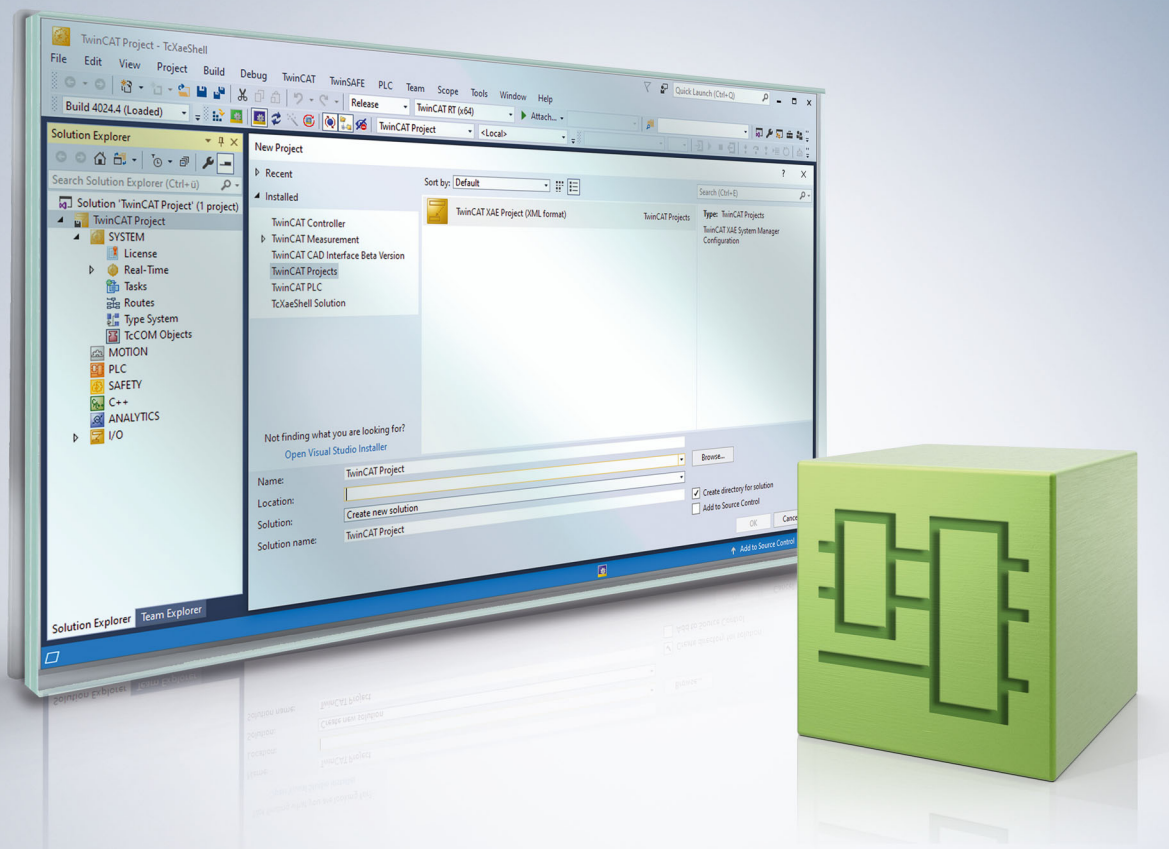
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC-Bibliothek: Tc3_EtherCATDiag



2024-07-10 | Version: 1.0.0

Table of Contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patents

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

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

Copyright

1.2 For your safety

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Signal words

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

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

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

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

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

1.3 Notes on information security

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

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

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

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

2 Overview

The PLC library Tc3_EtherCATDiag is used for extended diagnosis of EtherCAT slaves.

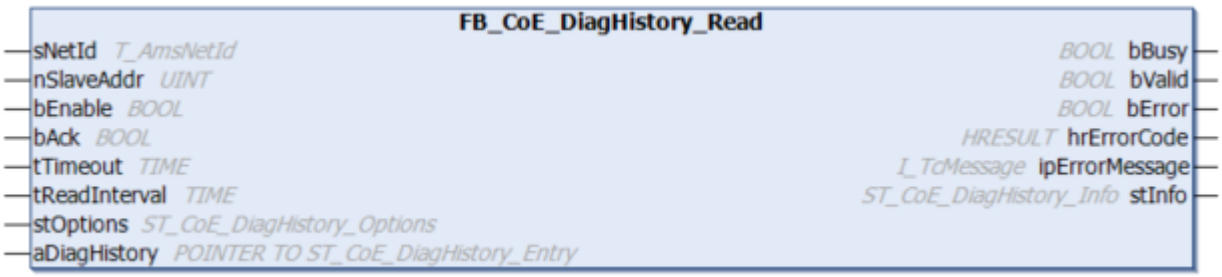
The function blocks FB_ReadCoEDiagHistory and FB_ReadCoEDiagHistory_Axis can be used to read EtherCAT slaves that support the CoE Diag History Object.

System requirements

Target System	Win10, Win11 IPC or CX, (x86, x64, ARM)
Min. TwinCAT version	3.1.4024.57
Min. TwinCAT level	TC1200 TC3 PLC

3 Function blocks

3.1 FB_CoE_DiagHistory_Read



The function block FB_ReadCoEDiagHistory can read EtherCAT slaves that support the CoE Diag History Object. The EtherCAT slave must comply with the ETG standard.

Syntax

```
VAR_INPUT
    sNetId          :  T_AmsNetId;
    nSlaveAddr      :  UINT;
    bEnable         :  BOOL;
    bAck            :  BOOL;
    tTimeout        :  TIME := DEFAULT_ADS_TIMEOUT;
    tReadInterval   :  TIME := T#10S;
    stOptions       :  ST_CoE_DiagHistory_Options;
END_VAR
VAR_IN_OUT
    aDiagHistory    :  ARRAY[*] OF ST_CoE_DiagHistory_Entry;
END_VAR
VAR_OUTPUT
    bBusy           :  BOOL;
    bValid          :  BOOL;
    bError          :  BOOL;
    hrErrorCode     :  HRESULT;
    ipErrorMessage  :  I_TcMessage := fbResult;
    stInfo          :  ST_CoE_DiagHistory_Info;
END_VAR
```

Inputs

Name	Type	Description
sNetId	T_AmsNetId	NetID of the slave
nSlaveAddr	UINT	Port address of the slave
bEnable	BOOL	Pulse: the Diag History is read out once. Continuous: the Diag History is read out repeatedly (see tReadInterval). With each new edge, everything is reset. The connection is then re-established and the slave is read.
bAck	BOOL	All messages are acknowledged when bAck is triggered.
tTimeout	TIME	Timeout for the respective ADS reads/writes in the function block. The default ADS timeout here is five seconds.
tReadInterval	TIME	Time interval between repeated reading of the slave with a permanent "bEnable" signal. The default is a time interval of 10 seconds.
stOptions	ST_CoE_DiagHistory_Options [▶ 14]	Controls for handling CoE Diag History

Inputs/outputs

Name	Type	Description
aDiagHistory	ARRAY[*] OF ST_CoE_DiagHistory_Entry [▶ 13]	An array of any length from ST_CoE_DiagHistory_Entry is assigned to this In-Out variable. The FB automatically adapts to the length of the array. If it is too short, older messages in the slave's Diag History Object are ignored. If it is too long, superfluous entries are deleted. A length greater than 250 makes no sense here, as the ETG standard defines a maximum of 250 messages for the Diag History Object.

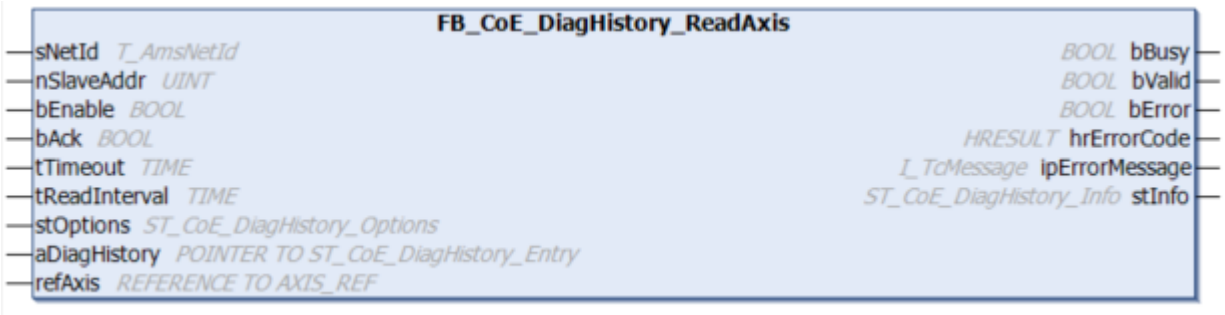
Outputs

Name	Type	Description
bBusy	BOOL	Simple busy signal, indicates that the function block is reading a slave.
bValid	BOOL	The Diag History Object has been successfully read and the messages issued are valid.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an error code if the bError output is set.
ipErrorMessage	I_TcMessage	Returns detailed information if a bError output is set. The interface pointer used here is always valid (not equal to zero) and is of the type I_TcMessage. In particular, the corresponding error is immediately visible as plain text in the PLC OnlineView.
stInfo	ST_CoE_DiagHistory_Info [▶ 13]	Provides information from the read EtherCAT Slave Diag History Object. This includes, among other things, the Buffer Mode used and information on the available messages. This output is only updated after a successful read operation.

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1 >= Build 4024.57	x86, x64, ARM	Tc3_EtherCATDiag

3.2 FB_CoE_DiagHistory_ReadAxis



The function block FB_ReadCoEDiagHistory_Axis is a variation of the function block FB_ReadCoEDiagHistory [▶ 9], which can be assigned an AXIS_REF structure instead of an address. Only messages that belong either to the device or to the specific axis are output (e.g. for an AX8206 dual-axis module).

Syntax

```
VAR_INPUT
    sNetId           :  T_AmsNetId;
    nSlaveAddr       :  UINT;
    bEnable          :  BOOL;
    bAck             :  BOOL;
    tTimeout         :  TIME := DEFAULT_ADS_TIMEOUT;
    tReadInterval    :  TIME := T#10S;
    stOptions        :  ST_CoE_DiagHistory_Options;
END_VAR
VAR_IN_OUT
    aDiagHistory     :  ARRAY[*] OF ST_CoE_DiagHistoryEntry;
END_VAR
VAR_INPUT
    refAxis          :  REFERNCE TO AXIS_REF;
END_VAR
VAR_OUTPUT
    bBusy            :  BOOL;
    bValid           :  BOOL;
    bError           :  BOOL;
    hrErrorCode      :  HRESULT;
    ipErrorMessage   :  I_TcMessage := fbResult;
    stInfo           :  ST_CoE_DiagHistory_Info;
END_VAR
```

Inputs

Name	Type	Description
sNetId	T_AmsNetId	This input is not used and should not be assigned.
nSlaveAddr	UINT	This input is not used and should not be assigned.
bEnable	BOOL	Pulse: the Diag History is read out once. Continuous: the Diag History is read out repeatedly (see tReadInterval). With each new edge, everything is reset first. The connection is then re-established and the slave is read.
bAck	BOOL	All messages are acknowledged when bAck is triggered.
tTimeout	TIME	Timeout for the respective ADS reads/writes in the function block. The default is the ADS timeout of five seconds.
tReadInterval	TIME	Time interval between repeated reading of the slave with a permanent "bEnable" signal. The default is a time interval of 10 seconds.
stOptions	ST_CoE_DiagHistory_Options [▶ 14]	Controls for handling CoE Diag History
refAxis	REFERNCE TO AXIS_REF	AXIS_REF of an axis, from which the address of the corresponding slave can be determined.

Inputs/outputs

Name	Type	Description
aDiagHistory	ARRAY[*] OF ST_CoE_DiagHistory_Entry [▶ 13]	An array of any length from ST_CoE_DiagHistory_Entry is assigned to this In-Out variable. The FB automatically adapts to the length of the array. If it is too short, older messages in the slave's Diag History Object are ignored. If it is too long, superfluous entries are deleted. A length greater than 250 makes no sense here, as the ETG standard defines a maximum of 250 messages for the Diag History Object.

Outputs

Name	Type	Description
bBusy	BOOL	Simple busy signal, indicates that the function block is reading a slave.
bValid	BOOL	The Diag History Object has been successfully read and the messages issued are valid.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an error code if the bError output is set.
ipErrorMessage	I_TcMessage	Returns detailed information if a bError output is set. The interface pointer used here is always valid (not equal to zero) and is of the type I_TcMessage. In particular, the corresponding error is immediately visible as plain text in the PLC OnlineView.
stInfo	ST_CoE_DiagHistory_Info [▶ 13]	Provides information from the read EtherCAT Slave Diag History Object. This includes, among other things, the Buffer Mode used and information on the available messages. This output is only updated after a successful read operation.

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1 >= Build 4024.57	x86, x64, ARM	Tc3_EtherCATDiag

4 Data types

4.1 ST_CoE_DiagHistory_Entry

Syntax

```
TYPE ST_CoE_DiagHistory_Entry :
STRUCT
    eMsgType      :  E_CoE_0x10F3_DiagHistory_MsgType;
    bAckStatus    :  BOOL;
    stTimeStamp   :  TIMESTRUCT;
    {attribute 'displaymode':='hex'}
    nDiagCode     :  UDINT;
    eDiagCodeType :  E_CoE_0x10F3_DiagHistory_DiagCodeType;
    {attribute 'displaymode':='hex'}
    nTextId       :  UINT;
    nModuleNo     :  UINT;
    sMsgText      :  STRING(DiagHistory_Parameters.cMsgStringLength);
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
eMsgType	E_CoE_0x10F3_DiagHistory_MsgType [▶ 15]	Type of message/diagnosis This can be converted into a string using the TO_STRING() method.
bAckStatus	BOOL	Status of the confirmed message
stTimeStamp	TIMESTRUCT	Timestamp
nDiagCode	UDINT	Diagnostic code - numerical value
eDiagCodeType	E_CoE_0x10F3_DiagHistory_DiagCodeType [▶ 15]	Typing the diagnostic code This can be converted into a string using the TO_STRING() method.
nTextId	UINT	Text ID
nModuleNo	UINT	Diagnostic code module number
sMsgText	STRING	Message text

4.2 ST_CoE_DiagHistory_Info

The structure provides information on the settings in the EtherCAT slave with regard to EtherCAT diagnosis.

Syntax

```
TYPE ST_CoE_DiagHistory_Info :
STRUCT
    nMaxNoOfMsgs            : UINT;
    eBufferMode             : E_CoE_0x10F3_DiagHistory_BufferMode;
    bNewMsgsAvail           : BOOL;
    bEmergSendingEnabled    : BOOL;
    bInfoMsgsDisabled       : BOOL;
    bWarnMsgsDisabled       : BOOL;
    bErrMsgsDisabled        : BOOL;
    bOldMsgsOverwritten     : BOOL;
    bNewMsgsDiscarded       : BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nMaxNoOfMsgs	UINT	Maximum number of messages in the buffer of the EtherCAT slave
eBufferMode	E_CoE_0x10F3_DiagHistory_BufferMode [▶ 15]	Overwrite or confirmation mode of the EtherCAT slave
bNewMsgsAvail	BOOL	New messages available since last read operation.
bEmergSendingEnabled	BOOL	Emergency messages enabled. New diagnostic messages must be sent as emergency messages.
bInfoMsgsDisabled	BOOL	Information messages disabled.
bWarnMsgsDisabled	BOOL	Warning messages disabled.
bErrMsgsDisabled	BOOL	Error messages disabled.
bOldMsgsOverwritten	BOOL	Old messages in the buffer were overwritten by new messages (Overwrite Mode).
bNewMsgsDiscarded	BOOL	New messages were discarded because the buffer is filled with unconfirmed messages (Acknowledge Mode).

4.3 ST_CoE_DiagHistory_Options

Syntax

```
TYPE ST_CoE_DiagHistory_Options :
STRUCT
    nLangId                                 : DINT := 1033;
    eBufferMode                             : E_CoE_0x10F3_DiagHistory_BufferMode :=
E_CoE_0x10F3_DiagHistory_BufferMode.OverwriteMode;
    bEnableEmergSending                     : BOOL := TRUE;
    bDisableInfoMsgs                        : BOOL := FALSE;
    bDisableWarnMsgs                        : BOOL := FALSE;
    bDisableErrMsgs                         : BOOL := FALSE;
    bDisableAutoRecoverAfterConnectionErr   : BOOL := FALSE;
    bExcludeTextIdFromMsgText               : BOOL := FALSE;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nLangId	DINT	Optional: Language ID, 1033 = English (US), 1031 = German etc.
eBufferMode	E_CoE_0x10F3_DiagHistory_BufferMode [▶ 15]	Overwrite or Acknowledge mode
bEnableEmergSending	BOOL	The sending of emergency messages is disabled. New diagnostic messages are not sent as emergency messages.
bDisableInfoMsgs	BOOL	Info messages are disabled.
bDisableWarnMsgs	BOOL	Warning messages are disabled.
bDisableErrMsgs	BOOL	Error messages are disabled.
bDisableAutoRecoverAfterConnectionErr	BOOL	This disables the automatic re-establishment of the connection to the EtherCAT diagnostic history. By default, the connection is re-established after a connection error if the bEnable input is still TRUE.
bExcludeTextIdFromMsgText	BOOL	Setting this option removes the text ID from the message text.

4.4 E_CoE_0x10F3_DiagHistory_BufferMode

The Buffer Mode determines how the EtherCAT slave behaves with regard to its error memory. The messages are stored in the form of a ring buffer.

```
TYPE E_CoE_0x10F3_DiagHistory_BufferMode :
(
UNKNOWN,
OverwriteMode,
AcknowledgeMode
) USINT;
END_TYPE
```

Value	Description
OverwriteMode	In Overwrite Mode, the oldest messages are always discarded if the ring buffer is full.
AcknowledgeMode	In Acknowledge Mode, if the ring buffer is full, the oldest acknowledged messages are discarded. If only non-acknowledged messages are in the list, these remain and the latest incoming messages are discarded. If messages are acknowledged with the function block input bAck, they can be overwritten by new messages.

4.5 E_CoE_0x10F3_DiagHistory_DiagCodeType

The enumeration specifies the type of DiagCode value output. The types are defined by the ETG standard.

```
TYPE E_CoE_0x10F3_DiagHistory_DiagCodeType :
(
    UNKNOWN,
    NotUsed,
    ManufacturerSpecific,
    EmergencyErrorCode,
    Reserved,
    ModuleRelated,
    DeviceRelated,
    ProfileSpecific
) USINT;
END_TYPE
```

Value	Description
UNKNOWN	Unknown type of message
NotUsed	The DiagCode is currently not used according to the standard.
ManufacturerSpecific	The DiadCode is specific and defined by the device manufacturer.
EmergencyErrorCode	The DiagCode is an emergency code.
Reserved	The DiagCode is reserved for future use. The type has not yet been defined.
ModuleRelated	The DiagCode is module-related.
DeviceRelated	The DiagCode is device-related.
ProfileSpecific	The DiagCode is profile-specific.

4.6 E_CoE_0x10F3_DiagHistory_MsgType

```
TYPE E_CoE_0x10F3_DiagHistory_MsgType :
(
    UNKNOWN,
    Info,
    Warning,
    Error,
    RESERVED_Value_x
) USINT;
END_TYPE
```

Value	Description
UNKNOWN	Unknown type of message
Info	This is an information message.
Warning	This is a warning message.
Error	This is not an error message.
RESERVED_Value_x	Other types of messages are currently not defined. The values are reserved for the future.

5 Global parameters

5.1 DiagHistory_Parameters

These parameters are constant at runtime, but can be changed on a project-specific basis via the library manager and are saved with the project properties. They allow longer message texts or more arguments within the message texts by adjustment. Due to the large, partly internal, arrays, the size of the function block instances and also the program size may increase considerably by adjusting the parameters.

Syntax

```
VAR_GLOBAL CONSTANT
    // FB-internal read buffer - Max.  parameters/arguments buffer size per message in bytes.
    // Reduce to optimize memory resources. Increase if needed.
    cMsgParsBufferSize    : UINT(0..1000) := 128;

    // FB-internal read buffer - Max number of messages in diag history buffer.
    // (ETG.1020 Norm states 250 max. possible messages. Not all Slaves hold this many messages).
    // Reduce to optimize memory resources. Increase if needed.
    cMaxMsgsBuffer        : UINT(0..250)  := 250;

    // FB-internal read buffer - Max length of message string. Reduce to optimize memory resources.
    // Increase if needed. Increase if needed.
    cMsgStringLength      : UINT(0..255) := 200;
END_VAR
```

6 Global constants

6.1 Global_Version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

```
VAR_GLOBAL CONSTANT
   stLibVersion_Tc3_EtherCATDiag : ST_LibVersion;
END_VAR
```

Name	Type	Description
stLibVersion_Tc3_EtherCATDiag	ST_LibVersion	Version information of the Tc3_EtherCATDiag library

To check if the version you have is the one you need, use the function F_CmpLibVersion (defined in the PLC library Tc2_System).

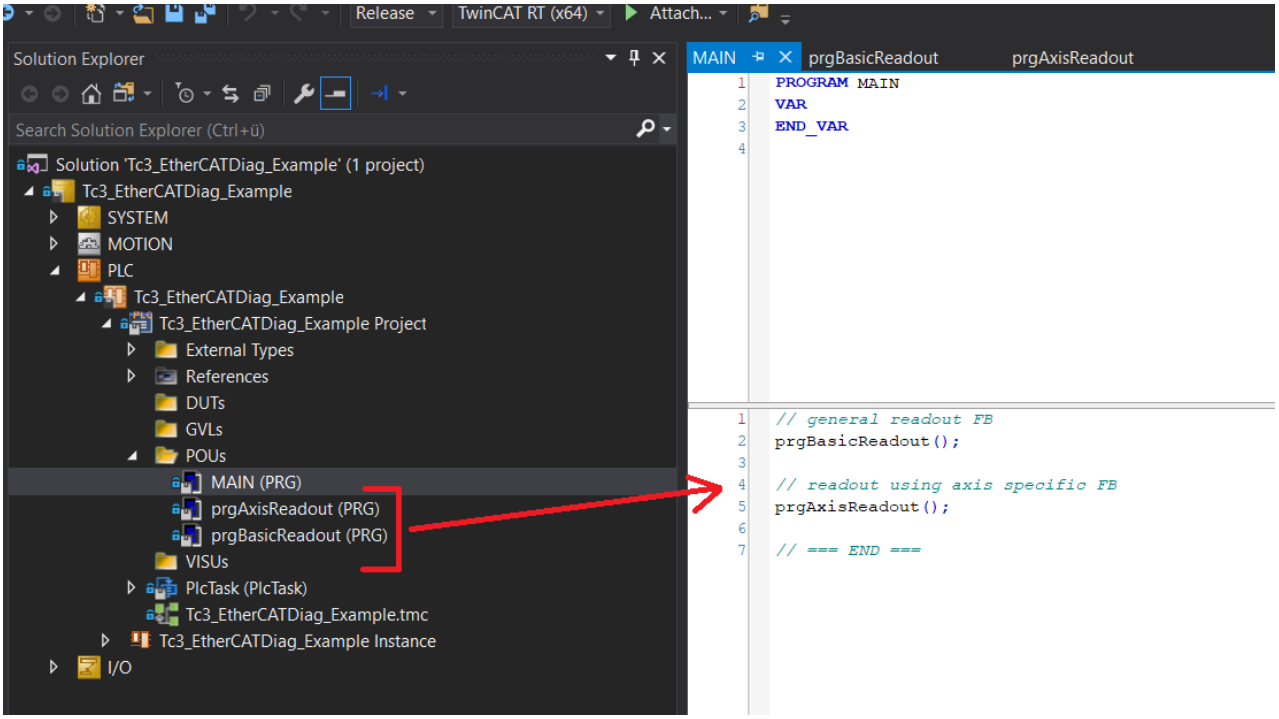
7 Examples

Documents about this

Tc3_EtherCATDiag_Example (Resources/zip/16600107531.zip)

7.1 Tc3_EtherCATDiag Example

The example project contains two program POUs, each of which contains an example call to `FB_CoE_DiagHistory_Read` and `FB_CoE_DiagHistory_ReadAxis`:



Example call FB_CoE_DiagHistory_Read

The example contains a simple call of the basic FB. The address of the slave is linked via the sNetId and nSlaveAddr inputs. At the bEnable input, the function block can be activated either once (bReadOnce) or permanently (bReadContinous). Messages can be acknowledged with the bAck input. Other controls or configurations can be controlled using stOptions. The messages from the slave are saved in the aMsgArray array. The signals bReadOnce and bAck are cyclically written FALSE.

prgBasicReadout | prgAxisReadout

```
PROGRAM prgBasicReadout
VAR
    bReadOnce                   : BOOL;
    bReadContinuous             : BOOL;
    bAck                        : BOOL;
    stOptions                   : ST_CoE_DiagHistory_Options;

    fbCoEDiagHist               : FB_CoE_DiagHistory_Read;
    IstAmsAdr           AT %I*  : AMSADDR;
    aMsgArray                   : ARRAY[1..100] OF ST_CoE_DiagHistory_Entry;
END_VAR
```

```
fbCoEDiagHist(
    sNetId                  :=  F_CreateAmsNetId(IstAmsAdr.netId),
    nSlaveAddr              :=  IstAmsAdr.port,
    bEnable                 :=  bReadOnce
                                OR bReadContinuous,
    bAck                    :=  bAck,
    tTimeout                :=  ,
    tReadInterval           :=  ,
    stOptions               :=  stOptions,
    aDiagHistory            :=  aMsgArray,
    bBusy                   =>  ,
    bValid                  =>  ,
    bError                  =>  ,
    hrErrorCode             =>  ,
    ipErrorMessage          =>  ,
    stInfo                  =>  );

bReadOnce   :=  FALSE;
bAck        :=  FALSE;

// === END ===
```

Example call FB_CoE_DiagHistory_ReadAxis

The example contains a simple call of the axis FB. The address of the slave is determined using the refAxis. At the bEnable input, the function block can be activated either once (bReadOnce) or permanently (bReadContinous). Messages can be acknowledged with the bAck input. Other controls or configurations can be controlled using stOptions. The messages from the slave are saved in the aMsgArray array. The signals bReadOnce and bAck are cyclically written FALSE.

MAIN prgBasicReadout prgAxisReadout

```
PROGRAM prgAxisReadout
VAR
    bReadOnce                   : BOOL;
    bReadContinuous             : BOOL;
    bAck                        : BOOL;
    stOptions                   : ST_CoE_DiagHistory_Options;

    fbCoEDiagHist_Axis          : FB_CoE_DiagHistory_ReadAxis;
    refAxis                     : AXIS_REF;
    aMsgArray                   : ARRAY[1..100] OF ST_CoE_DiagHistory_Entry;
END_VAR
```

```
fbCoEDiagHist_Axis(
    refAxis                 :=  refAxis,
    sNetId                  :=  ,
    nSlaveAddr              :=  ,
    bEnable                 :=  bReadOnce
                                OR bReadContinuous,
    bAck                    :=  bAck,
    tTimeout                :=  ,
    tReadInterval           :=  ,
    stOptions               :=  stOptions,
    aDiagHistory            :=  aMsgArray,
    bBusy                   =>  ,
    bValid                  =>  ,
    bError                  =>  ,
    hrErrorCode             =>  ,
    ipErrorMessage          =>  ,
    stInfo                  =>  );

bReadOnce   :=  FALSE;
bAck        :=  FALSE;

// === END ===
```

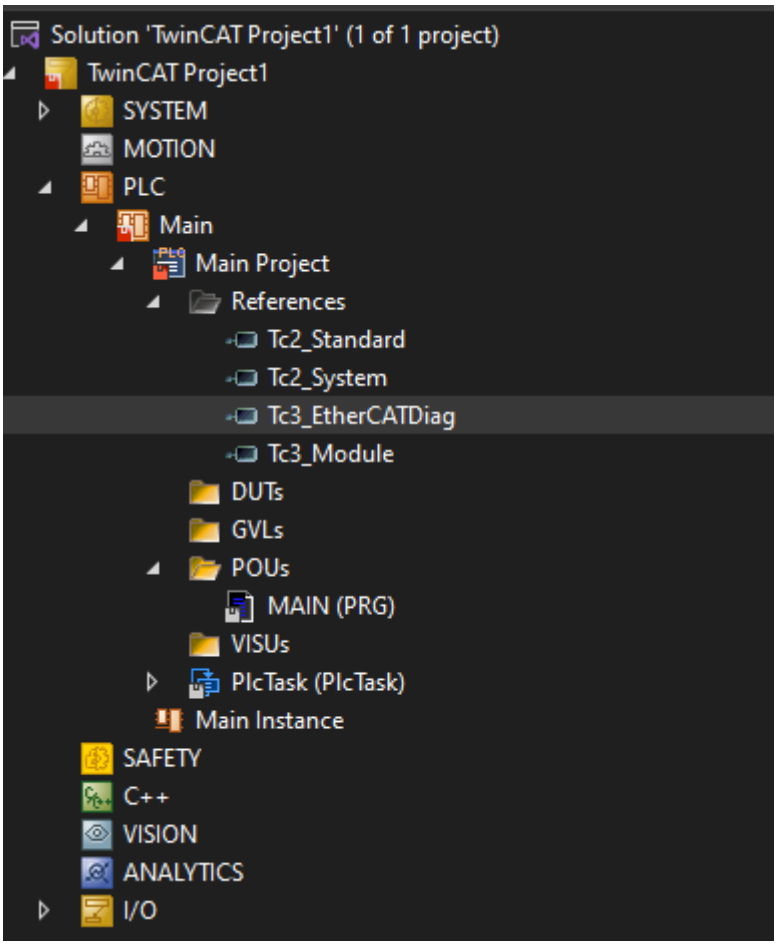
Download: https://infosys.beckhoff.com/content/1033/tcplclib_tc3_ethercatdiag/Resources/16600107531/.zip

7.2 Tutorial

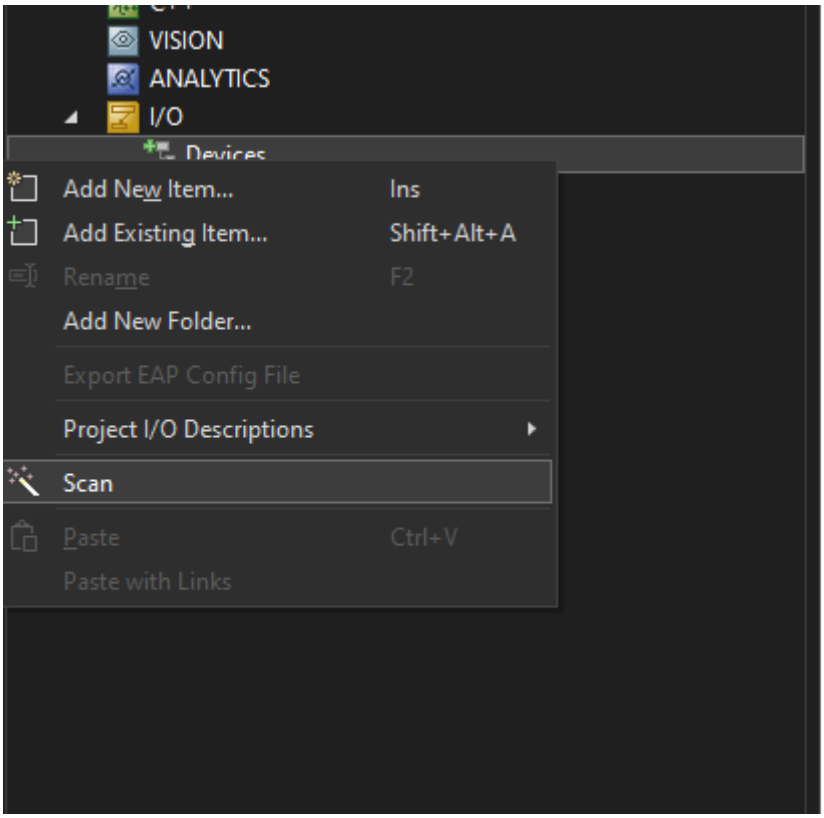
FB_CoE_DiagHistory_Read

The following tutorial documents a simple implementation of FB_CoE_DiagHistory_Read.

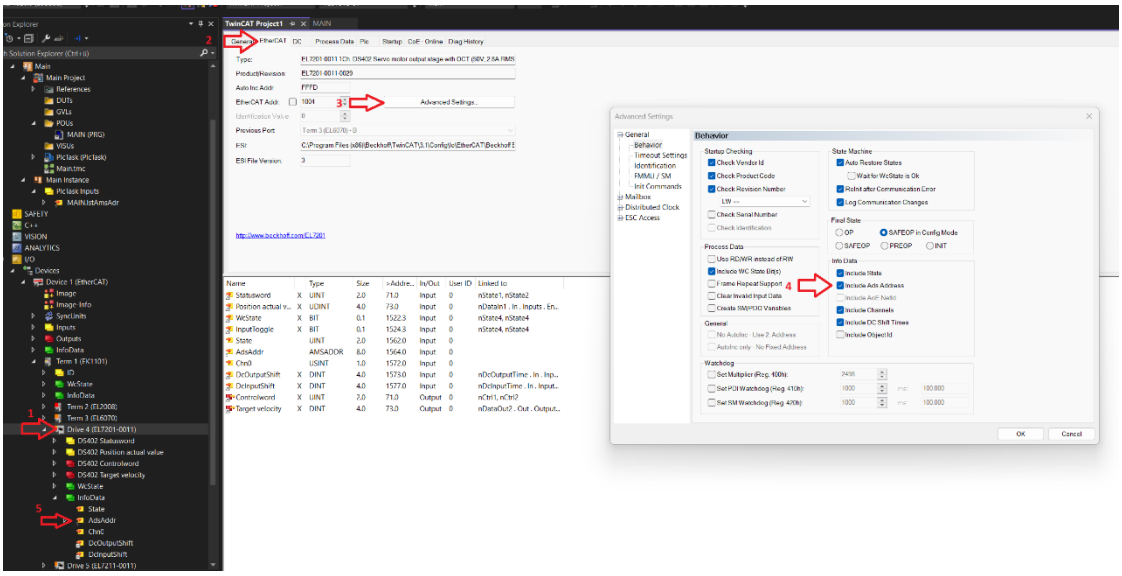
1. Create a new TwinCAT Solution and integrate the Tc3_EtherCATDiag libraries.



⇨ The connected hardware can be read via "Scan".



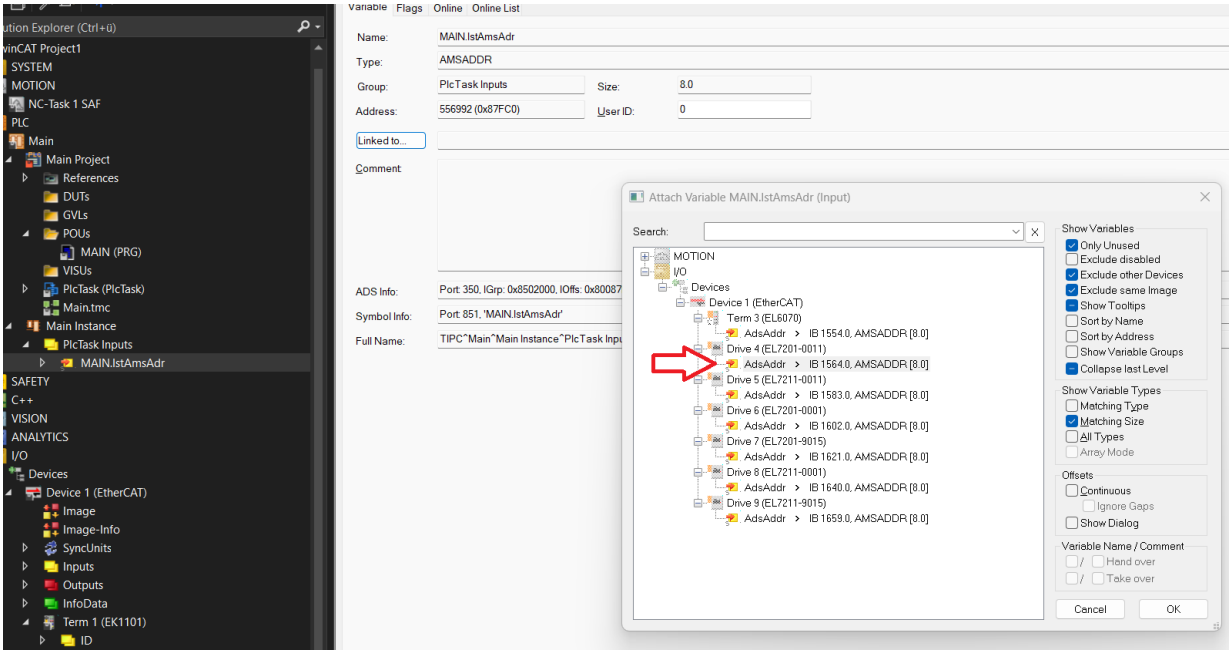
2. You can then make the ADS address of the desired slave visible/linkable in the "Advanced EtherCAT Settings".



3. You can now declare the following variables.

```
MAIN*
1   PROGRAM MAIN
2   VAR
3       bReadOnce                   : BOOL;
4       bReadContinuous             : BOOL;
5       bAck                        : BOOL;
6       stOptions                   : ST_CoE_DiagHistory_Options;
7
8       fbCoEDiagHist               : FB_CoE_DiagHistory_Read;
9       IstAmsAdr           AT %I*  : AMSADDR;
10      aMsgArray                   : ARRAY[1..100] OF ST_CoE_DiagHistory_Entry;
11  END_VAR
12
```

⇨ IstAmsAdr is linked to the AMS address of the desired slave.



⇨ The FB is called as in the example project.

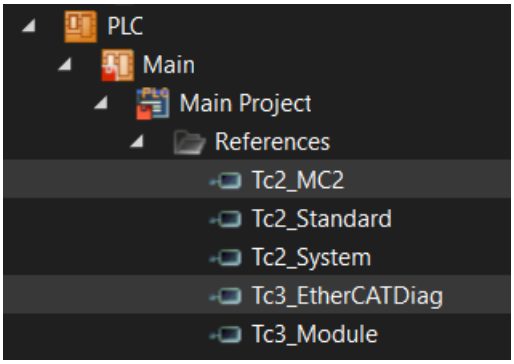
```
fbCoEDiagHist(
    sNetId                  :=  F_CreateAmsNetId(IstAmsAdr.netId),
    nSlaveAddr              :=  IstAmsAdr.port,
    bEnable                 :=  bReadOnce
                                OR bReadContinuous,
    bAck                    :=  bAck,
    tTimeout                :=  ,
    tReadInterval           :=  ,
    stOptions               :=  stOptions,
    aDiagHistory            :=  aMsgArray,
    bBusy                   =>  ,
    bValid                  =>  ,
    bError                  =>  ,
    hrErrorCode             =>  ,
    ipErrorMessage          =>  ,
    stInfo                  =>  );

bReadOnce    :=  FALSE;
bAck         :=  FALSE;
```

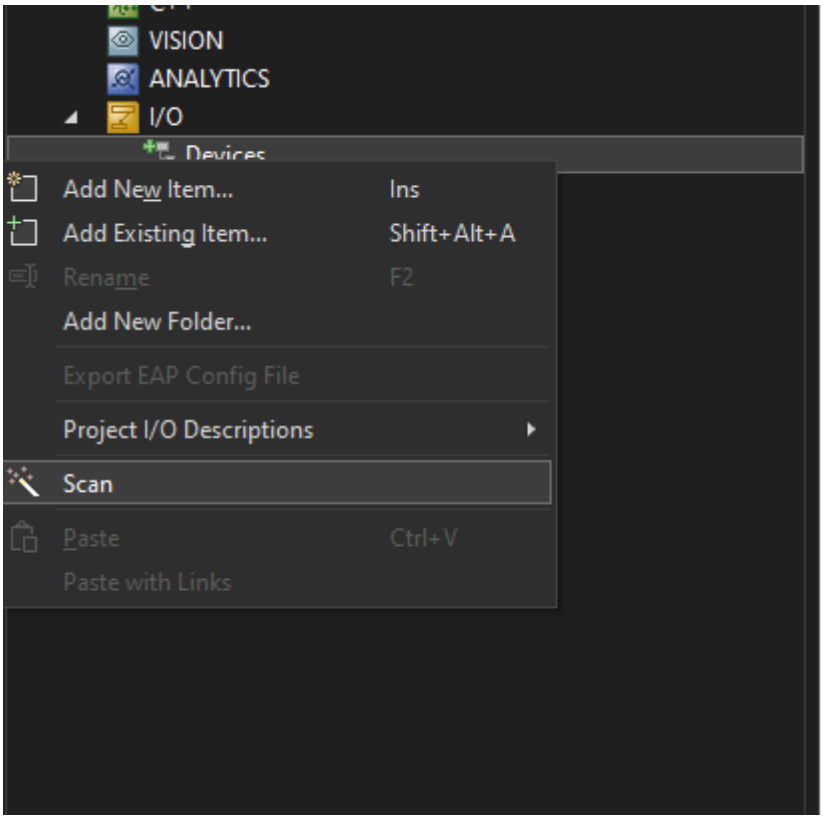
FB_CoE_DiagHistory_ReadAxis

The following tutorial documents a simple implementation of FB_CoE_DiagHistory_ReadAxis.

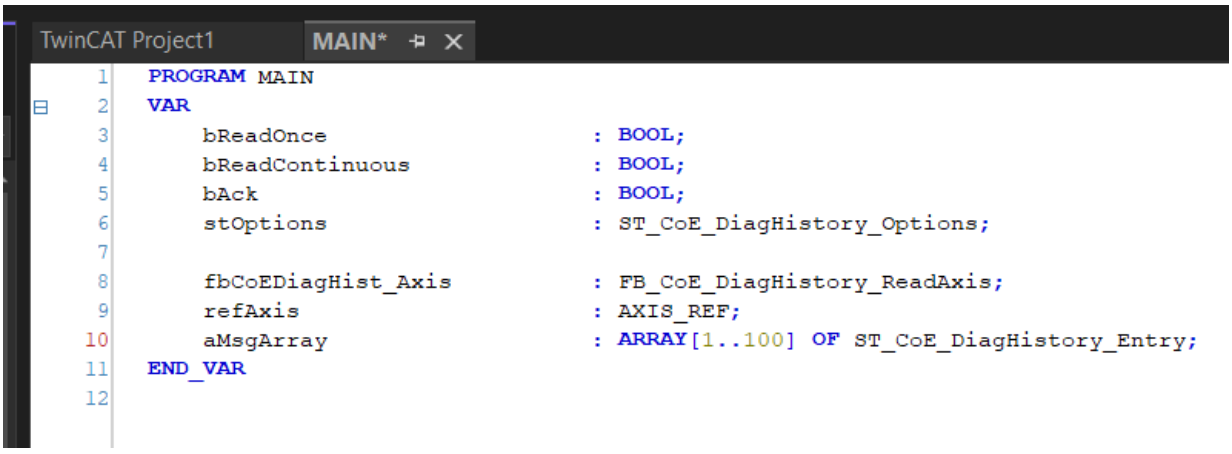
1. Create a new TwinCAT Solution and integrate the libraries Tc3_EtherCATDiag and Tc2_MC2.



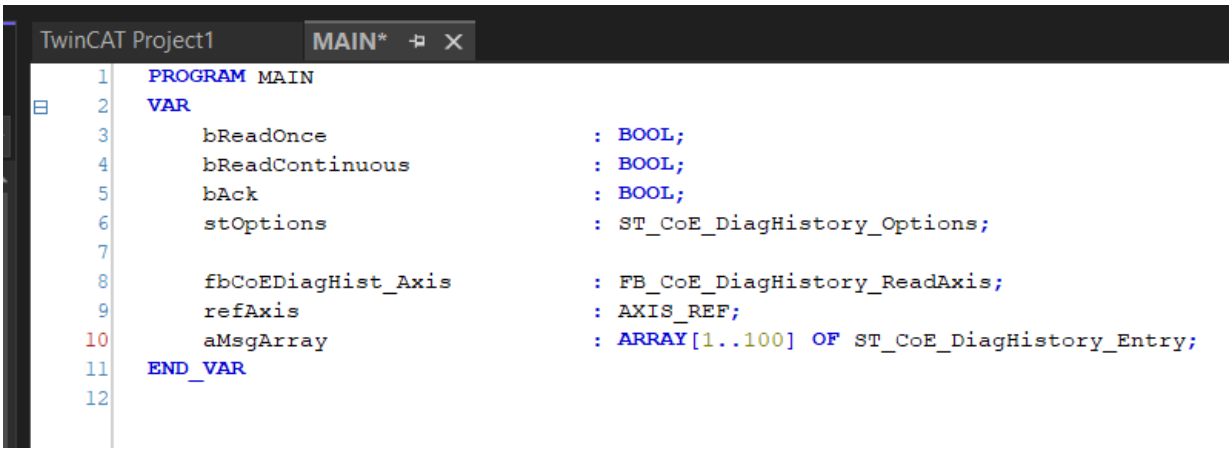
2. You can read the connected hardware via "Scan". Motion axes can be created automatically in this step.



3. You can declare the following variables.



4. Link the refAxis to the desired axis.



⇨ The FB is called as in the example project:

```
fbCoEDiagHist_Axis(
    refAxis                 :=  refAxis,
    sNetId                  :=  ,
    nSlaveAddr              :=  ,
    bEnable                 :=  bReadOnce
                                OR bReadContinuous,
    bAck                    :=  bAck,
    tTimeout                :=  ,
    tReadInterval           :=  ,
    stOptions               :=  stOptions,
    aDiagHistory            :=  aMsgArray,
    bBusy                   =>  ,
    bValid                  =>  ,
    bError                  =>  ,
    hrErrorCode             =>  ,
    ipErrorMessage          =>  ,
    stInfo                  =>  );

bReadOnce   :=  FALSE;
bAck        :=  FALSE;
```

More Information:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com