APPLICATION EXAMPLE

AC500 ETHERCAT

DIAGNOSIS WITH AC500 V3

Contents

1 Disclaimer

A. For customers domiciled outside Germany /

Für Kunden mit Sitz außerhalb Deutschlands

„Warranty, Liability:

The user shall be solely responsible for the use of this products described within this file. ABB shall be under no warranty whatsoever. ABB's liability in connection with application of the products or examples provided or the files included within this products, irrespective of the legal ground, shall be excluded. The exclusion of liability shall not apply in the case of intention or gross negligence. The present declaration shall be governed by and construed in accordance with the laws of Switzerland under exclusion of its conflict of laws rules and of the Vienna Convention on the International Sale of Goods (CISG)."

„Gewährleistung und Haftung:

Der Nutzer ist allein für die Verwendung des in diesem Dokument beschriebenen Produkte und beschriebenen Anwendungsbeispiele verantwortlich.

ABB unterliegt keiner Gewährleistung. Die Haftung von ABB im Zusammenhang mit diesem Anwendungsbeispiel oder den in dieser Datei enthaltenen Dateien - gleich aus welchem Rechtsgrund - ist ausgeschlossen. Dieser Ausschluss gilt nicht im Falle von Vorsatz oder grober Fahrlässigkeit. Diese Erklärung unterliegt Schweizer Recht unter Ausschluss der Verweisungsnormen und des UN-Kaufrechts (CISG)."

B. Nur für Kunden mit Sitz in Deutschland

„Gewährleistung und Haftung:

Die in diesem Dokument beschriebenen Anwendungsbeispiele oder enthaltenen Dateien beschreiben eine mögliche Anwendung der AC500 bzw. zeigen eine mögliche Einsatzart. Sie stellen nur Beispiele für Programmierungen dar, sind aber keine fertigen Lösungen. Eine Gewähr kann nicht übernommen werden.

Der Nutzer ist für die ordnungsgemäße, insbesondere vollständige und fehlerfreie Programmierung der Steuerungen selbst verantwortlich. Im Falle der teilweisen oder ganzen Übernahme der Programmierbeispiele können gegen ABB keine Ansprüche geltend gemacht werden.

Die Haftung von ABB, gleich aus welchem Rechtsgrund, im Zusammenhang mit den Anwendungsbeispielen oder den in dieser Datei enthaltenen Beschreibung wird ausgeschlossen. Der Haftungsausschluss gilt jedoch nicht in Fällen des Vorsatzes, der groben Fahrlässigkeit, bei Ansprüchen nach dem Produkthaftungsgesetz, im Falle der Verletzung des Lebens, des Körpers oder der Gesundheit oder bei schuldhafter Verletzung einer wesentlichen Vertragspflicht. Im Falle der Verletzung einer wesentlichen Vertragspflicht ist die Haftung jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht zugleich ein anderer der in Satz 2 dieses Unterabsatzes erwähnten Fälle gegeben ist. Eine Änderung der Beweislast zum Nachteil des Nutzers ist hiermit nicht verbunden.

Es gilt materielles deutsches Recht unter Ausschluss des UN-Kaufrechts."

2 Introduction

2.1 Scope of the document

This application example shows the structural procedure of the EtherCAT diagnostics in IEC code with an AC500 V3 PLC. The example does not replace one to one the diagnosis for each plant - much more the diagnosis depends on the application.

The example should show the general procedure of the diagnosis system and how to use it inside IEC programming.

2.2 Compatibility

The application example explained in this document have been used with the below engineering system versions. They should also work with other versions, nevertheless some small adaptations may be necessary, for future versions.

- Automation Builder 2.4.1 or newer
- AC500 V3 PLC
- CPU Firmware Version ≥ 3.4.1.278
- CM579-ETHCAT Firmware Version ≥ 4.5.7.21
- Libraries

AC500_EtherCAT,	≥ 1.3.1.5
AC500_EcatBase,	≥ 1.3.1.5
AC500_NetxEcat,	≥ 1.3.1.2

3 Automation Builder configuration and Project settings

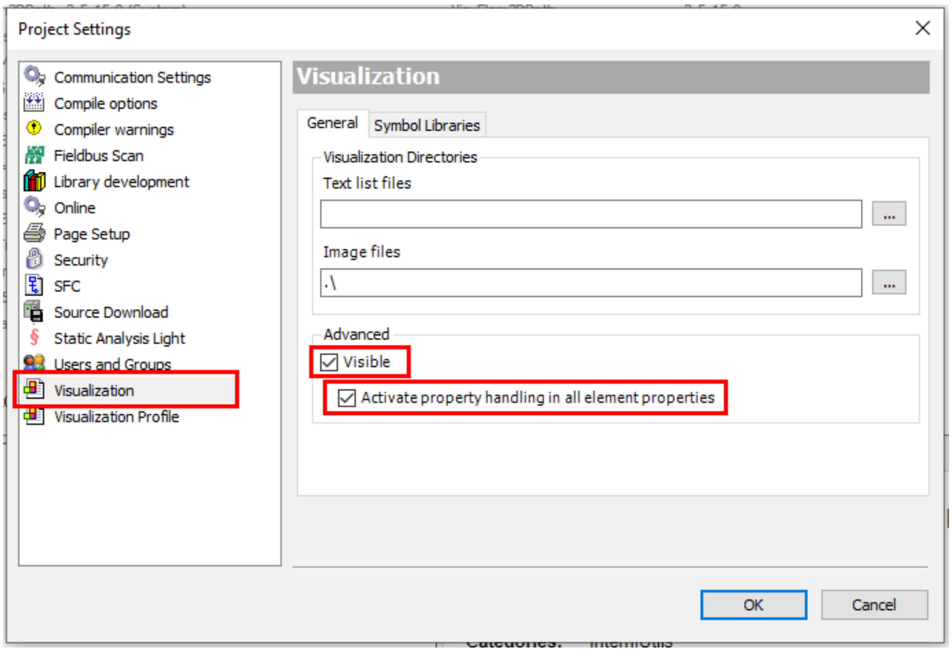
The configuration of the application example does only simulate an EtherCAT system with different modules for better understanding of the diagnosis system.

3.1 Project Settings

This application example uses the object-oriented programming style (OOP) whereby the Properties represents the state of different values and variables of the diagnosis information.

In order to use these Properties in the visualization the project settings must be modified accordingly by activate the property handling in all elements.

To do this, open the **Project > Project settings** menu and activate the **Visible** checkbox under the Visualizations tab. After that an additional checkbox for the **Active property handling in all element properties** can be activated.



Text list files:

Image files: .\

☑ Visible

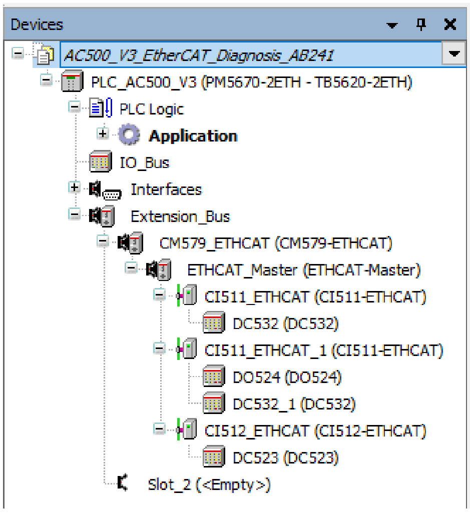
☑ Activate property handling in all element properties

Please take in mind, that these settings must be done for each project itself. It is not a setting for the complete Automation Builder, only for the project.

3.2 Configuration Overview

The system is built up with the following devices:

- AC500 PLC **PM5670-ETH**
- Communication Module **CM579-ETHCAT**
- Communication Interface **CI511-ETHCAT** and **CI512-ETHCAT**
- S500 Input/ Output modules **DC532**, **DC523**, **DO524**



Devices

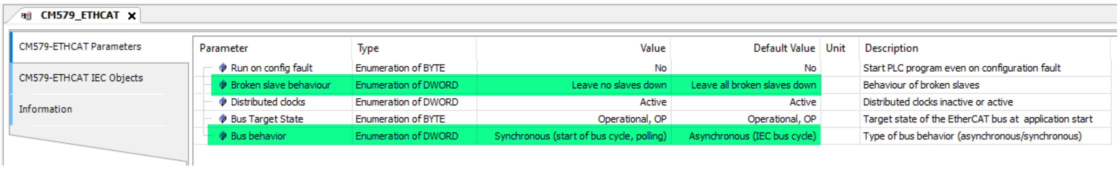
- AC500_V3_EtherCAT_Diagnosis_AB241
  - PLC_AC500_V3 (PM5670-2ETH - TB5620-2ETH)
    - PLC Logic
      - **Application**
    - IO_Bus
    - Interfaces
    - Extension_Bus
      - CM579_ETHCAT (CM579-ETHCAT)
        - ETHCAT_Master (ETHCAT-Master)
          - CI511_ETHCAT (CI511-ETHCAT)
            - DC532 (DC532)
          - CI511_ETHCAT_1 (CI511-ETHCAT)
            - DO524 (DO524)
            - DC532_1 (DC532)
          - CI512_ETHCAT (CI512-ETHCAT)
            - DC523 (DC523)
      - Slot_2 (<Empty>)

For a properly working EtherCAT system, the Topology of the configuration must match to the one of the setup.

The configuration of this example can be easily modified and adapted to other systems. The diagnostic will be initialized automatically for up to 256 slaves that are configured at the ETHCAT_Master. Only the visualization must be modified accordingly for visualization purposes.

3.3 CM579_ETHCAT Parameter

At the CM579_ETHCAT node, the Application Example uses the following parameter settings:



CM579_ETHCAT

CM579-ETHCAT Parameters

CM579-ETHCAT IEC Objects

Information

Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Broken slave behaviour	Enumeration of DWORD	Leave no slaves down	Leave all broken slaves down		Behaviour of broken slaves
Distributed clocks	Enumeration of DWORD	Active	Active		Distributed clocks inactive or active
Bus Target State	Enumeration of BYTE	Operational, OP	Operational, OP		Target state of the EtherCAT bus at application start
Bus behavior	Enumeration of DWORD	Synchronous (start of bus cycle, polling)	Asynchronous (IEC bus cycle)		Type of bus behavior (asynchronous/synchronous)

Broken slave behavior: Leave no slaves down

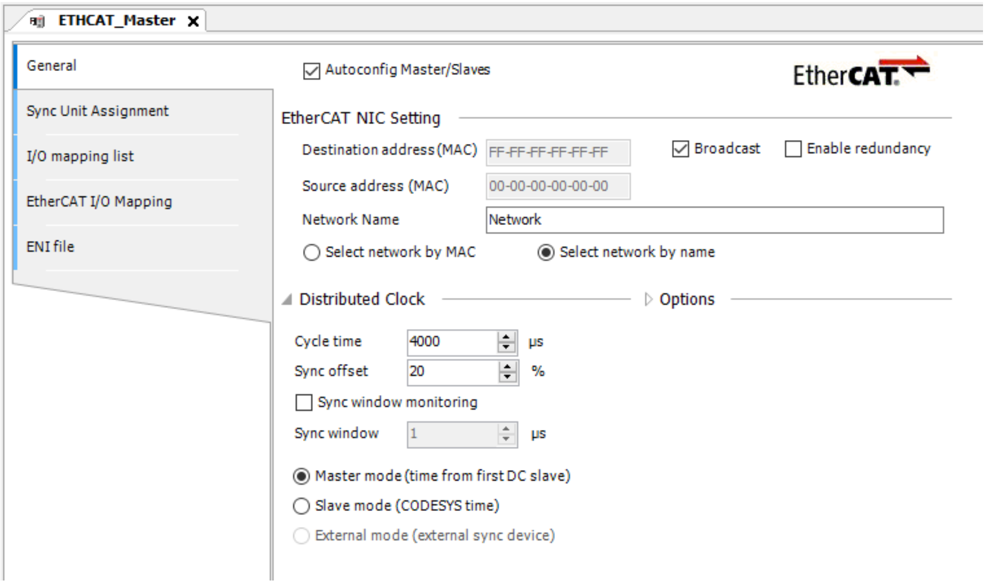
Using ‘Broken slave behavior’ with value ‘Leave no slaves down’ causes that broken slaves will be reintegrated into the system again automatically. Changing this behavior, the diagnostics must modify accordingly, as the reintegration needs to be done manually.

Bus behavior: Synchronous (start of bus cycle, polling)

There are three options for the Bus behavior available. Asynchronous, Synchronous (start of bus cycle) and Synchronous (end of bus cycle). Asynchronous means that the bus does not synchronize the data and a usual Cyclic Task can be chosen for EtherCAT. For real time applications, the data needs to be synchronized and therefore the Parameter must be selected accordingly. With the usage of a synchronize bus behavior, the Task configuration must be configured as external Event (please find more information here).

3.4 ETHCAT_Master

General EtherCAT bus settings can be done at the ETHCAT_Master node (e.g. Cycle Time for the EtherCAT bus). For the EtherCAT diagnostics there are no special settings to be done in the General tab, which is why the settings remains on default.



☑ Autoconfig Master/Slaves

EtherCAT NIC Setting

Destination address (MAC) FF-FF-FF-FF-FF-FF ☑ Broadcast ☐ Enable redundancy

Source address (MAC) 00-00-00-00-00-00

Network Name Network

○ Select network by MAC ● Select network by name

Distributed Clock

Options

Cycle time 4000 µs

Sync offset 20 %

☐ Sync window monitoring

Sync window 1 µs

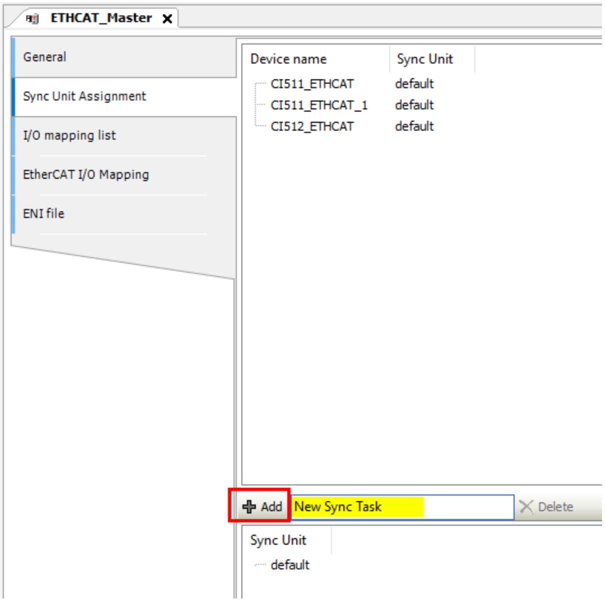
● Master mode (time from first DC slave)

○ Slave mode (CODESYS time)

○ External mode (external sync device)

Only the Sync Units are relevant for the diagnosis. Each Sync Unit has its own Working Counter which are monitoring the EtherCAT telegrams for errors and faults. As default all appended device will be assigned to the Sync Unit “default”.

The Application Example itself does only contains the default Sync Units but for other Applications it might be required to have additional Sync Units which will then of course need to be implemented inside the diagnosis.



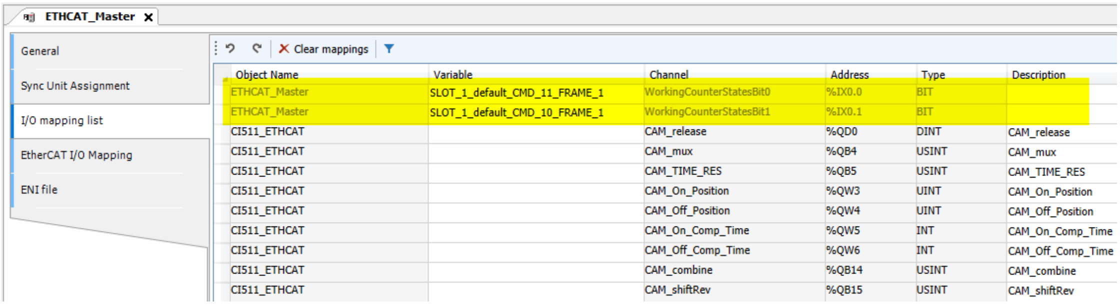
Device name	Sync Unit
CI511_ETHCAT	default
CI511_ETHCAT_1	default
CI512_ETHCAT	default

New Sync Task

Sync Unit
default

Additional Sync Units can be defined in the yellow highlighted Textbox and created by confirming with the “Add”-Button. The Sync Units can then be changed for each Device manually in the upper window.

One Sync Unit has two Working Counter to be monitored. The first one will monitor the Inputs (CMD10) and the second one the Outputs (CMD11) for errors and faults.

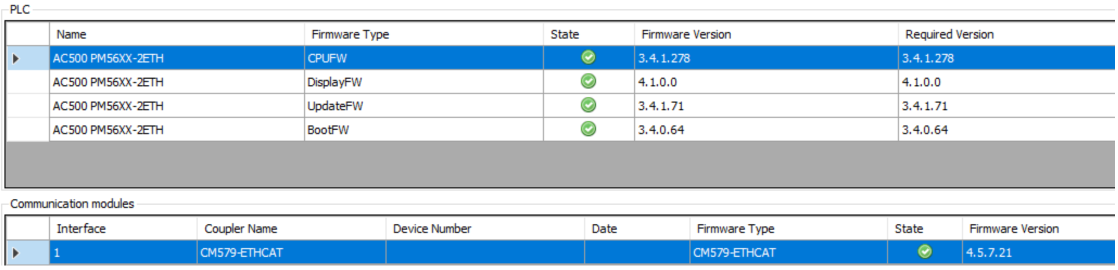


Object Name	Variable	Channel	Address	Type	Description
ETHCAT_Master	SLOT_1_default_CMD_11_FRAME_1	WorkingCounterStatesBit0	%IX0.0	BIT	
ETHCAT_Master	SLOT_1_default_CMD_10_FRAME_1	WorkingCounterStatesBit1	%IX0.1	BIT	
CI511_ETHCAT		CAM_release	%QD0	DINT	CAM_release
CI511_ETHCAT		CAM_mux	%QB4	USINT	CAM_mux
CI511_ETHCAT		CAM_TIME_RES	%QB5	USINT	CAM_TIME_RES
CI511_ETHCAT		CAM_On_Position	%QW3	UINT	CAM_On_Position
CI511_ETHCAT		CAM_Off_Position	%QW4	UINT	CAM_Off_Position
CI511_ETHCAT		CAM_On_Comp_Time	%QW5	INT	CAM_On_Comp_Time
CI511_ETHCAT		CAM_Off_Comp_Time	%QW6	INT	CAM_Off_Comp_Time
CI511_ETHCAT		CAM_combine	%QB14	USINT	CAM_combine
CI511_ETHCAT		CAM_shiftRev	%QB15	USINT	CAM_shiftRev

Adding a Sync Unit affects that two further Working Counters will be added to the ETHCAT_Master's IO mapping list. These variables can be used to monitor the bus in the IEC programming.

3.5 Firmware Version

This example supports the following firmware versions and higher.



PLC

Name	Firmware Type	State	Firmware Version	Required Version
AC500 PM56XX-2ETH	CPUFW	✓	3.4.1.278	3.4.1.278
AC500 PM56XX-2ETH	DisplayFW	✓	4.1.0.0	4.1.0.0
AC500 PM56XX-2ETH	UpdateFW	✓	3.4.1.71	3.4.1.71
AC500 PM56XX-2ETH	BootFW	✓	3.4.0.64	3.4.0.64

Communication modules

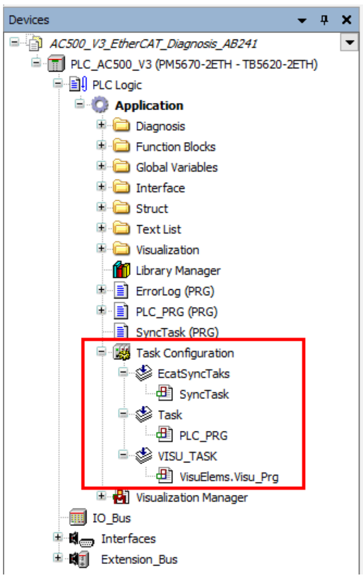
Interface	Coupler Name	Device Number	Date	Firmware Type	State	Firmware Version
1	CM579-ETHCAT			CM579-ETHCAT	✓	4.5.7.21

The full functionality of the example cannot be ensured with older firmware versions.

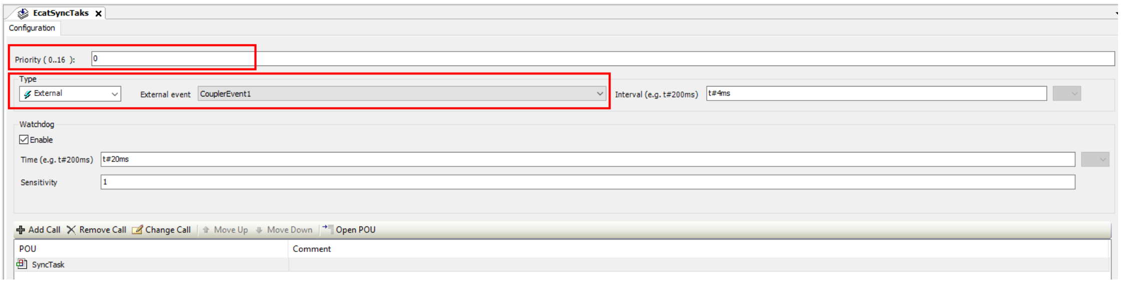
4 Codesys Program

4.1 Task Configuration

The Task configuration for this example consists of three different Tasks:



EcatSyncTask calling the *SyncTask* Program and is for real time calculations for the EtherCAT specific application.



Priority (0..16): 0

Type: External

External event: CouplerEvent1

Interval (e.g. t#200ms): t#4ms

Watchdog

☑ Enable

Time (e.g. t#200ms): t#20ms

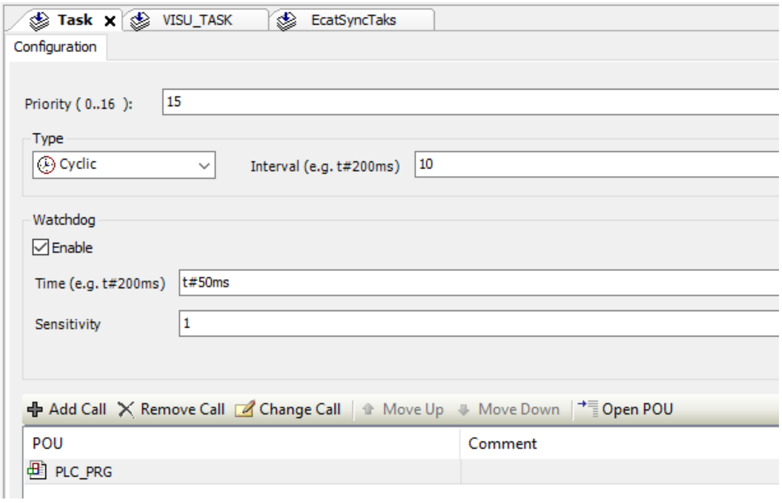
Sensitivity: 1

POU	Comment
SyncTask	

For a synchronized EtherCAT bus it is required to set the Task configuration as an external event, where External event – CouplerEvent1 stands for the first coupler slot next to the PLC. As soon as the Bus behavior in the CM579-ETHCAT Parameter is set to Synchronized, the External event must be configured for the Task, otherwise the bus will not run.

Priorities 0-15 are realtime capable, where 0 is the highest and 15 the lowest one. Priority 16 is a non-realtime capable priority and should never be selected for EtherCAT.

The ***Task***, which is calling the *PLC_PRG* Program is the slow task which will run most of the diagnostic to prevent high load on the EtherCAT bus.



Priority (0..16): 15

Type: Cyclic

Interval (e.g. t#200ms): 10

Watchdog

☑ Enable

Time (e.g. t#200ms): t#50ms

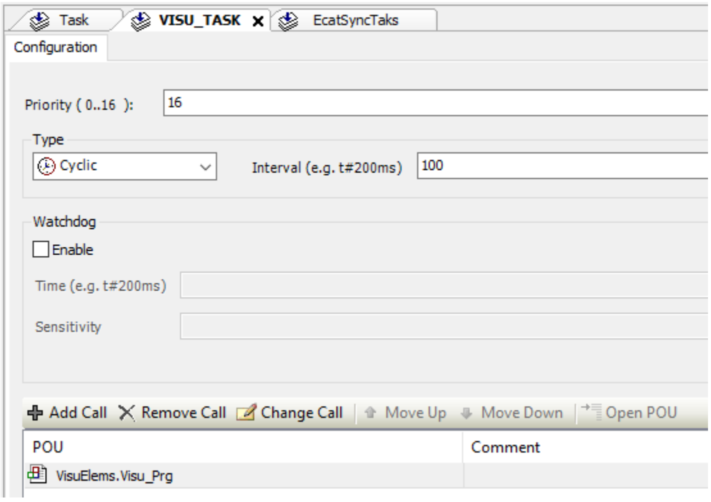
Sensitivity: 1

POU	Comment
PLC_PRG	

It is using the default settings of the Task configuration, except the Watchdog time, which was increased to 50 milli seconds.

In a real application with accuracy calculations for EtherCAT, the fast Task should have a higher Priority than the slow Task.

Finally, the **VISU_TASK** is generated by the system automatically as soon as a visualization will be added to the project.



Priority (0..16): 16

Type: Cyclic

Interval (e.g. t#200ms): 100

Watchdog

☐ Enable

Time (e.g. t#200ms):

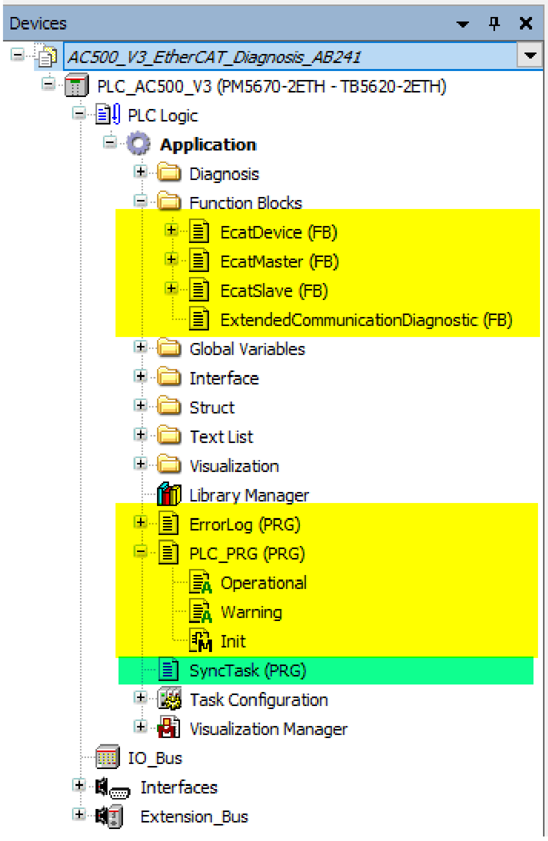
Sensitivity:

POU	Comment
VisuElems.Visu_Prg	

It uses the default settings as well and is based on a non-realtime Task (Priority 16) as the visualization must not affect the runtime and the calculation of an application.

4.2 Program structure

According to the task configuration, the program structure consists of the code for the fast synchronous EtherCAT calculation (highlighted green) and the part of the slow asynchronous calculation (highlighted yellow).

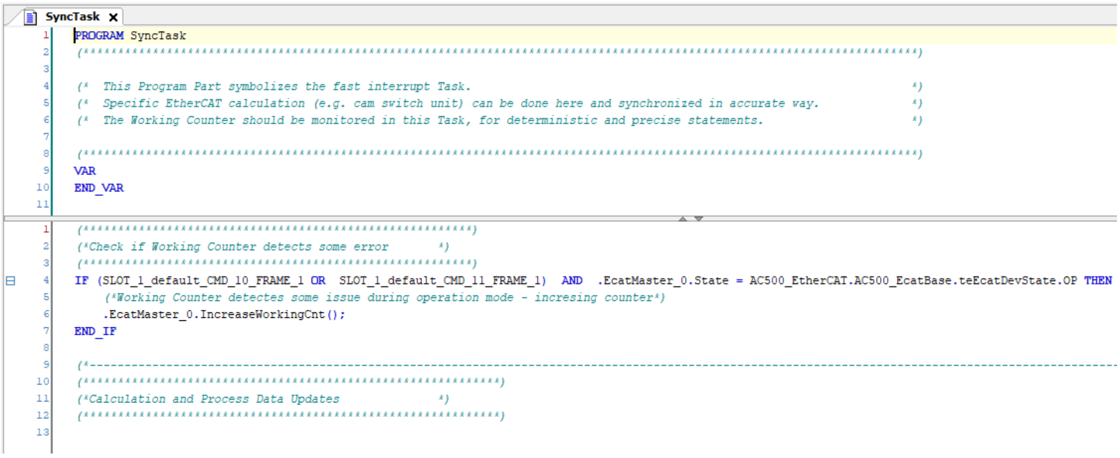


The complete EtherCAT diagnosis is executed in the slow task except for the working counter evaluation. This one will be monitored in the fast task to detect each faulty telegram or fault.

4.2.1 SyncTask (PRG)

The working counters are increasing in each device and in every frame. The master is expecting a predefined working counter value and validates the returned working counter. In case of an error or fault, the working counter does not increase, and the validation indicates a fault or error.

This validation is part of the specification and will be done automatically without any further settings, parameters or function block. As this is part of the EtherCAT frame, the validation inside the fast Task will not increase the Load. It is strongly recommended to query the working counter in the fast task to detect errors without any delay.



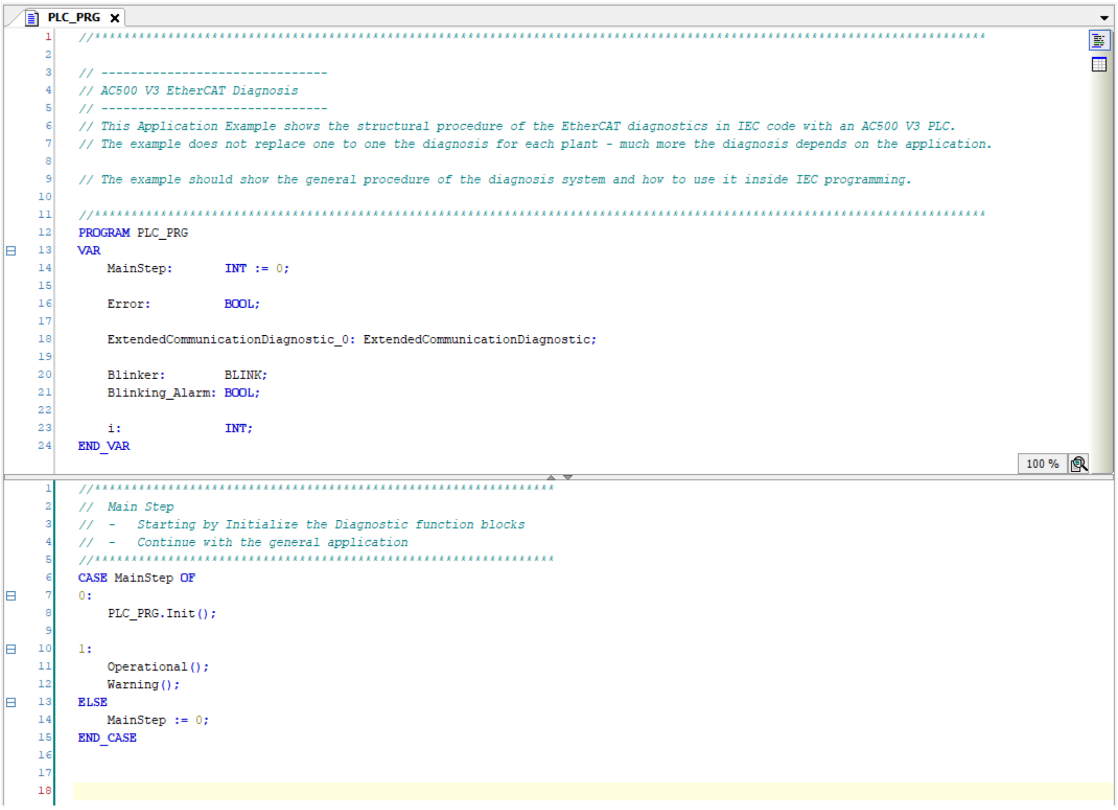
```
PROGRAM SyncTask
(**********************************************************************************************)

(*  This Program Part symbolizes the fast interrupt Task.                                       *)
(*  Specific EtherCAT calculation (e.g. cam switch unit) can be done here and synchronized in accurate way.          *)
(*  The Working Counter should be monitored in this Task, for deterministic and precise statements.                *)

(**********************************************************************************************)
VAR
END_VAR
```

```
(*********************************************************)
(*Check if Working Counter detects some error        *)
(*********************************************************)
IF (SLOT_1_default_CMD_10_FRAME_1 OR  SLOT_1_default_CMD_11_FRAME_1)  AND  .EcatMaster_0.State = AC500_EtherCAT.AC500_EcatBase.teEcatDevState.OP THEN
    (*Working Counter detectes some issue during operation mode - incresing counter*)
    .EcatMaster_0.IncreaseWorkingCnt();
END_IF

(*-----------------------------------------------------------------------------------------------------------------
(*************************************************************)
(*Calculation and Process Data Updates                *)
(*************************************************************)
```

4.2.2 PLC_PRG (PRG)

To prevent the fast EtherCAT Task for high load most of the diagnostic takes place at the PLC_PRG program part in the slow task.

PLC_PRG in general consists of 9 lines of code and does only navigate trough the different states of the system.



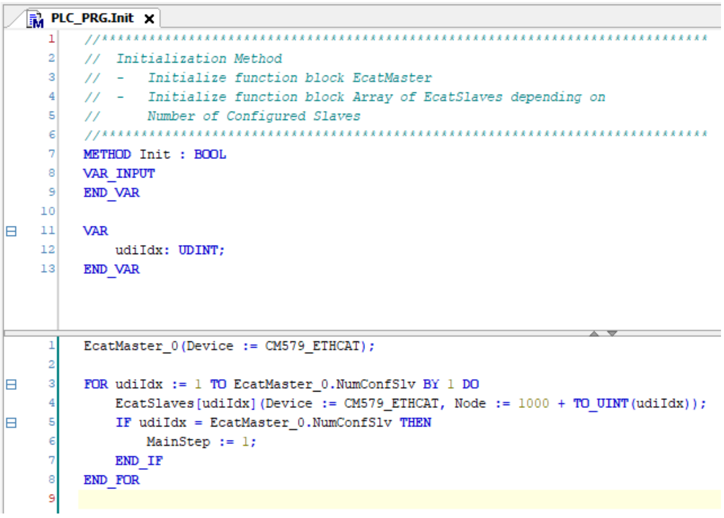
```
//****************************************************************************************************************

// ------------------------------
// AC500 V3 EtherCAT Diagnosis
// ------------------------------
// This Application Example shows the structural procedure of the EtherCAT diagnostics in IEC code with an AC500 V3 PLC.
// The example does not replace one to one the diagnosis for each plant - much more the diagnosis depends on the application.

// The example should show the general procedure of the diagnosis system and how to use it inside IEC programming.

//****************************************************************************************************************
PROGRAM PLC_PRG
VAR
    MainStep:       INT := 0;

    Error:          BOOL;

    ExtendedCommunicationDiagnostic_0: ExtendedCommunicationDiagnostic;

    Blinker:        BLINK;
    Blinking_Alarm: BOOL;

    i:              INT;
END_VAR
```

```
//****************************************************************
//  Main Step
//  -   Starting by Initialize the Diagnostic function blocks
//  -   Continue with the general application
//****************************************************************
CASE MainStep OF
0:
    PLC_PRG.Init();

1:
    Operational();
    Warning();
ELSE
    MainStep := 0;
END_CASE
```

4.2.2.1 Init (Method)

During the startup of the system, the PLC_PRG method Init() is called. Within this initialization the function blocks will be assigned their inputs.

Beginning with the EcatMaster function block, the system receives the information how many slaves are configured. This information is needed to initialize the number of EcatSlave function blocks to receive diagnosis of each slave.



```
//****************************************************************************
//  Initialization Method
//  -   Initialize function block EcatMaster
//  -   Initialize function block Array of EcatSlaves depending on
//      Number of Configured Slaves
//****************************************************************************
METHOD Init : BOOL
VAR_INPUT
END_VAR

VAR
    udiIdx: UDINT;
END_VAR
```

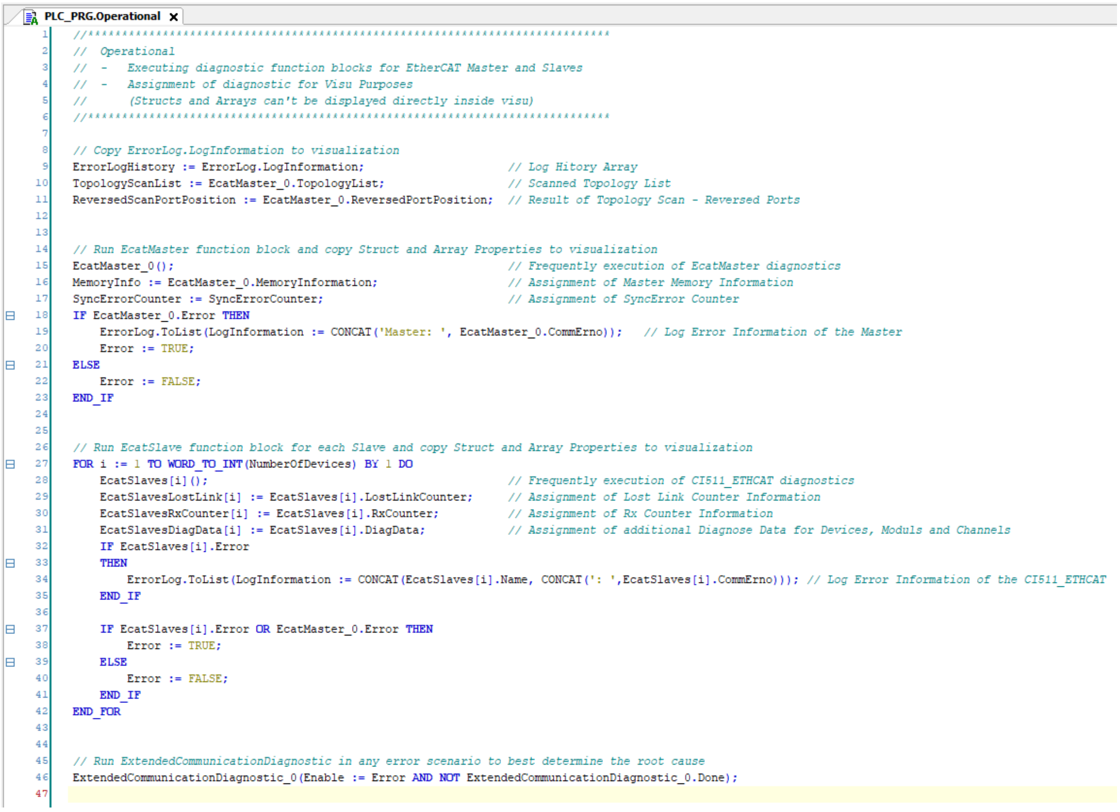
```
EcatMaster_0(Device := CM579_ETHCAT);

FOR udiIdx := 1 TO EcatMaster_0.NumConfSlv BY 1 DO
    EcatSlaves[udiIdx](Device := CM579_ETHCAT, Node := 1000 + TO_UINT(udiIdx));
    IF udiIdx = EcatMaster_0.NumConfSlv THEN
        MainStep := 1;
    END_IF
END_FOR
```

Independent of the configuration of the EtherCAT system, the instance of the function blocks and therefore of each slave will be done automatically for up to 256 slaves.

Only the initialization of the EcatMaster must be modified if the instance name of the CM579_ETHCAT is changed or a second coupler is added to the system.

4.2.2.2 Operational (Action)

After the initialization phase the MainStep continues with the Action *Operational*. During Operational the function blocks are called and some of the information will be assigned to local variables to be able to visualize them.



```
//****************************************************************************
//  Operational
//  -   Executing diagnostic function blocks for EtherCAT Master and Slaves
//  -   Assignment of diagnostic for Visu Purposes
//       (Structs and Arrays can't be displayed directly inside visu)
//****************************************************************************

// Copy ErrorLog.LogInformation to visualization
ErrorLogHistory := ErrorLog.LogInformation;                    // Log Hitory Array
TopologyScanList := EcatMaster_0.TopologyList;                 // Scanned Topology List
ReversedScanPortPosition := EcatMaster_0.ReversedPortPosition; // Result of Topology Scan - Reversed Ports


// Run EcatMaster function block and copy Struct and Array Properties to visualization
EcatMaster_0();                                                // Frequently execution of EcatMaster diagnostics
MemoryInfo := EcatMaster_0.MemoryInformation;                  // Assignment of Master Memory Information
SyncErrorCounter := SyncErrorCounter;                          // Assignment of SyncError Counter
IF EcatMaster_0.Error THEN
    ErrorLog.ToList(LogInformation := CONCAT('Master: ', EcatMaster_0.CommErno));    // Log Error Information of the Master
    Error := TRUE;
ELSE
    Error := FALSE;
END_IF


// Run EcatSlave function block for each Slave and copy Struct and Array Properties to visualization
FOR i := 1 TO WORD_TO_INT(NumberOfDevices) BY 1 DO
    EcatSlaves[i]();                                           // Frequently execution of CI511_ETHCAT diagnostics
    EcatSlavesLostLink[i] := EcatSlaves[i].LostLinkCounter;    // Assignment of Lost Link Counter Information
    EcatSlavesRxCounter[i] := EcatSlaves[i].RxCounter;         // Assignment of Rx Counter Information
    EcatSlavesDiagData[i] := EcatSlaves[i].DiagData;           // Assignment of additional Diagnose Data for Devices, Moduls and Channels
    IF EcatSlaves[i].Error
    THEN
        ErrorLog.ToList(LogInformation := CONCAT(EcatSlaves[i].Name, CONCAT(': ',EcatSlaves[i].CommErno))); // Log Error Information of the CI511_ETHCAT
    END_IF

    IF EcatSlaves[i].Error OR EcatMaster_0.Error THEN
        Error := TRUE;
    ELSE
        Error := FALSE;
    END_IF
END_FOR


// Run ExtendedCommunicationDiagnostic in any error scenario to best determine the root cause
ExtendedCommunicationDiagnostic_0(Enable := Error AND NOT ExtendedCommunicationDiagnostic_0.Done);
```

4.2.3 ErrorLog (PRG)

The error messages of the EtherCAT system will be stored inside an array which is visualized at Log History page of the Visualization.

The ErrorLog program handles current pending error messages inside the array. It contains the methods **Delete**, **DeleteAll**, **ToList** as well es the properties **LogInformation** and **SelectedRow**.

The property **LogInformation** contains the array for the diagnosis messages including a timestamp.

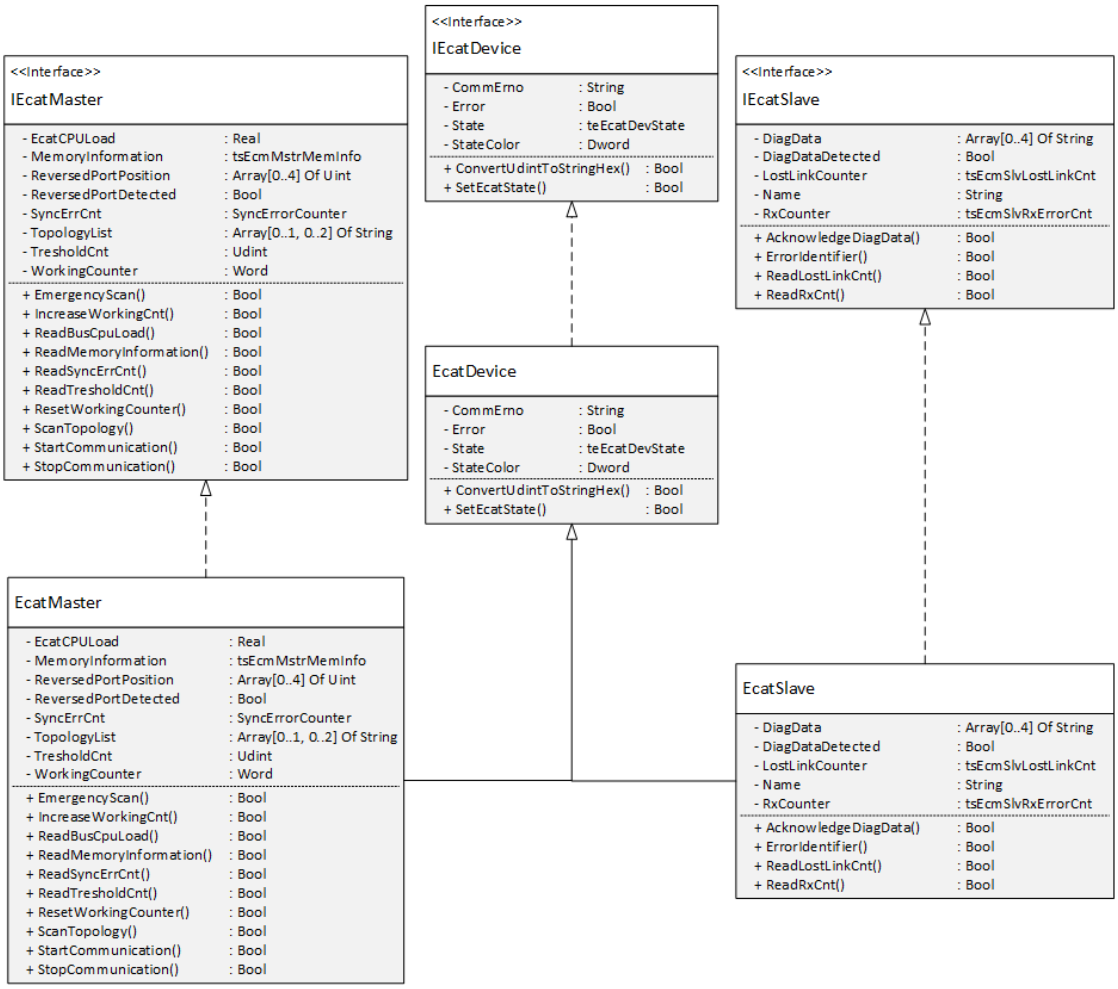
With the Method **ToList** the diagnosis information will be logged and the timestamp added. To avoid an overflow of error information, the method checks for duplicated diagnosis information.

Delete does delete single information, depending on the Selected Row of the visualized Table.

Delete All deletes the complete array accordingly.

4.2.4 Diagnostic function blocks

The program uses the Object-oriented programming style whereby the diagnostic is structured according to the following inheritance.

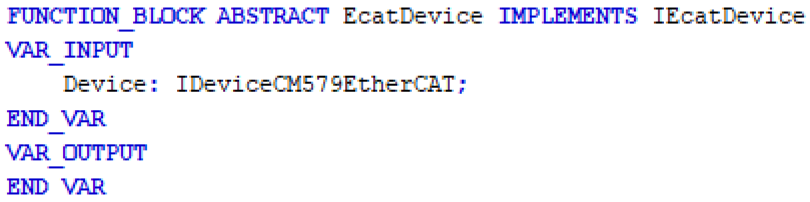

<<Interface>>
IEcatMaster
- EcatCPULoad : Real
- MemoryInformation : tsEcmMstrMemInfo
- ReversedPortPosition : Array[0..4] Of Uint
- ReversedPortDetected : Bool
- SyncErrCnt : SyncErrorCounter
- TopologyList : Array[0..1, 0..2] Of String
- TresholdCnt : Udint
- WorkingCounter : Word
+ EmergencyScan() : Bool
+ IncreaseWorkingCnt() : Bool
+ ReadBusCpuLoad() : Bool
+ ReadMemoryInformation() : Bool
+ ReadSyncErrCnt() : Bool
+ ReadTresholdCnt() : Bool
+ ResetWorkingCounter() : Bool
+ ScanTopology() : Bool
+ StartCommunication() : Bool
+ StopCommunication() : Bool
<<Interface>>
IEcatDevice
- CommErrno : String
- Error : Bool
- State : teEcatDevState
- StateColor : Dword
+ ConvertUdintToStringHex() : Bool
+ SetEcatState() : Bool
<<Interface>>
IEcatSlave
- DiagData : Array[0..4] Of String
- DiagDataDetected : Bool
- LostLinkCounter : tsEcmSlvLostLinkCnt
- Name : String
- RxCounter : tsEcmSlvRxErrorCnt
+ AcknowledgeDiagData() : Bool
+ ErrorIdentifier() : Bool
+ ReadLostLinkCnt() : Bool
+ ReadRxCnt() : Bool
EcatDevice
- CommErrno : String
- Error : Bool
- State : teEcatDevState
- StateColor : Dword
+ ConvertUdintToStringHex() : Bool
+ SetEcatState() : Bool
EcatMaster
- EcatCPULoad : Real
- MemoryInformation : tsEcmMstrMemInfo
- ReversedPortPosition : Array[0..4] Of Uint
- ReversedPortDetected : Bool
- SyncErrCnt : SyncErrorCounter
- TopologyList : Array[0..1, 0..2] Of String
- TresholdCnt : Udint
- WorkingCounter : Word
+ EmergencyScan() : Bool
+ IncreaseWorkingCnt() : Bool
+ ReadBusCpuLoad() : Bool
+ ReadMemoryInformation() : Bool
+ ReadSyncErrCnt() : Bool
+ ReadTresholdCnt() : Bool
+ ResetWorkingCounter() : Bool
+ ScanTopology() : Bool
+ StartCommunication() : Bool
+ StopCommunication() : Bool
EcatSlave
- DiagData : Array[0..4] Of String
- DiagDataDetected : Bool
- LostLinkCounter : tsEcmSlvLostLinkCnt
- Name : String
- RxCounter : tsEcmSlvRxErrorCnt
+ AcknowledgeDiagData() : Bool
+ ErrorIdentifier() : Bool
+ ReadLostLinkCnt() : Bool
+ ReadRxCnt() : Bool

Assumed that every EtherCAT Device (EcatDevice) has its own state, error and communication error the diagnosis of master and slave differs. Therefore, EcatMaster and EcatSlave inherit from EcatDevices and implements the specific diagnosis interfaces accordingly.

4.2.4.1 EcatDevice (FB)

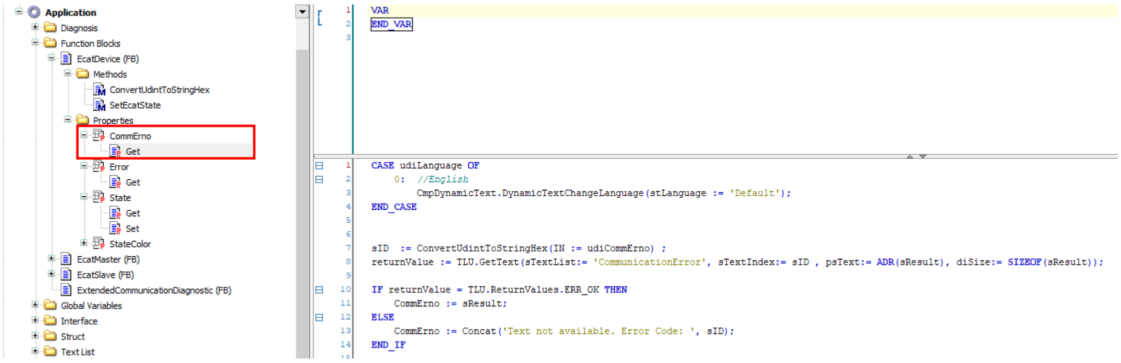
Inherit EcatDevice to EcatMaster and EcatSlave allows to use the code from the parent in the child function blocks. As there can be multiple masters at on system, the specific CM579-ETHCAT module must be defined for each device.

The CM579-ETHCAT module will be assigned by an interface inside Automation Builder. This assignment must be forwarded to the EcatDevice function bock as followed:



```
FUNCTION_BLOCK ABSTRACT EcatDevice IMPLEMENTS IEcatDevice
VAR_INPUT
    Device: IDeviceCM579EtherCAT;
END_VAR
VAR_OUTPUT
END_VAR
```

As an example for the common code inside the EcatDevice function block which will be then inherit to EcatMaster and EcatSlave the CommErno can be considered.



```
VAR
END_VAR
```

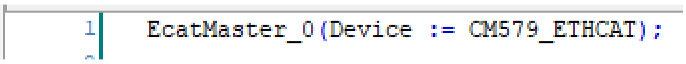
```
CASE udiLanguage OF
    0:  //English
        CmpDynamicText.DynamicTextChangeLanguage(stLanguage := 'Default');
END_CASE


sID  := ConvertUdintToStringHex(IN := udiCommErno) ;
returnValue := TLU.GetText(sTextList:= 'CommunicationError', sTextIndex:= sID , psText:= ADR(sResult), diSize:= SIZEOF(sResult));

IF returnValue = TLU.ReturnValues.ERR_OK THEN
    CommErno := sResult;
ELSE
    CommErno := Concat('Text not available. Error Code: ', sID);
END_IF
```

The communication error number tells about the communication state of each device, the master as well as the slaves. It gives detailed information about current faults that occurs during communication. This information will be hand out by each device as a hexadecimal code that needs to be decoded to interpret it’s meaning. Reading the Property CommErno of an EtherCAT device affects that the hexadecimal code will be decoded into a readable string, depending on the language which is selected by the system.

Once, the code must be extended, the complete change can be done inside the EcatDevice function block without any adaption inside the EcatMaster nor the EcatSlave.

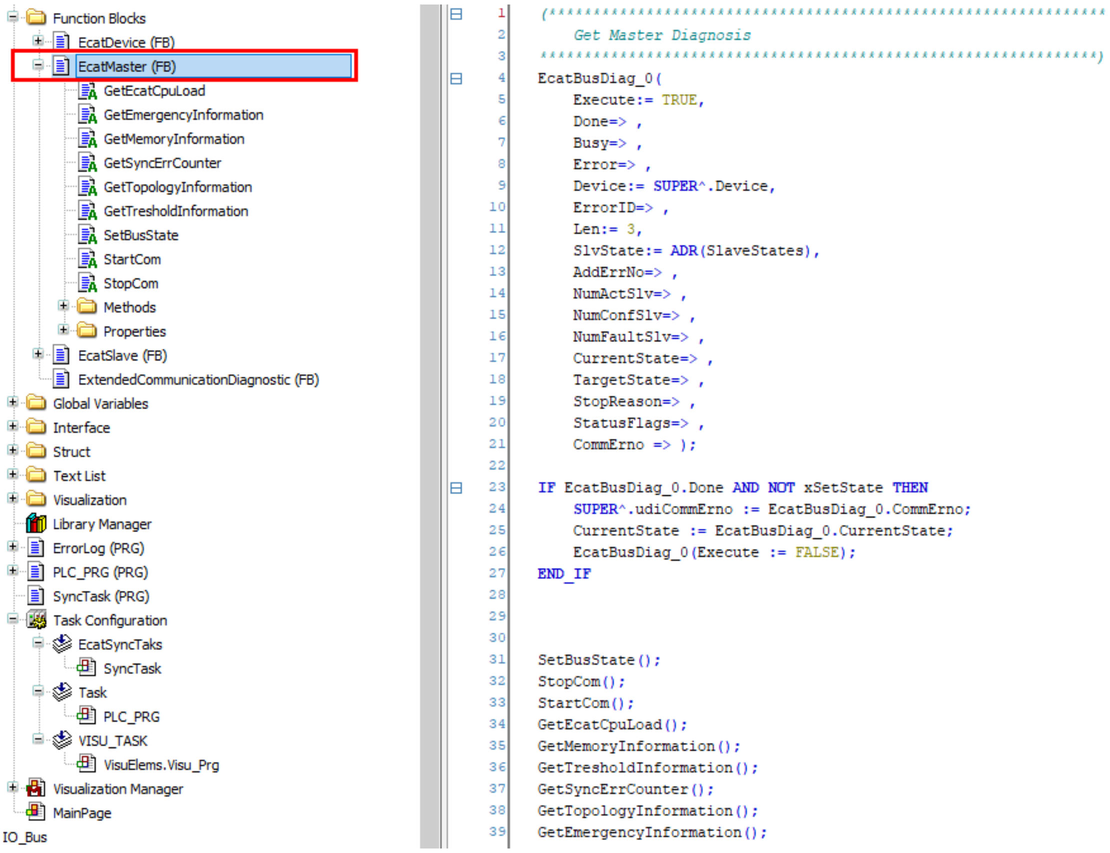
4.2.4.2 EcatMaster (FB)

At the initialization of the Application, the CM579-ETHCAT module will be assigned to the EcatMaster function block accordingly.



```
EcatMaster_0(Device := CM579_ETHCAT);
```

Inside the function block, the EcatBusDiag (Library – AC500_EtherCAT) will be called frequently to get the actual information about the Bus state.

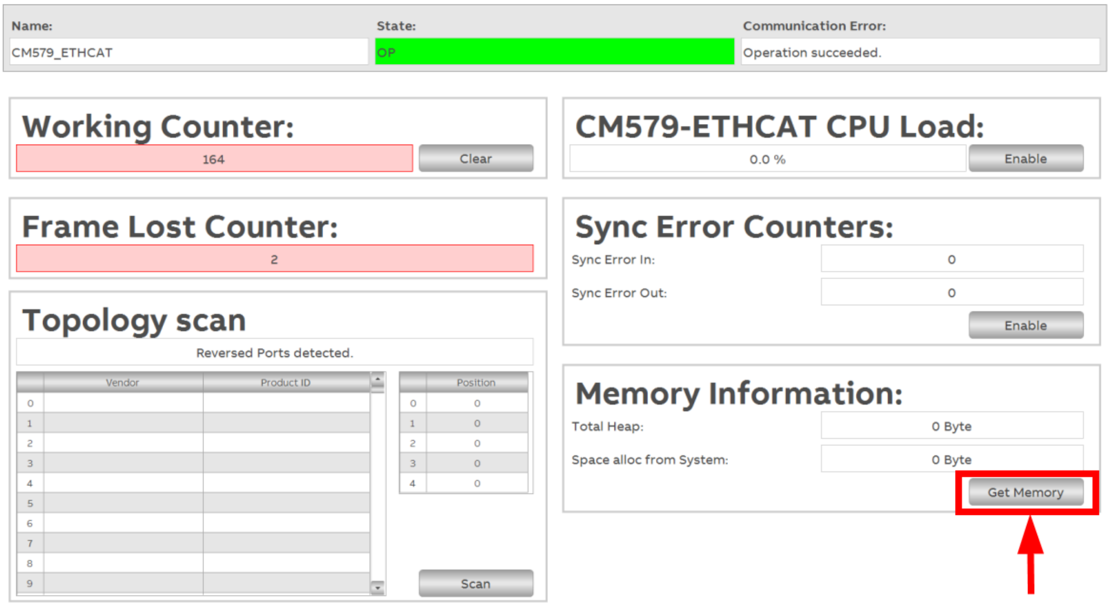


```
(****************************************************************
    Get Master Diagnosis
****************************************************************)
EcatBusDiag_0(
    Execute:= TRUE,
    Done=> ,
    Busy=> ,
    Error=> ,
    Device:= SUPER^.Device,
    ErrorID=> ,
    Len:= 3,
    SlvState:= ADR(SlaveStates),
    AddErrNo=> ,
    NumActSlv=> ,
    NumConfSlv=> ,
    NumFaultSlv=> ,
    CurrentState=> ,
    TargetState=> ,
    StopReason=> ,
    StatusFlags=> ,
    CommErno => );

IF EcatBusDiag_0.Done AND NOT xSetState THEN
    SUPER^.udiCommErno := EcatBusDiag_0.CommErno;
    CurrentState := EcatBusDiag_0.CurrentState;
    EcatBusDiag_0(Execute := FALSE);
END_IF



SetBusState();
StopCom();
StartCom();
GetEcatCpuLoad();
GetMemoryInformation();
GetTresholdInformation();
GetSyncErrCounter();
GetTopologyInformation();
GetEmergencyInformation();
```

All other diagnosis information will be queried on demand. For this, the methods will trigger the enable buttons, while the diagnosis function block will be called inside the Action routines.

E.g. the user requires the memory information of the CM579-ETHCAT master module:

- The user clicks on the button **Get Memory** at the page BusDiagnosis of the Application



Name:	State:	Communication Error:
CM579_ETHCAT	OP	Operation succeeded.

Working Counter: 164

CM579-ETHCAT CPU Load: 0.0 %

Frame Lost Counter: 2

Sync Error In: 0

Sync Error Out: 0

Reversed Ports detected.

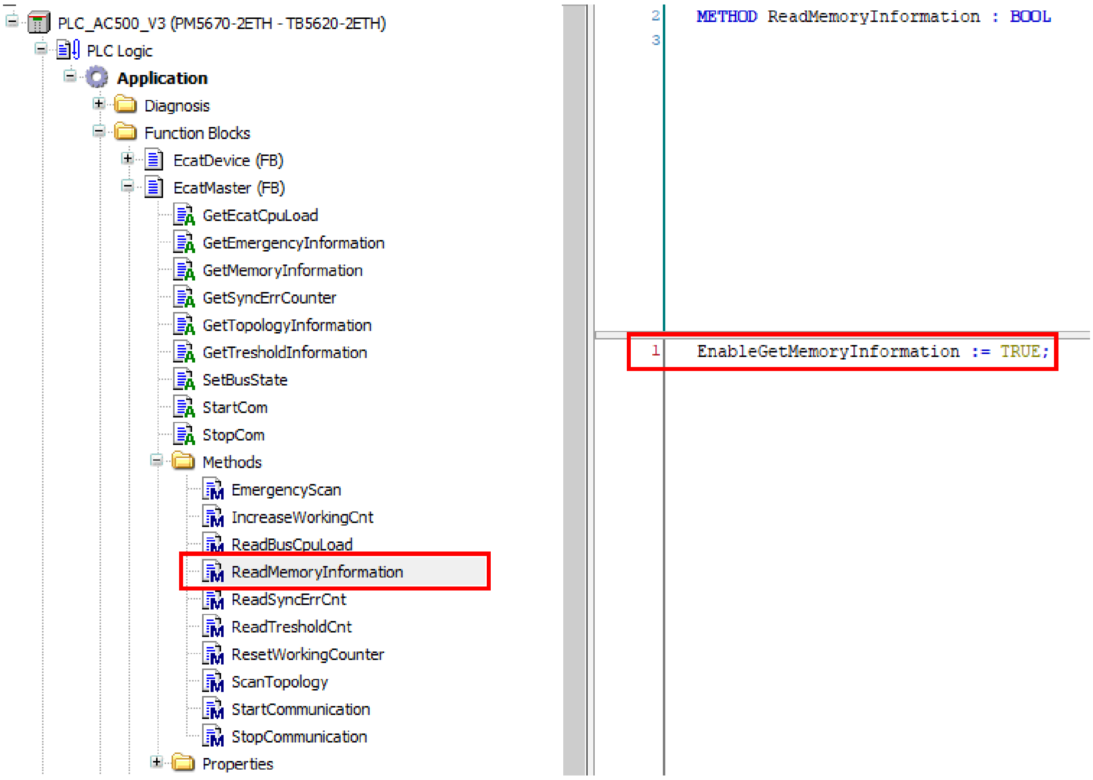
	Vendor	Product ID
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

	Position
0	0
1	0
2	0
3	0
4	0

Total Heap: 0 Byte

Space alloc from System: 0 Byte

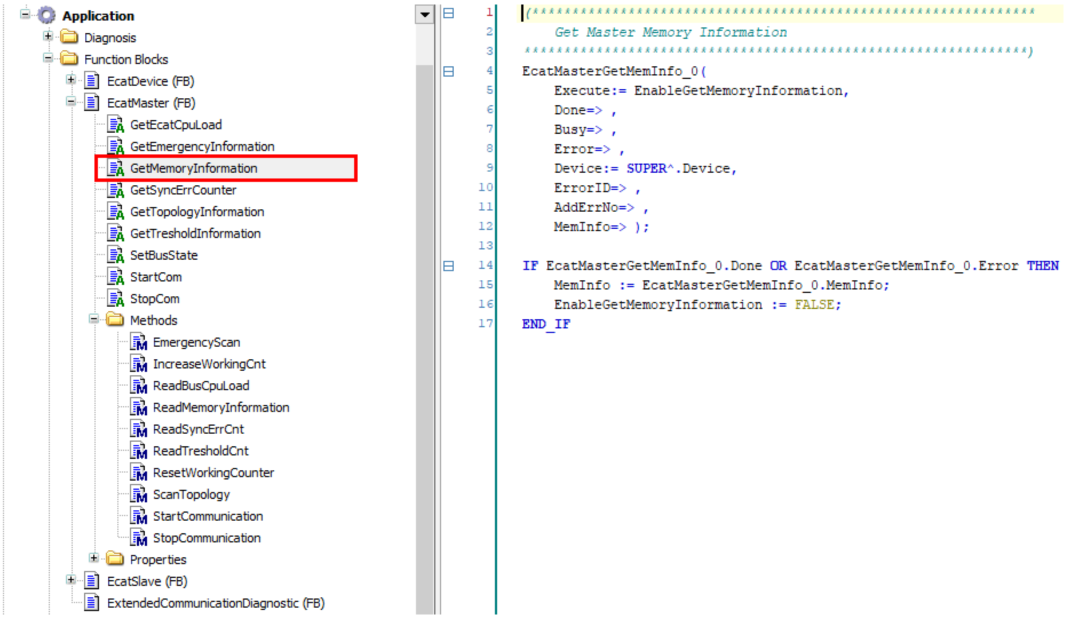
- This button will run the Method **ReadMemoryInformation** of the EcatMaster function block and the **EnableGetMemoryInformation** will be set to True



```
METHOD ReadMemoryInformation : BOOL
```

```
EnableGetMemoryInformation := TRUE;
```

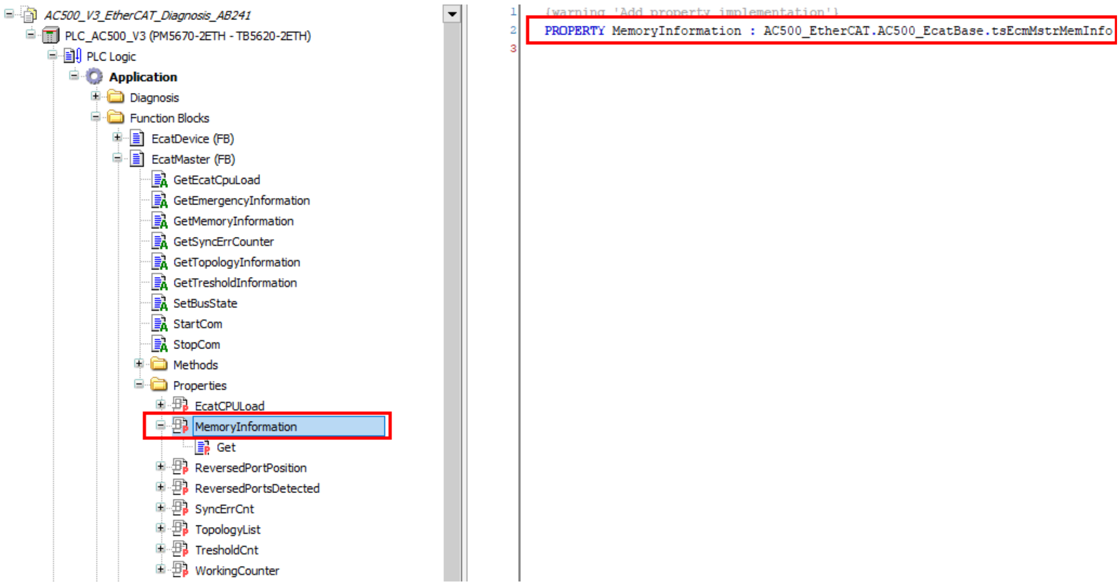
- Inside the Action **GetMemoryInformation** this **EnableGetMemoryInformation** executes the EcatMasterGetMemInfo to receive the desired information



```
(****************************************************************
    Get Master Memory Information
****************************************************************)
EcatMasterGetMemInfo_0(
    Execute:= EnableGetMemoryInformation,
    Done=> ,
    Busy=> ,
    Error=> ,
    Device:= SUPER^.Device,
    ErrorID=> ,
    AddErrNo=> ,
    MemInfo=> );

IF EcatMasterGetMemInfo_0.Done OR EcatMasterGetMemInfo_0.Error THEN
    MemInfo := EcatMasterGetMemInfo_0.MemInfo;
    EnableGetMemoryInformation := FALSE;
END_IF
```

Most of the Properties have only a Get() Accessor to make the values visible inside the visualization or to use them inside the application. The values in general are set by the diagnostic of each device and therefore, they are set inside the function block EcatMaster.

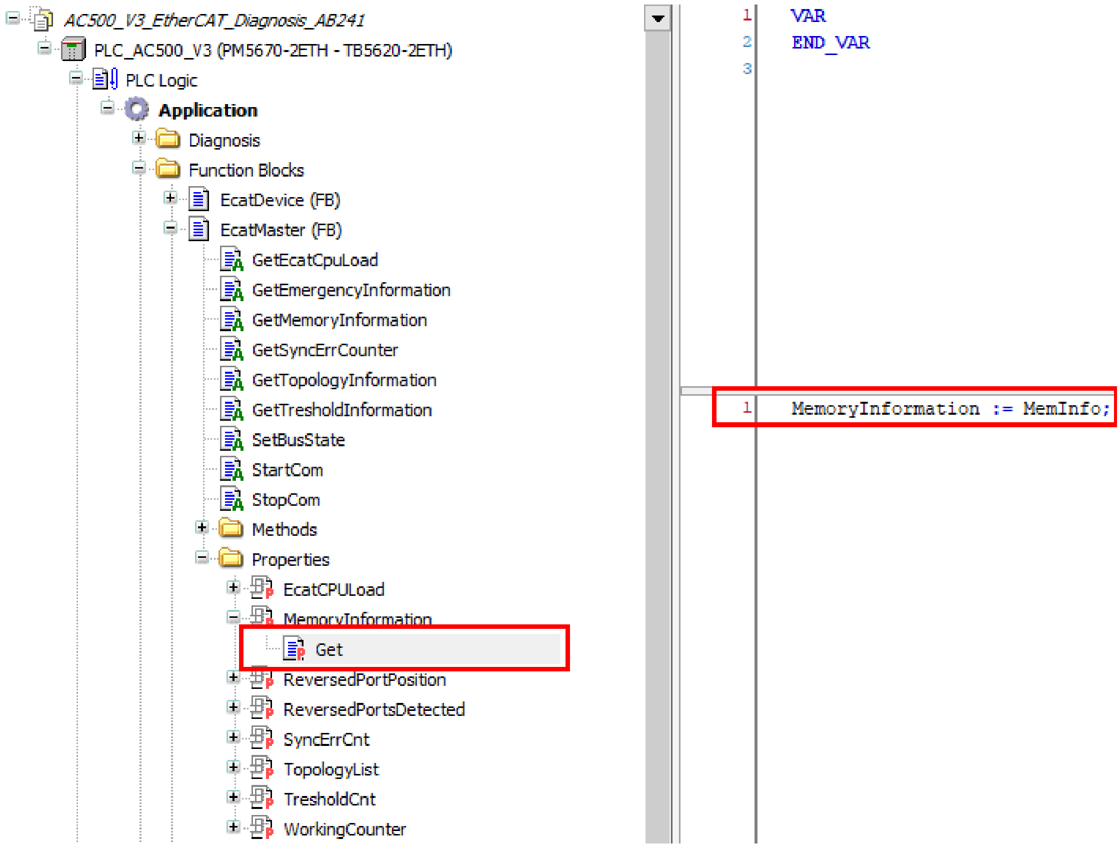
From the above example, where the Memory Information of the master is requested, the result is a STRUCT of the type **tsEcmMstrMemInfo** which will be then stored in an internal variable MemInfo.



```
{warning 'Add property implementation'}
PROPERTY MemoryInformation : AC500_EtherCAT.AC500_EcatBase.tsEcmMstrMemInfo
```

The Property **MemoryInformation** consists of the same type of STRUCT and as soon as the Property is called to print this value inside the visualization, the **MemInfo** will be assigned to the **MemoryInfo**.

This is then coded inside the Get() Accessor routine.



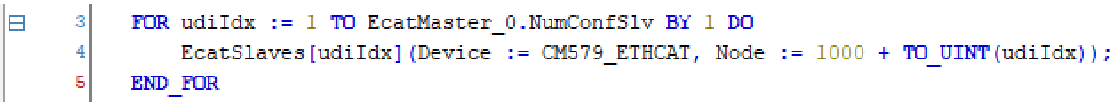
```
VAR
END_VAR
```

```
MemoryInformation := MemInfo;
```

4.2.4.3 EcatSlave (FB)

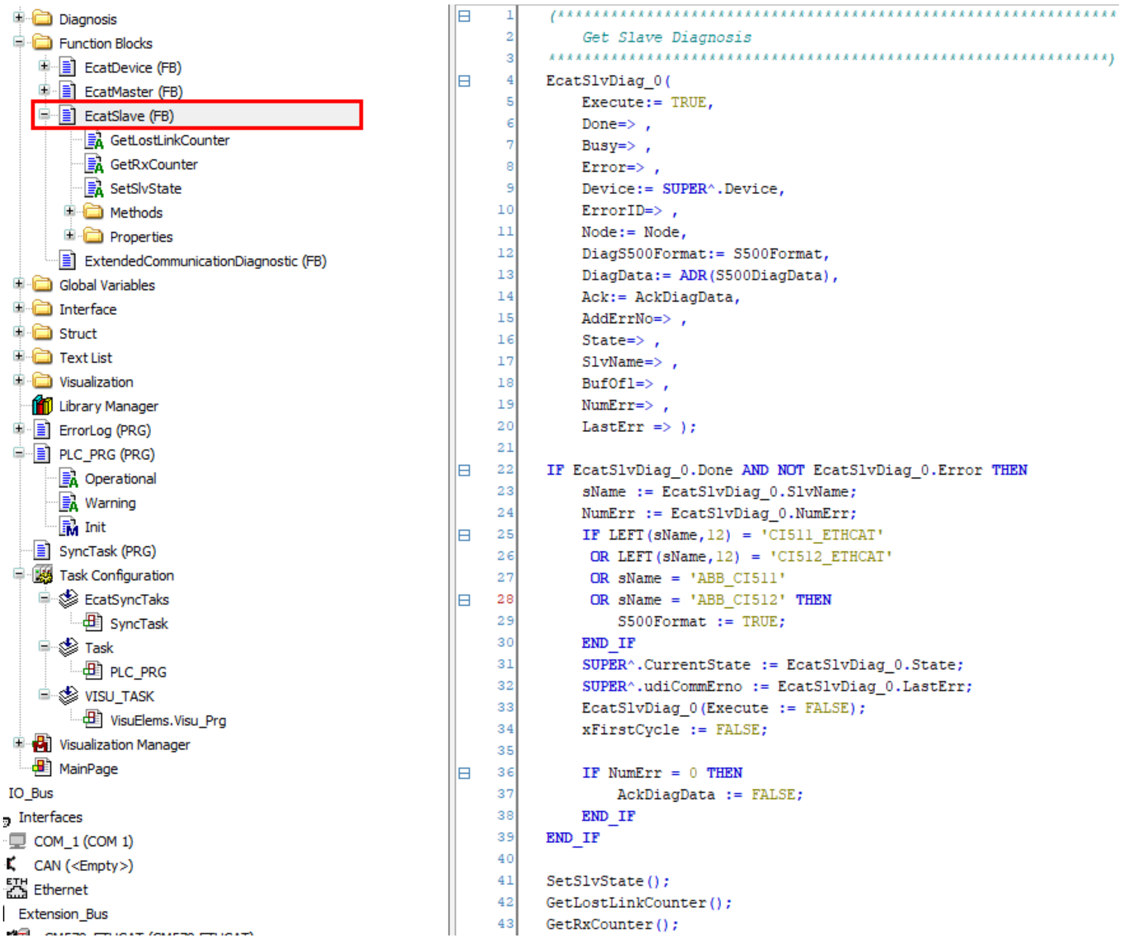
Like the EcatMaster, the EcatSlaves will be initialized at the beginning with the interface for the CM579-ETHCAT module and the node address of each slave.

Depending on the Number of configured Slaves, the number of EcatSlave instances will be created.



```
FOR udiIdx := 1 TO EcatMaster_0.NumConfSlv BY 1 DO
    EcatSlaves[udiIdx](Device := CM579_ETHCAT, Node := 1000 + TO_UINT(udiIdx));
END_FOR
```

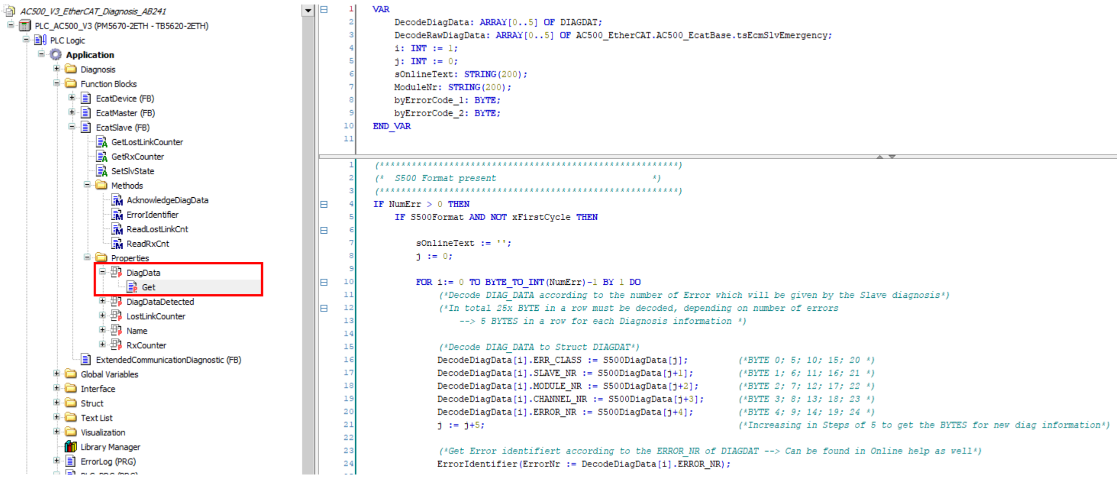
Again, the diagnosis information of the slaves will be queried frequently by the EcatSlvDiag to always receive the latest states.



```
(***************************************************************
    Get Slave Diagnosis
***************************************************************)
EcatSlvDiag_0(
    Execute:= TRUE,
    Done=> ,
    Busy=> ,
    Error=> ,
    Device:= SUPER^.Device,
    ErrorID=> ,
    Node:= Node,
    DiagS500Format:= S500Format,
    DiagData:= ADR(S500DiagData),
    Ack:= AckDiagData,
    AddErrNo=> ,
    State=> ,
    SlvName=> ,
    BufOfl=> ,
    NumErr=> ,
    LastErr => );

IF EcatSlvDiag_0.Done AND NOT EcatSlvDiag_0.Error THEN
    sName := EcatSlvDiag_0.SlvName;
    NumErr := EcatSlvDiag_0.NumErr;
    IF LEFT(sName,12) = 'CI511_ETHCAT'
     OR LEFT(sName,12) = 'CI512_ETHCAT'
     OR sName = 'ABB_CI511'
     OR sName = 'ABB_CI512' THEN
        S500Format := TRUE;
    END_IF
    SUPER^.CurrentState := EcatSlvDiag_0.State;
    SUPER^.udiCommErno := EcatSlvDiag_0.LastErr;
    EcatSlvDiag_0(Execute := FALSE);
    xFirstCycle := FALSE;

    IF NumErr = 0 THEN
        AckDiagData := FALSE;
    END_IF
END_IF

SetSlvState();
GetLostLinkCounter();
GetRxCounter();
```

For the EcatSlvDiag there is a special diagnosis information that gives emergency message for Modules and Channels faults and errors. These emergency messages are supplier-dependent and needs to be decoded according to the manual.

The input DiagData is an Array of Byte that contains the emergency messages. This array and the meaning of each Byte is decoded to a readable String inside the property DiagData.

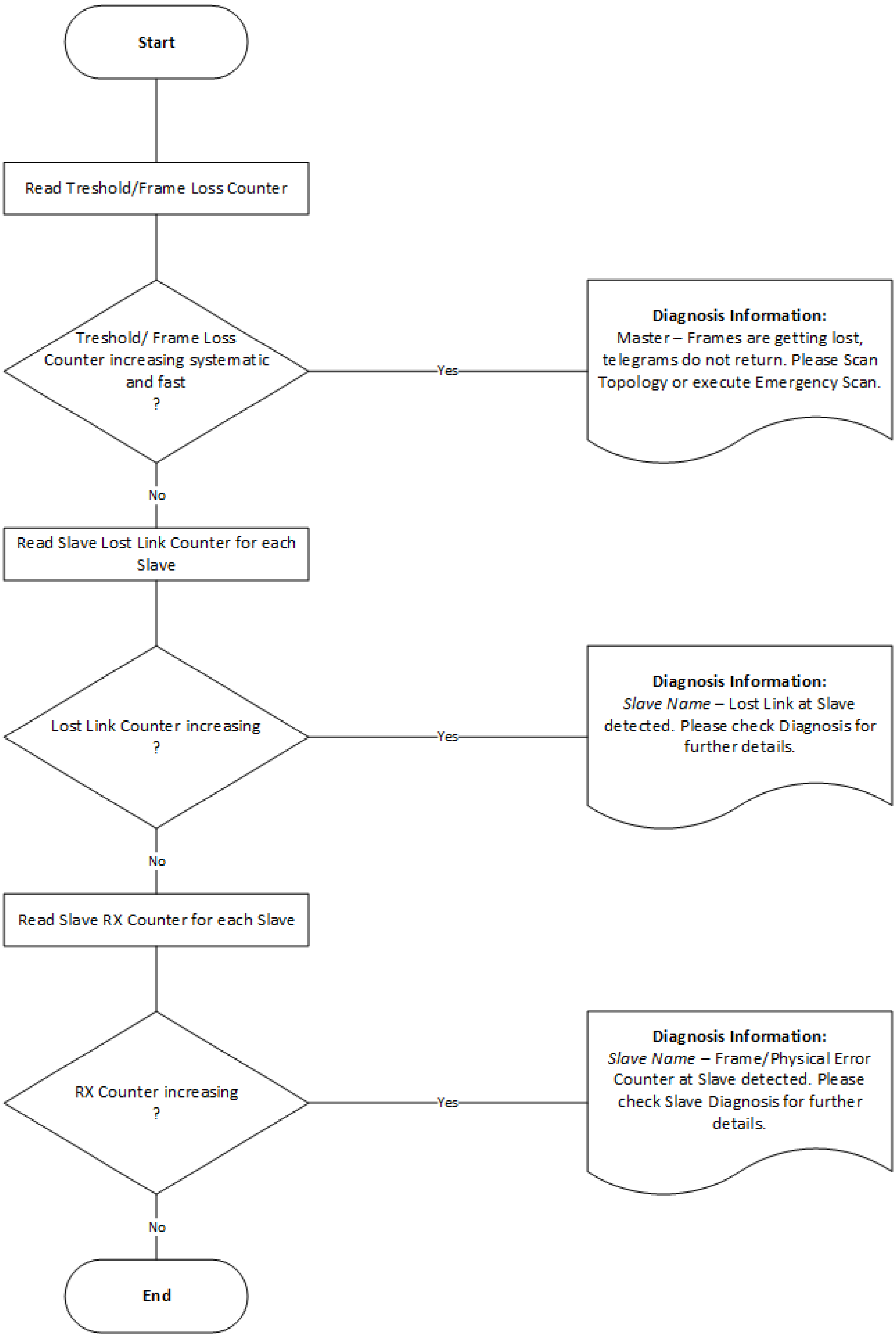


```
VAR
    DecodeDiagData: ARRAY[0..5] OF DIAGDAT;
    DecodeRawDiagData: ARRAY[0..5] OF AC500_EtherCAT.AC500_EcatBase.tsEcmSlvEmergency;
    i: INT := 1;
    j: INT := 0;
    sOnlineText: STRING(200);
    ModuleNr: STRING(200);
    byErrorCode_1: BYTE;
    byErrorCode_2: BYTE;
END_VAR
```

```
(*******************************************************)
(*  S500 Format present                              *)
(*******************************************************)
IF NumErr > 0 THEN
    IF S500Format AND NOT xFirstCycle THEN

        sOnlineText := '';
        j := 0;

        FOR i:= 0 TO BYTE_TO_INT(NumErr)-1 BY 1 DO
            (*Decode DIAG_DATA according to the number of Error which will be given by the Slave diagnosis*)
            (*In total 25x BYTE in a row must be decoded, depending on number of errors
                --> 5 BYTES in a row for each Diagnosis information *)

            (*Decode DIAG_DATA to Struct DIAGDAT*)
            DecodeDiagData[i].ERR_CLASS := S500DiagData[j];         (*BYTE 0; 5; 10; 15; 20 *)
            DecodeDiagData[i].SLAVE_NR := S500DiagData[j+1];        (*BYTE 1; 6; 11; 16; 21 *)
            DecodeDiagData[i].MODULE_NR := S500DiagData[j+2];       (*BYTE 2; 7; 12; 17; 22 *)
            DecodeDiagData[i].CHANNEL_NR := S500DiagData[j+3];      (*BYTE 3; 8; 13; 18; 23 *)
            DecodeDiagData[i].ERROR_NR := S500DiagData[j+4];        (*BYTE 4; 9; 14; 19; 24 *)
            j := j+5;                                               (*Increasing in Steps of 5 to get the BYTES for new diag information*)

            (*Get Error identifiert according to the ERROR_NR of DIAGDAT --> Can be found in Online help as well*)
            ErrorIdentifier(ErrorNr := DecodeDiagData[i].ERROR_NR);
```

4.2.4.4 ExtendedCommunicationDiagnostic (FB)

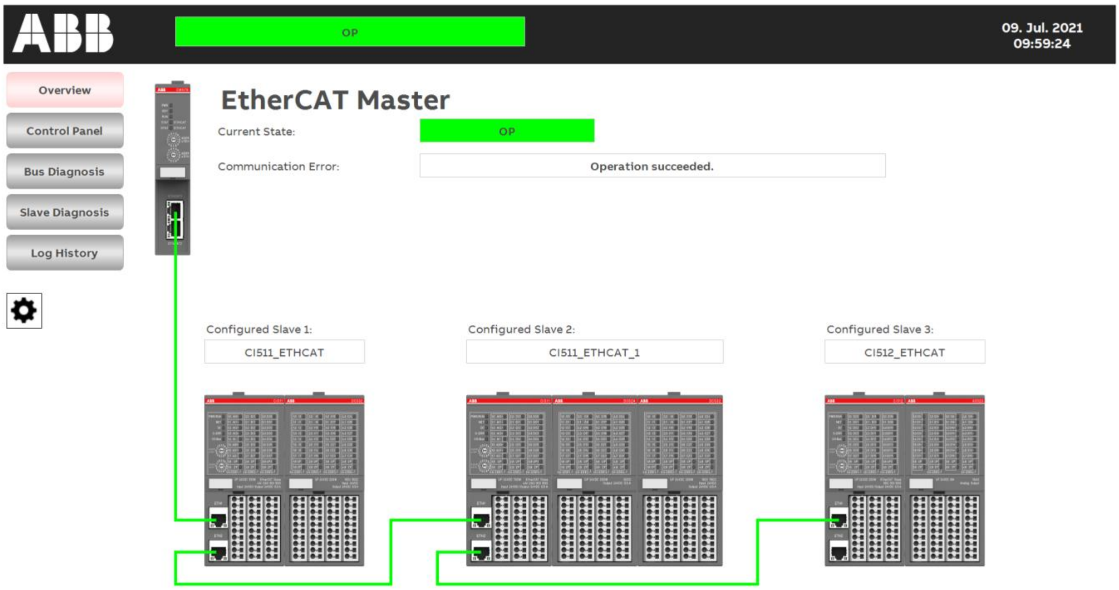
The ExtendedCommunicationDiagnostic function block will be executed in any communication error scenario to get deeper information about the fault of the system. In any case that the Error property of an EtherCAT device is set to true, the function block will run through a pattern of diagnostic to best determine the root cause of the error. This pattern is made by the following flow chart:


Start
Read Treshold/Frame Loss Counter
Treshold/ Frame Loss Counter increasing systematic and fast ?
Yes
Diagnosis Information:
Master – Frames are getting lost, telegrams do not return. Please Scan Topology or execute Emergency Scan.
No
Read Slave Lost Link Counter for each Slave
Lost Link Counter increasing ?
Yes
Diagnosis Information:
Slave Name – Lost Link at Slave detected. Please check Diagnosis for further details.
No
Read Slave RX Counter for each Slave
RX Counter increasing ?
Yes
Diagnosis Information:
Slave Name – Frame/Physical Error Counter at Slave detected. Please check Slave Diagnosis for further details.
No
End

4.3 Visualization

4.3.1 Main Page

The Main Page of the example gives a graphical overview about the actual state of the complete EtherCAT system.



09. Jul. 2021
09:59:24

EtherCAT Master

Current State: OP

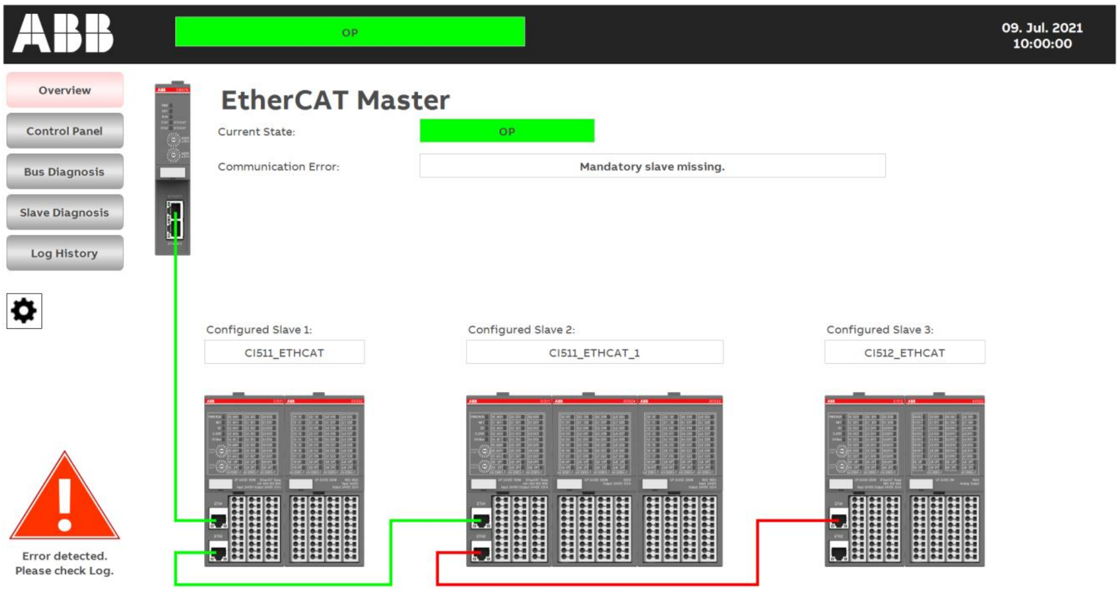
Communication Error: Operation succeeded.

Configured Slave 1: CI511_ETHCAT

Configured Slave 2: CI511_ETHCAT_1

Configured Slave 3: CI512_ETHCAT

General faults and errors, like disconnected cables or missing slaves, can be detected and localized at the Main Page directly.



09. Jul. 2021
10:00:00

EtherCAT Master

Current State: OP

Communication Error: Mandatory slave missing.

Configured Slave 1: CI511_ETHCAT

Configured Slave 2: CI511_ETHCAT_1

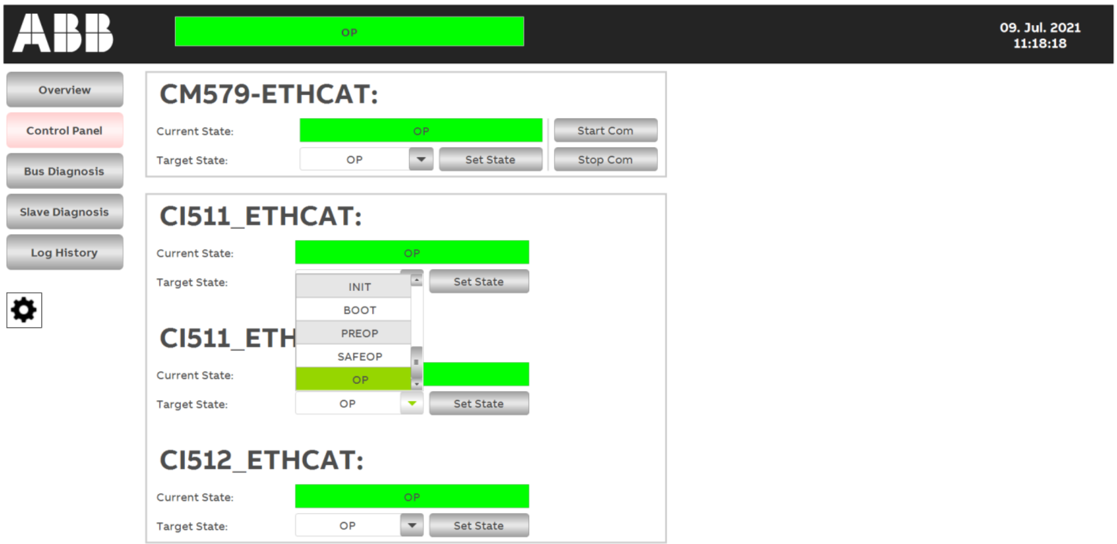
Configured Slave 3: CI512_ETHCAT

Error detected.
Please check Log.

4.3.2 Control Panel

At the Control Panel, the state of the complete EtherCAT bus or just single devices can be controlled by selecting a target state via the combobox and continue with the “Set State” button.

For the CM579-ETHCAT, the Communication can be started and stopped as well. Stopping the Communication will disable the data transfer and stop the machinery.



OP

09. Jul. 2021
11:18:18

CM579-ETHCAT:

Current State: OP

Target State: OP

CI511_ETHCAT:

Current State: OP

Target State:

CI511_ETH

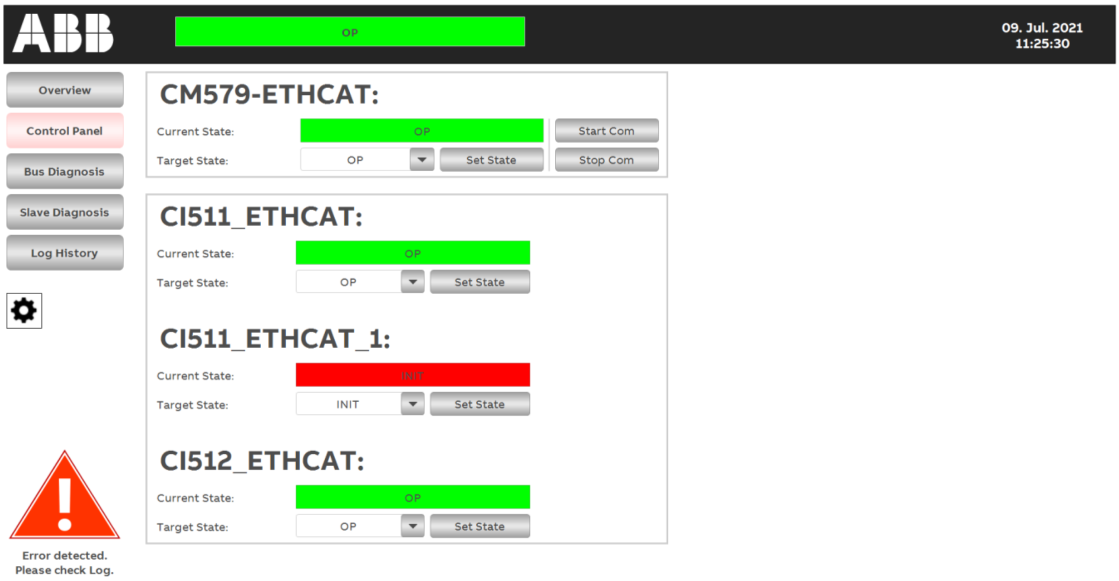
Current State:

Target State: OP

CI512_ETHCAT:

Current State: OP

Target State: OP



OP

09. Jul. 2021
11:25:30

CM579-ETHCAT:

Current State: OP

Target State: OP

CI511_ETHCAT:

Current State: OP

Target State: OP

CI511_ETHCAT_1:

Current State: INIT

Target State: INIT

CI512_ETHCAT:

Current State: OP

Target State: OP

Error detected.
Please check Log.

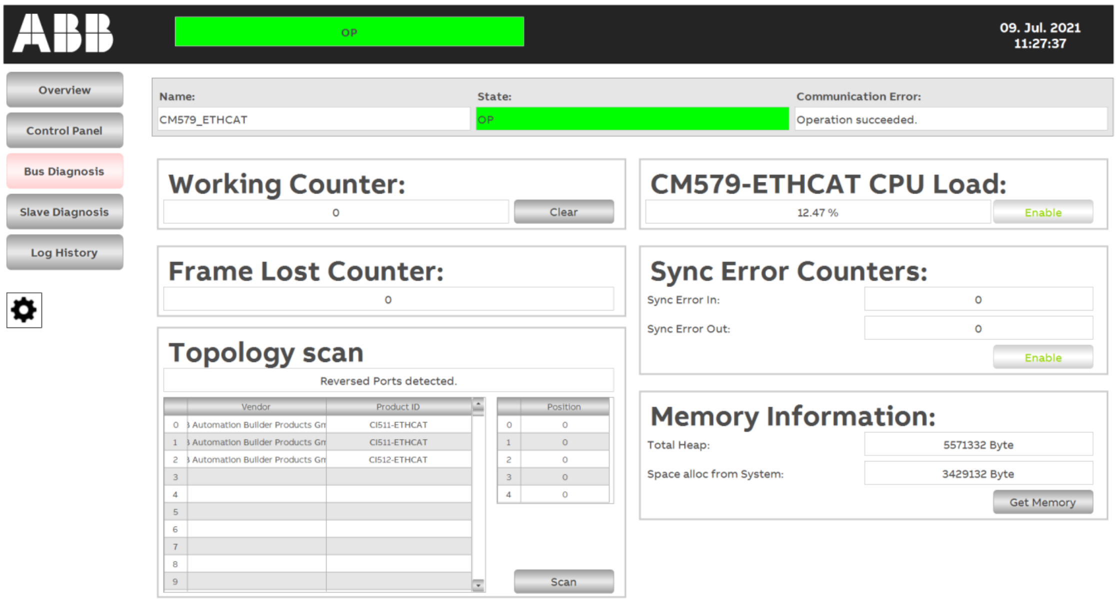
CAUTION!

Start and Stop of the Communication should never used during Operational of a Bus. This might cause hardware failure.

4.3.3 Bus Diagnosis

Inside the Bus Diagnosis detailed information about bus specific diagnosis are given, e.g. the Master Memory, bus CPU load or Synchronization.

The load and memory of the CM579-ETHCAT Master will not be monitored constantly and needs to be enabled on demand. Same for the Topology scan, it is executable only in Init state and should only be used if the master detects a Topology mismatch.



OP

09. Jul. 2021
11:27:37

Name: CM579_ETHCAT
State: OP
Communication Error: Operation succeeded.

Working Counter: 0

CM579-ETHCAT CPU Load: 12.47 %

Frame Lost Counter: 0

Sync Error Counters:
Sync Error In: 0
Sync Error Out: 0

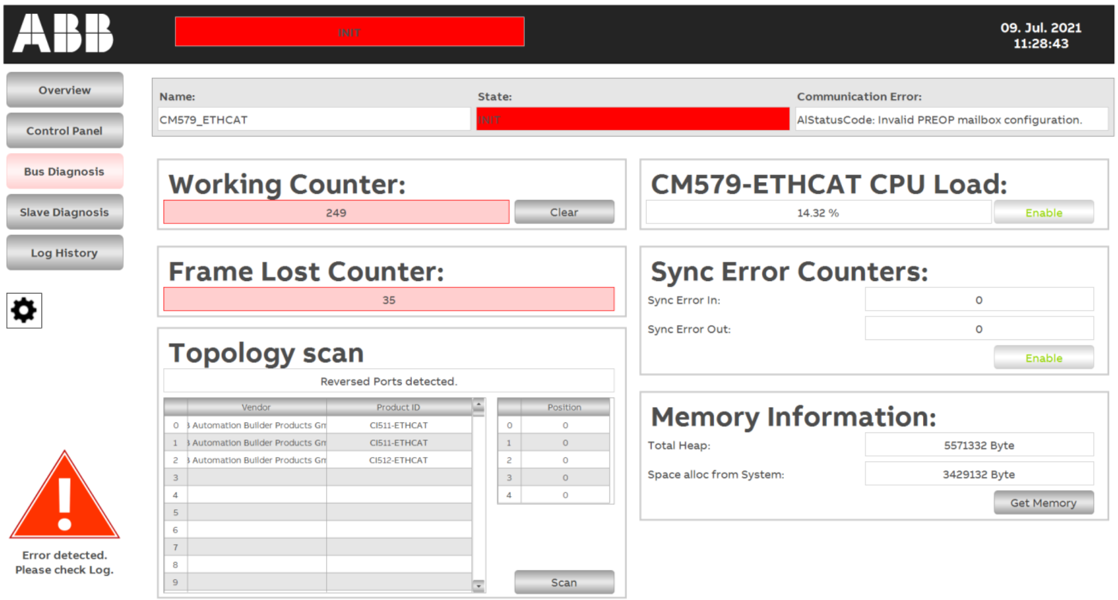
Topology scan

Reversed Ports detected.

	Vendor	Product ID
0	Automation Builder Products Gm	CI511-ETHCAT
1	Automation Builder Products Gm	CI511-ETHCAT
2	Automation Builder Products Gm	CI512-ETHCAT
3		
4		
5		
6		
7		
8		
9		

	Position
0	0
1	0
2	0
3	0
4	0

Memory Information:
Total Heap: 5571332 Byte
Space alloc from System: 3429132 Byte



INIT

09. Jul. 2021
11:28:43

Name: CM579_ETHCAT
State: INIT
Communication Error: AlStatusCode: Invalid PREOP mailbox configuration.

Working Counter: 249

CM579-ETHCAT CPU Load: 14.32 %

Frame Lost Counter: 35

Sync Error Counters:
Sync Error In: 0
Sync Error Out: 0

Topology scan

Reversed Ports detected.

	Vendor	Product ID
0	Automation Builder Products Gm	CI511-ETHCAT
1	Automation Builder Products Gm	CI511-ETHCAT
2	Automation Builder Products Gm	CI512-ETHCAT
3		
4		
5		
6		
7		
8		
9		

	Position
0	0
1	0
2	0
3	0
4	0

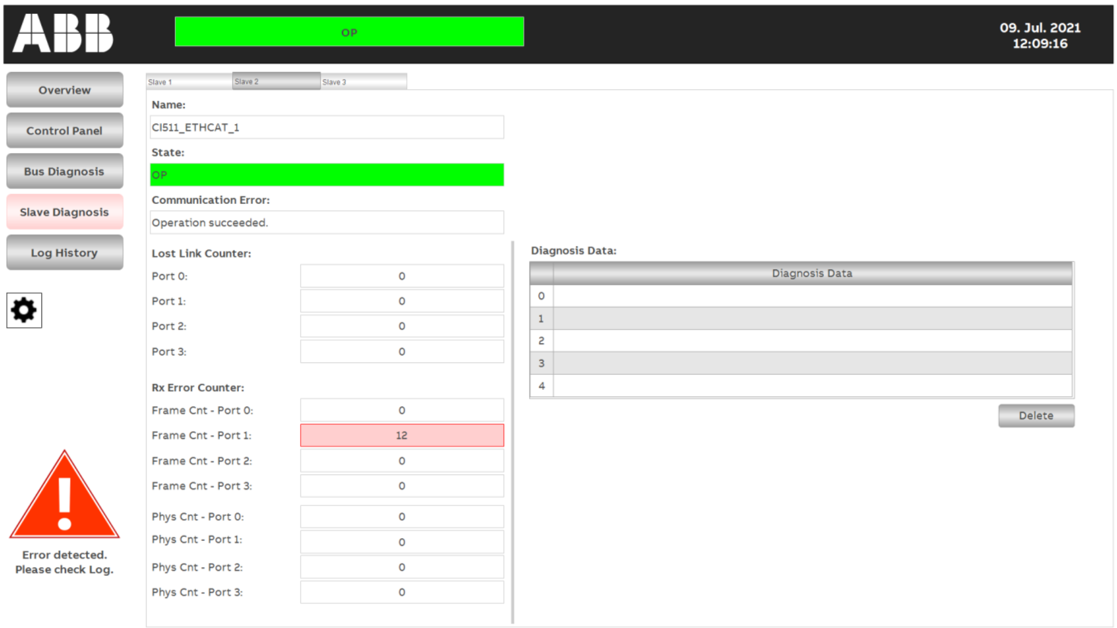
Memory Information:
Total Heap: 5571332 Byte
Space alloc from System: 3429132 Byte

Error detected.
Please check Log.

4.3.4 Slave Diagnosis

Like Bus Diagnosis, the register Slave Diagnosis gives detailed information about the Slave states, their error counters and diagnosis data of the Input and Output faults.

Lost Link and Rx Error counters will be displayed and visualized for each slave.



Name:
CI511_ETHCAT_1

State:
OP

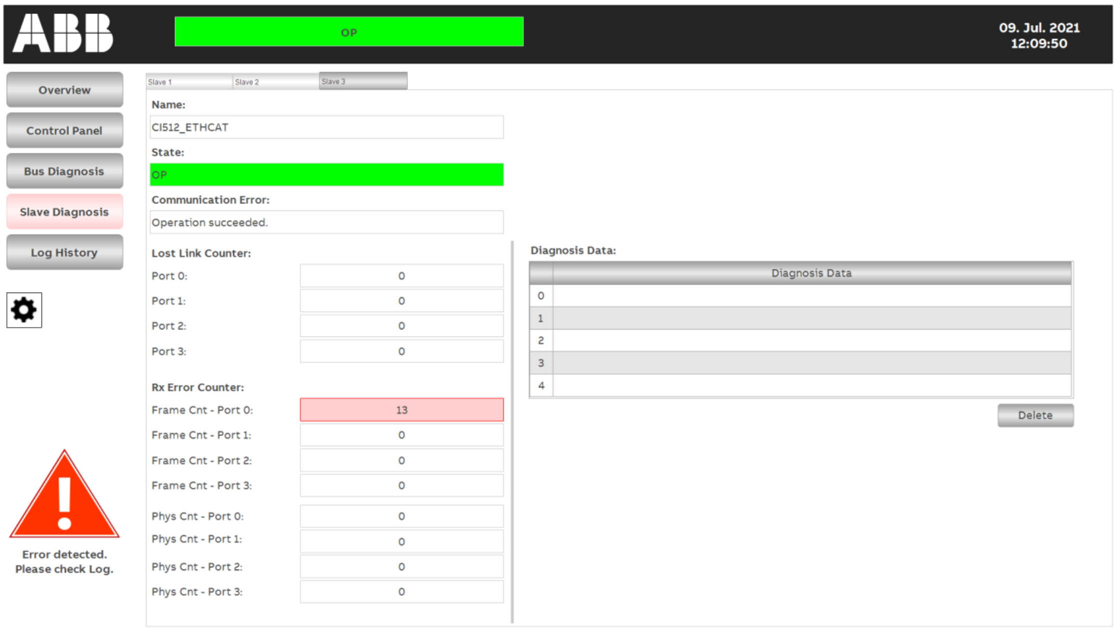
Communication Error:
Operation succeeded.

Lost Link Counter:
Port 0: 0
Port 1: 0
Port 2: 0
Port 3: 0

Rx Error Counter:
Frame Cnt - Port 0: 0
Frame Cnt - Port 1: 12
Frame Cnt - Port 2: 0
Frame Cnt - Port 3: 0
Phys Cnt - Port 0: 0
Phys Cnt - Port 1: 0
Phys Cnt - Port 2: 0
Phys Cnt - Port 3: 0

Diagnosis Data:

Error detected.
Please check Log.



Name:
CI512_ETHCAT

State:
OP

Communication Error:
Operation succeeded.

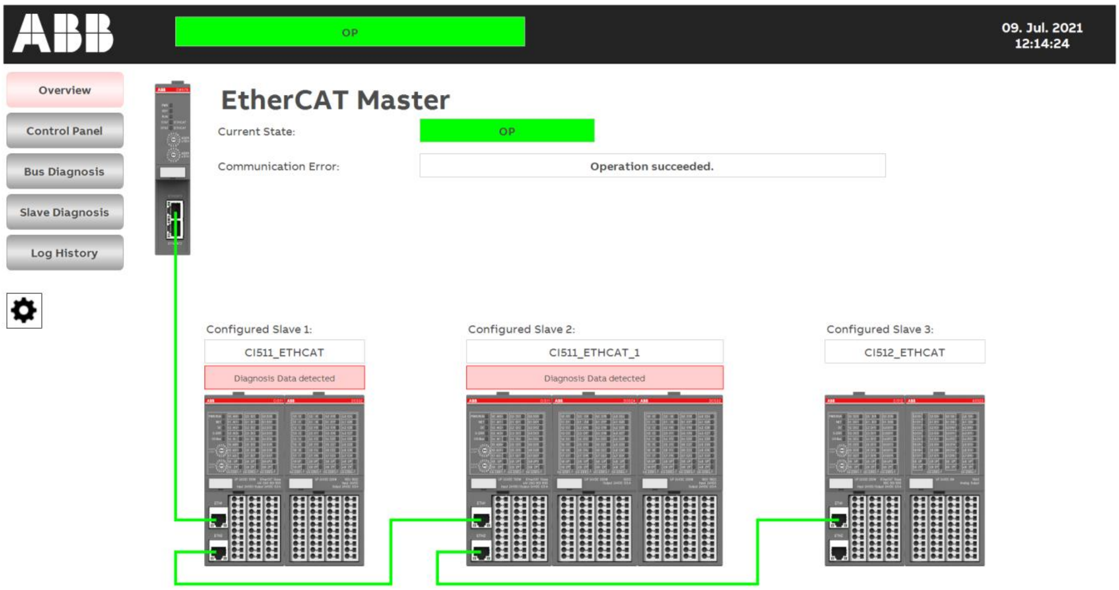
Lost Link Counter:
Port 0: 0
Port 1: 0
Port 2: 0
Port 3: 0

Rx Error Counter:
Frame Cnt - Port 0: 13
Frame Cnt - Port 1: 0
Frame Cnt - Port 2: 0
Frame Cnt - Port 3: 0
Phys Cnt - Port 0: 0
Phys Cnt - Port 1: 0
Phys Cnt - Port 2: 0
Phys Cnt - Port 3: 0

Diagnosis Data:

Error detected.
Please check Log.

If an error at a remote IO Module of an EtherCAT slave occurs, a warning information is shown at the overview page.



OP

09. Jul. 2021
12:14:24

EtherCAT Master

Current State: OP

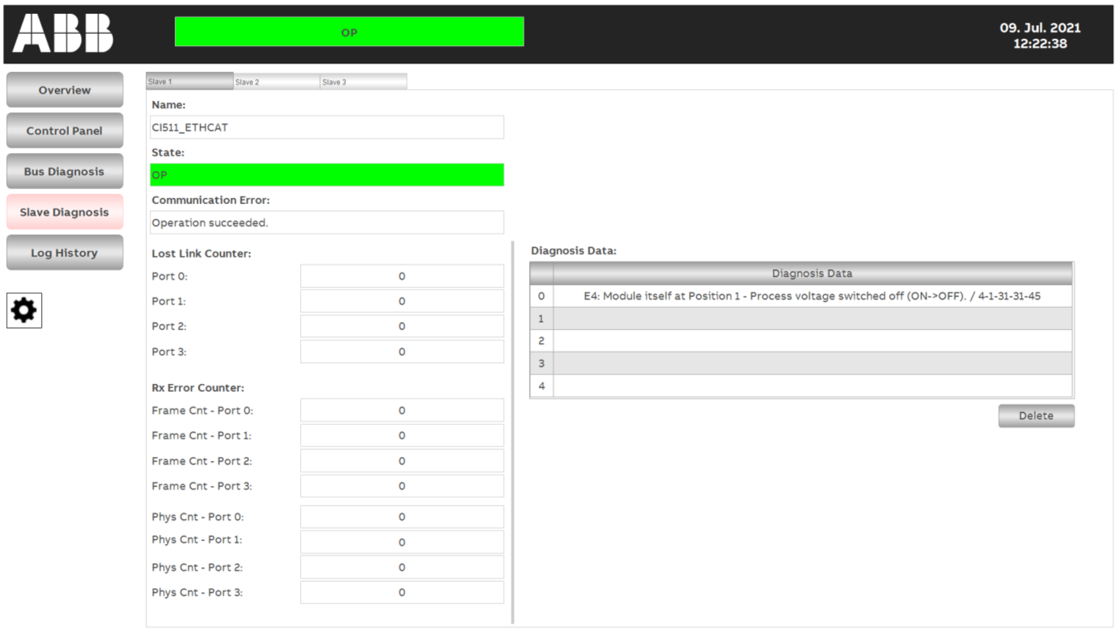
Communication Error: Operation succeeded.

Configured Slave 1: CI511_ETHCAT
Diagnosis Data detected

Configured Slave 2: CI511_ETHCAT_1
Diagnosis Data detected

Configured Slave 3: CI512_ETHCAT

Further diagnosis information can be found at the slave diagnosis page where the error is pending. The table Diagnosis Data shows the decoded Emergency Message which is sent by the slave.



OP

09. Jul. 2021
12:22:38

Name: CI511_ETHCAT

State: OP

Communication Error: Operation succeeded.

Lost Link Counter:

Port 0: 0
Port 1: 0
Port 2: 0
Port 3: 0

Rx Error Counter:

Frame Cnt - Port 0: 0
Frame Cnt - Port 1: 0
Frame Cnt - Port 2: 0
Frame Cnt - Port 3: 0

Phys Cnt - Port 0: 0
Phys Cnt - Port 1: 0
Phys Cnt - Port 2: 0
Phys Cnt - Port 3: 0

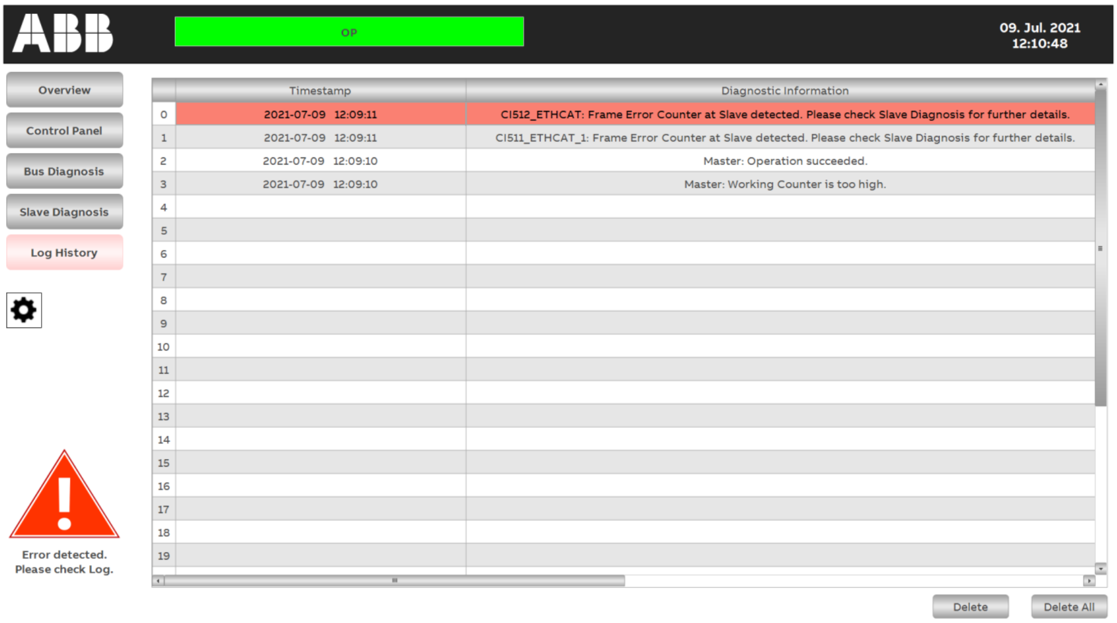
Diagnosis Data:

	Diagnosis Data
0	E4: Module itself at Position 1 - Process voltage switched off (ON->OFF). / 4-1-31-31-45
1	
2	
3	
4	

4.3.5 Log Entries

A history Log about the current faults and errors can be found at the “Log History” page. As long as an error is pending, the Error Warning is blinking.

New incoming Log Entries will be listed at the top of the table.



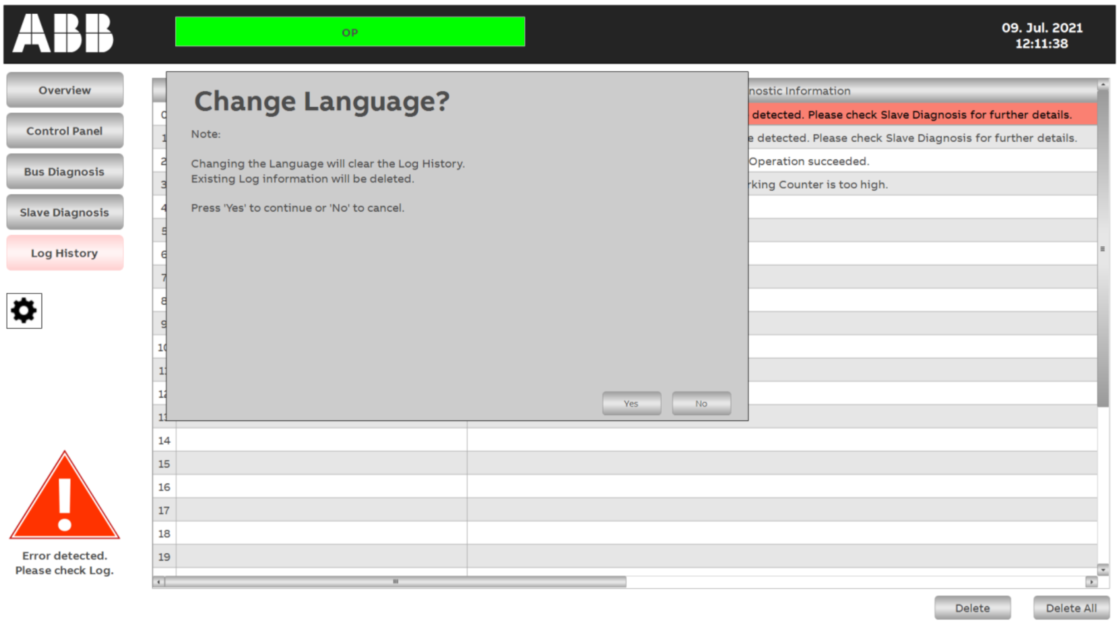
	Timestamp	Diagnostic Information
0	2021-07-09 12:09:11	CI512_ETHCAT: Frame Error Counter at Slave detected. Please check Slave Diagnosis for further details.
1	2021-07-09 12:09:11	CI511_ETHCAT_1: Frame Error Counter at Slave detected. Please check Slave Diagnosis for further details.
2	2021-07-09 12:09:10	Master: Operation succeeded.
3	2021-07-09 12:09:10	Master: Working Counter is too high.
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		

Note: The warning Information is blinking as long as an entry persists at the Log History table. Deleting the Log History table will also reset the Working counter.

4.3.6 Settings

On the left side below the navigation buttons, the language can be changed by clicking on the gear wheel.

An information is shown that changing the language will clear the current Log History table. Continue with “Yes” to change the language or click “No” to cancel.



Change Language?

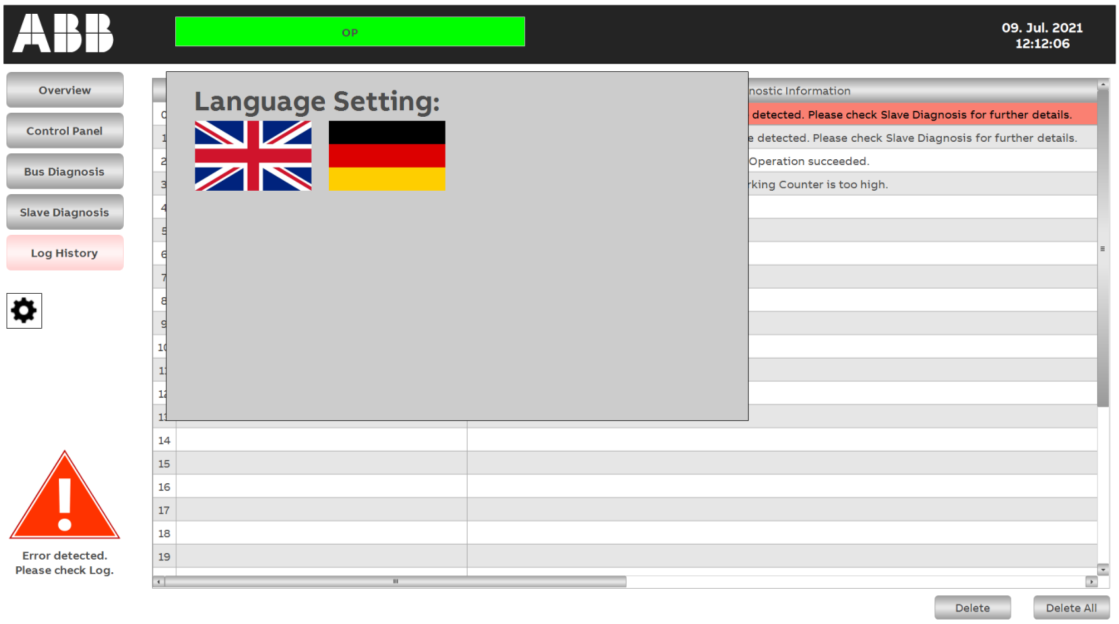
Note:

Changing the Language will clear the Log History.
Existing Log information will be deleted.

Press 'Yes' to continue or 'No' to cancel.

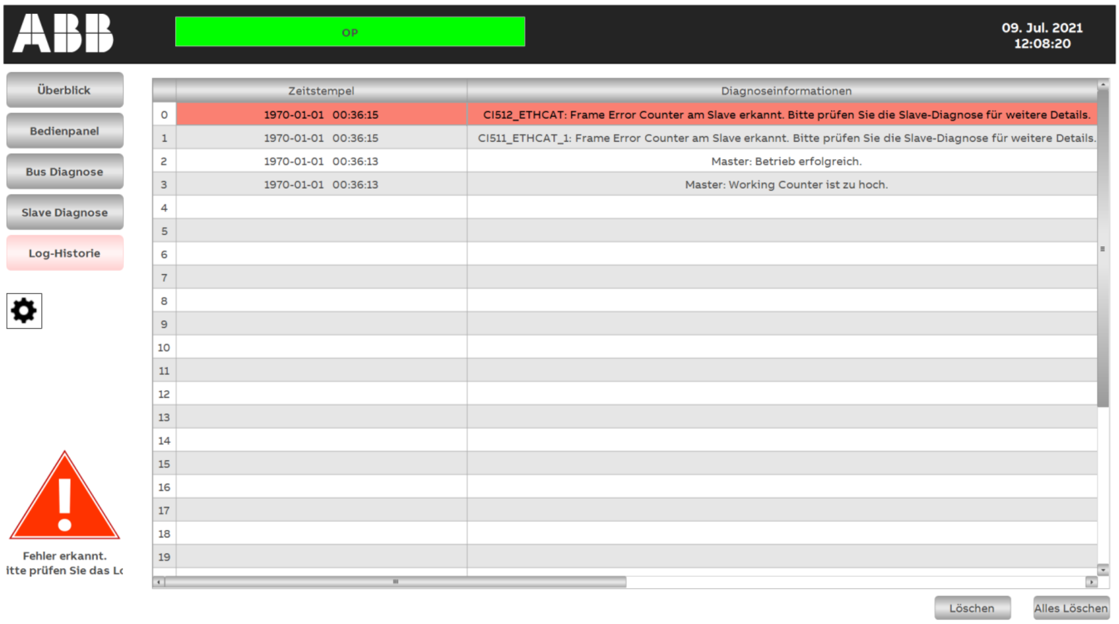
nostic Information
detected. Please check Slave Diagnosis for further details.
e detected. Please check Slave Diagnosis for further details.
Operation succeeded.
king Counter is too high.

Currently there are two available languages – English and German. Selecting one of these languages will close the setting window and change the complete text at the visualization will be changed accordingly.



Language Setting:

nostic Information
detected. Please check Slave Diagnosis for further details.
e detected. Please check Slave Diagnosis for further details.
Operation succeeded.
king Counter is too high.



	Zeitstempel	Diagnoseinformationen
0	1970-01-01 00:36:15	CI512_ETHCAT: Frame Error Counter am Slave erkannt. Bitte prüfen Sie die Slave-Diagnose für weitere Details.
1	1970-01-01 00:36:15	CI511_ETHCAT_1: Frame Error Counter am Slave erkannt. Bitte prüfen Sie die Slave-Diagnose für weitere Details.
2	1970-01-01 00:36:13	Master: Betrieb erfolgreich.
3	1970-01-01 00:36:13	Master: Working Counter ist zu hoch.
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		

Fehler erkannt.
itte prüfen Sie das L

ABB Automation Products GmbH
Eppelheimer Straße 82
69123 Heidelberg, Germany
Phone: +49 62 21 701 1444
Fax: +49 62 21 701 1382
E-Mail: plc.support@de.abb.com
www.abb.com/plc