Motion Control Products

Application note
Accessing drive error data via EtherCAT

AN00252

Rev B (EN)

Access MicroFlex e190 and MotiFlex e180 drive error description and error code at run-time via EtherCAT and ready-made function block for Automation Builder

Introduction

AC500 PLCs (PM585 and PM59x) can be used to perform real-time motion control of ABBs EtherCAT enabled servo drives. In most applications it will be necessary to retrieve error information from one or more axes when an axis fault occurs. Instead of consuming Process Data Object (PDO) mappings to achieve this, which would unnecessarily consume available EtherCAT cycle time, this data can be read from the drives using Service Data Object (SDO) access instead.

This application note details the available error objects and includes an export file for a simple to use function block that can be re-used in EtherCAT motion applications using the PS552-MC motion control libraries. A sample Automation Builder project is included to further illustrate the use of this function block.

Pre-requisites

You will need to have the following to work through this application note:

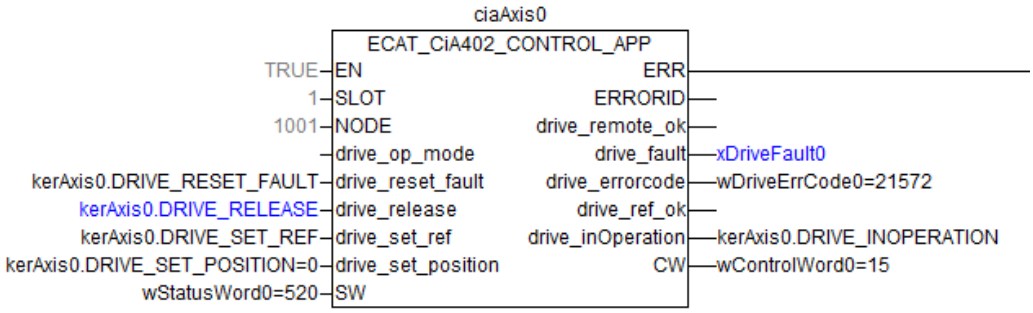
- Mint Workbench build 5812 or later (see new.abb.com/motion for latest downloads and support information)
- A MicroFlex e190 or MotiFlex e180 drive with build 5863 or later firmware
- A PC or laptop running Automation Builder 1.2 or later
- An installed copy of the ABB PLCopen motion control library (PS552-MC-E v3.2.0 or later)
- The SDO access export file from application note AN00242 (included with this application note too)
- Servo drive package for Automation Builder (try to use the latest version from the website)
- One of the following AC500 PLC processors…..PM585, PM590, PM591, PM592 or PM595 (PLC processors should be running firmware version 2.5.1 or later). The PM595 is provided with an integrated EtherCAT coupler (this should be running firmware version 4.2.32.2 or later). All other processors require a CM579-ECAT communication module (which must be running firmware version 2.6.9 or later, but ideally version 4.3.0.2 or later). Contact your local ABB PLC support team for details on how to check these requirements and update if necessary or visit http://new.abb.com/plc/programmable-logic-controllers-plcs and select the link for ‘Software’. For the purposes of the text in this application note we have assumed the use of a PM591 PLC with CM579-ETHCAT coupler
- Ethernet cable to connect the EtherCAT coupler to the drive

To follow the basic steps to create example code to read drive error data only requires a PC or laptop running Automation Builder 1.2 or later and an installed copy of the PS552-MC-E motion control libraries and the servo drive package (version 1.2.4.1 or later if using 5863 firmware). It is assumed the reader has a basic working knowledge of Mint Workbench, Automation Builder, CoDeSys and the AC500 PLC and that the reader has read and understood the contents of application note AN00205, which is also available for download from new.abb.com/motion, and has commissioned an EtherCAT based servo drive (MicroFlex e190 or MotiFlex e180 for example) ready for use with the AC500 PLC.

This application note includes the Mint servo drives package file suitable for use with 5863 firmware for convenience.

Available error code

The ECAT_CIA402_CONTROL_APP function block provided as part of the PS552-MC motion control library provides two outputs that will indicate some error information…



The drive_fault output will become TRUE in the event of a drive error occurring. At the same time drive_errorcode will report the appropriate DS402 error code. The table below (extracted from the Mint Help file) shows the list of possible errors and their associated DS402 and Mint error codes…

DS 402 error code	DS 402 description	Mint error code	Mint description
0x2310	Continuous over current	10014	Over current
0x2350	Load level fault (I2t, thermal state)	10011	Drive Overload
0x3110	Mains over voltage	10016	Bus over voltage
0x3120	Mains under voltage	10017	Bus under voltage
0x3130	Phase failure	10029	Supply phase loss
0x4210	Excess temperature device	10019	Motor temperature input
0x4310	Excess temperature drive	30001	Drive over-temperature
0x4320	Too low temperature drive	30029	Drive under-temperature
0x5110	Supply low voltage	30000	Internal power supply loss
0x5114	U4 = manufacturer specific	10023	Encoder supply lost
0x5400	Power section	10012	Power base not ready
0x5410	Output stages	10013	Power module fault
0x5441	Contact 1 - Manufacturer specific	10010	Drive Enable Input Inactive
0x5442	Contact 2 - Manufacturer specific	10001	Forward Hardware Limit
0x5443	Contact 3 - Manufacturer specific	10002	Reverse Hardware Limit
0x5444	Contact 4 - Manufacturer specific	10033	Safe Torque Off is active
0x5445	Contact 5 - Manufacturer specific	10007	Error Input active
0x7303	Manufacturer specific error	10039	Resolver signals lost or incorrect
0x7305	Incremental sensor 1 fault	10022	Encoder signals lost
0x7310	Speed	10015	Over speed
0x7500	Communication	10026	PDO data lost
0x8400	Velocity speed controller	10006	Fatal velocity exceeded
0x8611	Following Error	10005	Following Error
0x8612	Software limits	10003/10004	Fwd/Rev soft limit hit
0xFF00	Manufacturer specific error	10020	Phase search failed
0xFF01	Manufacturer specific error	10031	Heatsink too hot to Phase Search
0xFF02	Manufacturer specific error	10028	Encoder not ready
0xFF03	Manufacturer specific error	10018	Motor overload
0xFF04	Manufacturer specific error	30002	Production data not valid
0xFF05	Manufacturer specific error	10000	Motion aborted
0xFF06	Manufacturer specific error	10034	Safe Torque Off hardware is faulty
0xFF07	Manufacturer specific error	10035	Safe Torque Off inputs not same level
0xFF08	Manufacturer specific error	30009	Internal API error
0xFF09	Manufacturer specific error	10036	Encoder reading wrong
0xFF0A	Manufacturer specific error	20000	Axis has reached FolErrorWarning
0xFF0B	Manufacturer specific error	10038	Encoder battery dead
0xFF0C	Manufacturer specific error	20004	Encoder battery low

0xFF0D	Manufacturer specific error		10040	The DSL encoder is reporting an error
0xFF0E	Manufacturer specific error		10041	Drive output frequency limit exceeded
0xFF0F	Manufacturer specific error		20005	Phase loss detected
0xFF10	Manufacturer specific error		20006	Motor temperature has not been read
0xFF09	Manufacturer specific error			

The screenshot of our CIA402 function block shows an example of the drive reporting error code 21572 (decimal). In hexadecimal this equates to 0x5444, which the table above reveals to be “Contact 4 – Manufacturer specific” as far as DS402 is concerned but which can be decoded as Error Code 10033 – “Safe Torque Off is active” when looking at the equivalent Mint code/description (which is correct, for the example we attempted to enable the drive whilst the STO input was turned off).

Whilst it is possible to use the CIA402 function block to report application errors to the user its functionality is limited. It is necessary to hard code a look-up table to translate the DS402 error codes into meaningful error messages – although this might be a preferred solution if the error data must be presented to the user in a non-English language for example.

The following section details EtherCAT objects available within the drive that are also available to provide error information.

EtherCAT drive error objects

The MicroFlex e190 and MotiFlex e180 drives are provided with two objects that are able to present error data to an EtherCAT master such as the AC500 PLC.

Diagnosis history object
Object 0x10F3 is the standard “Diagnosis History Object” and operates as defined by the EtherCAT Technology Group document 1020 (EtherCAT Protocol Enhancements)…

EtherCAT Summary

Address	Name	Actual
Index: 10F3 - DGN_DiagnosisHistory_REC (22 items)		
10F3:00	DGN_DiagnosisHistory_REC.SubIndex 000	21 (16#15)
10F3:01	DGN_DiagnosisHistory_REC.MaximumMessages_U8	16 (16#10)
10F3:02	DGN_DiagnosisHistory_REC.NewestMessage_U8	19 (16#13)
10F3:03	DGN_DiagnosisHistory_REC.NewestAcknowledgedMessage_U8	18 (16#12)
10F3:04	DGN_DiagnosisHistory_REC.NewMessagesAvailable_BOOL	True
10F3:05	DGN_DiagnosisHistory_REC.Flags_U16	17 (16#0011)
10F3:06	DGN_DiagnosisHistory_REC.DiagnosisMessage_0_OSTR	00E000000202010047DDC8DEEF02000004204178697 31C205361666520546F72717565204F666620696E707
10F3:07	DGN_DiagnosisHistory_REC.DiagnosisMessage_1_OSTR	00E0000002020100AFD04C84F702000004204178697 31C205361666520546F72717565204F666620696E7075
10F3:08	DGN_DiagnosisHistory_REC.DiagnosisMessage_2_OSTR	00E0000002020100A7204EEDF902000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:09	DGN_DiagnosisHistory_REC.DiagnosisMessage_3_OSTR	00E0000002020100559F8C350403000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:0A	DGN_DiagnosisHistory_REC.DiagnosisMessage_4_OSTR	00E0000002020100CAAA85172B030000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:0B	DGN_DiagnosisHistory_REC.DiagnosisMessage_5_OSTR	00E0000002020100D97F4F844D030000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:0C	DGN_DiagnosisHistory_REC.DiagnosisMessage_6_OSTR	00E0000002020100855ADD096903000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:0D	DGN_DiagnosisHistory_REC.DiagnosisMessage_7_OSTR	00E000000202010084AC8D1AA104000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:0E	DGN_DiagnosisHistory_REC.DiagnosisMessage_8_OSTR	00E00000020201008D65B816BA040000042041786973 1C205361666520546F72717565204F666620696E7075
10F3:0F	DGN_DiagnosisHistory_REC.DiagnosisMessage_9_OSTR	00E00000020201005DC7B127C404000004204178697 31C205361666520546F72717565204F666620696E7075
10F3:10	DGN_DiagnosisHistory_REC.DiagnosisMessage_10_OSTR	00E00000020201003DBEDC8DC504000004204178697 31C205361666520546F72717565204F666620696E707
10F3:11	DGN_DiagnosisHistory_REC.DiagnosisMessage_11_OSTR	00E0000002020100B538B355C604000004204178697 31C205361666520546F72717565204F666620696E7075
10F3:12	DGN_DiagnosisHistory_REC.DiagnosisMessage_12_OSTR	00E00000020201008DA4CF22D504000004204178697 31C205361666520546F72717565204F666620696E7075
10F3:13	DGN_DiagnosisHistory_REC.DiagnosisMessage_13_OSTR	00E000000202010055EBDF57D604000004204178697 3222050444F20646174612069732 06E6F742070726573
10F3:14	DGN_DiagnosisHistory_REC.DiagnosisMessage_14_OSTR	00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
10F3:15	DGN_DiagnosisHistory_REC.DiagnosisMessage_15_OSTR	00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

This object will log up to 16 errors in a circular buffer (from subindex 0x06 to 0x15). It can operate in either overwrite mode or acknowledge mode. Acknowledge mode is the default (as bit 4 of subindex 0x05 is set by default to define this).

The PLC application can detect that new errors are available by reading subindex 0x04 (which will return TRUE if there are unacknowledged errors in the history). The PLC application can identify the latest error produced by the drive from subindex 0x02. Errors are acknowledged by writing the subindex of the message concerned to subindex 0x03. So for example, in the screenshot above the last error acknowledged by the PLC is held in subindex 18 (0x12) but it can be seen that the newest message (subindex 0x02) indicates 19 (0x13) and so there is an unacknowledged error available (and hence subindex 0x04 reports TRUE).

The diagnostic messages themselves (subindex 0x06 to 0x15) are encoded as follows (we will use the diagnosis message stored in subindex 0x13 as an example):

- The first 4 bytes are a diagnostic code to indicate what type of message this is. The message is always an 'Emergency error code' and so these bytes are always '00 E0 00 00' (note that these are encoded in little endian format so the actual value is 0x0000E000).
- The next 2 bytes are flags for the message content. These bytes are always '02 02' to indicate the message is an error message with two parameters.
- The next 2 bytes are a text id…. '00 01'…..again these will never change and can effectively be ignored
- The next 8 bytes are a timestamp for when the error occured….'55 EB DF 57 D6 04 00 00' ignoring the endian-ness….these can be ignored as the drive doesn't have a real time clock (RTC), you would use the PLC time (which can use a RTC is a battery is fitted) to timestamp the errors if required
- The next 2 bytes are parameters relating to the error information….'20 04'….bits 12-15 define the data type (2 = string….this will never change).….bits 0-11 define the length (in bytes) of the string that follows (4 bytes). So in this case the next 4 bytes can be decoded as a string….these aren't little endian…..'41 78 69 73' = "Axis" in ASCII…
- The next 2 bytes are parameters relating to the next piece of error information…..'20 22' ….so as before, bits 12-15 define the data type (2 = string….this will never change).….bits 0-11 define the length (in bytes) of the string that follows (22 hex = 34 bytes). If you then look at the next 34 bytes in subindex 0x13 you will see "50 44 4F 20 64 61 74 61 20 69 73 20 6E 6F 74 20 70 72 65 etc…." which in ASCII reads "PDO data is not pre…." If we could have screenshot the whole message this would have read "PDO data is not present (MN to CN)"

Whilst this object is very useful and will allow the PLC application to access error descriptions it is quite complex to decode, will most likely require an additional PDO mapping for each axis (to continually read the status of subindex 0x04 "NewMessagesAvailable") and doesn't provide any information about the error code that the e190 or e180 drive will be flashing via its seven segment display. This object is most likely to be used by the PC based PLC programming tool itself which may be able to more easily decode the object's contents (Note that Automation Builder does not include integrated support to access the Diagnosis History object). The sample project includes with this application note includes an example function block (FBReadDriveDiagHistory) that will decode drive errors as they occur via this object, but it is recommended that for most (if not all) applications an alternative object is used as described in the following paragraphs.

First error object
The first error object (0x4144) is an ABB (manufacturer) specific object that has been included to allow very simple retrieval of error information from the MicroFlex e190 and MotiFlex e180 servo drives….

EtherCAT

Address	Name	Actual
Index: 4144 - MML_FirstError_REC (8 items)		
4144:00	MML_FirstError_REC.SubIndex 000	7 (16#07)
4144:01	MML_FirstError_REC.Code_I32	10033 (16#00002731)
4144:02	MML_FirstError_REC.Group_VS	Axis
4144:03	MML_FirstError_REC.Text_VS	Safe Torque Off input active
4144:04	MML_FirstError_REC.Data_OSTR	00000000
4144:05	MML_FirstError_REC.Line_I32	-1 (16#FFFFFFFF)
4144:06	MML_FirstError_REC.ParamFamily_I16	-1 (16#FFFF)
4144:07	MML_FirstError_REC.ParamIndex_I16	-1 (16#FFFF)

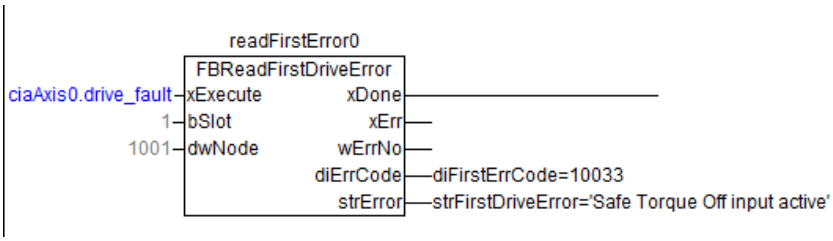
This object stores the Mint (MML) error description and error code (as indicated by the seven segment display on the drive) for the first error detected in sub-indexes of object 0x4144. If multiple errors occur (e.g. loss of an encoder input may also result in an over-speed trip or following error very shortly afterwards) only the first (root cause) error is recorded – this is sufficient for most, if not all, applications.

Subindex 0x03 contains the error description and subindex 0x01 contains the error code (as shown above).

The rising edge (from FALSE to TRUE) of the CIA402 function block 'drive_fault' output can be used to call additional function blocks to make SDO calls to these objects (see application note AN00242 for further information about the use of SDO access via EtherCAT).

This application note includes an export file for a pre-written function block that will return the error description and error code and this is also included in the sample Automation Builder project. This function block in turn makes use of another ABB function block created to simplify SDO reads of 32 bit integer objects. This function block can be included by importing the SDO access export file

from application note AN00242 (this exp file is also included with this application note for convenience). The screenshot below illustrates the typical use of this function block…



The following table details the input and output parameters for FBReadFirstDriveError:

Input parameter	Data type	Description
xExecute	BOOL	Rising edge on this input will cause the function block to attempt to read the drive error code and description
bSlot	BYTE	Slot number of the EtherCAT coupler being used by the PLC
dwNode	DWORD	Node ID for the drive to be accessed
Output parameter	**Data type**	**Description**
xDone	BOOL	Becomes TRUE when the function block completes (successfully or otherwise)
xErr	BOOL	Becomes TRUE if the function block encounters an error trying to read error data from the specified drive
wErrNo	WORD	Provides an error code to explain the reason for the function block returning TRUE on xErr (refer to the Automation Builder Help system for detailed explanations for these error codes
diErrCode	DINT	Provides the drive's (Mint/MML) error code (as will be indicated via the drive's seven segment display)
strError	STRING	This is the error description retrieved from the specified drive

Contact Us

For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC

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