode,

```
public struct HLI_HB_KEY
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Direction

Start/end of button press event and motion direction of buttons in manual mode. -1: Start of button press, negative motion direction 0: End of button press +1: Start of button press, positive motion direction

```
public short Direction
```

Field Value

[short](#)

F_Refresh

Retriggering "start of button press" event. If the element "Life time of a button signal" has a value unequal to 0, the "start of button press" event is retriggered if the "Lifetime of the button signal" has not yet expired.

```
public bool F_Refresh
```

Field Value

[bool](#)

FillUp1

Reserved for future use.

```
public bool FillUp1
```

Field Value

[bool](#)

FillUp2

Reserved for future use.

```
public bool FillUp2
```

Field Value

[bool](#)

FillUp3

Reserved for future use.

```
public bool FillUp3
```

Field Value

[bool](#)

FillUp4

Reserved for future use.

```
public int FillUp4
```

Field Value

[int](#)

LifeTime

Lifetime of the button signal. This is a time period defined by number of interpolator cycles. If this element has a value unequal to 0, the CNC independently generates the "End of button press" event after receiving a "Start of button press" event after the time period expires and which was defined by the number of specified interpolator cycles.

```
public uint LifeTime
```

Field Value

[uint](#)

LogicalKeyNumber

Logical button number from which the command comes.

```
public ushort LogicalKeyNumber
```

Field Value

[ushort](#)

Struct HLI_HB_RAPID_KEY

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to activate/deactivate rapid traverse mode by a normal button press in manual mode.

```
public struct HLI_HB_RAPID_KEY
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

FillUp1

Reserved for future use.

```
public uint FillUp1
```

Field Value

[uint](#)

KeyPressed

Rapid traverse mode of the button on/off. TRUE = Button for rapid traverse mode is active. The parameterised rapid traverse path motion is used for continuous jog mode. FALSE = Button not active in rapid traverse mode. The parameterised normal path velocity is used in continuous joy mode.]

```
public ushort KeyPressed
```

Field Value

[ushort](#)

LogicalKeyNumber

Logical button number for which the rapid traverse mode should be selected/deselected.

```
public ushort LogicalKeyNumber
```

Field Value

[ushort](#)

Struct HLI_HB_TIP_PARAMETER

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data for parameterising continuous jog mode in manual mode

```
public struct HLI_HB_TIP_PARAMETER
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

FillUp1

Reserved for future use.

```
public ushort FillUp1
```

Field Value

[ushort](#)

LogicalAxisNumber

Unique system-wide number of a logical axis. The specified logical axis is assigned the velocity at which it will be moved in manual mode in continuous jog mode.

```
public ushort LogicalAxisNumber
```

Field Value

[ushort](#)

Speed

Velocity to be assigned to the logical axis in continuous jog mode. Unit: 1 µm/s

```
public uint Speed
```

Field Value

[uint](#)

Struct HLI_IMCM_MODE_STATE

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Struct for operation mode management that combines mode and state into one struct.

```
public struct HLI_IMCM_MODE_STATE
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Mode

Operation mode.

```
public OperationMode Mode
```

Field Value

[OperationMode](#)

State

State within the operation mode.

```
public OperationState State
```

Field Value

[OperationState](#)

Struct HLI_INTF_VERSION_NAME

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Structure that contains the CNC version name.

```
public struct HLI_INTF_VERSION_NAME
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Name

CNC version name.

```
public string Name
```

Field Value

[string](#)

Struct HLI_MODUL_NAME

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Structure that contains the name of one module.

```
public struct HLI_MODUL_NAME
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Name

Name of the module.

```
public string Name
```

Field Value

[string](#)

Struct HLI_M_H_PROZESS

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Further describes the TECHNO_UNIT_CH struct when it is a M or H function.

```
public struct HLI_M_H_PROZESS
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

BlockNumber

NC Block number of the M or H technology function.

```
public uint BlockNumber
```

Field Value

[uint](#)

ExpectedTime

Expexted time for handling of the M or H technology function in [ms].

```
public int ExpectedTime
```

Field Value

[int](#)

Number

Number of the M or H technology function.

```
public uint Number
```

Field Value

[uint](#)

ProgramRow

NC program row of the M or H technology function.

```
public uint ProgramRow
```

Field Value

[uint](#)

additionalValue

Additional value, if programmed in NC program.

```
public int additionalValue
```

Field Value

[int](#)

fill_up_1

Reserved for future use.

```
public ushort fill_up_1
```

Field Value

[ushort](#) ↗

fill_up_2

Reserved for future use.

```
public int fill_up_2
```

Field Value

[int](#) ↗

nr_late_sync

Counter of written late sync, if active sync present.

```
public ushort nr_late_sync
```

Field Value

[ushort](#) ↗

synchMode

Synchronisation mode

```
public uint synchMode
```

Field Value

[uint](#) ↗

Struct HLI_PROC_TRANS_TO_MODE_STATE

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to switch over the operation mode and poll the current state of operation mode management, including flow control of user data.

```
public struct HLI_PROC_TRANS_TO_MODE_STATE
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

ChannelNumber

Not used (only for compatibility with the HÜMNOS standard).

```
public uint ChannelNumber
```

Field Value

[uint](#)

FromMode

Operation mode from which the operation mode is to be changed.

```
public OperationMode FromMode
```

Field Value

[OperationMode](#)

FromState

State within the operation mode from which the state switchover is to occur.

```
public OperationState FromState
```

Field Value

[OperationState](#)

Parameter

Parameters for operation mode change. It may be necessary to specify parameters when commanding an operation mode change to ensure the successful change to a specific state of an operation mode. These parameters are saved in this element.

```
public string Parameter
```

Field Value

[string](#) ↗

ToMode

Target operation mode when the operation mode is switched over.

```
public OperationMode ToMode
```

Field Value

[OperationMode](#)

ToState

Target state when operation mode is changed.

```
public OperationState ToState
```

Field Value

[OperationState](#)

fill_up_1

Reserved for future use.

```
public int fill_up_1
```

Field Value

[int](#)

fill_up_2

Reserved for future use.

```
public int fill_up_2
```

Field Value

[int](#)

Struct HLI_RUMPF_NC_PROG

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Error information in relation to NC program.

```
public struct HLI_RUMPF_NC_PROG
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

BlockNumber

Block number of current NC line.

```
public uint BlockNumber
```

Field Value

[uint](#)

FileEncrypted

Is file that gave the error encrypted.

```
public bool FileEncrypted
```

Field Value

[bool](#)

FileName

File offset in bytes.

```
public string FileName
```

Field Value

[string](#) ↗

FileOffset

```
public uint FileOffset
```

Field Value

[uint](#) ↗

FillUp1

Reserved for future use.

```
public bool FillUp1
```

Field Value

[bool](#) ↗

FillUp3

Reserved for future use.

```
public int FillUp3
```

Field Value

[int](#)

FillUp4

Reserved for future use.

```
public int FillUp4
```

Field Value

[int](#)

LogicalPathNumber

Logical path number (see start-up list).

```
public ushort LogicalPathNumber
```

Field Value

[ushort](#)

PositionOffsetNcBlock

Position in NC block in bytes.

```
public ushort PositionOffsetNcBlock
```

Field Value

[ushort](#)

ProgramName

File name.

```
public string ProgramName
```

Field Value

[string](#)

TokenOffsetNcBlock

Token offset in current NC line.

```
public ushort TokenOffsetNcBlock
```

Field Value

[ushort](#)

Struct HLI_WERT

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Error value.

```
public struct HLI_WERT
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Content

Content.

```
public HLI_WERT_DATA Content
```

Field Value

[HLI_WERT_DATA](#)

FillUp1

Reserved for future use.

```
public int FillUp1
```

Field Value

[int](#)

Type

Data type.

```
public uint Type
```

Field Value

[uint](#)

Struct HLI_WERT_B

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Additional error information.

```
public struct HLI_WERT_B
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Content

Datum itself.

```
public HLI_WERT_B_DATA Content
```

Field Value

[HLI_WERT_B_DATA](#)

Dimension

Dimension of datum.

```
public uint Dimension
```

Field Value

[uint](#)

FillUp1

Reserved for future use.

```
public int FillUp1
```

Field Value

[int](#)

Importance

Significance of datum.

```
public uint Importance
```

Field Value

[uint](#)

Type

Data type.

```
public uint Type
```

Field Value

[uint](#)

Struct HLI_WERT_B_DATA

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Contains the actual data of an error message.

```
public struct HLI_WERT_B_DATA
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Data

Data.

```
public byte[] Data
```

Field Value

[byte](#)[]

Struct HLI_WERT_DATA

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Error value data.

```
public struct HLI_WERT_DATA
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Data

Content data.

```
public byte[] Data
```

Field Value

[byte](#)[]

Class HandWheelInc

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage the count of handwheel increments for a handwheel index, including flow control of user data.

```
public class HandWheelInc
```

Inheritance

[object](#) ← HandWheelInc

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

EnableControlElement(bool)

Signal to CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

WriteCommandElement(short)

Write the count of handwheel increments to the CNC.

```
public void WriteCommandElement(short data)
```

Parameters

data [short ↗](#)

Class HcodeNeedsAcknowledgeEventArgs

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Event args supplied when a H code needs to be acknowledged.

```
public class HcodeNeedsAcknowledgeEventArgs
```

Inheritance

[object](#) ← HcodeNeedsAcknowledgeEventArgs

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

IndexToAcknowledge

Index of technology to acknowledge when you are done processing this H-code.

```
public readonly int IndexToAcknowledge
```

Field Value

[int](#)

MhProcess

Description of the H-code.

```
public readonly HLI_M_H_PROZESS MhProcess
```

Field Value

HLI M_H PROZESS

Class HrParameters

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data for parameterising handwheel mode in manual mode, including flow control of user data

```
public class HrParameters
```

Inheritance

[object](#) ← HrParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

EnableControlElement(bool)

Signal to CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

SignalCommandSemaphor(bool)

CNC accepts the commanded data if this element has the value TRUE and sets this element to the value FALSE after complete acceptance of the data. You should set this element to the value TRUE if all data to be commanded has been written.

```
public void SignalCommandSemaphor(bool signal)
```

Parameters

signal [bool](#)

WriteCommandElement(HLI_HB_HR_PARAMETER)

Write the parameters for the handwheel mode to the CNC.

```
public void WriteCommandElement(HLI_HB_HR_PARAMETER data)
```

Parameters

data [HLI HB HR PARAMETER](#)

Class Interpolator

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The Interpolator class provides an interface to interact with a CNC interpolator via an AdsClient. It allows reading and writing various properties related to the interpolator's state, such as axis count, moved path, velocity, tool life, and more. The class also provides addresses for device notifications and methods to retrieve interpolator addresses and properties.

```
public class Interpolator
```

Inheritance

[object](#) ← Interpolator

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualBlockCount

Actual block count.

```
public int ActualBlockCount { get; }
```

Property Value

[int](#)

ActualDWord

Actual DWord.

```
public uint ActualDWord { get; }
```

Property Value

[uint](#)

ActualStateOfInterpolator

Actual state of the interpolator.

```
public byte ActualStateOfInterpolator { get; }
```

Property Value

[byte](#)

ActualToolRadius

Actual tool radius. Unit: [0.1 μm]

```
public double ActualToolRadius { get; }
```

Property Value

[double](#)

ActualVelocityOnPath

Actual velocity on path. Unit: [μm/s]

```
public double ActualVelocityOnPath { get; }
```

Property Value

[double](#)

ActualZeroOffsetGroup

Actual zero offset group.

```
public ushort ActualZeroOffsetGroup { get; }
```

Property Value

[ushort](#)

AdditionalStatus

Additional status.

```
public uint AdditionalStatus { get; }
```

Property Value

[uint](#)

Adresses

Adresses for the interpolator class. Use this if you want to add device notifications for any of the properties.

```
public InterpolatorAdresses Adresses { get; }
```

Property Value

[InterpolatorAdresses](#)

AxisCount

Number of axes.

```
public ushort AxisCount { get; }
```

Property Value

[ushort](#) ↗

BendAngleAtBlockEnd

Bend angle at block end.

```
public int BendAngleAtBlockEnd { get; }
```

Property Value

[int](#) ↗

BlockEndActive

Block end active.

```
public bool BlockEndActive { get; }
```

Property Value

[bool](#) ↗

BlockLengthOnPath

Block length on path. Unit: [0.1 μm]

```
public double BlockLengthOnPath { get; }
```

Property Value

[double](#) ↗

BlockNumberActual

Actual block number.

```
public int BlockNumberActual { get; }
```

Property Value

[int](#)

CartesianTransformationActive

Cartesian transformation active.

```
public bool CartesianTransformationActive { get; }
```

Property Value

[bool](#)

CaxFunctionAligningActive

Cax function aligning active.

```
public bool CaxFunctionAligningActive { get; }
```

Property Value

[bool](#)

CommandBFOVERRIDE

Commanded BF override.

```
public ushort CommandBFOVERRIDE { get; }
```

Property Value

[ushort](#)

CommandFBOverride

Commanded FB override.

```
public ushort CommandFBOverride { get; }
```

Property Value

[ushort](#)

CoveredDistance

Covered distance.

```
public double CoveredDistance { get; }
```

Property Value

[double](#)

DistanceFromProgramStart

Distance from program start.

```
public double DistanceFromProgramStart { get; }
```

Property Value

[double](#)

Druckwinkel

Pressure angle.

```
public double Druckwinkel { get; }
```

Property Value

[double](#) ↗

DwellTimeCommanded

Commanded G04 dwell time. Unit: seconds.

```
public double DwellTimeCommanded { get; }
```

Property Value

[double](#) ↗

DwellTimeRemaning

Remaining G04 dwell time. Unit: seconds.

```
public double DwellTimeRemaning { get; set; }
```

Property Value

[double](#) ↗

DynamicLimit

Dynamic limit.

```
public uint DynamicLimit { get; }
```

Property Value

[uint](#) ↗

DynamicWeightG129

Dynamic weight G129. Unit: %

```
public double DynamicWeightG129 { get; }
```

Property Value

[double](#) ↗

DynamicWeightG131

Dynamic weight G131. Unit: %

```
public double DynamicWeightG131 { get; }
```

Property Value

[double](#) ↗

DynamicWeightG133

Dynamic weight G133. Unit: %

```
public double DynamicWeightG133 { get; }
```

Property Value

[double](#) ↗

DynamicWeightG134

Dynamic weight G134. Unit: %

```
public double DynamicWeightG134 { get; }
```

Property Value

[double](#) ↗

DynamicWeightG231

Dynamic weight G231. Unit: %

```
public double DynamicWeightG231 { get; }
```

Property Value

[double](#) ↗

DynamicWeightG233

Dynamic weight G233. Unit: %

```
public double DynamicWeightG233 { get; }
```

Property Value

[double](#) ↗

FeedOfSyncAxis

Feed of sync axis. Unit: [$\mu\text{m}/\text{s}$]

```
public double FeedOfSyncAxis { get; }
```

Property Value

[double](#) ↗

GlobalEnabledAxesCount

Global enabled axes count.

```
public ushort GlobalEnabledAxesCount { get; }
```

Property Value

[ushort](#)

HscFilterDelayTimelpo

HSC filter delay time Ipo. Unit: seconds.

```
public double HscFilterDelayTimeIpo { get; }
```

Property Value

[double](#)

HscFilterOn

HSC filter on.

```
public bool HscFilterOn { get; }
```

Property Value

[bool](#)

HscFilterOrder

HSC filter order.

```
public uint HscFilterOrder { get; }
```

Property Value

[uint](#)

HscSurfaceActive

HSC surface active.

```
public bool HscSurfaceActive { get; }
```

Property Value

[bool](#)

HscSurfaceCheckJerk

HSC surface check jerk.

```
public uint HscSurfaceCheckJerk { get; }
```

Property Value

[uint](#)

HscSurfaceFAutoOffG00

HSC surface F auto off G00.

```
public bool HscSurfaceFAutoOffG00 { get; }
```

Property Value

[bool](#)

HscSurfaceMaxAngle

HSC surface max angle. Unit: [deg]

```
public double HscSurfaceMaxAngle { get; }
```

Property Value

[double](#) ↗

HscSurfacePathDev

HSC surface path deviation. Unit: [mm]

```
public double HscSurfacePathDev { get; }
```

Property Value

[double](#) ↗

HscSurfaceTrackDev

HSC surface track deviation. Unit: [deg]

```
public double HscSurfaceTrackDev { get; }
```

Property Value

[double](#) ↗

IpoBufLevel

Interpolator buffer level.

```
public uint IpoBufLevel { get; }
```

Property Value

[uint](#) ↗

IpoSyncWaitState

Interpolator sync wait state.

```
public uint IpoSyncWaitState { get; }
```

Property Value

[uint](#)

KinematicalTransformationActive

Kinematical transformation active.

```
public bool KinematicalTransformationActive { get; }
```

Property Value

[bool](#)

LatestInputBlockCount

Latest input block count.

```
public int LatestInputBlockCount { get; }
```

Property Value

[int](#)

LimitingAxis

Limiting axis.

```
public uint LimitingAxis { get; }
```

Property Value

[uint](#) ↗

ManualAcsAxesMovementLimitation

Manual: ACS axes movement limitation.

```
public uint ManualAcsAxesMovementLimitation { get; }
```

Property Value

[uint](#) ↗

ManualPcsAxesMovementLimitation

Manual: PCS axes movement limitation.

```
public uint ManualPcsAxesMovementLimitation { get; }
```

Property Value

[uint](#) ↗

ManualWaitPositionInitializationDone

Manual: wait position initialization done.

```
public bool ManualWaitPositionInitializationDone { get; }
```

Property Value

[bool](#) ↗

MaximumVelocityOnPath

Maximum velocity on path.

```
public uint MaximumVelocityOnPath { get; }
```

Property Value

[uint](#)

MaximumVelocityOnPathAtBlockEnd

Maximum velocity on path at block end.

```
public uint MaximumVelocityOnPathAtBlockEnd { get; }
```

Property Value

[uint](#)

MeasuringActive

Measuring active.

```
public bool MeasuringActive { get; }
```

Property Value

[bool](#)

MotionCtrl

Motion control.

```
public uint MotionCtrl { get; }
```

Property Value

[uint](#)

MotionCtrl2

Motion control 2.

```
public uint MotionCtrl2 { get; }
```

Property Value

[uint](#)

MovedPath

Length of moved path. Unit: [0.1 μm]

```
public double MovedPath { get; }
```

Property Value

[double](#)

OverrideFeedFactor

Override feed factor.

```
public double OverrideFeedFactor { get; }
```

Property Value

[double](#)

PathfeedInSyncArea

Path feed in sync area. Unit: [μm/s]

```
public double PathfeedInSyncArea { get; }
```

Property Value

[double](#)

PosLahTimeToDist

Position lookahead time to distance.

```
public int PosLahTimeToDist { get; }
```

Property Value

[int](#)

PositionLookaheadDistance

Position lookahead distance.

```
public uint PositionLookaheadDistance { get; set; }
```

Property Value

[uint](#)

ProgrammedVelocityOnPath

Programmed velocity on path. Unit: [$\mu\text{m}/\text{s}$]

```
public double ProgrammedVelocityOnPath { get; }
```

Property Value

[double](#)

Rahmenwinkel

Frame angle.

```
public double Rahmenwinkel { get; }
```

Property Value

[double](#) ↗

RakelTransformationActive

Rakel transformation active.

```
public bool RakelTransformationActive { get; }
```

Property Value

[bool](#) ↗

RapidMovementBlock

Is current block a rapid G0 movement.

```
public bool RapidMovementBlock { get; }
```

Property Value

[bool](#) ↗

RemainingPathOfBlock

Length of the remaining path of the block. Unit: [0.1 µm]

```
public double RemainingPathOfBlock { get; }
```

Property Value

[double](#) ↗

SingleStepMode

Single step mode active.

```
public uint SingleStepMode { get; set; }
```

Property Value

[uint](#) ↗

SlopeBufLevel

Slope buffer level.

```
public uint SlopeBufLevel { get; }
```

Property Value

[uint](#) ↗

SlopeBufPath

Length of slope buffer path. Unit: [0.1 μm]

```
public uint SlopeBufPath { get; }
```

Property Value

[uint](#) ↗

Status

Status.

```
public uint Status { get; }
```

Property Value

[uint](#)

SuspendAxisOutputState

Suspend axis output state.

```
public uint SuspendAxisOutputState { get; }
```

Property Value

[uint](#)

ToolLifeDistance

Tool life distance. Unit: mm

```
public double ToolLifeDistance { get; }
```

Property Value

[double](#)

ToolLifeDistanceFactor

Tool life distance factor.

```
public double ToolLifeDistanceFactor { get; set; }
```

Property Value

[double](#)

ToolLifeTime

Tool life time. Unit: seconds

```
public double ToolLifeTime { get; }
```

Property Value

[double](#)

ToolLifeTimeFactor

Tool life time factor.

```
public double ToolLifeTimeFactor { get; set; }
```

Property Value

[double](#)

ToolLifeToolId

Tool life tool ID.

```
public uint ToolLifeToolId { get; }
```

Property Value

[uint](#)

TotalCsMatrixEx0

Total CS matrix ex0.

```
public double TotalCsMatrixEx0 { get; }
```

Property Value

[double ↗](#)

TotalCsMatrixEx1

Total CS matrix ex1.

```
public double TotalCsMatrixEx1 { get; }
```

Property Value

[double ↗](#)

TotalCsMatrixEx2

Total CS matrix ex2.

```
public double TotalCsMatrixEx2 { get; }
```

Property Value

[double ↗](#)

TotalCsMatrixEy0

Total CS matrix ey0.

```
public double TotalCsMatrixEy0 { get; }
```

Property Value

[double ↗](#)

TotalCsMatrixEy1

Total CS matrix ey1.

```
public double TotalCsMatrixEy1 { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixEy2

Total CS matrix ey2.

```
public double TotalCsMatrixEy2 { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixEz0

Total CS matrix ez0.

```
public double TotalCsMatrixEz0 { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixEz1

Total CS matrix ez1.

```
public double TotalCsMatrixEz1 { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixEz2

Total CS matrix ez2.

```
public double TotalCsMatrixEz2 { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixVx

Total CS matrix vx. Unit: [0.1 μm]

```
public double TotalCsMatrixVx { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixVy

Total CS matrix vy. Unit: [0.1 μm]

```
public double TotalCsMatrixVy { get; }
```

Property Value

[double](#) ↗

TotalCsMatrixVz

Total CS matrix vz. Unit: [0.1 μm]

```
public double TotalCsMatrixVz { get; }
```

Property Value

[double](#) ↗

WegBisSynPunkt

Weg bis Syn. Punkt.

```
public double WegBisSynPunkt { get; }
```

Property Value

[double](#) ↗

ZeitBisSynPunkt

Zeit bis Syn. Punkt.

```
public double ZeitBisSynPunkt { get; }
```

Property Value

[double](#) ↗

Class InterpolatorAdresses

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Adresses for the interpolator class.

```
public class InterpolatorAdresses
```

Inheritance

[object](#) ← InterpolatorAdresses

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

BlockNumberActual

Sub index for block number actual.

```
public uint BlockNumberActual { get; }
```

Property Value

[uint](#)

GroupIndex

Group index for interpolator adresses.

```
public uint GroupIndex { get; }
```

Property Value

Class JogParameters

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data for parameterising incremental jog mode in manual manual, including flow control of user data

```
public class JogParameters
```

Inheritance

[object](#) ← JogParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

EnableControlElement(bool)

Signal to CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

SignalCommandSemaphor(bool)

CNC accepts the commanded data if this element has the value TRUE and sets this element to the value FALSE after complete acceptance of the data. You should set this element to the value TRUE if all data to be commanded has been written.

```
public void SignalCommandSemaphor(bool signal)
```

Parameters

signal [bool](#)

WriteCommandElement(HLI_HB_JOG_PARAMETER)

Write the parameters for the incremental jog mode to the CNC.

```
public void WriteCommandElement(HLI_HB_JOG_PARAMETER data)
```

Parameters

data [HLI_HB_JOG_PARAMETER](#)

Class Key

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data to enforce a button press in manual mode, including flow control of user data.

```
public class Key
```

Inheritance

[object](#) ← Key

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

EnableControlElement(bool)

Signal to CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

SignalCommandSemaphor(bool)

CNC accepts the commanded data if this element has the value TRUE and sets this element to the value FALSE after complete acceptance of the data. You should set this element to the value TRUE if all data to be commanded has been written.

```
public void SignalCommandSemaphor(bool signal)
```

Parameters

signal [bool](#)

WriteCommandElement(HLI_HB_KEY)

Write the command data for the key to the CNC.

```
public void WriteCommandElement(HLI_HB_KEY data)
```

Parameters

data [HLI_HB_KEY](#)

Class ManualOperation

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Class that contains settings and functions for the manual operation of a CNC machine. Such as keys, rapid keys and hand wheels.

```
public class ManualOperation
```

Inheritance

[object](#) ← ManualOperation

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

HandWheelIncs

Array of control units to manage the counts of handwheel increments for all handwheels, including flow control of user data.

```
public HandWheelInc[] HandWheelIncs { get; }
```

Property Value

[HandWheelInc\[\]](#)

HrParameters

Control unit to manage data for parameterising handwheel mode in manual mode, including flow control of user data

```
public HrParameters HrParameters { get; }
```

Property Value

[HrParameters](#)

JogParameters

Control unit to manage data for parameterising incremental jog mode in manual manual, including flow control of user data

```
public JogParameters JogParameters { get; }
```

Property Value

[JogParameters](#)

Keys

Control unit to manage data to enforce a button press in manual mode, including flow control of user data.

```
public Key[] Keys { get; }
```

Property Value

[Key\[\]](#)

RapidKey

During continuous jog mode it is possible to switch between the normal velocity and rapid traverse velocity with this control unit.

```
public RapidKey RapidKey { get; }
```

Property Value

[RapidKey](#)

TipParameters

Control unit to manage data for parameterising continuous jog mode in manual mode, including flow control of user data.

```
public TipParameters TipParameters { get; }
```

Property Value

[TipParameters](#)

Methods

EnableControlElement(bool)

Present marker to the CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

GetControlElementNumber(int)

Logical number of the control element currently linked to the axis in question.

```
public ushort GetControlElementNumber(int axisIndex)
```

Parameters

axisIndex [int](#)

Returns

[ushort](#)

GetManualModeState(int)

Gets the manual mode state of an axis for the supplied axis index.

```
public ushort GetManualModeState(int axisIndex)
```

Parameters

axisIndex [int](#)

Returns

[ushort](#)

GetOperationModeState(int)

Gets the operation mode state of an axis for the supplied axis index.

```
public ushort GetOperationModeState(int axisIndex)
```

Parameters

axisIndex [int](#)

Returns

[ushort](#)

GetPathVelocityContinuous(int)

Path velocity of the axis in question when moved in continuous jog mode.

```
public int GetPathVelocityContinuous(int axisIndex)
```

Parameters

axisIndex [int](#)

Returns

[int](#)

SignalCommandSemaphor(bool)

CNC accepts the commanded data if this element has the value TRUE and sets this element to the value FALSE after complete acceptance of the data. You should set this element to the value TRUE if all data to be commanded has been written.

```
public void SignalCommandSemaphor(bool signal)
```

Parameters

[signal](#) [bool](#)

WriteCommandElement(HLI_HB_ACTIVATION)

Write the command data for one of the manual control elements to the CNC.

```
public void WriteCommandElement(HLI_HB_ACTIVATION controlElement)
```

Parameters

[controlElement](#) [HLI HB ACTIVATION](#)

Class McodeNeedsAcknowledgeEventArgs

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Event args supplied when a M code needs to be acknowledged.

```
public class McodeNeedsAcknowledgeEventArgs
```

Inheritance

[object](#) ← McodeNeedsAcknowledgeEventArgs

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

IndexToAcknowledge

Index of technology to acknowledge when you are done processing this M-code.

```
public readonly int IndexToAcknowledge
```

Field Value

[int](#)

MhProcess

Description of the M-code.

```
public readonly HLI_M_H_PROZESS MhProcess
```

Field Value

HLI M_H PROZESS

Class ObjectDescription

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Represents the description of a TASK COM object.

```
public class ObjectDescription
```

Inheritance

[object](#) ← ObjectDescription

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Id

Internal unique ID of an object.

```
public uint Id
```

Field Value

[uint](#)

IndexGroup

Index group for direct object access to the content.

```
public uint IndexGroup
```

Field Value

[uint](#)

IndexOffset

Index offset for direct object access to the content.

```
public uint IndexOffset
```

Field Value

[uint](#)

Name

Name of the object.

```
public string Name
```

Field Value

[string](#)

Size

Size of object in bytes.

```
public uint Size
```

Field Value

[uint](#)

Type

Object data type BOOL, BYTE, SINT, WORD, INT, DWORD, DINT, LWORD, LINT, REAL, LREAL, STRING

```
public string Type
```

Field Value

[string](#)

WriteAccess

TRUE if object is describable.

```
public ushort WriteAccess
```

Field Value

[ushort](#)

Methods

ToString()

Returns the name of the object.

```
public override string ToString()
```

Returns

[string](#)

Enum OperationMode

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The CNC distinguishes between five operating modes. It is possible to switch over between these operating modes via the operator-control and/or PLC interface, whereby only one operating mode may be active at any one time. This enumeration lists the operating modes that are defined.

```
public enum OperationMode
```

Fields

AUTOMATIC_MODE = 2

The control can run a complete NC program automatically. In this case, program execution can be interrupted and resumed.

MANUAL_MODE = 4

Movements are commanded by peripherals connected directly to the control (keys, handwheels).

MDI_MODE = 3

Movements are commanded by the operator-control computer via a single NC block. The NC block is transferred as a string to the control and executed via a START command. Interruption and resumption of movement are possible in this case.

REFERENCE_MODE = 5

The axes can be referenced. An NC program of name rpf.nc is started in this case.

STANDBY_MODE = 1

No operating mode is selected. Default after starting the control.

Class OperationModeAdresses

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

This class contains the addresses of objects used by the operation mode manager.

```
public class OperationModeAdresses
```

Inheritance

[object](#) ← OperationModeAdresses

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

GeoIndexGroup

Main index group for all addresses.

```
public uint GeoIndexGroup { get; }
```

Property Value

[uint](#)

OperationModeActual

Sub index group of the actual operation mode.

```
public uint OperationModeActual { get; }
```

Property Value

[uint](#) ↗

OperationStateActual

Sub index group of the actual operation state.

```
public uint OperationStateActual { get; }
```

Property Value

[uint](#) ↗

Class OperationModeManager

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The OperationModeManager class provides an interface to manage and control the operation modes and states of the CNC via an AdsClient. It allows setting and retrieving the current operation mode and state, resetting the operation mode, and managing the interface existence.

```
public class OperationModeManager : IDisposable
```

Inheritance

[object](#) ← OperationModeManager

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Adresses

Adresses used by the operation mode manager. Contains the main index group and sub index groups of common properties. These can be used to add device notifications by the user.

```
public OperationModeAdresses Adresses { get; }
```

Property Value

[OperationModeAdresses](#)

InterfaceExists

Signal to the CNC that the interface exists and we want to use it.

```
public bool InterfaceExists { set; }
```

Property Value

[bool](#)

OperationModeActual

Gets the actual operation mode.

```
public OperationMode OperationModeActual { get; }
```

Property Value

[OperationMode](#)

OperationModeAndStateActual

Gets the actual operation mode and state.

```
public HLI_IMCM_MODE_STATE OperationModeAndStateActual { get; }
```

Property Value

[HLI_IMCM_MODE_STATE](#)

OperationStateActual

Gets the actual operation state.

```
public OperationState OperationStateActual { get; }
```

Property Value

[OperationState](#)

RequestedModeAndState

Gets the requested operation mode and state.

```
public HLI_PROC_TRANS_TO_MODE_STATE RequestedModeAndState { get; }
```

Property Value

[HLI PROC TRANS TO MODE STATE](#)

Methods

Dispose()

Dispose the operation mode manager. Deletes all ADS device notifications.

```
public void Dispose()
```

Reset()

Reset the operation mode to standby mode.

```
public void Reset()
```

SetModeAndState(HLI_PROC_TRANS_TO_MODE_STATE)

Sets a new mode and state.

```
public void SetModeAndState(HLI_PROC_TRANS_TO_MODE_STATE unit)
```

Parameters

unit [HLI PROC TRANS TO MODE STATE](#)

SetModeAndState(OperationMode, OperationState, string)

```
public void SetModeAndState(OperationMode mode, OperationState state, string parameter)
```

Parameters

mode [OperationMode](#)

state [OperationState](#)

parameter [string](#)

It may be necessary to specify parameters when commanding an operation mode change to ensure the successful change to a specific state of an operation mode.

Enum OperationState

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Enumeration of the possible states of an operation mode. Depending on the actually selected operation mode, these states may contain a further meaning.

```
public enum OperationState
```

Fields

NoSignificance = 0

PROCESS_ACTIVE = 4

NC program is running, or the MDI NC block(s) are running, or manual mode is running or homing is running.

PROCESS_DESELECTED = 1

Operation mode is deselected.

PROCESS_ERROR = 6

An error occurred while the NC program is executed, or error state for other modes.

PROCESS_HOLD = 5

NC program is interrupted, or the MDI NC block(s) are stopped, or manual mode is stopped, or homing is stopped.

PROCESS_READY = 3

NC/Program is selected, or MDI block is selected, or manual mode is programmed or homing is programmed.

PROCESS_SELECTED = 2

Operation mode is selected.

Class RapidKey

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

During continuous jog mode it is possible to switch between the normal velocity and rapid traverse velocity. Here the rapid traverse is a button-specific feature and only becomes effective when the corresponding button is pushed and linked to an axis.

```
public class RapidKey
```

Inheritance

[object](#) ← RapidKey

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

EnableControlElement(bool)

Signal to CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

SignalCommandSemaphor(bool)

CNC accepts the commanded data if this element has the value TRUE and sets this element to the value FALSE after complete acceptance of the data. You should set this element to the value TRUE if all data to be commanded has been written.

```
public void SignalCommandSemaphor(bool signal)
```

Parameters

signal [bool](#)

WriteCommandElement(HLI_HB_RAPID_KEY)

Write the command data for the rapid key to the CNC.

```
public void WriteCommandElement(HLI_HB_RAPID_KEY data)
```

Parameters

data [HLI HB RAPID KEY](#)

Struct SOLLKONT_VISU_ACHS_DATA_STD

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Struct that contains axis-specific visualisation data.

```
public struct SOLLKONT_VISU_ACHS_DATA_STD
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

AlignmentBytes

Bytes for alignment.

```
public ushort AlignmentBytes
```

Field Value

[ushort](#)

CommandPosition

Current Command position in [0.1 μm]

```
public int CommandPosition
```

Field Value

[int](#)

LogicalAxisNumber

Logical axis number of the axis that the commanded position belongs to.

```
public ushort LogicalAxisNumber
```

Field Value

[ushort](#)

Struct SOLLKONT_VISU_CH_DATA_STD

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Struct that contains standard visualisation data.

```
public struct SOLLKONT_VISU_CH_DATA_STD
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

BlockNumber

Block number in the NC program

```
public int BlockNumber
```

Field Value

[int](#)

ChannelNumber

ChannelNumber

```
public ushort ChannelNumber
```

Field Value

[ushort](#)

CircleCenterPoint

Absolute position of circle centre point in the active machining plane (G17,G18,G19) in [0.1 µm] for G2 / G3 blocks (as of CNC Build V2.10.1032.03 and V2.10.1505.05)

```
public double[] CircleCenterPoint
```

Field Value

[double](#)[]

CircleRadius

Radius in [0.1 µm] for G2 / G3 blocks.

```
public uint CircleRadius
```

Field Value

[uint](#)[]

FileOffset

File offset from file start in bytes.

= 0 : valid data offset when program is active. == -1 : Offset not valid since no program is active

```
public int FileOffset
```

Field Value

[int](#)[]

GFunction

G Function.

= 0 : G function : G0, G1, G2, G3, G61 for polynomial blocks. == -1 : no G function active

public short GFunction

Field Value

short ↗

Struct SOLLKONT_VISU_DATA_V0

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Version 0 format of visualisation data.

```
public struct SOLLKONT_VISU_DATA_V0
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Simu_achs_data_std

Axis-specific visualisation data.

```
public SOLLKONT_VISU_ACHS_DATA_STD[] Simu_achs_data_std
```

Field Value

[SOLLKONT_VISU_ACHS_DATA_STD\[\]](#)

Visu_data_std

Visualization data

```
public SOLLKONT_VISU_CH_DATA_STD Visu_data_std
```

Field Value

[SOLLKONT_VISU_CH_DATA_STD](#)

Struct SOLLKONT_VISU_PDU

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Container that hold visualization data in any of the 11 existing versions.

```
public struct SOLLKONT_VISU_PDU
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

Count

number of structures SOLLKONT_VISU_DATA_V0 ... SOLLKONT_VISU_DATA_V5 in the current message

```
public int Count
```

Field Value

[int](#)

Version

Version identifier of visualisation data P-STUP-00039

```
public uint Version
```

Field Value

[uint](#)

v0

Structure with visualisation data if P-STUP-00039 has the value 0.

```
public SOLLKONT_VISU_DATA_V0[] v0
```

Field Value

[SOLLKONT VISU DATA V0\[\]](#)

Class SingleBlock

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

When single-step mode is active, the machine operator has the option to execute an NC program step by step. The operator releases every NC line one by one. Comment lines or comment blocks and skipped blocks are skipped.

```
public class SingleBlock
```

Inheritance

[object](#) ← SingleBlock

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Enabled

This object defines whether the single block mode is active.

```
public bool Enabled { get; set; }
```

Property Value

[bool](#)

Enum SkipModes

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Enumeration that describes the skip mode at interpreter level. Skip levels active simultaneously are enabled by bitwise ORing.

```
public enum SkipModes : uint
```

Fields

Off = 0

Skip mode NC block OFF.

SkipLevel1 = 1

Skip level 1

SkipLevel10 = 512

Skip level 10

SkipLevel2 = 2

Skip level 2

SkipLevel3 = 4

Skip level 3

SkipLevel4 = 8

Skip level 4

SkipLevel5 = 16

Skip level 5

SkipLevel6 = 32

Skip level 6

SkipLevel17 = 64

Skip level 7

SkipLevel18 = 128

Skip level 8

SkipLevel19 = 256

Skip level 9

Struct TECHNO_UNIT_CH

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

A technology control unit contains elements for commanding, acknowledging and transferring any required parameters

```
public struct TECHNO_UNIT_CH
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

MSTH_PROCESS_CH

Depending on the content of the element TechnologyType this element contains the parameters of an M function/H function if the technology function type is HLI_INTF_M_FKT or HLI_INTF_H_FKT. S function (spindle) if the technology function type is HLI_INTF_SPINDEL. T function if the technology function type is HLI_INTF_TOOL.

```
public byte[] MSTH_PROCESS_CH
```

Field Value

[byte](#)[]

TechnologyType

The type of technology function is transferred here. M code, H code, S or T.

```
public TechnologyFunction TechnologyType
```

Field Value

done_w

Consumption data item. The CNC refreshes the data of the technology function only if this element is FALSE. After updating, the CNC sets this element to TRUE and so element done_w is set to FALSE. The PLC reads the data of the technology function if this element has the value TRUE. After the data is transferred, the PLC sets the value to FALSE.

```
public bool done_w
```

Field Value

[bool](#) ↗

fill_up_1

Reserved for future use.

```
public int fill_up_1
```

Field Value

[int](#) ↗

please_rw

By setting please_rw, the CNC signals to the PLC that the technology control unit is to be executed.

```
public bool please_rw
```

Field Value

[bool](#) ↗

Enum TechnologyFunction

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Enum for the selection of a technology function (M/H/S/T)

```
public enum TechnologyFunction : ushort
```

Fields

HLI_INTF_H_FKT = 2

H-Function

HLI_INTF_M_FKT = 1

M-Function

HLI_INTF_SPINDEL = 3

S-Function (spindle)

HLI_INTF_TOOL = 4

T-Function (tool)

Class TechnologyProcesses

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The TechnologyProcesses class manages the interaction with CNC technology units via an ADS client. It handles the acknowledgment of M and H codes, processes notifications from the CNC, and provides methods to read and acknowledge technology units.

```
public class TechnologyProcesses : IDisposable
```

Inheritance

[object](#) ← TechnologyProcesses

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

CountToBeAcknowledgedInThisBlock

Number of entries in the array ATechnoUnitChannel_Std (= number of technology functions to be acknowledged in this block)

```
public ushort CountToBeAcknowledgedInThisBlock { get; }
```

Property Value

[ushort](#)

Methods

AcknowledgeTechnologyUnit(int)

Acknowledge a technology unit. This is necessary to inform the CNC that processing of the NC program should continue.

```
public void AcknowledgeTechnologyUnit(int unitsIndex)
```

Parameters

unitsIndex [int](#)

Dispose()

Dispose the object. Deletes device notifications and handles on ADS side.

```
public void Dispose()
```

GetBlockByBlockSynchTechnologies()

Array of M/H/S/T technology functions with block-by-block synchronisation.

```
public TECHNO_UNIT_CH[] GetBlockByBlockSynchTechnologies()
```

Returns

[TECHNO_UNIT_CH\[\]](#)

Events

HcodeNeedsAcknowledge

Event that is triggered when a H code needs to be acknowledged.

```
public event EventHandler<HcodeNeedsAcknowledgeEventArgs>? HcodeNeedsAcknowledge
```

Event Type

[EventHandler](#) <HcodeNeedsAcknowledgeEventArgs>

McodeNeedsAcknowledge

Event that is triggered when a M code needs to be acknowledged.

```
public event EventHandler<McodeNeedsAcknowledgeEventArgs>? McodeNeedsAcknowledge
```

Event Type

[EventHandler](#) <McodeNeedsAcknowledgeEventArgs>

Class TipParameters

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

Control unit to manage data for parameterising continuous jog mode in manual mode, including flow control of user data.

```
public class TipParameters
```

Inheritance

[object](#) ← TipParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

EnableControlElement(bool)

Signal to CNC that the interface exists and we want to use it.

```
public void EnableControlElement(bool enabled)
```

Parameters

enabled [bool](#)

SignalCommandSemaphor(bool)

CNC accepts the commanded data if this element has the value TRUE and sets this element to the value FALSE after complete acceptance of the data. You should set this element to the value TRUE if all data to be commanded has been written.

```
public void SignalCommandSemaphor(bool signal)
```

Parameters

signal [bool](#)

WriteCommandElement(HLI_HB_TIP_PARAMETER)

Write the parameters for continuous jog mode to the CNC.

```
public void WriteCommandElement(HLI_HB_TIP_PARAMETER data)
```

Parameters

data [HLI HB TIP PARAMETER](#)

Class ZeroOffsets

Namespace: [TwinSharp.CNC](#)

Assembly: TwinSharp.dll

The ZeroOffsets class is responsible for handling zero offsets (G54 etc) in the CNC system.

```
public class ZeroOffsets
```

Inheritance

[object](#) ← ZeroOffsets

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

GetZeroOffsets(int)

Gets the zero offsets for a specific zero point index. G54 is index 1.

```
public int[] GetZeroOffsets(int zeroPointIndex)
```

Parameters

zeroPointIndex [int](#)

Returns

[int](#)[]

An array of all axes zero point.

SetZeroOffset(int, int, int)

Sets the zero offsets for a specific zero point index. G54 is index 1.

```
public void SetZeroOffset(int zeroPointIndex, int axisIndex, int offset)
```

Parameters

zeroPointIndex [int](#)

axisIndex [int](#)

offset [int](#)

Namespace TwinSharp.IPC

Classes

[IPC](#)

The IPC class represents a Beckhoff Industrial PC (IPC) and provides access to various hardware modules such as network cards, CPU, memory, display devices, operating system, fans, mainboard, UPS, and miscellaneous modules. It uses the AdsClient to connect to the IPC and read the available MDP modules, initializing the corresponding module objects based on their types. The class implements IDisposable to ensure proper disposal of the AdsClient.

[IpcCpu](#)

The IpcCpu class provides methods to interact with the CPU of a device via ADS (Automation Device Specification). It allows reading the CPU frequency, current CPU usage percentage, and current CPU temperature in Celsius.

[IpcDisplayDevice](#)

The IpcDisplayDevice class provides an interface to interact with a display device connected via TwinCAT ADS. It allows reading and writing various properties of the display such as active display mode, display mode description, primary display status, COM port, version, brightness, and light enabled status. It also provides a method to save the brightness setting persistently across power cycles.

[IpcFan](#)

Each fan for which information is available is represented by a dedicated MDP module instance (not all devices support this).

[IpcMainBoard](#)

This module provides mainboard information, such as type, serial number, production date, boot count, operating time, and temperature. This module is not supported by all devices, since it requires a special BIOS.

[IpcMemory](#)

The IpcMemory class provides methods to interact with the memory of a TwinCAT ADS device. It allows reading the allocated and available program memory, as well as storage memory and memory division. The class handles different subindexes for devices with more than 4 GB of RAM and WindowsCE devices.

[IpcMiscellaneous](#)

The IpcMiscellaneous class allows reading and writing of settings such as the startup state of the Numlock key, CE remote display state, security wizard enabled state, auto logon username, and auto-generate certificates. The class uses an AdsClient to communicate with the system and perform these operations.

[IpcNIC](#)

The IpcNIC class provides an interface to interact with network interface card (NIC) settings through the TwinCAT ADS protocol. It allows reading and writing of various NIC properties such as MAC address, IPv4 address, subnet mask, DHCP status, default gateway, DNS servers, and virtual device name. The class handles specific behaviors for different operating systems like Windows, WinCE, TC/BSD, and TC/RTOS.

[IpcOperatingSystem](#)

Represents the operating system running on the IPC. Provides properties to retrieve OS version information, build number, CSD version, and system uptime.

[IpcTime](#)

The IpcTime class provides methods to interact with time settings on the IPC. It allows getting and setting various time-related properties such as SNTP server address, SNTP refresh interval, seconds since 1970, textual date-time representation, timezone, and time offset.

[IpcTwinCAT](#)

The IpcTwinCAT class provides an interface to interact with a TwinCAT system using an AdsClient. It allows reading various system properties such as version, status, and configuration details. The class handles specific TwinCAT system attributes and provides methods to read these attributes from the TwinCAT system using ADS (Automation Device Specification) protocol.

[IpcUps](#)

The IpcUps class provides an interface to interact with an Uninterruptible Power Supply (UPS) device using the TwinCAT ADS protocol. It allows reading various properties of the UPS such as model, vendor name, version, revision, build, serial number, power status, communication status, battery status, battery capacity, battery runtime, persistent power fail count, power fail counter, fan error, no battery status, battery replace date, and interval service status. The class uses an AdsClient to communicate with the UPS device.

Class IPC

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IPC class represents a Beckhoff Industrial PC (IPC) and provides access to various hardware modules such as network cards, CPU, memory, display devices, operating system, fans, mainboard, UPS, and miscellaneous modules. It uses the AdsClient to connect to the IPC and read the available MDP modules, initializing the corresponding module objects based on their types. The class implements IDisposable to ensure proper disposal of the AdsClient.

```
public class IPC : IDisposable
```

Inheritance

[object](#) ← IPC

Implements

[IDisposable](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

IPC(AmsNetId)

Creates an representation of a Beckhoff IPC.

```
public IPC(AmsNetId target)
```

Parameters

target AmsNetId

Target where to find IPC. Use AmsNetId.Local for local access.

Properties

Cpu

Represents the CPU module of the IPC.

```
public IpcCpu? Cpu { get; }
```

Property Value

[IpcCpu](#)

DisplayDevices

Represents the display devices of the IPC.

```
public IpcDisplayDevice[] DisplayDevices { get; }
```

Property Value

[IpcDisplayDevice\[\]](#)

Fans

Represents the fans of the IPC.

```
public IpcFan[] Fans { get; }
```

Property Value

[IpcFan\[\]](#)

MainBoard

Represents the mainboard module of the IPC.

```
public IpcMainBoard? MainBoard { get; }
```

Property Value

[IpcMainBoard](#)

Memory

Represents the memory module of the IPC.

```
public IpcMemory? Memory { get; }
```

Property Value

[IpcMemory](#)

Miscellaneous

Represents the miscellaneous module of the IPC.

```
public IpcMiscellaneous? Miscellaneous { get; }
```

Property Value

[IpcMiscellaneous](#)

NICs

Represents the network cards of the IPC.

```
public IpcNIC[] NICs { get; }
```

Property Value

[IpcNIC\[\]](#)

OperatingSystem

Represents the operating system module of the IPC.

```
public IpcOperatingSystem? OperatingSystem { get; }
```

Property Value

[IpcOperatingSystem](#)

Time

Module for viewing and setting the time on the IPC.

```
public IpcTime? Time { get; }
```

Property Value

[IpcTime](#)

TwinCAT

Represents the TwinCAT module of the IPC.

```
public IpcTwinCAT? TwinCAT { get; }
```

Property Value

[IpcTwinCAT](#)

UPS

Represents the UPS module of the IPC.

```
public IpcUps? UPS { get; }
```

Property Value

[IpcUps](#)

Methods

Dispose()

Disposes the ads client.

```
public void Dispose()
```

Class IpcCpu

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcCpu class provides methods to interact with the CPU of a device via ADS (Automation Device Specification). It allows reading the CPU frequency, current CPU usage percentage, and current CPU temperature in Celsius.

```
public class IpcCpu
```

Inheritance

[object](#) ← IpcCpu

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Frequency

CPU frequency (constant)

```
public uint Frequency { get; }
```

Property Value

[uint](#)

TemperatureCelsius

Current CPU Temperature (°C). Requires BIOS API.

```
public short TemperatureCelsius { get; }
```

Property Value

[short](#) ↗

UsagePercent

Current CPU Usage (%)

```
public ushort UsagePercent { get; }
```

Property Value

[ushort](#) ↗

Class IpcDisplayDevice

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcDisplayDevice class provides an interface to interact with a display device connected via TwinCAT ADS. It allows reading and writing various properties of the display such as active display mode, display mode description, primary display status, COM port, version, brightness, and light enabled status. It also provides a method to save the brightness setting persistently across power cycles.

```
public class IpcDisplayDevice
```

Inheritance

[object](#) ← IpcDisplayDevice

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActiveDisplayModeID

ID of active display mode

```
public byte ActiveDisplayModeID { get; set; }
```

Property Value

[byte](#)

Brightness

Valid values: 20-100 (20 lowest brightness, 100 maximum brightness)

```
public uint Brightness { get; set; }
```

Property Value

[uint](#)

ComPort

Windows Embedded Standard (WES): e.g. "Com4" Windows CE: under Windows CE, the Com Port must end with a colon, e.g. "COM4:"

```
public string ComPort { get; set; }
```

Property Value

[string](#)

DisplayModeDescription

Description of active display mode

```
public string DisplayModeDescription { get; }
```

Property Value

[string](#)

IsPrimaryDisplay

True if this display is the primary display

```
public bool IsPrimaryDisplay { get; }
```

Property Value

[bool](#)

LightEnabled

Valid values: TRUE = background light ON, FALSE = background light OFF

```
public bool LightEnabled { get; set; }
```

Property Value

[bool](#)

Version

Version of the display device

```
public uint Version { get; }
```

Property Value

[uint](#)

Methods

SaveBrightnessPersistent()

Save brightness persistent between power cycles.

```
public void SaveBrightnessPersistent()
```

Class IpcFan

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

Each fan for which information is available is represented by a dedicated MDP module instance (not all devices support this).

```
public class IpcFan
```

Inheritance

[object](#) ← IpcFan

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

FanSpeedRPM

Fan speed (rpm)

```
public short FanSpeedRPM { get; }
```

Property Value

[short](#)

Class IpcMainBoard

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

This module provides mainboard information, such as type, serial number, production date, boot count, operating time, and temperature. This module is not supported by all devices, since it requires a special BIOS.

```
public class IpcMainBoard
```

Inheritance

[object](#) ← IpcMainBoard

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

BiosMajorVersion

BIOS major version

```
public byte BiosMajorVersion { get; }
```

Property Value

[byte](#)

BiosMinorVersion

BIOS minor version

```
public byte BiosMinorVersion { get; }
```

Property Value

[byte](#) ↗

BiosVersion

BIOS version

```
public string BiosVersion { get; }
```

Property Value

[string](#) ↗

BootCount

Number of times the device has been booted.

```
public uint BootCount { get; }
```

Property Value

[uint](#) ↗

MainBoardRevision

Mainboard revision

```
public byte MainBoardRevision { get; }
```

Property Value

[byte](#) ↗

MaxBoardTemperatureCelsius

Highest measured temperature.

```
public int MaxBoardTemperatureCelsius { get; }
```

Property Value

[int ↗](#)

MaxInputMilliVolts

Highest measured voltage.

```
public int MaxInputMilliVolts { get; }
```

Property Value

[int ↗](#)

MinBoardTemperatureCelsius

Lowest measured temperature.

```
public int MinBoardTemperatureCelsius { get; }
```

Property Value

[int ↗](#)

MinInputMilliVolts

Lowest measured voltage.

```
public int MinInputMilliVolts { get; }
```

Property Value

[int](#)

OperatingTimeMinutes

Operating time in minutes.

```
public uint OperatingTimeMinutes { get; }
```

Property Value

[uint](#)

ProductionDate

Production date of the mainboard.

```
public string ProductionDate { get; }
```

Property Value

[string](#)

SerialNumber

Serial number of the mainboard.

```
public string SerialNumber { get; }
```

Property Value

[string](#)

TemperatureCelsius

Current mainboard temperature °C.

```
public short TemperatureCelsius { get; }
```

Property Value

[short](#)

Type

Type of the mainboard.

```
public string Type { get; }
```

Property Value

[string](#)

Class IpcMemory

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcMemory class provides methods to interact with the memory of a TwinCAT ADS device. It allows reading the allocated and available program memory, as well as storage memory and memory division. The class handles different subindexes for devices with more than 4 GB of RAM and WindowsCE devices.

```
public class IpcMemory
```

Inheritance

[object](#) ← IpcMemory

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

MemoryDivision

Memory division. Only for WindowsCE.

```
public uint MemoryDivision { get; }
```

Property Value

[uint](#)

ProgramMemoryAllocated

Program Memory Allocated.

```
public ulong ProgramMemoryAllocated { get; }
```

Property Value

[ulong](#) ↗

ProgramMemoryAvailable

Program Memory Available.

```
public ulong ProgramMemoryAvailable { get; }
```

Property Value

[ulong](#) ↗

StorageMemoryAllocated

Storage Memory Allocated. Only for WindowsCE.

```
public uint StorageMemoryAllocated { get; }
```

Property Value

[uint](#) ↗

StorageMemoryAvailable

Storage Memory Available. Only for WindowsCE.

```
public uint StorageMemoryAvailable { get; }
```

Property Value

[uint](#) ↗

Class IpcMiscellaneous

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcMiscellaneous class allows reading and writing of settings such as the startup state of the Numlock key, CE remote display state, security wizard enabled state, auto logon username, and auto-generate certificates. The class uses an AdsClient to communicate with the system and perform these operations.

```
public class IpcMiscellaneous
```

Inheritance

[object](#) ← IpcMiscellaneous

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AutoGenerateCertificates

Auto Generate Certificates

```
public bool AutoGenerateCertificates { get; set; }
```

Property Value

[bool](#)

AutoLogonUsername

Auto Logon Username

```
public string AutoLogonUsername { get; }
```

Property Value

[string](#) ↗

CEremoteDisplayEnabled

CE Remote Display Enabled (WinCE only)

```
public bool CEremoteDisplayEnabled { get; set; }
```

Property Value

[bool](#) ↗

CEremoteDisplayState

CE remote display state shows whether a client is connected via CERHost. From MDP 1.6.x (WinCE only)

```
public bool CEremoteDisplayState { get; }
```

Property Value

[bool](#) ↗

SecurityWizardEnabled

Security Wizard Enabled

```
public bool SecurityWizardEnabled { get; set; }
```

Property Value

[bool](#) ↗

StartupNumlockState

Startup Numlock State. State of the Numlock key at system start

```
public bool StartupNumlockState { get; set; }
```

Property Value

[bool](#)

Class IpcNIC

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcNIC class provides an interface to interact with network interface card (NIC) settings through the TwinCAT ADS protocol. It allows reading and writing of various NIC properties such as MAC address, IPv4 address, subnet mask, DHCP status, default gateway, DNS servers, and virtual device name. The class handles specific behaviors for different operating systems like Windows, WinCE, TC/BSD, and TC/RTOS.

```
public class IpcNIC
```

Inheritance

[object](#) ← IpcNIC

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

DHCP

DHCP active.

```
public bool DHCP { get; set; }
```

Property Value

[bool](#)

IPv4Address

With WinCE a reboot may be required in order to obtain a correct value. Without a reboot WinCE may still supply the previous value!

```
public string IPv4Address { get; set; }
```

Property Value

[string](#)

IPv4DNSServers

Not for WinCE.

```
public string IPv4DNSServers { get; set; }
```

Property Value

[string](#)

IPv4DNSServersActive

Only for TC/BSD and TC/RTOS

```
public string IPv4DNSServersActive { get; }
```

Property Value

[string](#)

IPv4DefaultGateway

With WinCE a reboot may be required in order to obtain a correct value. Without a reboot WinCE may still supply the previous value! WinCE: depending on the DHCP status, a "Read" operation has the return value "DefaultGateway" or "DhcpDefaultGateway".

```
public string IPv4DefaultGateway { get; set; }
```

Property Value

[string](#)

IPv4SubNetMask

With WinCE a reboot may be required in order to obtain a correct value. Without a reboot WinCE may still supply the previous value!

```
public string IPv4SubNetMask { get; set; }
```

Property Value

[string](#)

MACAddress

MAC address of the card.

```
public string MACAddress { get; }
```

Property Value

[string](#)

VirtualDeviceName

Only for Windows.

```
public string VirtualDeviceName { get; set; }
```

Property Value

[string](#)

Class IpcOperatingSystem

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

Represents the operating system running on the IPC. Provides properties to retrieve OS version information, build number, CSD version, and system uptime.

```
public class IpcOperatingSystem
```

Inheritance

[object](#) ← IpcOperatingSystem

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

BuildNumber

OS Build Number

```
public uint BuildNumber { get; }
```

Property Value

[uint](#)

CSDVersion

OS CSD Version

```
public string CSDVersion { get; }
```

Property Value

[string](#)

MajorVersion

OS Major Version

```
public uint MajorVersion { get; }
```

Property Value

[uint](#)

MinorVersion

OS Minor Version

```
public uint MinorVersion { get; }
```

Property Value

[uint](#)

UpTimeSeconds

Uptime in seconds

```
public ulong UpTimeSeconds { get; }
```

Property Value

[ulong](#)

Class IpcTime

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcTime class provides methods to interact with time settings on the IPC. It allows getting and setting various time-related properties such as SNTP server address, SNTP refresh interval, seconds since 1970, textual date-time representation, timezone, and time offset.

```
public class IpcTime
```

Inheritance

[object](#) ← IpcTime

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

SNTPRefreshInSeconds

SNTP Refresh1 in Seconds On WindowsCE lowest allowed value is 5 Seconds. The system must be rebooted in order for the changes to take effect.

```
public uint SNTPRefreshInSeconds { get; set; }
```

Property Value

[uint](#)

SNTPServer

Name or IP Address of the timeserver "NoSync" = No synchronization "NT5DS" = Use domain hierarchy settings(Win32 only – no WinCE) May contain the following flags: See "NtpServer" msdn(Win32 only – no WinCE) The system must be rebooted in order for the changes to take effect.

```
public string SNTPServer { get; set; }
```

Property Value

[string](#)

SecondsSince1970

Seconds since midnight January 1, 1970 (local time)

```
public uint SecondsSince1970 { get; set; }
```

Property Value

[uint](#)

TextualDateTime

Textual DateTime presentation(local time) (ISO 8601) YYYY-MM-DDThh:mm:ss.sTZD

```
public string TextualDateTime { get; set; }
```

Property Value

[string](#)

TimeOffset

Time Offset – offset in seconds of the current local time relative to the coordinated universal time (UTC) (supports only steps of 15 minutes = 900 seconds) only for TC/RTOS

```
public int TimeOffset { get; set; }
```

Property Value

[int](#)

Timezone

Timezone - Zero based index of currently active timezone as listed in object 0x8nn2.Sub indizes in Object 0x8nn2 are one based.To lookup timezone information you need to query sub idx @ "this value"+1 Not for TC/RTOS

```
public ushort Timezone { get; set; }
```

Property Value

[ushort](#)

Class IpcTwinCAT

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcTwinCAT class provides an interface to interact with a TwinCAT system using an AdsClient. It allows reading various system properties such as version, status, and configuration details. The class handles specific TwinCAT system attributes and provides methods to read these attributes from the TwinCAT system using ADS (Automation Device Specification) protocol.

```
public class IpcTwinCAT
```

Inheritance

[object](#) ← IpcTwinCAT

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AmsNetID

AMS Net ID of the TwinCAT system. A restart of the computer is required in order to make a change to the Ams Net ID.

```
public string AmsNetID { get; }
```

Property Value

[string](#)

BuildNumber

TwinCAT build number.

```
public ushort BuildNumber { get; }
```

Property Value

[ushort](#)

Length

Length of the structure

```
public ushort Length { get; }
```

Property Value

[ushort](#)

LogFilePath

Only for WindowsCE

```
public string LogFilePath { get; }
```

Property Value

[string](#)

LogFileSize

Only for WindowsCE

```
public uint LogFileSize { get; }
```

Property Value

[uint](#)

MajorVersion

TwinCAT major version.

```
public ushort MajorVersion { get; }
```

Property Value

[ushort](#)

MinorVersion

TwinCAT minor version.

```
public ushort MinorVersion { get; }
```

Property Value

[ushort](#)

RegLevel

Only for TwinCAT 2.

```
public uint RegLevel { get; }
```

Property Value

[uint](#)

Revision

TwinCAT Revision

```
public ushort Revision { get; }
```

Property Value

[ushort](#) ↗

RunAsDevice

Only for WindowsCE

```
public ushort RunAsDevice { get; }
```

Property Value

[ushort](#) ↗

SecondsSinceLastStatusChange

Seconds since last TwinCAT status change

```
public ulong SecondsSinceLastStatusChange { get; }
```

Property Value

[ulong](#) ↗

ShowTargetVisu

Only for WindowsCE

```
public ushort ShowTargetVisu { get; }
```

Property Value

[ushort](#) ↗

Status

Status of the TwinCAT system.

```
public ushort Status { get; }
```

Property Value

[ushort](#)

SystemID

TwinCAT System ID

```
public string SystemID { get; }
```

Property Value

[string](#)

Class IpcUps

Namespace: [TwinSharp.IPC](#)

Assembly: TwinSharp.dll

The IpcUps class provides an interface to interact with an Uninterruptible Power Supply (UPS) device using the TwinCAT ADS protocol. It allows reading various properties of the UPS such as model, vendor name, version, revision, build, serial number, power status, communication status, battery status, battery capacity, battery runtime, persistent power fail count, power fail counter, fan error, no battery status, battery replace date, and interval service status. The class uses an AdsClient to communicate with the UPS device.

```
public class IpcUps
```

Inheritance

[object](#) ← IpcUps

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

BatteryCapacityPercent

Battery capacity in percent.

```
public byte BatteryCapacityPercent { get; }
```

Property Value

[byte](#)

BatteryReplaceDate

Date of the last battery change.

```
public string BatteryReplaceDate { get; }
```

Property Value

[string](#)

BatteryRuntimeSeconds

Battery runtime in seconds.

```
public uint BatteryRuntimeSeconds { get; }
```

Property Value

[uint](#)

BatteryStatus

Battery status.

```
public byte BatteryStatus { get; }
```

Property Value

[byte](#)

Build

Build.

```
public ushort Build { get; }
```

Property Value

[ushort](#)

CommunicationStatus

Communication status.

```
public byte CommunicationStatus { get; }
```

Property Value

[byte](#) ↗

FanError

Fan error.

```
public bool FanError { get; }
```

Property Value

[bool](#) ↗

IntervalServiceStatus

Interval Service Status indicates whether the configured service interval has elapsed.

```
public bool IntervalServiceStatus { get; }
```

Property Value

[bool](#) ↗

NoBattery

No battery.

```
public bool NoBattery { get; }
```

Property Value

[bool](#) ↗

PersistentPowerFailCount

Persistent Power Fail Count

```
public bool PersistentPowerFailCount { get; }
```

Property Value

[bool](#) ↗

PowerFailCounter

Power fail counter.

```
public uint PowerFailCounter { get; }
```

Property Value

[uint](#) ↗

PowerStatus

Power status.

```
public byte PowerStatus { get; }
```

Property Value

[byte](#) ↗

Revision

Revision.

```
public byte Revision { get; }
```

Property Value

[byte](#)

SerialNumber

Serial number.

```
public string SerialNumber { get; }
```

Property Value

[string](#)

UPSModel

UPS Model.

```
public string UPSModel { get; }
```

Property Value

[string](#)

VendorName

Vendor name.

```
public string VendorName { get; }
```

Property Value

[string](#) ↴

Version

Version.

```
public byte Version { get; }
```

Property Value

[byte](#) ↴

Namespace TwinSharp.NC

Classes

[Axis](#)

Represents an axis in a TwinCAT NC system, encapsulating its functions, parameters, state, cyclic process data, and associated sub-elements such as encoders, controllers, and drives.

[AxisCyclicProcessData](#)

The AxisCyclicProcessData class provides properties to interact with the cyclic process data of an axis in a TwinCAT NC system. It uses an AdsClient to read and write various control and status parameters of the axis, such as control word, controller enable, feed enable, referencing cam, velocity override, operation mode, actual position correction value, and external controller component. If the axis is linked to a PLC object, most of these values will be refused.

[AxisFunctions](#)

The AxisFunctions class provides a set of methods to control and manage the behavior of an axis in a TwinCAT NC system. It allows for operations such as resetting, stopping, referencing, and setting positions of the axis. Additionally, it supports advanced operations like controlled ramps, emergency stops, reversing operations, and sinus oscillation sequences. The class interacts with the TwinCAT system using an AdsClient to send commands and data to the specified axis.

[AxisParameters](#)

Represents the parameters of an axis in a TwinCAT NC system. This class provides properties to get and set various parameters of an axis, such as ID, name, type, cycle time, physical unit, velocities, monitoring settings, error reaction mode, and more. It uses an AdsClient to read and write these parameters from a TwinCAT system. The class also includes methods to read all sub-elements like encoder IDs, controller IDs, and drive IDs.

[AxisState](#)

Represents the state of an NC axis. Provides access to various properties and methods to read and manipulate the state of the axis, including position, velocity, acceleration, torque, and error codes.

[Channel](#)

Represents a channel in the TwinCAT NC system.

[ChannelCyclicProcessData](#)

The ChannelCyclicProcessData class provides methods to interact with the cyclic process data of a channel in a TwinCAT system. It allows reading and writing of speed override values for both the channel axis and the spindle.

[ChannelFunctions](#)

The ChannelFunctions class provides methods to interact with and control NC (Numerical Control) channels using an AdsClient. It allows loading NC programs by number or name, starting the interpreter, setting the interpreter operation mode, setting paths for subroutines, and controlling the channel with reset, stop, retry, and skip functionalities.

[ChannelParameters](#)

Represents the parameters of a channel in the TwinCAT NC system. This class provides properties to access various channel parameters such as ID, Name, Type, InterpreterType, ProgramLoadBufferSize, ProgramNumberJobList, InterpolationLoadLogMode, InterpolationTraceMode, RecordAllFeederEntries, NcLoggerLevel, G70Factor, G71Factor, ActivationOfDefaultGcode, and GroupId. These properties interact with the TwinCAT ADS client to read and write the respective values.

[ChannelState](#)

Represents the state of an NC (Numerical Control) channel, providing access to various channel properties such as error codes, group count, interpreter state, operation mode, and program information using an AdsClient.

[Controller](#)

Represents a controller in the TwinCAT NC system. This class provides access to the controller's parameters and state, allowing for the retrieval and manipulation of various control parameters and state information. It interacts with an AdsClient to communicate with the underlying system.

[ControllerParameters](#)

Represents the parameters of a controller, providing properties to get and set various control parameters such as ID, name, type, and control weights. This class interacts with an AdsClient to read and write parameter values from a specified index group.

[ControllerState](#)

The ControllerState class provides an interface to interact with a controller's state using an AdsClient. It allows reading various parameters of the controller such as error state, output in absolute units, output in percent, and output in volts.

[Drive](#)

The Drive class represents a drive in a TwinCAT system. It provides access to the drive's parameters and state through the DriveParameters and DriveState classes, respectively. The class is initialized with an AdsClient and a drive ID, which are used to interact with the drive's properties and state.

[DriveFunctions](#)

The DriveFunctions class provides methods to interact with and manipulate drive tables. It allows for the removal and deletion of characteristic drive tables.

[DriveParameters](#)

The DriveParameters class provides an interface to interact with drive parameters in a TwinCAT system using the AdsClient. It allows reading and writing various properties of a drive such as its ID, name, type, and motor polarity inversion status.

[DriveState](#)

Represents the state of a drive in a TwinCAT system. Provides properties to access various drive parameters such as error state, total output in absolute units, percent, and volts.

[Encoder](#)

The Encoder class represents an encoder device and provides access to its functions, parameters, and state. It uses an AdsClient to communicate with the encoder via the TwinCAT ADS protocol. The class contains three main properties:

- Functions: Provides methods to interact with and control the encoder.
- Parameters: Allows reading and writing various encoder settings.
- State: Provides access to various encoder states and properties.

[EncoderFunctions](#)

The EncoderFunctions class provides methods to interact with and control encoder devices via the TwinCAT ADS protocol. It includes functionalities to set and reinitialize the actual position of the encoder, activate and deactivate touch probes and external latches, and set external latch events. The class uses an AdsClient to communicate with the encoder and sends commands using specific index groups and offsets.

[EncoderParameters](#)

The EncoderParameters class provides an interface to interact with encoder parameters via an AdsClient. It allows reading and writing various encoder settings such as ID, name, type, scaling factor, position offset, count direction, modulo factor, mode, soft end monitoring, soft end positions, evaluation direction, and filter times. The class uses an AdsClient to communicate with the encoder and retrieve or update these parameters.

[EncoderState](#)

The EncoderState class provides access to various encoder states and properties through an AdsClient instance. It allows reading and writing of encoder-related data such as error codes, actual positions, velocities, accelerations, and other relevant metrics.

[Group](#)

Represents a group in the TwinCAT NC system.

[GroupFunctions](#)

The GroupFunctions class provides methods to control and manage an axis group in a TwinCAT NC (Numerical Control) system. It allows for resetting, stopping, clearing, and performing an emergency

stop on the group. Additionally, it supports starting and managing FIFO (First In, First Out) operations for the group.

[GroupParameters](#)

The GroupParameters class provides access to various parameters of a group in the TwinCAT NC system. It allows reading and writing of group-specific settings such as ID, name, type, cycle times, and FIFO configurations. This class interacts with the TwinCAT ADS client to perform read and write operations on the group parameters.

[GroupState](#)

The GroupState class provides properties to interact with and retrieve various states and information from a TwinCAT NC group via an AdsClient. It includes properties for error codes, axis counts, group states, and emergency stop status, among others. Each property reads or writes data from the TwinCAT system using specific index groups and offsets.

[NC](#)

The NC class provides access to the NC (Numerical Control) system using TwinCAT ADS protocol. It initializes and manages the Ring0Manager, Axes, Channels, Groups and Tables components.

[Ring0Manager](#)

Manages the low level Ring 0 operations for the NC system.

[Ring0Parameters](#)

Represents the parameters for Ring 0, providing access to various settings and configurations related to the SAF and SVB tasks, global time compensation, and cyclic data consistency.

[Ring0State](#)

Represents the state of the Ring 0 system, providing access to various counts and IDs for channels, groups, axes, encoders, controllers, drives, and tables.

[Table](#)

The **Table** class encapsulates various functionalities, parameters, and states related to a table in the context of TwinCAT ADS. It provides a structured way to interact with tables, which are likely used for motion control or other automation tasks.

[TableFunctions](#)

The TableFunctions class provides methods to generate and delete various types of tables with specified dimensions and interpolation types. It interacts with a TwinCAT AdsClient to perform these operations. The class supports generating general tables, valve diagram tables, and motion function tables, each with specific table types and dimensions.

[TableParameters](#)

The TableParameters class provides methods to interact with table parameters in a TwinCAT ADS system. It allows reading and writing various table properties such as ID, Name, SubType, MainType, LineCount, ColumnCount, TotalCount, StepWidth, MasterPeriod, and SlaveDifferencePerMasterPeriod. It also provides methods to get and set the activation mode for online changes, read and write single values in the table, and convert slave positions to master positions.

[TableState](#)

Represents the state of a table in a TwinCAT NC (Numerical Control) system. This class provides access to the 'User Counter'. It uses an AdsClient to communicate with the TwinCAT system and read the necessary data.

Structs

[NCAXISSTATE_ONLINESTRUCT](#)

AXIS ONLINE STRUCTURE (NC/CNC)

Enums

[ActualPositionType](#)

Enumeration of the different types of actual position that exist in TwinCAT NC.

[AxisType](#)

TwinCAT supports different axis types, which are defined in the enum AxisType.

[CamScalingMode](#)

Enumeration of type and scope of the scaling of a cam plate coupling.

[ChannelType](#)

Enumeration of the different types of channels that exists in TwinCAT NC.

[ControllerType](#)

Enumeration of the different types of controllers that exist in TwinCAT NC.

[CoupleState](#)

Coupling state of the axis

[DriveOutputStartType](#)

Enumeration of the different types of drive output start that exist in TwinCAT NC.

[EncoderEvaluationDirection](#)

Specifies the mode of evaluation for encoder signals.

[EncoderMode](#)

Enumeration of the different types of encoder modes that exist in TwinCAT NC.

[EncoderType](#)

Enumeration of all possible encoder types in TwinCAT NC.

[EndPositionType](#)

Enumeration of the different types of end position types (new end position) that exist in TwinCAT NC.

[ErrorReactionMode](#)

Enumeration of the possible error reaction modes that exist in TwinCAT NC.

[FifoInterpolationType](#)

Enumeration of the different ways that the TwinCAT NC FIFO can interpolate between supplied points.

[FifoOverrideType](#)

Determines the behaviour how quickly the FIFO should adapt to a new override.

[GroupAxisStartType](#)

Used by NC axis functions such as StandardAxisStart to define the start type of the axis.

[GroupType](#)

Enumeration of all possible group types in TwinCAT NC.

[InterpolationLoadLogMode](#)

Enumeration of the different types of interpolation load log modes that exist in TwinCAT NC.

[InterpolationTraceMode](#)

Enumeration of the different types of interpolation trace modes that exist in TwinCAT NC.

[InterpreterOperationMode](#)

Enumeration of the different modes that the TwinCAT NC interpreter can operate in.

[InterpreterState](#)

Enumeration of all possible states of the interpreter in TwinCAT NC. The interpreter state reflects the current state of the Interpreter State Machine. The complete list is given below.

[InterpreterType](#)

Enumeration of the different types of interpreters that exist in TwinCAT NC.

[NCTOPLC_AXIS_REF_OPMODE](#)

The AxisControlDWord is a 32 bit data word in the axis interface PLC->NC.

[NcDriveType](#)

TwinCAT supports different drives, these are defined in the enum DriveType.

[ProbeMode](#)

Enumeration of the different types of probe modes that exist in TwinCAT NC.

[SignalEdge](#)

Enumeration of the two types of signal edges that exist. Rising edge and falling edge.

[StateDWordFlags](#)

The StateDWord is a 32 bit data word in the axis interface NC->PLC.

[TableActivationMode](#)

Enumeration of the different types of table activation modes that exist in TwinCAT NC.

[TableInterpolationType](#)

Interpolation mode for position tables (cam plates). Position tables consist of a list of master and slave positions between which interpolation can take place in different ways. The interpolation type is not used for extended cam plates(motion functions).

[TableMainType](#)

Enumeration of the different types of main tables that exist in TwinCAT NC.

[TableSubType](#)

Enumeration of the different types of table subtypes that exist in TwinCAT NC.

Enum ActualPositionType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of actual position that exist in TwinCAT NC.

```
public enum ActualPositionType
```

Fields

AbsolutePosition = 1

Absolute position

ModuloPosition = 5

Modulo position

NOT_DEFINED = 0

Actual position type not defined.

RelativePosition = 2

Relative position

Class Axis

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents an axis in a TwinCAT NC system, encapsulating its functions, parameters, state, cyclic process data, and associated sub-elements such as encoders, controllers, and drives.

```
public class Axis
```

Inheritance

[object](#) ← Axis

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

Axis(AdsClient, uint)

Creates a new axis object representation of the given ID.

```
public Axis(AdsClient client, uint id)
```

Parameters

client AdsClient

id [uint](#)

Properties

Controllers

Gets the array of controllers associated with the axis.

```
public Controller[] Controllers { get; }
```

Property Value

[Controller\[\]](#)

CyclicProcessData

Gets the cyclic process data of the axis.

```
public AxisCyclicProcessData CyclicProcessData { get; }
```

Property Value

[AxisCyclicProcessData](#)

Drives

Gets the array of drives associated with the axis.

```
public Drive[] Drives { get; }
```

Property Value

[Drive\[\]](#)

Encoders

Gets the array of encoders associated with the axis.

```
public Encoder[] Encoders { get; }
```

Property Value

[Encoder\[\]](#)

Functions

Gets the functions available for the axis.

```
public AxisFunctions Functions { get; }
```

Property Value

[AxisFunctions](#)

Parameters

Gets the parameters of the axis.

```
public AxisParameters Parameters { get; }
```

Property Value

[AxisParameters](#)

State

Gets the state of the axis.

```
public AxisState State { get; }
```

Property Value

[AxisState](#)

Methods

ToString()

Returns the name of the axis.

```
public override string ToString()
```

Returns

[string](#)

Class AxisCyclicProcessData

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The AxisCyclicProcessData class provides properties to interact with the cyclic process data of an axis in a TwinCAT NC system. It uses an AdsClient to read and write various control and status parameters of the axis, such as control word, controller enable, feed enable, referencing cam, velocity override, operation mode, actual position correction value, and external controller component. If the axis is linked to a PLC object, most of these values will be refused.

```
public class AxisCyclicProcessData
```

Inheritance

[object](#) ← AxisCyclicProcessData

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualPositionCorrectionValue

Actual position correction value (measurement system error correction)

```
public double ActualPositionCorrectionValue { get; set; }
```

Property Value

[double](#)

ControlWord

Control double word

```
public int ControlWord { get; set; }
```

Property Value

[int](#)

ControllerEnable

Controller enable

```
public bool ControllerEnable { get; set; }
```

Property Value

[bool](#)

ExternalControllerComponent

External controller component (position controller component)

```
public double ExternalControllerComponent { get; set; }
```

Property Value

[double](#)

FeedEnableMinus

Feed enable minus

```
public bool FeedEnableMinus { get; set; }
```

Property Value

[bool](#)

FeedEnablePlus

Feed enable plus

```
public bool FeedEnablePlus { get; set; }
```

Property Value

[bool](#) ↗

OperationMode

Operation mode axis

```
public uint OperationMode { get; set; }
```

Property Value

[uint](#) ↗

ReferencingCam

Referencing cam

```
public bool ReferencingCam { get; set; }
```

Property Value

[bool](#) ↗

VelocityOverride

Velocity override (1000000 == 100%)

```
public uint VelocityOverride { get; set; }
```

Property Value

[uint](#) ↗

Class AxisFunctions

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The AxisFunctions class provides a set of methods to control and manage the behavior of an axis in a TwinCAT NC system. It allows for operations such as resetting, stopping, referencing, and setting positions of the axis. Additionally, it supports advanced operations like controlled ramps, emergency stops, reversing operations, and sinus oscillation sequences. The class interacts with the TwinCAT system using an AdsClient to send commands and data to the specified axis.

```
public class AxisFunctions
```

Inheritance

[object](#) ← AxisFunctions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

ActivateCompleteAxis()

Activate complete axis (enable)

```
public void ActivateCompleteAxis()
```

ActivateDriveOutput()

Activate drive output (enable).

```
public void ActivateDriveOutput()
```

ChangeDriveOutput(DriveOutputStartType, double)

Change the drive output.

```
public void ChangeDriveOutput(DriveOutputStartType startType, double newValue)
```

Parameters

startType [DriveOutputStartType](#)

newValue [double](#)

Required output value (e.g. %)

ClearAxisTask()

Clear axis (task)

```
public void ClearAxisTask()
```

DeactivateCompleteAxis()

Deactivate complete axis (disable)

```
public void DeactivateCompleteAxis()
```

DeactivateDriveOutput()

Deactivate drive output (disable).

```
public void DeactivateDriveOutput()
```

EmergencyStopWithControlledRamp(double, double)

Emergency stop with controlled ramp

```
public void EmergencyStopWithControlledRamp(double deceleration, double jerk)
```

Parameters

deceleration [double](#)

Deceleration (must be greater than or equal to the original deceleration)

jerk [double](#)

Jerk (must greater than or equal to the original jerk)

ExtendedAxisStart(GroupAxisStartType, double, double, double, double)

Extended axis start.

```
public void ExtendedAxisStart(GroupAxisStartType startType, double targetPosition, double requireVelocity, double acceleration = 0, double deceleration = 0, double jerk = 0)
```

Parameters

startType [GroupAxisStartType](#)

targetPosition [double](#)

requireVelocity [double](#)

acceleration [double](#)

0 if internal TwinCAT acceleration should be used.

deceleration [double](#)

0 if internal TwiNCAT deceleration should be used.

jerk [double](#)

0 if internal TwinCAT jerk should be used.

NewEndPositionAxis(EndPositionType, double)

Set new end position (axis).

```
public void NewEndPositionAxis(EndPositionType endPositionType, double newEndPosition)
```

Parameters

endPositionType [EndPositionType](#)

newEndPosition [double](#)

New end position (target position)

OrientedStop(double, double, double)

Oriented stop (oriented end position). Only for PTP axes.

```
public void OrientedStop(double moduloEndPosition, double deceleration, double jerk)
```

Parameters

moduloEndPosition [double](#)

deceleration [double](#)

jerk [double](#)

ReferenceAxis()

Reference axis (calibration).

```
public void ReferenceAxis()
```

ReleaseParkingBrake(ushort)

Release parking brake? 0: automatic activation(default) 1: mandatorily always released Note: Reset to '0' when resetting the axis!

```
public void ReleaseParkingBrake(ushort release)
```

Parameters

release [ushort](#)

Reset()

Reset axis

```
public void Reset()
```

SetActualAxisPosition(ActualPositionType, double)

Set actual axis position

```
public void SetActualAxisPosition(ActualPositionType actualPositionType,  
double actualPosition)
```

Parameters

actualPositionType [ActualPositionType](#)

actualPosition [double](#)

SetActualPositionOnTheFly(ActualPositionType, int, double)

Set actual axis position on the fly (in motion of the axis)

```
public void SetActualPositionOnTheFly(ActualPositionType positionType, int controlword,  
double newActualPosition)
```

Parameters

`positionType` [ActualPositionType](#)

`controlword` [int](#)

Control double word, e.g. for "clearing the lag error"

`newActualPosition` [double](#)

SetExternalAxisError(uint)

Set external axis error (runtime error)

```
public void SetExternalAxisError(uint errorCode)
```

Parameters

`errorCode` [uint](#)

StandardAxisStart(GroupAxisStartType, double, double)

Standard axis start.

```
public void StandardAxisStart(GroupAxisStartType startType, double endPosition,
    double velocity)
```

Parameters

`startType` [GroupAxisStartType](#)

`endPosition` [double](#)

`velocity` [double](#)

StartDriveOutput(DriveOutputStartType, double)

Start drive output.

```
public void StartDriveOutput(DriveOutputStartType startType, double value)
```

Parameters

startType [DriveOutputStartType](#)

value [double](#)

StartReversingOperation(GroupAxisStartType, double, double, double, double)

Start reversing operation for positioning (SERVO).

```
public void StartReversingOperation(GroupAxisStartType startType, double targetPosition1,  
double targetPosition2, double velocity, double idleSeconds)
```

Parameters

startType [GroupAxisStartType](#)

targetPosition1 [double](#)

targetPosition2 [double](#)

velocity [double](#)

idleSeconds [double](#)

StartReversingOperationVelocityJumps(GroupAxisStartType, double, double, double, uint)

Start reversing operation with velocity jumps (SERVO): (can be used to determine the velocity step response)

```
public void StartReversingOperationVelocityJumps(GroupAxisStartType startType, double  
velocity1, double velocity2, double travelSeconds, double idleSeconds, uint repetitionCount)
```

Parameters

startType [GroupAxisStartType](#)

`velocity1` [double](#)

`velocity2` [double](#)

`travelSeconds` [double](#)

`idleSeconds` [double](#)

`repetitionCount` [uint](#)

`StartSinusOscillationSequence(double, double, double, double, double, double, uint)`

Sine oscillation sequence

- used as single sinus oscillation(sinus generator)
- used as sinus oscillation sequence(e.g. for bode plot)

```
public void StartSinusOscillationSequence(double baseAmplitude, double baseFrequency, double startAmplitude, double feedConstantMotor, double startFrequency, double stopFrequency, double stepDurationSeconds, uint stepCycles)
```

Parameters

`baseAmplitude` [double](#)

`baseFrequency` [double](#)

`startAmplitude` [double](#)

`feedConstantMotor` [double](#)

`startFrequency` [double](#)

`stopFrequency` [double](#)

`stepDurationSeconds` [double](#)

`stepCycles` [uint](#)

`Stop()`

Stop axis

```
public void Stop()
```

StopDriveOutput()

Stop drive output.

```
public void StopDriveOutput()
```

StopWithControlledRamp(double, double)

Parameterizable stop (with controlled ramp). Only for PTP axes.

```
public void StopWithControlledRamp(double deceleration, double jerk)
```

Parameters

deceleration [double](#)

jerk [double](#)

Class AxisParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the parameters of an axis in a TwinCAT NC system. This class provides properties to get and set various parameters of an axis, such as ID, name, type, cycle time, physical unit, velocities, monitoring settings, error reaction mode, and more. It uses an AdsClient to read and write these parameters from a TwinCAT system. The class also includes methods to read all sub-elements like encoder IDs, controller IDs, and drive IDs.

```
public class AxisParameters
```

Inheritance

[object](#) ← AxisParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Acceleration

Default data set (e.g. mm/s²)

```
public double Acceleration { get; set; }
```

Property Value

[double](#)

ChannelID

Channel ID

```
public uint ChannelID { get; }
```

Property Value

[uint](#)

ChannelName

Channel name

```
public string ChannelName { get; }
```

Property Value

[string](#)

ChannelType

Channel type

```
public ChannelType ChannelType { get; }
```

Property Value

[ChannelType](#)

ControllerCount

Number of controllers that belong to this axis.

```
public uint ControllerCount { get; }
```

Property Value

[uint](#)

CycleTime

Cycle time axis (SEC)

```
public uint CycleTime { get; }
```

Property Value

[uint](#)

Deceleration

Default data set (e.g. mm/s²)

```
public double Deceleration { get; set; }
```

Property Value

[double](#)

DriveCount

Number of drives that belong to this axis.

```
public uint DriveCount { get; }
```

Property Value

[uint](#)

EncoderCount

Number of encoders that belong to this axis.

```
public uint EncoderCount { get; }
```

Property Value

[uint](#)

EncoderIDs

Axis encoder IDs

```
public uint[] EncoderIDs { get; }
```

Property Value

[uint](#)[]

ErrorDelaySeconds

Error delay time (if delayed error reaction is selected)

```
public double ErrorDelaySeconds { get; set; }
```

Property Value

[double](#)

ErrorReactionMode

Error reaction mode: 0: instantaneous (default) 1: delayed (e.g. for Master/Slave-coupling)

```
public ErrorReactionMode ErrorReactionMode { get; set; }
```

Property Value

[ErrorReactionMode](#)

GroupID

Group ID

```
public uint GroupID { get; }
```

Property Value

[uint](#)

GroupName

Group name

```
public string GroupName { get; }
```

Property Value

[string](#)

GroupType

Group type

```
public GroupType GroupType { get; }
```

Property Value

[GroupType](#)

ID

Axis ID

```
public uint ID { get; }
```

Property Value

[uint](#)

Jerk

Default data set (e.g. mm/s³)

```
public double Jerk { get; set; }
```

Property Value

[double](#)

LoopEnabled

Loop enabled

```
public bool LoopEnabled { get; set; }
```

Property Value

[bool](#)

LoopingDistance

Looping distance (\pm) e.g. mm

```
public double LoopingDistance { get; set; }
```

Property Value

[double](#)

MaxPermittedAcceleration

Maximum permitted acceleration

```
public double MaxPermittedAcceleration { get; set; }
```

Property Value

[double](#) ↗

MaxPermittedDeceleration

Maximum permitted deceleration

```
public double MaxPermittedDeceleration { get; set; }
```

Property Value

[double](#) ↗

MotionMonitoringEnabled

Motion monitoring enabled

```
public bool MotionMonitoringEnabled { get; set; }
```

Property Value

[bool](#) ↗

MotionMonitoringSeconds

Motion monitoring time.

```
public double MotionMonitoringSeconds { get; set; }
```

Property Value

[double](#) ↗

Name

Axis name

```
public string Name { get; }
```

Property Value

[string](#) ↗

PhysicalUnit

Physical unit

```
public string PhysicalUnit { get; }
```

Property Value

[string](#) ↗

PositionRangeMonitoringEnabled

Position range monitoring?

```
public bool PositionRangeMonitoringEnabled { get; set; }
```

Property Value

[bool](#) ↗

PositionRangeMonitoringWindow

Position range monitoring window

```
public double PositionRangeMonitoringWindow { get; set; }
```

Property Value

double ↗

PulseWayNegativeDirection

Pulse way in neg. direction e.g. mm

```
public double PulseWayNegativeDirection { get; set; }
```

Property Value

double ↗

PulseWayPositiveDirection

Pulse way in pos. direction e.g. mm

```
public double PulseWayPositiveDirection { get; set; }
```

Property Value

double ↗

RefVelocityAtRefOutput

Reference velocity at reference output (velocity pre-control)

```
public double RefVelocityAtRefOutput { get; set; }
```

Property Value

double ↗

RefVelocityCamDirection

Ref. velocity in cam direction

```
public double RefVelocityCamDirection { get; set; }
```

Property Value

[double](#) ↗

RefVelocitySyncDirection

Ref. velocity in sync direction

```
public double RefVelocitySyncDirection { get; set; }
```

Property Value

[double](#) ↗

TargetPositionMonitoringEnabled

Target position monitoring enabled

```
public bool TargetPositionMonitoringEnabled { get; set; }
```

Property Value

[bool](#) ↗

TargetPositionMonitoringSeconds

Target position monitoring time in seconds.

```
public double TargetPositionMonitoringSeconds { get; set; }
```

Property Value

[double](#)

TargetPositionMonitoringWindow

Target position monitoring window e.g. mm

```
public double TargetPositionMonitoringWindow { get; set; }
```

Property Value

[double](#)

Type

Axis type

```
public AxisType Type { get; }
```

Property Value

[AxisType](#)

VelocityHandFast

Velocity hand fast

```
public double VelocityHandFast { get; set; }
```

Property Value

[double](#)

VelocityHandSlow

Velocity hand slow

```
public double VelocityHandSlow { get; set; }
```

Property Value

[double](#) ↗

VelocityRapidTraverse

Velocity rapid traverse

```
public double VelocityRapidTraverse { get; set; }
```

Property Value

[double](#) ↗

Class AxisState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the state of an NC axis. Provides access to various properties and methods to read and manipulate the state of the axis, including position, velocity, acceleration, torque, and error codes.

```
public class AxisState
```

Inheritance

[object](#) ← AxisState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualAcceleration

Optional: Actual acceleration "ActAcc"

```
public double ActualAcceleration { get; }
```

Property Value

[double](#)

ActualPosition

Actual position (charge with actual position compensation value) ("ActPos")

```
public double ActualPosition { get; }
```

Property Value

[double](#)

ActualPositionModulo

Modulo actual position "ActPosModulo"

```
public double ActualPositionModulo { get; }
```

Property Value

[double](#)

ActualRotationModulo

Modulo actual rotation

```
public int ActualRotationModulo { get; }
```

Property Value

[int](#)

ActualVelocity

Optional: Actual velocity "ActVelo"

```
public double ActualVelocity { get; }
```

Property Value

[double](#)

CouplingState

Coupling state

```
public uint CouplingState { get; }
```

Property Value

[uint](#)

CouplingTableIndex

Coupling table index

```
public uint CouplingTableIndex { get; }
```

Property Value

[uint](#)

CycleCounter

Set cycle counter (SAF timestamp)

```
public uint CycleCounter { get; }
```

Property Value

[uint](#)

DelayedErrorCode

Delayed error code (error pre-warning) in case of a delayed error reaction (see bit ErrorPropagationDelayed)

```
public uint DelayedErrorCode { get; }
```

Property Value

[uint](#)

ErrorCode

Axis state error code.

```
public uint ErrorCode { get; }
```

Property Value

[uint](#)

ExpectedTargetPosition

Expected target position

```
public double ExpectedTargetPosition { get; }
```

Property Value

[double](#)

InitializeCommandCounter

Counter for initialization command (InitializeCommandCounter)

```
public uint InitializeCommandCounter { get; }
```

Property Value

[uint](#)

LagErrorPeakMaximum

Peak hold value for maximum negative lag error of the position

```
public double LagErrorPeakMaximum { get; }
```

Property Value

[double](#)

LagErrorPeakMinimum

Peak hold value for minimum positive lag error of the position

```
public double LagErrorPeakMinimum { get; }
```

Property Value

[double](#)

LagErrorPosition

Lag error position(without dead time compensation)

```
public double LagErrorPosition { get; }
```

Property Value

[double](#)

LagErrorPositionWithDeadTimeCompensation

Lag error position(with dead time compensation) "PosDiff"

```
public double LagErrorPositionWithDeadTimeCompensation { get; }
```

Property Value

[double](#)

OnlineData

The online data of the axis.

```
public NCAXISSTATE_ONLINESTRUCT OnlineData { get; }
```

Property Value

[NCAXISSTATE_ONLINESTRUCT](#)

PositioningTimeLastMotionCommand

Positioning time of the last motion command (start → target position window)

```
public double PositioningTimeLastMotionCommand { get; }
```

Property Value

[double](#)

RemainingTravelDistance

Remaining travel distance e.g. mm (SERVO).

```
public double RemainingTravelDistance { get; }
```

Property Value

[double](#)

RemaniningTravelTime

Remaining travel time seconds (SERVO).

```
public double RemaniningTravelTime { get; }
```

Property Value

[double](#)

ResetCommandCounter

Counter for reset command (ResetCommandCounter)

```
public uint ResetCommandCounter { get; }
```

Property Value

[uint](#)

SetAcceleration

Set acceleration

```
public double SetAcceleration { get; }
```

Property Value

[double](#)

SetCommandNumber

Set command number ("CmdNo")

```
public int SetCommandNumber { get; }
```

Property Value

[int](#)

SetCouplingFactor

Set coupling factor (set gear ratio)

```
public double SetCouplingFactor { get; }
```

Property Value

[double](#)

SetJerk

Set jerk (time derivative of the set acceleration)

```
public double SetJerk { get; }
```

Property Value

[double](#)

SetModuloRotation

Modulo set rotation

```
public int SetModuloRotation { get; }
```

Property Value

[int](#)

SetPosition

Set position

```
public double SetPosition { get; }
```

Property Value

[double](#)

SetPositionModulo

Modulo set position

```
public double SetPositionModulo { get; }
```

Property Value

[double](#)

SetTorque

Set torque (rot. motor) or set force(linear motor) ("SetTorque")

```
public double SetTorque { get; }
```

Property Value

[double](#)

SetTorqueChange

Set torque change (rot. motor) or set force change (linear motor) (time derivative of the set torque or set force)

```
public double SetTorqueChange { get; }
```

Property Value

[double](#)

SetTravelDirection

Set travel direction [-1.0, 0.0, 1.0]

```
public double SetTravelDirection { get; }
```

Property Value

[double](#) ↗

SetVelocity

Set velocity

```
public double SetVelocity { get; }
```

Property Value

[double](#) ↗

SetVelocityOverride

Set override value for velocity [0.0...1.0] 1.0=100%

```
public double SetVelocityOverride { get; }
```

Property Value

[double](#) ↗

TorqueOffset

Additive set torque (rot. motor) or additive set force (linear motor) for pre-control. ("TorqueOffset")

```
public double TorqueOffset { get; set; }
```

Property Value

[double](#)

UncorrectedSetAcceleration

Uncorrected set acceleration e.g. mm/s²

```
public double UncorrectedSetAcceleration { get; }
```

Property Value

[double](#)

UncorrectedSetPosition

Uncorrected set position

```
public double UncorrectedSetPosition { get; }
```

Property Value

[double](#)

UncorrectedSetTravelDirection

Uncorrected set travel direction [-1.0, 0.0, 1.0]

```
public double UncorrectedSetTravelDirection { get; }
```

Property Value

[double](#)

UncorrectedSetVelocity

Uncorrected set velocity e.g. mm/s

```
public double UncorrectedSetVelocity { get; }
```

Property Value

[double](#) ↗

Enum AxisType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

TwinCAT supports different axis types, which are defined in the enum AxisType.

```
public enum AxisType
```

Fields

ContinuousAxisServo = 1

Continuous axis (also SERCOS)

ContinuousAxisStepper = 3

Stepper motor axis (without PWM terminal KL2502/30 and without pulse train KL2521)

DiscreteAxisHighLow = 2

Discrete axis (high/low speed)

EncoderAxis = 5

Encoder axis

Hydraulic = 6

Continuous axis with operation mode switching for position/pressure control

NOT_DEFINED = 0

Not defined

Specific = 100

Specific.

TimeBaseGenerator = 7

Time Base Generator

Enum CamScalingMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of type and scope of the scaling of a cam plate coupling.

```
public enum CamScalingMode
```

Fields

Off = 2

No modificaiton accepted.

UserDefined = 0

User defines scaling parameters -scaling and -offset

WithAutoOffset = 1

Offset is calculated automatically for best result

Class Channel

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents a channel in the TwinCAT NC system.

```
public class Channel
```

Inheritance

[object](#) ← Channel

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

Channel(AdsClient, uint)

Initializes a new instance of the [Channel](#) class.

```
public Channel(AdsClient client, uint id)
```

Parameters

client AdsClient

The ADS client connected to the target.

id [uint](#)

The channel ID.

Properties

CyclicProcessData

Gets the cyclic process data of the channel.

```
public ChannelCyclicProcessData CyclicProcessData { get; }
```

Property Value

[ChannelCyclicProcessData](#)

Functions

Gets the channel functions.

```
public ChannelFunctions Functions { get; }
```

Property Value

[ChannelFunctions](#)

Parameters

Gets the channel parameters.

```
public ChannelParameters Parameters { get; }
```

Property Value

[ChannelParameters](#)

State

Gets the channel state.

```
public ChannelState State { get; }
```

Property Value

ChannelState

Class ChannelCyclicProcessData

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The ChannelCyclicProcessData class provides methods to interact with the cyclic process data of a channel in a TwinCAT system. It allows reading and writing of speed override values for both the channel axis and the spindle.

```
public class ChannelCyclicProcessData
```

Inheritance

[object](#) ← ChannelCyclicProcessData

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

SpeedOverrideChannel

Speed override channel (Axis in the Channel). 1000000 = 100%

```
public uint SpeedOverrideChannel { get; set; }
```

Property Value

[uint](#)

SpeedOverrideSpindle

Speed override spindle. 1000000 = 100%

```
public uint SpeedOverrideSpindle { get; set; }
```

Property Value

[uint](#) ↗

Class ChannelFunctions

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The ChannelFunctions class provides methods to interact with and control NC (Numerical Control) channels using an AdsClient. It allows loading NC programs by number or name, starting the interpreter, setting the interpreter operation mode, setting paths for subroutines, and controlling the channel with reset, stop, retry, and skip functionalities.

```
public class ChannelFunctions
```

Inheritance

[object](#) ← ChannelFunctions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

LoadProgramByName(string)

Load NC program by name. The standard NC path does not have to be given although it may. Other paths are also permitted.

```
public void LoadProgramByName(string programName)
```

Parameters

programName [string](#)

LoadProgramByNumber(uint)

Load NC program with program number

```
public void LoadProgramByNumber(uint programNumber)
```

Parameters

programNumber [uint](#)

ResetChannel()

Reset channel

```
public void ResetChannel()
```

RetryChannel()

"Retry" Channel(restart Channel)

```
public void RetryChannel()
```

SetInterpreterOperationMode(InterpreterOperationMode)

Set the interpreter/channel operation mode

```
public void SetInterpreterOperationMode(InterpreterOperationMode mode)
```

Parameters

mode [InterpreterOperationMode](#)

SetPathForSubRoutines(string)

Set the path for subroutines

```
public void SetPathForSubRoutines(string path)
```

Parameters

path [string ↗](#)

SkipChannel()

"Skip" Channel (skip task/block)

```
public void SkipChannel()
```

StartInterpreter()

Start interpreter

```
public void StartInterpreter()
```

StopChannel()

Stop channel

```
public void StopChannel()
```

Class ChannelParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the parameters of a channel in the TwinCAT NC system. This class provides properties to access various channel parameters such as ID, Name, Type, InterpreterType, ProgramLoadBufferSize, ProgramNumberJobList, InterpolationLoadLogMode, InterpolationTraceMode, RecordAllFeederEntries, NcLoggerLevel, G70Factor, G71Factor, ActivationOfDefaultGcode, and GroupId. These properties interact with the TwinCAT ADS client to read and write the respective values.

```
public class ChannelParameters
```

Inheritance

[object](#) ← ChannelParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActivationOfDefaultGcode

Activation of default G-code. 0/1 default: FALSE

```
public ushort ActivationOfDefaultGcode { get; set; }
```

Property Value

[ushort](#)

G70Factor

Factor for G70.

```
public double G70Factor { get; set; }
```

Property Value

[double](#)

G71Factor

Factor for G71.

```
public double G71Factor { get; set; }
```

Property Value

[double](#)

GroupId

Group ID (only explicit for 3D and FIFO channel)

```
public uint GroupId { get; }
```

Property Value

[uint](#)

ID

Channel ID

```
public uint ID { get; }
```

Property Value

[uint](#)

InterpolationLoadLogMode

Load log mode

```
public InterpolationLoadLogMode InterpolationLoadLogMode { get; set; }
```

Property Value

[InterpolationLoadLogMode](#)

InterpolationTraceMode

Trace mode

```
public InterpolationTraceMode InterpolationTraceMode { get; set; }
```

Property Value

[InterpolationTraceMode](#)

InterpreterType

Interpreter type

```
public InterpreterType InterpreterType { get; }
```

Property Value

[InterpreterType](#)

Name

Channel name

```
public string Name { get; }
```

Property Value

[string](#) ↗

NcLoggerLevel

Channel specific level for NC logger messages 0: errors only 1: all NC messages

```
public uint NcLoggerLevel { get; set; }
```

Property Value

[uint](#) ↗

ProgramLoadBufferSize

Program load buffer size in bytes

```
public uint ProgramLoadBufferSize { get; }
```

Property Value

[uint](#) ↗

ProgramNumberJobList

Program no. according to job list

```
public uint ProgramNumberJobList { get; }
```

Property Value

[uint](#) ↗

RecordAllFeederEntries

Records all feeder entries in a log file named "TcNci.log"

```
public uint RecordAllFeederEntries { get; set; }
```

Property Value

[uint](#)

Type

Channel type

```
public ChannelType Type { get; }
```

Property Value

[ChannelType](#)

Class ChannelState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the state of an NC (Numerical Control) channel, providing access to various channel properties such as error codes, group count, interpreter state, operation mode, and program information using an AdsClient.

```
public class ChannelState
```

Inheritance

[object](#) ← ChannelState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

CurrentLoadedProgramName

Program name of currently loaded program (100 characters, null-terminated)

```
public string CurrentLoadedProgramName { get; }
```

Property Value

[string](#)

CurrentLoadedProgramNumber

Current loaded program number

```
public uint CurrentLoadedProgramNumber { get; }
```

Property Value

[uint](#)

ErrorCode

Error code Channel

```
public int ErrorCode { get; }
```

Property Value

[int](#)

GroupCount

Number of groups in the Channel

```
public uint GroupCount { get; }
```

Property Value

[uint](#)

InterpreterOperationMode

Interpreter/channel operation mode

```
public InterpreterOperationMode InterpreterOperationMode { get; }
```

Property Value

[InterpreterOperationMode](#)

InterpreterSimulationMode

Interpreter simulation mode 0: off (default) 1: on

```
public uint InterpreterSimulationMode { get; }
```

Property Value

[uint](#)

InterpreterState

Interpreter status

```
public InterpreterState InterpreterState { get; }
```

Property Value

[InterpreterState](#)

TextIndex

If the interpreter is in the aborted state, the current text index can be read out here

```
public uint TextIndex { get; }
```

Property Value

[uint](#)

Enum ChannelType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of channels that exists in TwinCAT NC.

```
public enum ChannelType
```

Fields

FIFO = 3

FIFO channel.

Interpreter = 2

Interpreter channel.

KinematicTransformation = 4

Kinematic transformation channel.

Standard = 1

Standard channel.

Class Controller

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents a controller in the TwinCAT NC system. This class provides access to the controller's parameters and state, allowing for the retrieval and manipulation of various control parameters and state information. It interacts with an AdsClient to communicate with the underlying system.

```
public class Controller
```

Inheritance

[object](#) ← Controller

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Parameters

Gets the parameters of the controller, which include various control parameters such as ID, name, type, and control weights.

```
public ControllerParameters Parameters { get; }
```

Property Value

[ControllerParameters](#)

State

Gets the state of the controller, which includes various state parameters such as error state, output in absolute units, output in percent, and output in volts.

```
public ControllerState State { get; }
```

Property Value

[ControllerState](#)

Class ControllerParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the parameters of a controller, providing properties to get and set various control parameters such as ID, name, type, and control weights. This class interacts with an AdsClient to read and write parameter values from a specified index group.

```
public class ControllerParameters
```

Inheritance

[object](#) ← ControllerParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

DampingTimeTd

Damping time Td. Position control. [0.0 ... 60.0]. Seconds.

```
public double DampingTimeTd { get; set; }
```

Property Value

[double](#)

DerivativeActionTimeTv

Derivative action time Tv. Position control. [0.0 ... 60.0]. Seconds.

```
public double DerivativeActionTimeTv { get; set; }
```

Property Value

[double](#)

ID

Controller ID

```
public uint ID { get; }
```

Property Value

[uint](#)

IntegralActionTimeTn

Integral action time Tn. Position control. [0.0 ... 60.0]. Seconds.

```
public double IntegralActionTimeTn { get; set; }
```

Property Value

[double](#)

MaxOutputLimitation

Maximum output limitation () for controller total output. (Standard value: 0.5 == 50%)

```
public double MaxOutputLimitation { get; set; }
```

Property Value

[double](#)

Name

Controller name

```
public string Name { get; }
```

Property Value

[string](#)

ProportionalGainKpOrKv

Proportional amplification factor kp resp. kv

```
public double ProportionalGainKpOrKv { get; set; }
```

Property Value

[double](#)

Type

Controller type

```
public ControllerType Type { get; }
```

Property Value

[ControllerType](#)

VelocityPreControlWeight

Weight of the velocity pre control (standard value: 1.0 = 100 %) [0.0 ... 1.0]

```
public double VelocityPreControlWeight { get; set; }
```

Property Value

double ↴

Class ControllerState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The ControllerState class provides an interface to interact with a controller's state using an AdsClient. It allows reading various parameters of the controller such as error state, output in absolute units, output in percent, and output in volts.

```
public class ControllerState
```

Inheritance

[object](#) ← ControllerState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ErrorState

Error state controller

```
public int ErrorState { get; }
```

Property Value

[int](#)

OutputAbsoluteUnits

Controller output in absolute units. Base Unit / s Symbolic access possible "CtrlOutput".

```
public double OutputAbsoluteUnits { get; }
```

Property Value

[double](#) ↗

OutputPercent

Controller output in percent

```
public double OutputPercent { get; }
```

Property Value

[double](#) ↗

OutputVolts

Controller output in volts

```
public double OutputVolts { get; }
```

Property Value

[double](#) ↗

Enum ControllerType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of controllers that exist in TwinCAT NC.

```
public enum ControllerType
```

Fields

High_LowSpeedController = 7

High/low speed controller (position).

NOT_DEFINED = 0

Controller not defined.

PID_Controller = 3

PID Controller (with ka) (position).

PI_ControllerVelocity = 6

PI Controller (velocity).

PP_Controller = 2

PP Controller (with ka) (position).

P_ControllerPosition = 1

P-controller (standard) (position).

P_ControllerVelocity = 5

P Controller (velocity).

SercosController = 9

Sercos Controller (Position in the drive)

StepperMotorController = 8

Stepper motor controller (position).

TComController = 14

TCom Controller (Soft Drive) (Position in the drive)

Enum CoupleState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Coupling state of the axis

```
public enum CoupleState
```

Fields

Master = 1

Master axis with any number of slaves (MASTER)

MasterSlave = 2

Slave axis that is the master of another slave (MASTERSLAVE)

Single = 0

Single axis that is neither a master nor a slave (SINGLE)

Slave = 3

Just a slave axis (SLAVE)

Class Drive

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The Drive class represents a drive in a TwinCAT system. It provides access to the drive's parameters and state through the DriveParameters and DriveState classes, respectively. The class is initialized with an AdsClient and a drive ID, which are used to interact with the drive's properties and state.

```
public class Drive
```

Inheritance

[object](#) ← Drive

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Functions

Gets the functions of the drive.

```
public DriveFunctions Functions { get; }
```

Property Value

[DriveFunctions](#)

Parameters

Gets the parameters of the drive.

```
public DriveParameters Parameters { get; }
```

Property Value

[DriveParameters](#)

State

Gets the state of the drive.

```
public DriveState State { get; }
```

Property Value

[DriveState](#)

Class DriveFunctions

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The DriveFunctions class provides methods to interact with and manipulate drive tables. It allows for the removal and deletion of characteristic drive tables.

```
public class DriveFunctions
```

Inheritance

[object](#) ← DriveFunctions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

RemoveAndDeleteCharacteristicDriveTable(ulong)

Remove and delete the characteristic drive table

```
public void RemoveAndDeleteCharacteristicDriveTable(ulong tableId)
```

Parameters

tableId [ulong](#)

Table-ID

Enum DriveOutputStartType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of drive output start that exist in TwinCAT NC.

```
public enum DriveOutputStartType
```

Fields

NOT_DEFINED = 0

Drive start type not defined.

Percent = 1

Output in percent.

Velocity = 2

Output in velocity.

Class DriveParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The DriveParameters class provides an interface to interact with drive parameters in a TwinCAT system using the AdsClient. It allows reading and writing various properties of a drive such as its ID, name, type, and motor polarity inversion status.

```
public class DriveParameters
```

Inheritance

[object](#) ← DriveParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ID

The ID of the drive.

```
public uint ID { get; }
```

Property Value

[uint](#)

MotorPolarityInverted

Writing is not allowed if the controller enable has been issued.

```
public bool MotorPolarityInverted { get; set; }
```

Property Value

[bool](#) ↗

Name

The name of the drive.

```
public string Name { get; }
```

Property Value

[string](#) ↗

Type

The type of the drive.

```
public NcDriveType Type { get; }
```

Property Value

[NcDriveType](#)

Class DriveState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the state of a drive in a TwinCAT system. Provides properties to access various drive parameters such as error state, total output in absolute units, percent, and volts.

```
public class DriveState
```

Inheritance

[object](#) ← DriveState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ErrorState

Error state of the drive.

```
public int ErrorState { get; }
```

Property Value

[int](#)

TotalOutputAbsoluteUnits

Total output in absolute units. Base unit / s Symbolic access possible: "DriveOutput"

```
public double TotalOutputAbsoluteUnits { get; }
```

Property Value

[double](#)

TotalOutputPercent

Total output in percent.

```
public double TotalOutputPercent { get; }
```

Property Value

[double](#)

TotalOutputVolts

Total output in volts.

```
public double TotalOutputVolts { get; }
```

Property Value

[double](#)

Class Encoder

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The Encoder class represents an encoder device and provides access to its functions, parameters, and state. It uses an AdsClient to communicate with the encoder via the TwinCAT ADS protocol. The class contains three main properties:

- Functions: Provides methods to interact with and control the encoder.
- Parameters: Allows reading and writing various encoder settings.
- State: Provides access to various encoder states and properties.

```
public class Encoder
```

Inheritance

[object](#) ← Encoder

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Functions

The EncoderFunctions class provides methods to interact with and control encoder devices via the TwinCAT ADS protocol. It includes functionalities to set and reinitialize the actual position of the encoder, activate and deactivate touch probes and external latches, and set external latch events. The class uses an AdsClient to communicate with the encoder and sends commands using specific index groups and offsets.

```
public EncoderFunctions Functions { get; }
```

Property Value

[EncoderFunctions](#)

Parameters

The EncoderParameters class provides an interface to interact with encoder parameters via an AdsClient. It allows reading and writing various encoder settings such as ID, name, type, scaling factor, position offset, count direction, modulo factor, mode, soft end monitoring, soft end positions, evaluation direction, and filter times. The class uses an AdsClient to communicate with the encoder and retrieve or update these parameters.

```
public EncoderParameters Parameters { get; }
```

Property Value

[EncoderParameters](#)

State

The EncoderState class provides access to various encoder states and properties through an AdsClient instance. It allows reading and writing of encoder-related data such as error codes, actual positions, velocities, accelerations, and other relevant metrics.

```
public EncoderState State { get; }
```

Property Value

[EncoderState](#)

Enum EncoderEvaluationDirection

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Specifies the mode of evaluation for encoder signals.

```
public enum EncoderEvaluationDirection
```

Fields

EvaluationBlocked = 3

Evaluation neither in positive nor in negative counting direction (evaluation blocked)

Negative = 2

Evaluation only in negative counting direction

Positive = 1

Evaluation only in positive counting direction

PositiveAndNegative = 0

Evaluation in positive and negative counting direction (default configuration, i.e. compatible with the previous state)

Class EncoderFunctions

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The EncoderFunctions class provides methods to interact with and control encoder devices via the TwinCAT ADS protocol. It includes functionalities to set and reinitialize the actual position of the encoder, activate and deactivate touch probes and external latches, and set external latch events. The class uses an AdsClient to communicate with the encoder and sends commands using specific index groups and offsets.

```
public class EncoderFunctions
```

Inheritance

[object](#) ← EncoderFunctions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

ActivateExternalLatchFallingEdge()

Activate "External Latch" or activate "measuring probe function" (falling edge)

KL5101,SERCOS,AX2xxx,PROFIDrive

```
public void ActivateExternalLatchFallingEdge()
```

ActivateExternalLatchFallingEdgeCANopen(bool, bool, bool, bool)

Activate "External Latch" 1 to 4 or activate "measuring probe function" 1 to 4 (falling edge) CANopen

```
public void ActivateExternalLatchFallingEdgeCANopen(bool latch1, bool latch2, bool latch3,  
bool latch4)
```

Parameters

latch1 [bool](#)

latch2 [bool](#)

latch3 [bool](#)

latch4 [bool](#)

ActivateExternalLatchRisingEdge()

Activate "External Latch" or activate "measuring probe function" (typically rising edge)

KL5101,SERCOS,AX2xxx,PROFIDrive

```
public void ActivateExternalLatchRisingEdge()
```

ActivateExternalLatchRisingEdgeCANopen(bool, bool, bool, bool)

Activate "External Latch" 1 to 4 or activate "measuring probe function" 1 to 4 (typically rising edge)

CANopen

```
public void ActivateExternalLatchRisingEdgeCANopen(bool latch1, bool latch2, bool latch3, bool latch4)
```

Parameters

latch1 [bool](#)

latch2 [bool](#)

latch3 [bool](#)

latch4 [bool](#)

ActivateTouchProbe(uint, SignalEdge, ProbeMode, uint)

Function group "TouchProbeV2": SERCOS/SoE, EtherCAT/CoE(CANopen DS402), SoftDrive(TCom), MDP 511 (EL5101, EL5151, EL5021, EL7041, EL7342)

```
public void ActivateTouchProbe(uint probeUnit, SignalEdge signalEdge, ProbeMode probeMode,  
uint signalSource)
```

Parameters

probeUnit [uint](#)

signalEdge [SignalEdge](#)

probeMode [ProbeMode](#)

signalSource [uint](#)

DeactivateExternalLatch()

Deactivate "external latch" or deactivate "measuring probe function" KL5101,SERCOS,AX2xxx,PROFIDrive

```
public void DeactivateExternalLatch()
```

DeactivateExternalLatchCANopen(bool, bool, bool, bool)

Deactivate "external latch" or deactivate "measuring probe function" CANopen

```
public void DeactivateExternalLatchCANopen(bool latch1, bool latch2, bool latch3,  
bool latch4)
```

Parameters

latch1 [bool](#)

latch2 [bool](#)

latch3 [bool](#)

latch4 [bool](#)

DeactivateTouchProbe(uint, SignalEdge)

Deactivate "touch probe" (external latch) Function group "TouchProbeV2": SERCOS/SoE, EtherCAT/CoE(CANopen DS402), SoftDrive(TCom), MDP 511 (EL5101, EL5151, EL5021, EL7041, EL7342)

```
public void DeactivateTouchProbe(uint probeUnit, SignalEdge signalEdge)
```

Parameters

probeUnit [uint](#)

signalEdge [SignalEdge](#)

ReInitActualEncoderPosition()

Re-initialization of the actual encoder position. Note: Takes effect for reference system „ABSOLUTE MULTITURN RANGE(with single overflow)" and „ABSOLUTE SINGLETURN RANGE(with single overflow)". NEW from TC3

```
public void ReInitActualEncoderPosition()
```

SetActualPosition(ActualPositionType, double)

Set actual position encoder/axis. Caution when using!

```
public void SetActualPosition(ActualPositionType actualPositionType, double position)
```

Parameters

actualPositionType [ActualPositionType](#)

position [double](#)

SetExternalLatchEvent(double)

Set "External latch event" and "External latch position" KL5101,SERCOS,AX2xxx,PROFIDrive Only for EtherCAT.

```
public void SetExternalLatchEvent(double position)
```

Parameters

position [double](#)

Enum EncoderMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of encoder modes that exist in TwinCAT NC.

```
public enum EncoderMode
```

Fields

NotDefined = 0

Encoder mode not defined

Position = 1

Determination of position.

PositionVelocity = 2

Determination of position and velocity.

PositionVelocityAcceleration = 3

Determination of position, velocity and acceleration.

Class EncoderParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The EncoderParameters class provides an interface to interact with encoder parameters via an AdsClient. It allows reading and writing various encoder settings such as ID, name, type, scaling factor, position offset, count direction, modulo factor, mode, soft end monitoring, soft end positions, evaluation direction, and filter times. The class uses an AdsClient to communicate with the encoder and retrieve or update these parameters.

```
public class EncoderParameters
```

Inheritance

[object](#) ← EncoderParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

EncoderCountDirectionInverted

Encoder count direction. Writing is not allowed if the controller enable has been issued.

```
public bool EncoderCountDirectionInverted { get; set; }
```

Property Value

[bool](#)

EncoderEvaluationDirection

```
public EncoderEvaluationDirection EncoderEvaluationDirection { get; set; }
```

Property Value

[EncoderEvaluationDirection](#)

EncoderMode

Encoder mode.

```
public EncoderMode EncoderMode { get; set; }
```

Property Value

[EncoderMode](#)

FilterSecondsAcceleration

Filter time for actual acceleration value in seconds (P-T1). [0.0 ... 60.0].

```
public double FilterSecondsAcceleration { get; set; }
```

Property Value

[double](#)

FilterSecondsPosition

Filter time for actual position value in seconds (P-T1). [0.0 ... 60.0].

```
public double FilterSecondsPosition { get; set; }
```

Property Value

[double](#)

FilterSecondsVelocity

Filter time for actual velocity value in seconds (P-T1). [0.0 ... 60.0].

```
public double FilterSecondsVelocity { get; set; }
```

Property Value

[double](#)

ID

Encoder ID [1 ... 255]

```
public uint ID { get; }
```

Property Value

[uint](#)

ModuloFactor

Modulo factor.

```
public double ModuloFactor { get; set; }
```

Property Value

[double](#)

Name

Encoder name

```
public string Name { get; }
```

Property Value

[string](#)

PositionOffset

Position offset. Writing is not allowed if the controller enable has been issued.

```
public double PositionOffset { get; set; }
```

Property Value

[double](#)

ScalingFactor

Resulting scaling factor (numerator / denominator) Note: from TC3 the scaling factor consists of two components – numerator and denominator (default: 1.0). Writing is not allowed if the controller enable has been issued.

```
public double ScalingFactor { get; set; }
```

Property Value

[double](#)

SoftEndMaxMonitoring

Soft end max. monitoring?

```
public bool SoftEndMaxMonitoring { get; set; }
```

Property Value

[bool](#)

SoftEndMinMonitoring

Soft end min. monitoring?

```
public bool SoftEndMinMonitoring { get; set; }
```

Property Value

[bool](#)

SoftEndPositionMax

Soft end position max.

```
public double SoftEndPositionMax { get; set; }
```

Property Value

[double](#)

SoftEndPositionMin

Soft end position min.

```
public double SoftEndPositionMin { get; set; }
```

Property Value

[double](#)

Type

Encoder type

```
public EncoderType Type { get; }
```

Property Value

[EncoderType](#)

Class EncoderState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The EncoderState class provides access to various encoder states and properties through an AdsClient instance. It allows reading and writing of encoder-related data such as error codes, actual positions, velocities, accelerations, and other relevant metrics.

```
public class EncoderState
```

Inheritance

[object](#) ← EncoderState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ActualAcceleration

Optional: Actual acceleration. Symbol: "ActAcc".

```
public double ActualAcceleration { get; }
```

Property Value

[double](#)

ActualDriveVelocity

Optional: actual drive velocity(transferred directly from SoE, CoE or MDP 742 drive) Base Unit / s New from TC3.1 B4020.30

```
public double ActualDriveVelocity { get; }
```

Property Value

[double](#)

ActualIncrements

Encoder actual increments.

```
public int ActualIncrements { get; }
```

Property Value

[int](#)

ActualModuloRotation

Modulo actual rotation.

```
public int ActualModuloRotation { get; }
```

Property Value

[int](#)

ActualPosition

Actual position (charge with actual position compensation value). Symbol: "ActPos".

```
public double ActualPosition { get; }
```

Property Value

[double](#)

ActualPositionCorrectionValue

Actual position correction value (measuring system error correction).

```
public double ActualPositionCorrectionValue { get; }
```

Property Value

[double](#) ↗

ActualPositionDueToDeadTimeCompensation

Actual position compensation value due to the dead time compensation.

```
public double ActualPositionDueToDeadTimeCompensation { get; }
```

Property Value

[double](#) ↗

ActualPositionFiltered

Filtered actual position (offset with actual position correction value, without dead time compensation)

```
public double ActualPositionFiltered { get; }
```

Property Value

[double](#) ↗

ActualPositionModulo

Modulo actual position. Symbol: "ActPosModulo".

```
public double ActualPositionModulo { get; }
```

Property Value

[double](#)

ActualPositionUnfiltered

Charge with actual position compensation value.

```
public double ActualPositionUnfiltered { get; }
```

Property Value

[double](#)

ActualPositionWithoutCompensation

Actual position without actual position compensation value.

```
public double ActualPositionWithoutCompensation { get; }
```

Property Value

[double](#)

ActualVelocity

Optional: Actual velocity. Symbol: "ActVel".

```
public double ActualVelocity { get; }
```

Property Value

[double](#)

ActualVelocityUnfiltered

Optional: Unfiltered actual velocity Base Unit / s

```
public double ActualVelocityUnfiltered { get; }
```

Property Value

[double](#) ↗

ErrorCode

Error state encoder

```
public uint ErrorCode { get; }
```

Property Value

[uint](#) ↗

ReferenceFlag

"Calibrate flag"

```
public bool ReferenceFlag { get; set; }
```

Property Value

[bool](#) ↗

SoftwareActualIncrements

Software actual increment counter.

```
public int SoftwareActualIncrements { get; }
```

Property Value

[int](#) ↗

TimeShiftSumDueToDeadTimeCompensation

Sum of time shift for encoder dead time (parameterized and variable dead time). Note: A dead time is specified in the system as a positive value.

```
public double TimeShiftSumDueToDeadTimeCompensation { get; }
```

Property Value

[double](#)

Methods

ReadActualPositionBuffer(out uint, out double)

Read the actual position buffer.

```
public void ReadActualPositionBuffer(out uint timestamp, out double position)
```

Parameters

timestamp [uint](#)

position [double](#)

Enum EncoderType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of all possible encoder types in TwinCAT NC.

```
public enum EncoderType
```

Fields

ABS_FOX50 = 12

T and R Fox 50 module (24 bits absolute (SSI))

ABS_KL30XX = 7

Absolute analog input with 16 bits (KL30xx)

ABS_KL5001_SSI = 5

MDP 500/501: Absolute SSI with 24 bits (KL5001, IP5009) (MDP 501: EL5001)

ABS_M2510 = 11

Absolute analog input with 12 bits (M2510)

ABS_M3000 = 2

Absolute, with 24 or 25 bits, and 12 and 13 bits single turn encoders (M3000)

AX2000_B200 = 14

Incremental AX2000-B200 Lightbus with 16/20 bits (AX2000)

AX2000_B900 = 20

Incremental AX2000-B900 Ethernet

CANOPEN_DS402_MDP513_MDP742 = 19

MDP 513 / MDP 742 (DS402): CANopen and EtherCAT (AX2000-B510, AX2000-B1x0, EL7201, EL5032/32 Bit)

CANOPEN_LENZE = 18

Incremental CANopen Lenze

CANOPEN_MDP513_64BIT = 27

MDP 513 (DS402, EnDat2.2, 64 Bit): EL5032/64 Bit

HYDRAULIC_FORCE = 13

MMW type: force acquisition from Pa, Pb, Aa, Ab

INC_Binary = 10

Binary incremental encoders (0/1)

INC_KL5051 = 6

MDP 510: Absolute/Incremental BISSI with 16 bits (KL5051, PWM KL2502_30K(Frq-Cnt pulse mode), Pulse-Train KL2521, IP2512)

INC_KL5101 = 4

MDP 511: Incremental with 16 bits and latch (MDP511: EL7041, EL5101, EL5151, EL2521, EL5021(SinCos); KL5101, IP5109, KL5111)

INC_M31X0 = 3

Incremental, with 24 bits (M31x0, M3200, M3100, M2000)

INC_Sercos_P = 8

SERCOS "Encoder" position

INC_Sercos_PV = 9

SERCOS "Encoder" position and velocity

INC_TCOM = 26

TCOM Encoder -> Interface to Soft Drive Encoder

IP5209 = 22

Incremental with 16-bit counter and int. 32-bit latch (IP5209)

KL2531_Stepper = 23

Incremental with 16-bit counter and int. + ext. 15-bit latch (only switchable) (stepper motor terminal KL2531/KL2541)

KL2532_DC = 24

Incremental with 16-bit counter and ext. 16-bit latch (only switchable) (2-channel DC motor output stage KL2532/KL2542), 2-channel PWM DC motor output stage KL2535/KL2545

KL5151 = 21

Incremental with 16-bit counter and int. + ext 32-bit latch (KL5151_0000) (only switchable), the 2-channel KL5151_0050 has no latch.

NCBACKPLANE = 17

Incremental NC backplane

NOT_DEFINED = 0

Not defined

PROFIDRIVE = 15

Incremental with 32 Bit

SPECIFIC = 28

Specific

Simulation = 1

Simulation encoder

TIMEBASEGENERATOR = 25

Time Base Generator

UNIVERSAL = 16

Incremental with variable bit mask (max. 32 bits)

Enum EndPositionType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of end position types (new end position) that exist in TwinCAT NC.

```
public enum EndPositionType
```

Fields

AbsolutePosition = 1

Absolute position

ContinousPositionNegative = 4

Continuous position negative

ContinousPositionPositive = 3

Continuous position positive

ModuloPosition = 5

Modulo position

NOT_DEFINED = 0

End position not defined.

RelativePosition = 2

Relative position

Enum ErrorReactionMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the possible error reaction modes that exist in TwinCAT NC.

```
public enum ErrorReactionMode
```

Fields

Delayed = 1

Delayed. E.g. for master/slave coupling.

Instantaneous = 0

Immediate, default.

Enum FifoInterpolationType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different ways that the TwinCAT NC FIFO can interpolate between supplied points.

```
public enum FifoInterpolationType
```

Fields

INTERPOLATIONTYPE_4POINT = 1

4 point interpolation.

INTERPOLATIONTYPE_CUBICSPLINE = 4

Cubic spline interpolation.

INTERPOLATIONTYPE_LINEAR = 0

Linear interpolation between points.

Enum FifoOverrideType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Determines the behaviour how quickly the FIFO should adapt to a new override.

```
public enum FifoOverrideType
```

Fields

OVERRIDETYPE_INSTANTANEOUS = 1

Instantaneous override. Changes the override instantaneously to the desired value. This means that it is up to the user to modify the override slowly enough that excessive following errors do not result from the change in override.

OVERRIDETYPE_PT2 = 2

Gradually adapt to the new override value, using the parameter P-T2 time.

Class Group

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents a group in the TwinCAT NC system.

```
public class Group
```

Inheritance

[object](#) ← Group

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

Group(AdsClient, uint)

Initializes a new instance of the [Group](#) class.

```
public Group(AdsClient client, uint id)
```

Parameters

client AdsClient

The ADS client.

id [uint](#)

The group ID.

Properties

Functions

Gets the functions of the group.

```
public GroupFunctions Functions { get; }
```

Property Value

[GroupFunctions](#)

Parameters

Gets the parameters of the group.

```
public GroupParameters Parameters { get; }
```

Property Value

[GroupParameters](#)

State

Gets the state of the group.

```
public GroupState State { get; }
```

Property Value

[GroupState](#)

Enum GroupAxisStartType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Used by NC axis functions such as StandardAxisStart to define the start type of the axis.

```
public enum GroupAxisStartType
```

Fields

AbsoluteStart = 1

Absolute start.

ContinousStartNegative = 4

Continuous start negative.

ContinousStartPositive = 3

Continuous start positive.

Halt = 8192

Halt (without motion lock).

JogNegativeFast = 529

Jog negative fast. Undocumented.

JogNegativeSlow = 273

Jog negative slow. Undocumented.

JogPositiveFast = 528

Jog positive fast. Undocumented.

JogPositiveSlow = 272

Jog positive slow. Undocumented.

ModuloStartNegativeDirection = 773

Modulo start in negative direction (with modulo tolerance window).

ModuloStartOLD = 5

Modulo start (OLD).

ModuloStartPositiveDirection = 517

Modulo start in positive direction (with modulo tolerance window).

ModuloStartShortestDistance = 261

Modulo start on the shortest distance.

NOT_DEFINED = 0

Undefined.

RelativeStart = 2

Relative start.

StopAndLock = 4096

Stop and lock (axis locked for motion commands).

Class GroupFunctions

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The GroupFunctions class provides methods to control and manage an axis group in a TwinCAT NC (Numerical Control) system. It allows for resetting, stopping, clearing, and performing an emergency stop on the group. Additionally, it supports starting and managing FIFO (First In, First Out) operations for the group.

```
public class GroupFunctions
```

Inheritance

[object](#) ← GroupFunctions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

Clear()

Clear group (buffer/task)

```
public void Clear()
```

EmergencyStop(double, double)

Emergency stop(E-stop) (emergency stop with controlled ramp)

```
public void EmergencyStop(double deceleration, double jerk)
```

Parameters

deceleration [double](#)

Deceleration (must be greater than or equal to the original deceleration)

jerk double[]

Jerk (must greater than or equal to the original jerk)

FifoOverwrite(double[])

Overwrite the last x FIFO entries (lines): (x*m)-values (one or more lines) n: FIFO length (number of lines)
m: FIFO dimension (number of columns) range of values x: [1 ... n]

```
public void FifoOverwrite(double[] entries)
```

Parameters

entries double[]

FifoStart()

Start FIFO group(FIFO table must have been filled in advance)

```
public void FifoStart()
```

FifoWrite(double[])

Write x FIFO entries (lines): (x*m)-values (one or more lines) n: FIFO length (number of lines) m: FIFO dimension (number of columns) range of values x: [1 ... n]

```
public void FifoWrite(double[] entries)
```

Parameters

entries double[]

Reset()

Reset group

```
public void Reset()
```

Stop()

Stop group

```
public void Stop()
```

Class GroupParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The GroupParameters class provides access to various parameters of a group in the TwinCAT NC system. It allows reading and writing of group-specific settings such as ID, name, type, cycle times, and FIFO configurations. This class interacts with the TwinCAT ADS client to perform read and write operations on the group parameters.

```
public class GroupParameters
```

Inheritance

[object](#) ← GroupParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

FifoDimension

FIFO dimension (m = number of axes)

```
public uint FifoDimension { get; }
```

Property Value

[uint](#)

FifoInterpolationType

Interpolation type for FIFO setpoint generator

```
public FifoInterpolationType FifoInterpolationType { get; }
```

Property Value

[FifoInterpolationType](#)

FifoLength

FIFO size(length) (n = number of FIFO entries)

```
public uint FifoLength { get; }
```

Property Value

[uint](#)

FifoOverrideType

Override type for FIFO setpoint generator

```
public FifoOverrideType FifoOverrideType { get; set; }
```

Property Value

[FifoOverrideType](#)

FifoTimeDelta

Time delta for two sequenced FIFO entries (FIFO entry timebase)

```
public double FifoTimeDelta { get; set; }
```

Property Value

[double](#)

FifoTimeForOverrideChange

P-T2 time for override change(T1= T2 = T0)

```
public double FifoTimeForOverrideChange { get; set; }
```

Property Value

[double](#)

ID

Group ID

```
public uint ID { get; }
```

Property Value

[uint](#)

MaxSafEntries

Size of the SAF table (max. number of SAF entries

```
public uint MaxSafEntries { get; }
```

Property Value

[uint](#)

MaxSvbEntries

Size of the SVB table (max. number of SVB entries

```
public uint MaxSvbEntries { get; }
```

Property Value

[uint](#)

Name

Group name

```
public string Name { get; }
```

Property Value

[string](#)

NumberInTheChannel

Number in the channel

```
public uint NumberInTheChannel { get; }
```

Property Value

[uint](#)

ParentChannelID

Channel ID that this group belongs to.

```
public uint ParentChannelID { get; }
```

Property Value

[uint](#)

ParentChannelName

Channel name that this group belongs to.

```
public string ParentChannelName { get; }
```

Property Value

[string](#)

ParentChannelType

Channel type that this group belongs to.

```
public ChannelType ParentChannelType { get; }
```

Property Value

[ChannelType](#)

SafCycleTime

SAF cycle time group

```
public uint SafCycleTime { get; }
```

Property Value

[uint](#)

SafCycleTimeDivisor

Internal SAF cycle time divisor (divides the internal SAF cycle time by this factor)

```
public uint SafCycleTimeDivisor { get; }
```

Property Value

[uint](#)

SingleBlockOperationMode

Single block operation mode.

```
public bool SingleBlockOperationMode { get; set; }
```

Property Value

[bool](#)

SvbCycleTime

SVB cycle time group

```
public uint SvbCycleTime { get; }
```

Property Value

[uint](#)

Type

Group type

```
public GroupType Type { get; }
```

Property Value

[GroupType](#)

Class GroupState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The GroupState class provides properties to interact with and retrieve various states and information from a TwinCAT NC group via an AdsClient. It includes properties for error codes, axis counts, group states, and emergency stop status, among others. Each property reads or writes data from the TwinCAT system using specific index groups and offsets.

```
public class GroupState
```

Inheritance

[object](#) ← GroupState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

CouplingState

Coupling state (state)

```
public uint CouplingState { get; }
```

Property Value

[uint](#)

CouplingTableIndex

Coupling table index

```
public uint CouplingTableIndex { get; }
```

Property Value

[uint](#) ↗

CurrentBlockNumber

Current block number (only active for interpolation group)

```
public uint CurrentBlockNumber { get; }
```

Property Value

[uint](#) ↗

CurrentFreeSafEntries

Current number of free SAF entries/tasks

```
public uint CurrentFreeSafEntries { get; }
```

Property Value

[uint](#) ↗

CurrentFreeSvbEntries

Current number of free SVB entries/tasks

```
public uint CurrentFreeSvbEntries { get; }
```

Property Value

[uint](#) ↗

CurrentSafEntries

Current number of SAF entries/tasks

```
public uint CurrentSafEntries { get; }
```

Property Value

[uint](#)

CurrentSvbEntries

Current number of SVB entries/tasks

```
public uint CurrentSvbEntries { get; }
```

Property Value

[uint](#)

EmergencyStopActive

Emergency Stop (E-Stop) active?

```
public bool EmergencyStopActive { get; set; }
```

Property Value

[bool](#)

ErrorCode

Error code group

```
public int ErrorCode { get; }
```

Property Value

[int](#)

MasterAxisCount

Number of master axes

```
public uint MasterAxisCount { get; }
```

Property Value

[uint](#)

MovingState

Moving state (state)

```
public uint MovingState { get; }
```

Property Value

[uint](#)

ReferencingState

Referencing state (state)

```
public uint ReferencingState { get; }
```

Property Value

[uint](#)

SafGroupState

SAF group state (main state)

```
public uint SafGroupState { get; }
```

Property Value

[uint](#)

SafSubGroupState

SAF sub-group state (sub state)

```
public uint SafSubGroupState { get; }
```

Property Value

[uint](#)

SlaveAxisCount

Number of slave axes

```
public uint SlaveAxisCount { get; }
```

Property Value

[uint](#)

SvbGroupState

SVB group state (state)

```
public uint SvbGroupState { get; }
```

Property Value

[uint](#)

Enum GroupType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of all possible group types in TwinCAT NC.

```
public enum GroupType
```

Fields

EncoderGroup = 9

Encoder Group + x Slave

FIFOGroup = 11

FIFO Group + x Slave

Group1D = 2

1D-Group + x Slave

Group2D = 3

2D-Group + x Slave

Group3D = 4

3D-Group + x Slave

HighLowSpeed = 5

High/low speed + x Slave

KinematicTransformationGroup = 12

Kinematic Transformation Group + x Slave

LowCostStepperMotor = 6

Low cost stepper motor (dig. IO) + x Slave

NOT_DEFINED = 0

Group not defined.

PTPGroup = 1

PTP-Group + x Slave

TableGroup = 7

Table Group + x Slave

Enum InterpolationLoadLogMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of interpolation load log modes that exist in TwinCAT NC.

```
public enum InterpolationLoadLogMode
```

Fields

LoaderLogOff = 0

Loader log off.

SourceAndCompiled = 2

Source and Compiled.

SourceOnly = 1

Source only.

Enum InterpolationTraceMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of interpolation trace modes that exist in TwinCAT NC.

```
public enum InterpolationTraceMode
```

Fields

TraceLineNumbers = 1

Trace line numbers

TraceOff = 0

Trace off

TraceSource = 2

Trace Source

Enum InterpreterOperationMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different modes that the TwinCAT NC interpreter can operate in.

```
public enum InterpreterOperationMode
```

Fields

Default = 0

Default (deactivates the other modes)

SingleBlockInterpreter = 16384

Single block mode in the interpreter

SingleBlockNC = 1

Single block mode in the NC core (Block execution task/SAF)

Enum InterpreterState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of all possible states of the interpreter in TwinCAT NC. The interpreter state reflects the current state of the Interpreter State Machine. The complete list is given below.

```
public enum InterpreterState
```

Fields

ITP_STATE_ABORTED = 12

If a runtime error occurs during the processing of an NC program, the interpreter goes into aborted state. The actual error code is given in Channel State.

ITP_STATE_ABORTING = 11

Interpreter aborting.

ITP_STATE_END = 9

Interpreter reached end.

ITP_STATE_FAULT = 13

Interpreter has a fault.

ITP_STATE_FLUSHBUFFERS = 17

Interpreter is flushing buffers.

ITP_STATE_IDLE = 1

The interpreter is in idle state if no NC program is loaded yet or if a group reset is executed. The interpreter also goes into idle state when a current program is stopped. In the case a group reset must be executed in order to prevent error 0x42C5. It is therefore recommended to execute a group reset after stopping via the PLC.

ITP_STATE_INITFAILED = 0

Initialization of interpreter failed.

ITP_STATE_READY = 2

After successful loading of an NC program, the interpreter is in ready state. After a program has been successfully processed and exited, the interpreter goes into ready state. In the meantime, however, other states are accepted.

ITP_STATE_RESET = 14

Interpreter is reset.

ITP_STATE_RUNNING = 5

Interpreter running.

ITP_STATE_SCANNING = 4

Interpreter scanning.

ITP_STATE_SEARCHLINE = 8

Interpreter block searching.

ITP_STATE_SINGLESTOP = 10

This state is only accepted in Single Block Mode. As soon as the entry has been sent from the interpreter to the NC core, the interpreter goes into this mode.

ITP_STATE_STARTED = 3

Interpreter started.

ITP_STATE_STAY_RUNNING = 6

Interpreter stay running.

ITP_STATE_STOP = 15

Interpreter is stopped.

ITP_STATE_WAITFUNC = 16

Interpreter is waiting for acknowledgement of M-function.

ITP_STATE_WRITETABLE = 7

Interpreter writing table.

Enum InterpreterType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of interpreters that exist in TwinCAT NC.

```
public enum InterpreterType
```

Fields

NCInterpreterDIN66025ClassicDialect = 2

NC Interpreter DIN 66025 (Classic Dialect).

NCInterpreterDIN66025GST = 1

NC Interpreter DIN 66025 (GST).

NotDefined = 0

Interpreter type not defined.

Class NC

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The NC class provides access to the NC (Numerical Control) system using TwinCAT ADS protocol. It initializes and manages the Ring0Manager, Axes, Channels, Groups and Tables components.

```
public class NC
```

Inheritance

[object](#) ← NC

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

NC(AmsNetId)

Initializes a new instance of the [NC](#) class. This constructor sets up the Ring0Manager and establishes a connection to the TwinCAT ADS client. It also initializes the Axes, Channels, Groups, and Tables components using the provided target AmsNetId.

```
public NC(AmsNetId target)
```

Parameters

target AmsNetId

The AmsNetId of the target device to connect to.

Properties

Axes

Gets the array of Axis objects representing the axes in the NC system.

```
public Axis[] Axes { get; }
```

Property Value

[Axis\[\]](#)

Channels

Gets the array of Channel objects representing the channels in the NC system.

```
public Channel[] Channels { get; }
```

Property Value

[Channel\[\]](#)

Groups

Gets the array of Group objects representing the groups in the NC system.

```
public Group[] Groups { get; }
```

Property Value

[Group\[\]](#)

Ring0Manager

Gets the Ring0Manager instance which manages the low-level operations and state of the NC system.

```
public Ring0Manager Ring0Manager { get; }
```

Property Value

Tables

Gets the array of Table objects representing the tables in the NC system.

```
public Table[] Tables { get; }
```

Property Value

[Table](#)[]

Struct NCAXISSTATE_ONLINESTRUCT

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

AXIS ONLINE STRUCTURE (NC/CNC)

```
public struct NCAXISSTATE_ONLINESTRUCT
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

ActualAcceleration

Actual acceleration. e.g. mm/s²

```
public double ActualAcceleration
```

Field Value

[double](#)

ActualModuloPosition

Axis actual modulo position

```
public double ActualModuloPosition
```

Field Value

[double](#)

ActualPosition

Axis Actual position

```
public double ActualPosition
```

Field Value

[double](#)

ActualTorque

Actual torque or actual force (new from TwinCAT 3.1 B4013). e.g. 100% = 1000

```
public double ActualTorque
```

Field Value

[double](#)

ActualVelocity

Optional: Actual velocity. e.g. mm/s

```
public double ActualVelocity
```

Field Value

[double](#)

AxisControlDWord

Axis control double word

```
public NCTOPLC_AXIS_REF_OPMODE AxisControlDWord
```

Field Value

[NCTOPLC_AXIS_REF_OPMODE](#)

AxisStatusDWord

Axis state double word

```
public StateDWordFlags AxisStatusDWord
```

Field Value

[StateDWordFlags](#)

ControlLoopIndex

Axis control loop index

```
public int ControlLoopIndex
```

Field Value

[int](#)

ControllerOutputPercent

Controller output in percent

```
public double ControllerOutputPercent
```

Field Value

[double](#)

ErrorState

Axis error state.

```
public int ErrorState
```

Field Value

[int](#)

FillUp

Reserved for future use.

```
public byte[] FillUp
```

Field Value

[byte](#)[]

FollowingErrorPeakMaximum

Peak hold value for max. pos. position lag (pos.) e.g. mm

```
public double FollowingErrorPeakMaximum
```

Field Value

[double](#)

FollowingErrorPeakMinimum

PeakHold value for max. neg. position lag (pos.) e.g. mm

```
public double FollowingErrorPeakMinimum
```

Field Value

[double](#)

FollowingErrorPosition

Lag error position. e.g. mm

```
public double FollowingErrorPosition
```

Field Value

[double](#)

SetAcceleration

Set acceleration. e.g. mm/s²

```
public double SetAcceleration
```

Field Value

[double](#)

SetJerk

Set jerk (new from TwinCAT 3.1 B4013). e.g. mm/s³

```
public double SetJerk
```

Field Value

[double](#)

SetModuloPosition

Axis modulo set position.

```
public double SetModuloPosition
```

Field Value

[double](#) ↗

SetPosition

Axis set position.

```
public double SetPosition
```

Field Value

[double](#) ↗

SetTorque

Set torque or set force. Symbol "SetTorque". e.g. 100% = 1000

```
public double SetTorque
```

Field Value

[double](#) ↗

SetTorqueChange

Set torque change or set force change (time derivative of the set torque or set force) (from TwinCAT 3.1 B4024.2). e.g. %/s

```
public double SetTorqueChange
```

Field Value

[double](#)

SetVelocity

Set velocity. e.g. mm/s

```
public double SetVelocity
```

Field Value

[double](#)

SlaveCouplingState

Slave coupling state (state)

```
public CoupleState SlaveCouplingState
```

Field Value

[CoupleState](#)

TorqueOffset

Additive set torque or additive set force ("TorqueOffset") (from TwinCAT 3.1 B4024.2). e.g. 100% = 1000

```
public double TorqueOffset
```

Field Value

[double](#)

TotalOutputPercent

Total output in percent

```
public double TotalOutputPercent
```

Field Value

[double](#) ↗

VelocityOverride

Velocity override (1000000 == 100%)

```
public int VelocityOverride
```

Field Value

[int](#) ↗

Enum NCTOPLC_AXIS_REF_OPMODE

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The AxisControlIDWord is a 32 bit data word in the axis interface PLC->NC.

```
[Flags]
public enum NCTOPLC_AXIS_REF_OPMODE : uint
```

Fields

AllowExtSetAxisCommands = 4194304

Allow motion commands to an axis that is fed by an external setpoint generator.

AllowSlaveCommands = 2097152

Allow motion commands to slave axes.

BacklashCompensation = 32

Backlash compensation.

DelayedErrorReaction = 64

Delayed error reaction of the NC.

LoopMode = 4

Loop movement.

ModuloPositioning = 128

Modulo axis (modulo display).

MotionMonitoring = 8

Physical movement monitoring.

NcApplicationRequested = 8388608

Request bit for the application software (PLC code), e.g. for an "ApplicationHomingRequest".

None = 0

No active bits set.

PEHTimeMonitoring = 16

PEH time monitoring.

PositionAreaMonitoring = 1

Position range monitoring.

PositionCorrection = 1048576

Position correction ("Measuring system error compensation")

PositionLagMonitoring = 65536

Lag monitoring - position.

SimulationAxis = 256

Simulation axis.

SoftLimitMaxMonitoring = 524288

End position monitoring max.

SoftLimitMinMonitoring = 262144

End position monitoring min.

StopMonitoring = 4096

Standstill monitoring.

TargetPositionMonitoring = 2

Target position window monitoring.

Unused1 = 512

Reserved for future use.

Unused2 = 1024

Reserved for future use.

Unused3 = 2048

Reserved for future use.

Unused4 = 8192

Reserved for future use.

Unused5 = 16384

Reserved for future use.

Unused6 = 32768

Reserved for future use.

VeloLagMonitoring = 131072

Lag monitoring - velocity.

Enum NcDriveType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

TwinCAT supports different drives, these are defined in the enum DriveType.

```
public enum NcDriveType
```

Fields

AX2000_B200 = 11

AX2000-B200 Lightbus, incremental with 32 bits (AX2000)

AX2000_B900 = 17

AX2000-B900 Ethernet (max. 32 bits, signed value)

CANopen_DS402_MDP742 = 16

MDP 742 (DS402): CANopen and EtherCAT (AX2000-B510, AX2000-B1x0, EL7201, AX8000)

CANopen_Lenze = 15

CANopen Lenze (max. 32 bits, signed value)

Discrete_TwoSpeed = 7

High/low speed drive (digital)

KL2531_Stepper = 20

Stepper motor terminal KL2531/KL2541

KL2532_DC = 21

2-channel DC motor stage KL2532/KL2542, 2-channel PWM DC motor stage KL2535/KL2545

KL4xxx = 5

MDP 252/253: KL4xxx, PWM KL2502_30K (Frq-Cnt pulse mode), KL4132 (16-bit), Pulse-Train KL2521, IP2512-

KL4xxx_NonLinear = 6

MDP 252/253: analog type for non-linear characteristics.

KL5051 = 10

MDP 510: BISSI Drive KL5051 with 32 bits (see KL4xxx)

M2400_DAC1 = 1

Analog Servo Drive: M2400 DAC 1.

M2400_DAC2 = 2

Analog Servo Drive: M2400 DAC 2.

M2400_DAC3 = 3

Analog Servo Drive: M2400 DAC 3.

M2400_DAC4 = 4

Analog Servo Drive: M2400 DAC 4.

MDP_703 = 24

MDP 703: Modular Device Profile MDP 703 for stepper (e.g. EL7031/EL7041)

MDP_733 = 23

MDP 733: Modular Device Profile MDP 733 for DC (e.g. EL7332/EL7342)

NOT_DEFINED = 0

Not defined.

NcBackplane = 14

Variable bit mask (max. 32 bits, signed value)

ProfilDrive = 12

Incremental with 32 Bit

Sercos = 9

SERCOS-Drive (digital)

Stepper = 8

Stepper motor drive (digital)

TCOM = 22

TCOM Drive -> Interface to Soft Drive

Universal = 13

Variable bit mask (max. 32 bits, signed value)

Enum ProbeMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of probe modes that exist in TwinCAT NC.

```
public enum ProbeMode
```

Fields

Continous = 2

Continous probe mode.

Single = 1

Single probe mode.

Class Ring0Manager

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Manages the low level Ring 0 operations for the NC system.

```
public class Ring0Manager
```

Inheritance

[object](#) ← Ring0Manager

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

Ring0Manager(AmsNetId)

Initializes a new instance of the [Ring0Manager](#) class. Connects to the specified target using the ADS client.

```
public Ring0Manager(AmsNetId target)
```

Parameters

target AmsNetId

The target AmsNetId to connect to.

Properties

Parameters

Gets the parameters for the Ring 0 operations.

```
public Ring0Parameters Parameters { get; }
```

Property Value

[Ring0Parameters](#)

State

Gets the state information for the Ring 0 operations.

```
public Ring0State State { get; }
```

Property Value

[Ring0State](#)

Class Ring0Parameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the parameters for Ring 0, providing access to various settings and configurations related to the SAF and SVB tasks, global time compensation, and cyclic data consistency.

```
public class Ring0Parameters
```

Inheritance

[object](#) ← Ring0Parameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

CycleTimeSaf

Cycle time SAF task Unit: 100 ns

```
public uint CycleTimeSaf { get; }
```

Property Value

[uint](#)

CycleTimeSvb

Cycle time SVB task. Unit: 100 ns

```
public uint CycleTimeSvb { get; }
```

Property Value

[uint](#)

CyclicDataConsistenceCheck

Cyclic data consistence check and correction of the NC setpoint values Unit: 0/1

```
public ushort CyclicDataConsistenceCheck { get; set; }
```

Property Value

[ushort](#)

GlobalTimeCompensationShift

Global Time Compensation Shift (for SAF Task). Unit: ns

```
public int GlobalTimeCompensationShift { get; }
```

Property Value

[int](#)

Class Ring0State

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the state of the Ring 0 system, providing access to various counts and IDs for channels, groups, axes, encoders, controllers, drives, and tables.

```
public class Ring0State
```

Inheritance

[object](#) ← Ring0State

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AxisCount

Quantity of Axis 0, 1...255

```
public uint AxisCount { get; }
```

Property Value

[uint](#)

AxisIds

Supplies the axis IDs for all axes in the system

```
public uint[] AxisIds { get; }
```

Property Value

[uint](#)[]

ChannelCount

Quantity of Channel 0, 1...255

```
public uint ChannelCount { get; }
```

Property Value

[uint](#)[]

ChannelIds

Supplies the Channel IDs for all Channels in the system

```
public uint[] ChannelIds { get; }
```

Property Value

[uint](#)[]

ControllerCount

Quantity of controller 0, 1...255

```
public uint ControllerCount { get; }
```

Property Value

[uint](#)[]

ControllerIds

Supplies the controller IDs for all controllers in the system

```
public uint[] ControllerIds { get; }
```

Property Value

[uint](#)[]

DriveCount

Quantity of Drives 0, 1...255

```
public uint DriveCount { get; }
```

Property Value

[uint](#)

DriveIds

Supplies the drive IDs for all drives in the system

```
public uint[] DriveIds { get; }
```

Property Value

[uint](#)[]

EncoderCount

Quantity of Encoder 0, 1...255

```
public uint EncoderCount { get; }
```

Property Value

[uint](#)

EncoderIds

Supplies the encoder IDs for all encoders in the system

```
public uint[] EncoderIds { get; }
```

Property Value

[uint](#)[]

GroupCount

Quantity of group 0, 1...255

```
public uint GroupCount { get; }
```

Property Value

[uint](#)

GroupIds

Supplies the group IDs for all groups in the system

```
public uint[] GroupIds { get; }
```

Property Value

[uint](#)[]

TableCount

Quantity of table (n x m) 0, 1...255

```
public uint TableCount { get; }
```

Property Value

[uint](#)

TableIds

Supplies the table IDs for all tables in the system

```
public uint[] TableIds { get; }
```

Property Value

[uint](#)[]

Enum SignalEdge

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the two types of signal edges that exist. Rising edge and falling edge.

```
public enum SignalEdge
```

Fields

FallingEdge = 1

Falling edge. Signal goes from high to low.

RisingEdge = 0

Rising edge. Signal goes from low to high.

Enum StateDWordFlags

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The StateDWord is a 32 bit data word in the axis interface NC->PLC.

```
[Flags]
public enum StateDWordFlags : uint
```

Fields

CamDataQueued = 4194304

CAM data (only MF) are queued for "Online Change" and waiting for activation

CamScalingPending = 8388608

CAM scaling are queued for "Online Change" and waiting for activation

CamTableQueued = 2097152

CAM table is queued for "Online Change" and waiting for activation

CmdBuffered = 16777216

Following command is queued in then command buffer (s. Buffer Mode)

Compensating = 8192

Section compensation passive[0]/active[1] (s. "MC_MoveSuperImposed")

ConstantVelocity = 4096

Axis has reached its constant velocity or rotary speed

ContinuousMotion = 524288

Axis has target position (\pm) endless

ControlLoopClosed = 1048576

Axis is ready for operation and axis control loop is closed (e.g. position control)

DriveDeviceError = 268435456

Hardware drive device error (no warning), interpretation only possible when drive is data exchanging,
e.g. EtherCAT "OP"-state

Error = 2147483648

Axis is in a fault state

ErrorPropagationDelayed = 64

Axis signals an error pre warning (from TC 2.11)

ExtSetPointGenEnabled = 16384

External setpoint generator enabled

ExternalLatchValid = 65536

External latch value or sensing switch has become valid

HasBeenStopped = 128

Axis has been stopped or is presently executing a stop

HasJob = 256

Axis has instructions, is carrying instructions out

Homed = 2

Axis has been referenced/ homed ("Axis calibrated")

HomingBusy = 2048

Axis referenced ("Axis being calibrated")

InPositionArea = 8

Axis is in position window (physical feedback)

InTargetPosition = 16

Axis is at target position (PEH) (physical feedback)

IoDataInvalid = 1073741824

IO data invalid (e.g. 'WcState' or 'CdlState')

MotionCommandsLocked = 536870912

Axis is locked for motion commands (TcMc2)

NegativeDirection = 1024

Axis moving to logically smaller values

NewTargetPosition = 131072

Axis has a new target position or a new velocity

None = 0

None.

NotImplementedYet = 32768

Operating mode not yet executed (Busy). Not implemented yet!

NotImplementedYet2 = 262144

Axis is not at target position or cannot reach the target position (e.g. stop).Not implemented yet!

NotMoving = 4

Axis is logically stationary ("Axis not moving")

PTPmode = 33554432

Axis in PTP mode (no slave, no NCI axis, no FIFO axis) (from TC 2.10 Build 1326)

PositiveDirection = 512

Axis moving to logically larger values

Protected = 32

Axis is in a protected operating mode (e.g. as a slave axis)

ReadyForOperation = 1

Axis is ready for operation

SoftLimitMaxExceeded = 134217728

Position software limit switch maximum is exceeded (from TC 2.10 Build 1327)

SoftLimitMinExceeded = 67108864

Position software limit switch minimum is exceeded (from TC 2.10 Build 1327)

Class Table

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The **Table** class encapsulates various functionalities, parameters, and states related to a table in the context of TwinCAT ADS. It provides a structured way to interact with tables, which are likely used for motion control or other automation tasks.

```
public class Table
```

Inheritance

[object](#) ← Table

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

Functions

Gets the functions related to table operations such as generating and deleting tables.

```
public TableFunctions Functions { get; }
```

Property Value

[TableFunctions](#)

Parameters

Gets the parameters of the table including ID, name, types, dimensions, and other properties.

```
public TableParameters Parameters { get; }
```

Property Value

[TableParameters](#)

State

Gets the state of the table including the user counter.

```
public TableState State { get; }
```

Property Value

[TableState](#)

Enum TableActivationMode

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of table activation modes that exist in TwinCAT NC.

```
public enum TableActivationMode
```

Fields

AsSoonAsPossible = 5

Modified cam plate data take effect as soon as system dynamics allow.

DeleteQueuedData = 7

Queued cam plate data are deleted. Data are queued if the change was requested at a certain master position or at the end of the cycle, for example.

Instantaneous = 0

The change takes effect immediately. (default)

MasterAxisPosition = 2

The change takes effect at a certain absolute position of the master axis. The command must be issued ahead of this position.

MasterCamPosition = 1

The change takes effect at a certain cam plate position (master position within the cam plate). The command must be issued ahead of this position

NextCycle = 3

For a cyclic cam plate, the change takes effect at the transition to the next period.

NextCycleOnce = 4

Modify the data at the beginning of the next cam table cycle, activation is valid for one cycle only

Off = 6

Changes in cam plate data are ignored.

Class TableFunctions

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The TableFunctions class provides methods to generate and delete various types of tables with specified dimensions and interpolation types. It interacts with a TwinCAT AdsClient to perform these operations. The class supports generating general tables, valve diagram tables, and motion function tables, each with specific table types and dimensions.

```
public class TableFunctions
```

Inheritance

[object](#) ← TableFunctions

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

DeleteTable()

Deletes table with dimension (n*m) Table types: 1,2,3,4

```
public void DeleteTable()
```

GenerateMotionFunctionTable(TableInterpolationType, uint, uint)

Generates "Motion Function" table with dimension (n*m): Table types: 3, 4 Dimension: at least 2x1

```
public void GenerateMotionFunctionTable(TableInterpolationType tableType, uint lineCount,  
uint columnCount)
```

Parameters

tableType [TableInterpolationType](#)

lineCount [uint](#)

columnCount [uint](#)

GenerateTable(TableInterpolationType, uint, uint)

Generates table with dimension (n*m): Table types: 1,2,3,4 Dimension: at least 2x1

```
public void GenerateTable(TableInterpolationType tableType, uint lineCount,  
uint columnCount)
```

Parameters

tableType [TableInterpolationType](#)

lineCount [uint](#)

columnCount [uint](#)

GenerateValveDiagramTable(TableInterpolationType, uint, uint)

Generates valve diagram table with dimension (n*m): Table types: 1,3 Dimension: at least 2x1

```
public void GenerateValveDiagramTable(TableInterpolationType tableType, uint lineCount,  
uint columnCount)
```

Parameters

tableType [TableInterpolationType](#)

lineCount [uint](#)

columnCount [uint](#)

Enum TableInterpolationType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Interpolation mode for position tables (cam plates). Position tables consist of a list of master and slave positions between which interpolation can take place in different ways. The interpolation type is not used for extended cam plates(motion functions).

```
public enum TableInterpolationType
```

Fields

CubicSpline = 2

Cubic spline interpolation of all reference points ("global spline") (NC_INTERPOLATIONTYPE_SPLINE)

FourPoint = 1

4-point interpolation (NC_INTERPOLATIONTYPE_4POINT) (for equidistant table types only)

Linear = 0

Linear interpolation (NC_INTERPOLATIONTYPE_LINEAR) (Standard)

SlidingCubicSpline = 3

Moving cubic spline interpolation with n sampling points ('local spline')

Enum TableMainType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of main tables that exist in TwinCAT NC.

```
public enum TableMainType
```

Fields

CamPlate = 1

(n*m) Cam plate tables (Camming)

CharacteristicCurve = 10

(n*m) Characteristic curves tables (Characteristics) (e.g. hydraulic valve characteristic curves). Only non-cyclic table sub-types (1, 3) are supported!

MotionFunction = 16

(n*m) "Motion Function" tables (MF). Only non-equidistant table sub-types (3, 4) are supported!

Class TableParameters

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

The TableParameters class provides methods to interact with table parameters in a TwinCAT ADS system. It allows reading and writing various table properties such as ID, Name, SubType, MainType, LineCount, ColumnCount, TotalCount, StepWidth, MasterPeriod, and SlaveDifferencePerMasterPeriod. It also provides methods to get and set the activation mode for online changes, read and write single values in the table, and convert slave positions to master positions.

```
public class TableParameters
```

Inheritance

[object](#) ← TableParameters

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

ColumnCount

Number of columns (m)

```
public uint ColumnCount { get; }
```

Property Value

[uint](#)

ID

Table ID

```
public uint ID { get; }
```

Property Value

[uint](#)

LineCount

Number of lines (n)

```
public uint LineCount { get; }
```

Property Value

[uint](#)

MainType

Table main type

```
public TableMainType MainType { get; }
```

Property Value

[TableMainType](#)

MasterPeriod

Master period (cyclic table). Base unit (e.g. degree)

```
public double MasterPeriod { get; }
```

Property Value

[double](#)

Name

Table name

```
public string Name { get; }
```

Property Value

[string](#)

SlaveDifferencePerMasterPeriod

Slave difference per master period (cyclic table). Base unit (e.g. degree)

```
public double SlaveDifferencePerMasterPeriod { get; }
```

Property Value

[double](#)

SlavePositionToMaster

Read slave position to the given master position (relates only to the "row values" of the table)

```
public double SlavePositionToMaster { get; set; }
```

Property Value

[double](#)

StepWidth

Step width (position delta) (equidistant table) Equidistant Tables Base unit (e.g. mm)

```
public double StepWidth { get; }
```

Property Value

[double](#) ↗

SubTupe

Table sub type

```
public TableSubType SubTupe { get; }
```

Property Value

[TableSubType](#)

TotalCount

Total number of elements (n*m)

```
public uint TotalCount { get; }
```

Property Value

[uint](#) ↗

Methods

GetActivationMode(out TableActivationMode, out double, out CamScalingMode, out CamScalingMode)

Get activation mode for online change from table data (only MF).

```
public void GetActivationMode(out TableActivationMode activationMode,  
    out double activationPosition, out CamScalingMode masterScalingType, out  
    CamScalingMode slaveScalingType)
```

Parameters

`activationMode` [TableActivationMode](#)

Activation mode: 0: 'instantaneous' (default) 1: 'master cam pos.' 2: 'master' axis pos.' 3: 'next cycle' 4: 'next cycle once' 5: 'as soon as possible' 6: 'off' 7: 'delete queued data'

`activationPosition` [double](#)

Activation position (e.g. mm)

`masterScalingType` [CamScalingMode](#)

Master scaling type 0: user defined (default) 1: scaling with auto offset 2: off

`slaveScalingType` [CamScalingMode](#)

Slave scaling type 0: user defined (default) 1: scaling with auto offset 2: off

`GetSingleValue(uint, uint)`

```
public double GetSingleValue(uint line, uint column)
```

Parameters

`line` [uint](#)

`column` [uint](#)

Returns

[double](#)

`SetActivationMode(TableActivationMode, double, CamScalingMode, CamScalingMode)`

Set Activation mode for online change from table data (only MF).

```
public void SetActivationMode(TableActivationMode activationMode, double activationPosition, CamScalingMode masterScalingType, CamScalingMode slaveScalingType)
```

Parameters

activationMode [TableActivationMode](#)

Activation mode: 0: 'instantaneous' (default) 1: 'master cam pos.' 2: 'master' axis pos.' 3: 'next cycle' 4: 'next cycle once' 5: 'as soon as possible' 6: 'off' 7: 'delete queued data'

activationPosition [double](#)

Activation position (e.g. mm)

masterScalingType [CamScalingMode](#)

Master scaling type 0: user defined (default) 1: scaling with auto offset 2: off

slaveScalingType [CamScalingMode](#)

Slave scaling type 0: user defined (default) 1: scaling with auto offset 2: off

SetSingleValue(uint, uint, double)

Write single value [n,m].

```
public void SetSingleValue(uint line, uint column, double value)
```

Parameters

line [uint](#)

n-th line

column [uint](#)

m-th column

value [double](#)

Single value. Base unit (e.g. mm)

Class TableState

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Represents the state of a table in a TwinCAT NC (Numerical Control) system. This class provides access to the 'User Counter'. It uses an AdsClient to communicate with the TwinCAT system and read the necessary data.

```
public class TableState
```

Inheritance

[object](#) ← TableState

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

UserCounter

'User Counter' (number of table user)

```
public int UserCounter { get; }
```

Property Value

[int](#)

Enum TableSubType

Namespace: [TwinSharp.NC](#)

Assembly: TwinSharp.dll

Enumeration of the different types of table subtypes that exist in TwinCAT NC.

```
public enum TableSubType
```

Fields

EquidistantMasterCyclicContinuation = 2

(n*m) Table with equidistant master positions and cyclic continuation of the master profile (equidistant cyclic)

EquidistantMasterNonCyclicContinuation = 1

(n*m) Table with equidistant master positions and no cyclic continuation of the master profile (equidistant linear)

NonEquidistantMasterCyclicContinuation = 4

(n*m) Table with non-equidistant, but strictly monotonously increasing master positions and a cyclic continuation of the master profile (monotonously cyclic)

NonEquidistantMasterNonCyclicContinuation = 3

(n*m) Table with non-equidistant, but strictly monotonously increasing master positions and a non-cyclic continuation of the master profile (monotonously linear)

Namespace TwinSharp.PLC

Classes

[PLC](#)

The PLC class provides methods to control a TwinCAT PLC runtime system using an AdsClient. It allows starting, stopping, and resetting the PLC, as well as accessing various system information and status variables through the PlcAppSystemInfo instance.

[PlcAppSystemInfo](#)

The PlcAppSystemInfo class provides access to various system information and status variables of a TwinCAT PLC application. It uses an AdsClient to read and write these variables, which include object ID, flags, ADS port, boot data status, application timestamp, and more.

Class PLC

Namespace: [TwinSharp.PLC](#)

Assembly: TwinSharp.dll

The PLC class provides methods to control a TwinCAT PLC runtime system using an AdsClient. It allows starting, stopping, and resetting the PLC, as well as accessing various system information and status variables through the PlcAppSystemInfo instance.

```
public class PLC
```

Inheritance

[object](#) ← PLC

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

PLC(AdsClient)

Initializes a new instance of the PLC class with the specified AdsClient. The AdsClient should typically be connected to the target system at port AmsPort.PlcRuntime_851 (851).

```
public PLC(AdsClient client)
```

Parameters

client AdsClient

Properties

AppInfo

Gets the PlcAppSystemInfo instance which provides access to various system information and status variables of a TwinCAT PLC application.

```
public PlcAppSystemInfo AppInfo { get; }
```

Property Value

[PlcAppSystemInfo](#)

Methods

Reset()

Can be used to reset a PLC runtime system. When the PLC is reset, the PLC variables are filled with their initial values, and the execution of the PLC program is stopped. Equivalent to the function block PLC_Reset.

```
public void Reset()
```

Start()

Can be used to start a PLC runtime system on a TwinCAT system. The function block can, for instance, be used to start the PLC on a remote PC. Equivalent to the function block PLC_Start.

```
public void Start()
```

Stop()

Can be used to stop a PLC runtime system on a TwinCAT system. The function block can, for instance, be used to stop the PLC on a remote or a local PC. Equivalent to the function block PLC_Stop.

```
public void Stop()
```

Class PlcAppSystemInfo

Namespace: [TwinSharp.PLC](#)

Assembly: TwinSharp.dll

The PlcAppSystemInfo class provides access to various system information and status variables of a TwinCAT PLC application. It uses an AdsClient to read and write these variables, which include object ID, flags, ADS port, boot data status, application timestamp, and more.

```
public class PlcAppSystemInfo
```

Inheritance

[object](#) ← PlcAppSystemInfo

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Properties

AdsPort

ADS port of the PLC application.

```
public uint AdsPort { get; }
```

Property Value

[uint](#)

AppName

Name generated by TwinCAT, which contains the port.

```
public string AppName { get; }
```

Property Value

[string](#)

AppTimestamp

Time at which the PLC application was compiled

```
public DateTime AppTimestamp { get; }
```

Property Value

[DateTime](#)

BSODOccured

This variable has the value TRUE if Windows is in a BSOD.

```
public bool BSODOccured { get; }
```

Property Value

[bool](#)

BootDataLoaded

PERSISTENT variables: LOADED (without error).

```
public bool BootDataLoaded { get; }
```

Property Value

[bool](#)

Flags

TwinCAT internal use.

```
public uint Flags { get; }
```

Property Value

[uint](#)

KeepOutputsOnBP

The flag can be set and prevents that the outputs are zeroed when a breakpoint is reached. In this case the task continues to run. Only the execution of the PLC code is interrupted.

```
public bool KeepOutputsOnBP { get; set; }
```

Property Value

[bool](#)

LicensesPending

This variable has the value TRUE if not all licenses that are provided by license dongles have been validated yet.

```
public bool LicensesPending { get; }
```

Property Value

[bool](#)

ObjId

Object ID of the PLC project instance.

```
public ulong ObjId { get; }
```

Property Value

[ulong](#) ↗

OldBootData

PERSISTENT variables: INVALID (the back-up copy was loaded, since no valid file was present).

```
public bool OldBootData { get; }
```

Property Value

[bool](#) ↗

OnlineChangeCnt

Number of online changes since the last complete download

```
public ulong OnlineChangeCnt { get; }
```

Property Value

[ulong](#) ↗

ProjectName

Name of the project.

```
public string ProjectName { get; }
```

Property Value

[string](#) ↗

ShutdownInProgress

This variable has the value TRUE if a shutdown of the TwinCAT system is in progress. Some parts of the TwinCAT system may already have been shut down.

```
public bool ShutdownInProgress { get; }
```

Property Value

[bool](#)

TaskCnt

Number of tasks in the runtime system

```
public ulong TaskCnt { get; }
```

Property Value

[ulong](#)