Anybus CompactCom 40 Diagnostic Events for PROFIBUS

APPLICATION NOTE

SCM-1202–025 1.1 ENGLISH

Connecting Devices™

Important User Information

Liability

Every care has been taken in the preparation of this document. Please inform HMS Industrial Networks AB of any inaccuracies or omissions. The data and illustrations found in this document are not binding. We, HMS Industrial Networks AB, reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks AB. HMS Industrial Networks AB assumes no responsibility for any errors that may appear in this document.

There are many applications of this product. Those responsible for the use of this device must ensure that all the necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

HMS Industrial Networks AB will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features, timing, or functional side effects found outside the documented scope of this product. The effects caused by any direct or indirect use of such aspects of the product are undefined, and may include e.g. compatibility issues and stability issues.

The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, HMS Industrial Networks AB cannot assume responsibility for actual use based on these examples and illustrations.

Intellectual Property Rights

HMS Industrial Networks AB has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the USA and other countries.

1 Preface

1.1 About this Document

This application note is intended to provide a description about how diagnostic events are presented in the engineering tool for the industrial network PROFIBUS.

It is divided into two parts:

Part one provides a short overview of the Anybus CompactCom 40 Diagnostic Object (02h) and its diagnostic events. Part two is an example, showing how to get diagnostic messages displayed for PROFIBUS using the PLC engineering tool Siemens SIMATIC STEP 7 V5.5.

1.1.1 Target Audience

This document is meant for trained and skilled personnel working with the equipment described.

You need electrical engineering skills for the installation and commissioning of electrical equipment.

You also need general knowledge of automation and programmable logic controllers, in particular about Siemens SIMATIC software. Additionally, knowledge on the PROFIBUS industrial fieldbus protocol and the Anybus CompactCom object model is necessary.

1.2 History

Revision	Date	Description	Responsible
1.0	2016-10-17	First version	OLB
1.1	2016-12–01	Converted to DOX	KaD

1.3 Referenced Documents

Description	Name / Type	Version
HMS Starter Kit	HMS Development Board 2 Rev 1.07 P/N	0314-1.1.1
Anybus CompactCom 40 module	ABCC-M40-DPV1 P/N	V.1.04
Siemens S7 PLC PROFIBUS	CPU 315-2 PN/DP	V 3.2.10
GSD-file for the Anybus CompactCom ABCC-M40-DPV1	HMS_1815.gsd	1.5
ABCC_40_PROFIBUS_DP-V1	Network Guide	V1.30
ABCC Software Design guide	Software Guide	V3.0
PC with Siemens PLC programming software	Siemens SIMATIC STEP 7	V.5.5 + SP4
How to configure an Anybus Profibus slave module with Siemens Step 7	Application note	2.1
Profile Guidelines, Part 3 Diagnostics, Alarms and Time Stamping	PROFIBUS Network Specification /Profile Guidelines	Version 1.0
IDE	KEIL uVision 5	V5.20
Anybus CompactCom Driver	Anybus CompactCom Host Application Example Code	V2.01

1.4 More Information about Networks and Products

The latest manuals and GSD files can be found on the HMS website, www.anybus.com

The PROFIBUS user organisation has a website on the Internet, www.profibus.org. Several technical guides are available in or via this site.

1.5 Trademark Information

2 General Information

Diagnostics is an important feature in industrial automation, production automation and process automation. Diagnostics helps to manage maintenance, avoid damage and increase the reliability of the installations, reduce downtimes and – at the end — save money.

This document describes how the user of the Anybus CompactCom 40 PROFIBUS DP-V1 can use its features to create valuable diagnostics describing the state of a field device where the CompactCom is implemented.

3 The Diagnostic Object

The Anybus CompactCom concept is based on an object model.

For detailed information about this, please refer to the Anybus CompactCom 40 Software Design Guide.

For creating diagnostics (information from a field device to a PLC) the CompactCom concept contains the diagnostic object (02h) which is located inside the CompactCom. A diagnostic event is created by sending a **create *(03h)*** command to the **diagnostic object (02h)** of the CompactCom.

The Anybus CompactCom PROFIBUS DP-V1 contains an additional diagnostic object called *PROFIBUS DP-V0 Diagnostic Object (10h)* which has to be used if the CompactCom is used as a DP-V0 interface.

Focus of this application note is creating diagnostics using the CompactCom as a DP-V1 interface exclusively.

This application note also describes the extended diagnostics possibilities offered by the CompactCom, for standard implementation without implementing the Modular Device Object.

3.1 Create a Diagnostic Event

At the beginning the diagnostic object (02h) is empty. The process of reporting a diagnostic to the network starts in the host application. For that a ***create (03h)*** event command message is sent from the host application to the diagnostic object (02h).

The CompactCom will internally create a new instance inside the diagnostic object (02h). This new instance corresponds to a **diagnostic event**.

The ***create (03h)*** command must contain info for describing the diagnostic event: a **Severity** (CMDExt[0]) and an **Event Code** (CMDExt[1]). The **severity** indicates how critical the event is and if it will recover by itself or not. CMDExt[0] additionally contains a bit called ***Extended Diagnostic***, which informs the CompactCom if the event message contains additional user specific data. If the ***Extended Diagnostic*** bit is set, the additional user specific data will be found in MsgData[0..7], and the network specific data will follow in MsgData[8..n].

An **Event Code** informs about the nature of the event for example *temperature (40h)*.

Command Details: Create

Details

Command Code: 03h

Valid For: Object

Description

Creates a new instance, in this case representing a new diagnostic event in the host application.

- Command details:

Field	Contents	Note
CMDExt[0]	Bit 0: Extended Diagnostic Bit 4–6: Severity Other bits Reserved. Set to zero.	
CMDExt[1]	Event Code, see previous page	
MsgData[0...1]	Slot number associated with the event Set to “0” if unknown or unsupported	These fields only exist if bit 0 (Extended Diagnostic) is set
MsgData[2...3]	ADI associated with the event Set to “0” if unknown or unsupported	
MsgData[4]	Element associated with the event Set to “255” if unknown or unsupported	
MsgData[5]	Bit in element associated with the event Set to “255”if unknown or unsupported	
MsgData[6...7]	Reserved. Set to zero	
MsgData[0/8...n]	Network specific extension (optional, definition is network specific)	MsgData[8-n] if bit 0 in CmdExt[0] is set MsgData[0-n] if bit 0 in CmdExt[0] is not set

- Response details (Success):

Field	Contents
MsgData[0...1]	The number of the created instance

- Response details (Error):

Error	Contents	Comment
Object Specific Error	MsgData[1] = 02h	Error code (Latching event not supported) The event could not be created since the module does not support latching events
	MsgData[1] = FFh	Error code (Network specific error) The event could not be created due to a network specific reason. Information about the event is found in response MsgData[2-n]

3.2 Severity Codes of Diagnostic Events

The following severity codes are defined for the CompactCom:

Severity

This parameter indicates the severity level of the event. Only bits 4 - 6 are used for severity level information.

Severity Levels

Bit Combination	Severity	Comment
000	Minor, recoverable	-
001	Minor, unrecoverable	Unrecoverable events cannot be deleted
010	Major, recoverable	-
011	Major, unrecoverable	Causes a state-shift to EXCEPTION
101	Minor, latching	
110	Major, latching	
(other)	-	(reserved for future use)

Typically, *recoverable* events are generated by the process e.g. if a temperature exceeds a limit value defined by the device manufacturer (e.g. internal temperature of the device exceeds 50° C). The character of this event is typically a warning when the event is defined as *minor*. The temperature of the device is recoverable as the device can cool down again when some heat producers are cut off.

The device manufacturer will add a *major recoverable* event when he wants to inform the PLC that the temperature has reached a critical high temperature which can damage the device.

An ***unrecoverable*** diagnostic event is typically created when the device detects that a sensor is broken. The sensor has to be replaced, it will - normally – not recover by itself. If the device has only one sensor and this sensor is broken the device will create a ***major unrecoverable*** event. The device has to be stopped and to be repaired (replace the broken sensor) before used again otherwise there will be a big risk that the device will be damaged after restart. If the device has some redundant sensors it will create a ***minor unrecoverable*** event informing the users that the broken sensor should be replaced by another one within the next scheduled inspection.

Minor latching **and** ***major latching*** allow the creation of *latching diagnostic event.*

Latching events trigger alarms on PROFIBUS. See *Diagnostics Alarm, p. 12* for more information. The latching events remain active until explicitly acknowledged by the network master. If the network does not support acknowledgment of latching diagnostic events, the module shall refuse the creation of latching diagnostic events.

When the network master acknowledges one or more latching events, the module shall send a “Reset Diagnostic” request to the application object (FFh). The request contains a list of diagnostic instances which the master wishes to acknowledge. The application object shall respond with a list of diagnostic instances which it allows the module to delete. The module will then delete the allowed instances, and report the appropriate information to the network master.

For more information, see the Software Design Guide and the Network Guide of Anybus CompactCom 40 device.

> **!** The device manufacturer has to define which event will be reported as a diagnostic event to the PLC and which severity has to be used.

3.3 Anybus CompactCom Event Codes

The table below shows a list of the event codes applicable for Anybus CompactCom 40 device.

Event Codes

#	Meaning	Comment
10h	Generic Error	-
20h	Current	-
21h	Current, device input side	-
22h	Current, inside the device	-
23h	Current, device output side	-
30h	Voltage	-
31h	Mains Voltage	-
32h	Voltage inside the device	-
33h	Output Voltage	-
40h	Temperature	-
41h	Ambient Temperature	-
42h	Device Temperature	-
50h	Device Hardware	-
60h	Device Software	-
61h	Internal Software	-
62h	User Software	-
63h	Data Set	-
70h	Additional Modules	-
80h	Monitoring	-
81h	Communication	-
82h	Protocol Error	-
90h	External Error	-
F0h	Additional Functions	-
FFh	NW specific	Definition is network-specific; consult separate network guide for further information.

The event code **FFh** is used if network specific diagnostics are reported (not considered within this application note).

4 PROFIBUS Diagnostics

This section shows how PROFIBUS handles diagnostic data.

PROFIBUS provides two types of diagnostics: ***Standard Diagnostics*** and ***Extended Diagnostics***

Standard Diagnostics provide general information about the device type and a flag when Extended Diagnostics are present (more details see “Standard Diagnostics” below).

Extended Diagnostics report more precisely the origin of the error and the type of error and differs 3 types:

Device Related Diagnostics, Identifier Related Diagnostics, Channel Related Diagnostics

A Channel Related Diagnostics is related to one channel (signal input, or output) of a slave or a module of a slave. Identifier Related Diagnostics is related to modules of a slave.

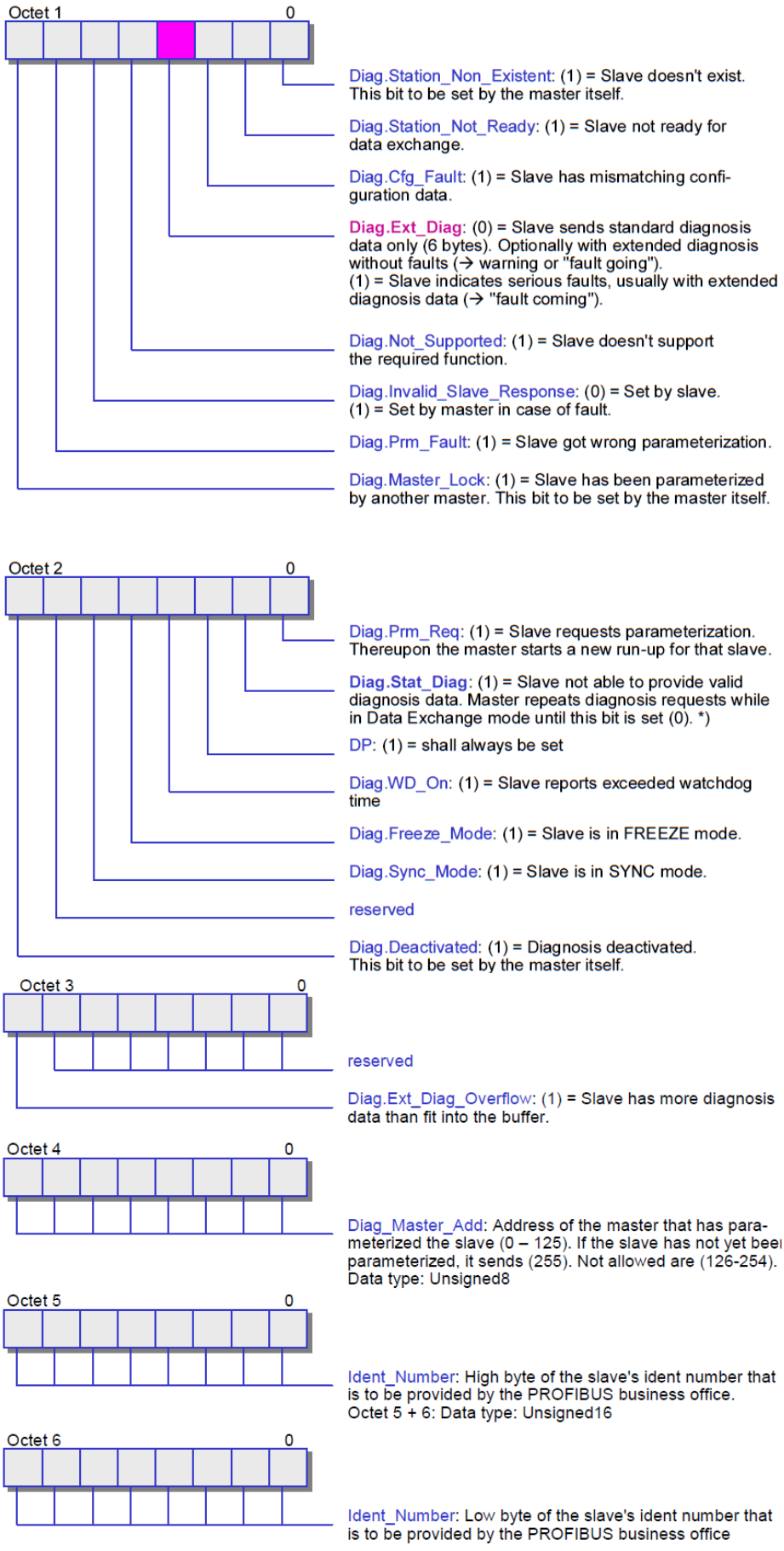
Within the scope of this application note is the ***Device Related Diagnostics***.

Device Related Diagnostics can be **diagnostics alarm** or **status messages** (details see below).

For more details about diagnostics mechanisms of PROFIBUS DP Diagnostics see **PROFIBUS profile guidelines part 3: Diagnostics, Alarms and Time Stamping**.

4.1 Standard Diagnostics

It consists of 6 octets. Their meaning is specified in the standard and is fixed. The diagnostics information is related to the communication layer and covers diagnostics scenarios such as the device identification, communication mode information (FREEZE, SYNC), readiness, availabilities, watchdogs, and parameterization and configuration faults.


Octet 1
0
Diag.Station_Non_Existent: (1) = Slave doesn't exist. This bit to be set by the master itself.
Diag.Station_Not_Ready: (1) = Slave not ready for data exchange.
Diag.Cfg_Fault: (1) = Slave has mismatching configuration data.
Diag.Ext_Diag: (0) = Slave sends standard diagnosis data only (6 bytes). Optionally with extended diagnosis without faults (→ warning or "fault going").
(1) = Slave indicates serious faults, usually with extended diagnosis data (→ "fault coming").
Diag.Not_Supported: (1) = Slave doesn't support the required function.
Diag.Invalid_Slave_Response: (0) = Set by slave.
(1) = Set by master in case of fault.
Diag.Prm_Fault: (1) = Slave got wrong parameterization.
Diag.Master_Lock: (1) = Slave has been parameterized by another master. This bit to be set by the master itself.
Octet 2
0
Diag.Prm_Req: (1) = Slave requests parameterization. Thereupon the master starts a new run-up for that slave.
Diag.Stat_Diag: (1) = Slave not able to provide valid diagnosis data. Master repeats diagnosis requests while in Data Exchange mode until this bit is set (0). *)
DP: (1) = shall always be set
Diag.WD_On: (1) = Slave reports exceeded watchdog time
Diag.Freeze_Mode: (1) = Slave is in FREEZE mode.
Diag.Sync_Mode: (1) = Slave is in SYNC mode.
reserved
Diag.Deactivated: (1) = Diagnosis deactivated. This bit to be set by the master itself.
Octet 3
0
reserved
Diag.Ext_Diag_Overflow: (1) = Slave has more diagnosis data than fit into the buffer.
Octet 4
0
Diag_Master_Add: Address of the master that has parameterized the slave (0 – 125). If the slave has not yet beer parameterized, it sends (255). Not allowed are (126-254). Data type: Unsigned8
Octet 5
0
Ident_Number: High byte of the slave's ident number that is to be provided by the PROFIBUS business office. Octet 5 + 6: Data type: Unsigned16
Octet 6
0
Ident_Number: Low byte of the slave's ident number that is to be provided by the PROFIBUS business office

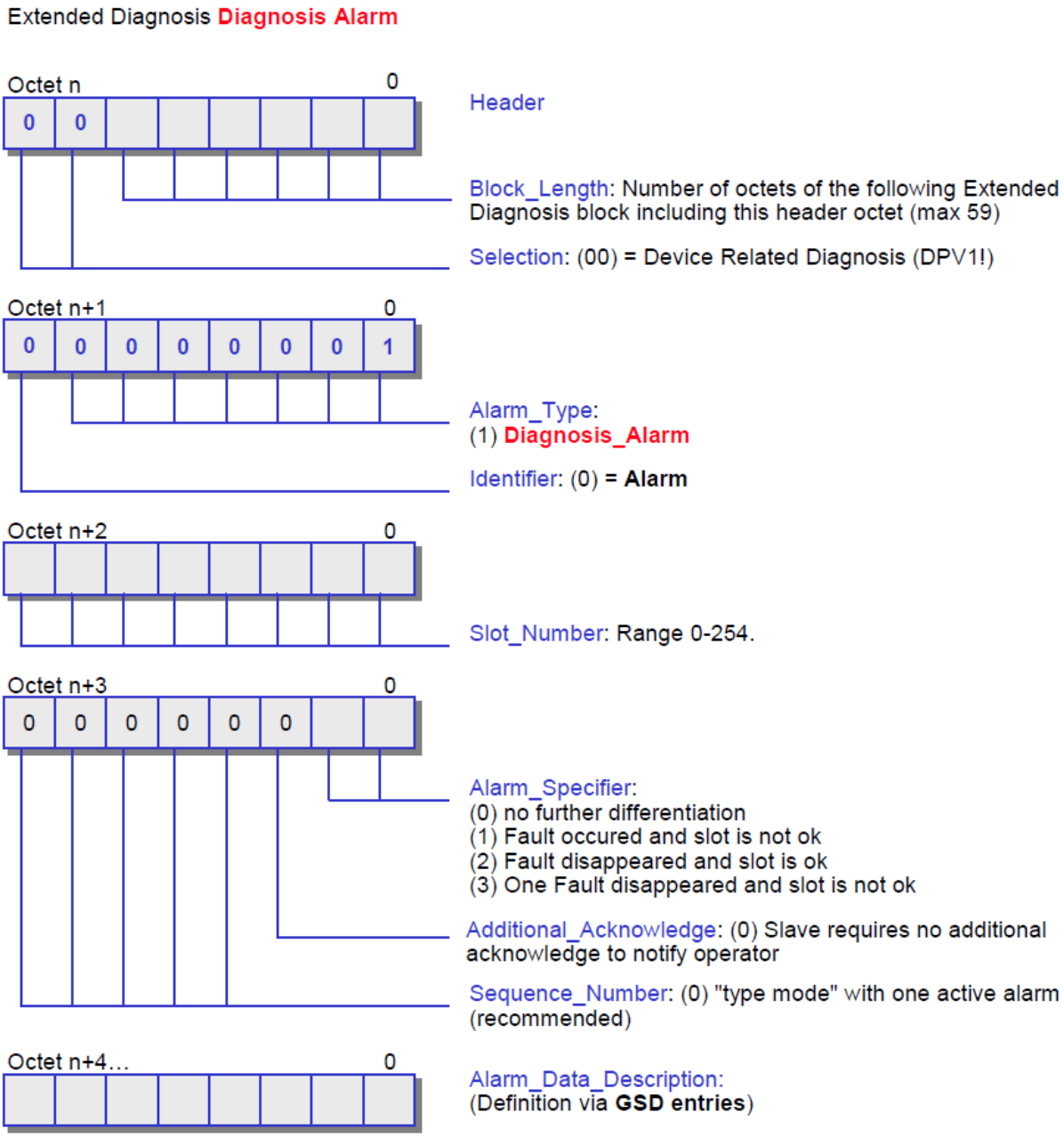
4.2 Extended Diagnostics

Here only the Device Related Diagnostics - one of the three possible types of Extended Diagnostics - is presented.

Device Related Diagnostics is divided into two types: ***Diagnostics Alarm*** and ***Status***.

4.2.1 Diagnostics Alarm

The ***Diagnostics Alarm*** covers failures and errors of slave technology in a more detailed way as shown in the following:

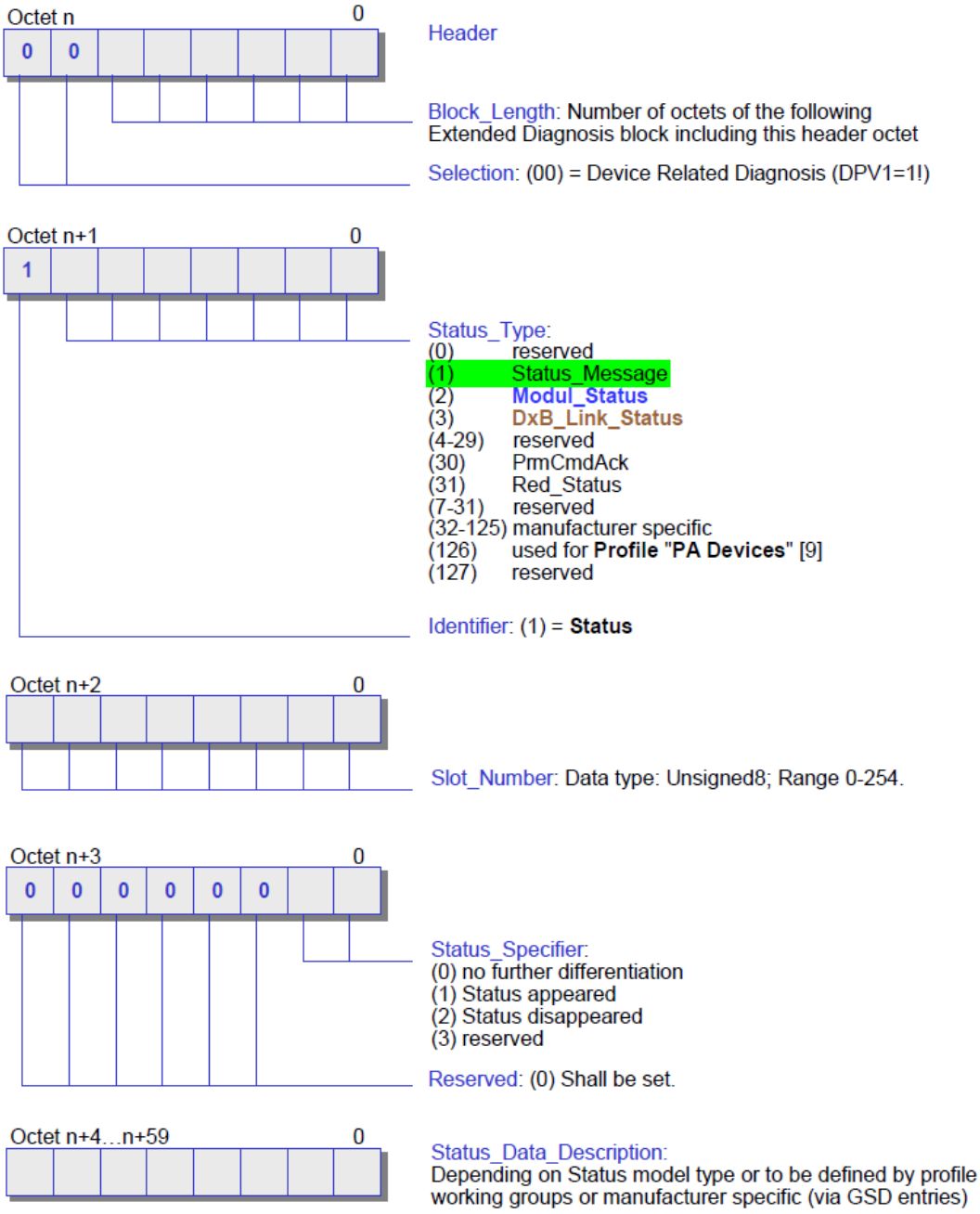


The CompactCom will create an Extended Diagnostics Alarm when the host application selects ***latching severities*** in the create event command.

4.2.2 Status

The *Extended Diagnostics Status* provides several different Status Types, e.g. *Status_Message, Modul_Status*, … Within this application note the Status_Type *Status_Message* is in focus.

The CompactCom will create an Extended Diagnostics Status with the Status_Type *Status_Message* (value of Octet n+1 = 81h) when the host application selects ***recoverable / unrecoverable severities*** in the create event command (as long as the Modular Device Object is not implemented on host application side).



5 Application Example

In this example we are using the Anybus CompactCom 40 Starter Kit in which we integrate the Anybus CompactCom 40 device. We use the host application example code for the windows platform simulating the host application. We also run the automation engineering tool SIMATIC STEP 7 by Siemens, for configuring the PLC. In this example we employ a S7 PLC-CPU 315-2 PN/DP controller. The engineering tool will also report the diagnostic events sent by the CompactCom device to the PLC.

The host application example code can be found here: Host application sample code.

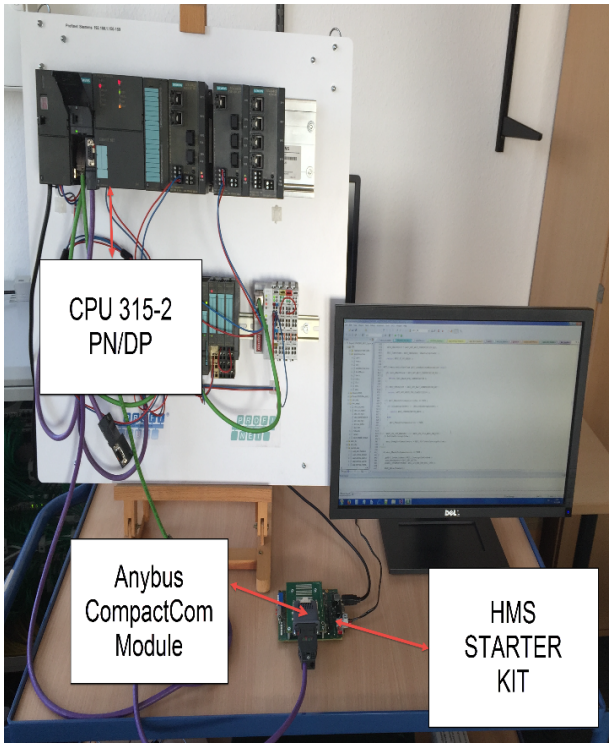


Fig. 1 Hardware Connection Overview

The following section will explain what has to be done from the application side to make the CompactCom PROFIBUS device start to send ***Device Related Diagnostics*** to the network master.

Each diagnostic event created by the host application - independent of whether the bit ***Extended Diagnostics*** of CMDExt[0] is set or not - will be reported as Extended Diagnostics on PROFIBUS.

5.1 Code Sample for Creating a Diagnostic Event

This sample is intended to show how to create a diagnostic event in the CompactCom, using the CompactCom host application example code. The example below shows the structure of a ***create (03h) event command message*** that generates a diagnostic event defined as *minor recoverable* for the event code *voltage*.

The section 6.1 will show how this event is displayed on STEP 7.

```
// message header part
ABCC_SetMsgHeader (psMsg,                          // buffer
                   ABP_OBJ_NUM_DI,                 // diagnostic object
                   0,                              // instance
                   0,                              // attribute
                   ABP_CMD_CREATE,                 // command type
                   8,                              // Message data size
                   ABCC_GetNewSourceId ());        // source id

// severity
psMsg->sHeader.bCmdExt0 = ABP_DI_EVENT_SEVERITY_MINOR_REC;

// event code
psMsg->sHeader.bCmdExt1 = ABP_DI_EVENT_VOLTAGE;


// message data part (little endian)
ABCC_SetMsgData8 (psMsg, 0x00, 0); // slot number associated with the event
ABCC_SetMsgData8 (psMsg, 0x00, 1);
ABCC_SetMsgData8 (psMsg, 0x00, 2); // ADI associated with the event
ABCC_SetMsgData8 (psMsg, 0x00, 3);

// big endian for the rest of data
ABCC_SetMsgData8 (psMsg, 0xFF, 4); // element associated with the event
ABCC_SetMsgData8 (psMsg, 0xFF, 5); // bit associated with the event
ABCC_SetMsgData8 (psMsg, 0x00, 6); // reserved
ABCC_SetMsgData8 (psMsg, 0x00, 7); // reserved
```

The *create (03h)* event command above has created a diagnostic event including the severity code (***minor recoverable (0x00)***) and the event code (***the voltage (0x30)***).

The message data field contains 8 octets.

The first and second message data octets (indices 0 and 1) indicate the slot affected by the event -in this example they are set to the default value “0”-. The third and fourth message data octets (indices 2 and 3) reveal which process data (ADI) is involved –here set to “0” because it is not associated with any particular ADI-.

The fifth message data octet (index 4) signals which element of the ADI is concerned –here set to “255” because it is associated with the entire ADI-. The sixth octet (index 5) tells which bit of the element is affected –here set to “255” because it is associated with the entire element-. The seventh and eighth octets (indices 6 and 7) are reserved.

See section 12.4 in the Anybus CompactCom 40 Software Design Guide and the network guide for more details.

6 Relation Between Severity / Event Code and PROFIBUS GSD Files

The PROFIBUS GSD file of the ABCC contains a section called ***Status diagnostic messages*** where **alarm diagnostics** or **status messages** are defined and in this area is done the relation between Severity and Event Code of the diagnostic event created by the host application and the text (Diag_Texts) which will be displayed in the engineering tool environment (SIMATIC STEP 7).

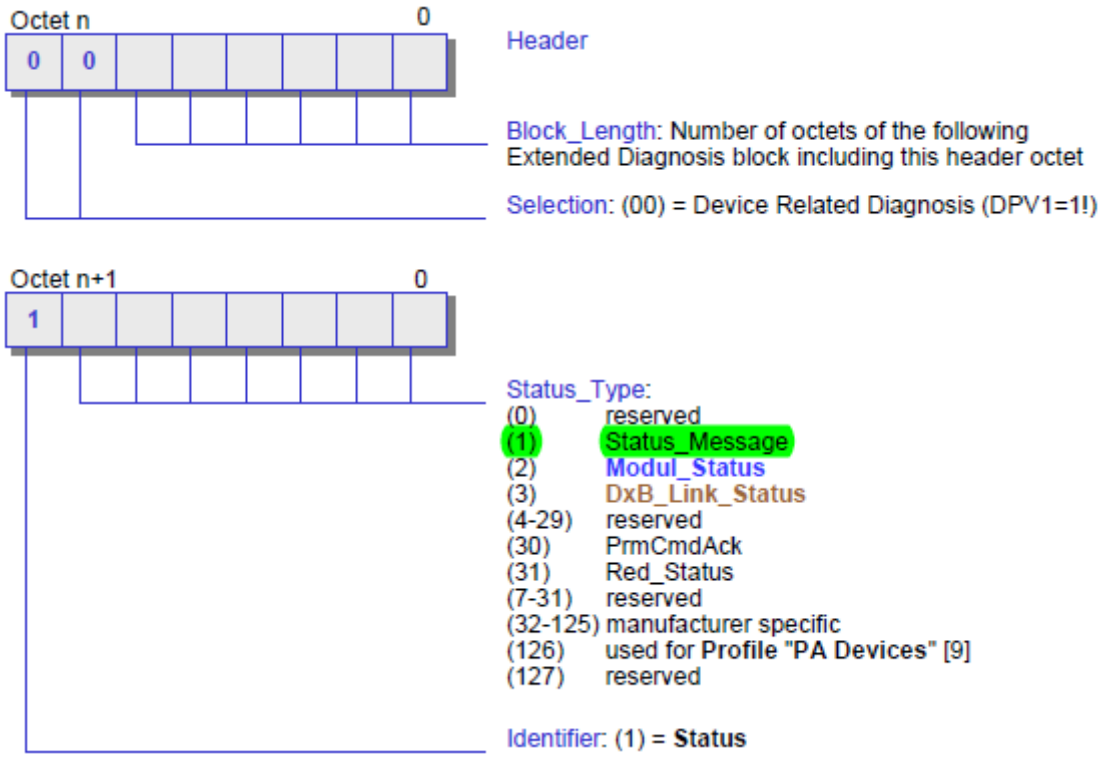
```
;Status diagnostic messages
Unit_Diag_Area=16-17
Value(0)   = "Status not changed"
Value(1)   = "Status appears"
Value(2)   = "Status disappears"
Unit_Diag_Area_End

Unit_Diag_Area=24-31
Value(0)    = "Minor, recoverable"
Value(16)   = "Minor, unrecoverable"
Value(32)   = "Major, recoverable"
Unit_Diag_Area_End

Unit_Diag_Area=32-39
Value(16)   = "Generic Error"
Value(32)   = "Current"
Value(33)   = "Current, device input side"
Value(34)   = "Current, inside the device"
Value(35)   = "Current, device output side"
Value(48)   = "Voltage"
Value(49)   = "Mains Voltage"
Value(50)   = "Voltage inside the device"
Value(51)   = "Output Voltage"
Value(64)   = "Temperature"
Value(65)   = "Ambient Temperature"
Value(66)   = "Device Temperature"
Value(80)   = "Device Hardware"
Value(96)   = "Device Software"
Value(97)   = "Internal Software"
Value(98)   = "User Software"
Value(99)   = "Data Set"
Value(112)  = "Additional Modules"
Value(128)  = "Monitoring"
Value(129)  = "Communication"
Value(130)  = "Protocol Error"
Value(144)  = "External Error"
Value(240)  = "Additional Functions"
Unit_Diag_Area_End
```

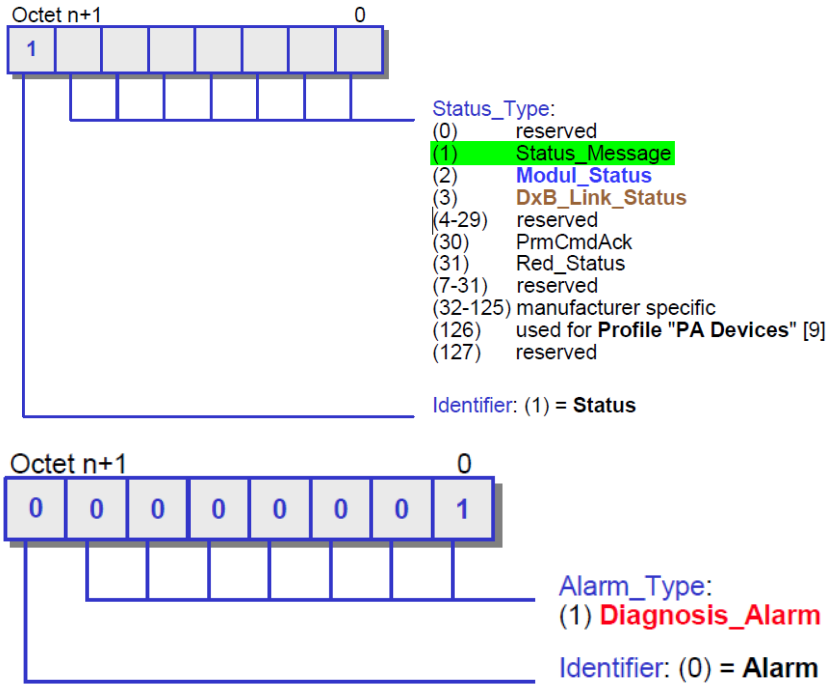
Between the keywords Unit_Diag_Area and Unit_Diag_Area_End the assignment of values in the bit fields to texts is done. These values are in decimal format.

The Unit_Diag_Area is that part of the diagnostics frame which follows the header and counts the bits starting with 1 for bit 0 of the “Type” octet:



Octet	Name	7	6	5	4	3	2	1	0
1	Header_ Octet								
2	Type	7	6	5	4	3	2	1	0
3	Slot	15	14	13	12	11	10	9	8
4	Specifier	23	22	21	20	19	18	17	16
5	Diagnosis_User_Data (0..59 Byte)	31	30	29	28	27	26	25	24
5		39	38	37	36	35	34	33	32
6						..	42	41	40
:						:			

Bits 0 to 7 of the Unit_Diag_Area refer to Octet n+1 (Status_Type / Alarm_Type) and define the type of the diagnostics: alarm or status.



In case of *Status* bit 7 is set, in case of *Alarm* bit 7 is cleared.

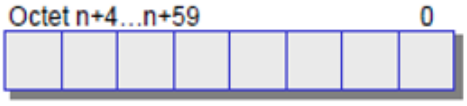
Bits 8 to 15 of the Unit_Diag_Area refer to Octet n+2 (Slot number) and are not listed explicitly in the GSD file. The slot number is given by the host application when sending the create event command.

Bits 16 to 23 of the Unit_Diag_Area refer to Octet n+3 (Status specifier / Alarm specifier). They inform if an event appears or disappears. This info is updated by the CompactCom internally, depending on if the host application sent a create or delete event command.

```
;Status diagnostic messages
Unit_Diag_Area=16-17
Value(0)   = "Status not changed"
Value(1)   = "Status appears"
Value(2)   = "Status disappears"
Unit_Diag_Area_End
```

Bits 24 to 31 of the Unit_Diag_Area refer to Octet n+4 (Status_Data_Description, **first octet**) and are listed explicitly in the GSD file and correspond to the value of **severity** which is given by the host application when sending the create event command:

```
Unit_Diag_Area=24-31
Value(0)    = "Minor, recoverable"
Value(16)   = "Minor, unrecoverable"
Value(32)   = "Major, recoverable"
Unit_Diag_Area_End
```



Status_Data_Description:
Depending on Status model type or to be defined by profile working groups or manufacturer specific (via GSD entries)

```
/*---------------------------------------------------------------------------
**
** Diagnostic object event severity.
**
**---------------------------------------------------------------------------
*/
typedef enum ABP_DiEventSeverityType
{
   ABP_DI_EVENT_SEVERITY_MINOR_REC   = 0x00, /* Minor, recoverable            */
   ABP_DI_EVENT_SEVERITY_MINOR_UNREC = 0x10, /* Minor, unrecoverable          */
   ABP_DI_EVENT_SEVERITY_MAJOR_REC   = 0x20, /* Major, recoverable            */
   ABP_DI_EVENT_SEVERITY_MAJOR_UNREC = 0x30, /* Major, unrecoverable          */
   ABP_DI_EVENT_SEVERITY_MINOR_LATCH = 0x50, /* Minor, recoverable latching (ABCC40)  */
   ABP_DI_EVENT_SEVERITY_MAJOR_LATCH = 0x60  /* Major, recoverable latching (ABCC40)  */
}
ABP_DiEventSeverityType;
```

Bits 32 to 39 of the Unit_Diag_Area refer to Octet n+5 (Status_Data_Description, **second octet**) and are listed explicitly in the GSD file and correspond to the value of **event code** which is given by the host application when sending the create event command.

```
Unit_Diag_Area=32-39
Value(16)  = "Generic Error"
Value(32)  = "Current"
Value(33)  = "Current, device input side"
Value(34)  = "Current, inside the device"
Value(35)  = "Current, device output side"
Value(48)  = "Voltage"
Value(49)  = "Mains Voltage"
Value(50)  = "Voltage inside the device"
Value(51)  = "Output Voltage"
Value(64)  = "Temperature"
Value(65)  = "Ambient Temperature"
Value(66)  = "Device Temperature"
Value(80)  = "Device Hardware"
Value(96)  = "Device Software"
Value(97)  = "Internal Software"
Value(98)  = "User Software"
Value(99)  = "Data Set"
Value(112) = "Additional Modules"
Value(128) = "Monitoring"
Value(129) = "Communication"
Value(130) = "Protocol Error"
Value(144) = "External Error"
Value(240) = "Additional Functions"
Unit_Diag_Area_End
```

```
typedef enum ABP_DiEventCodeType
{
   ABP_DI_EVENT_NONE                   = 0x00, /* No event                      */
   ABP_DI_EVENT_GENERIC_ERROR          = 0x10, /* Generic Error                 */
   ABP_DI_EVENT_CURRENT                = 0x20, /* Current                       */
   ABP_DI_EVENT_CURRENT_DEVICE_INPUT   = 0x21, /* Current, device input side    */
   ABP_DI_EVENT_CURRENT_INSIDE         = 0x22, /* Current, inside the device    */
   ABP_DI_EVENT_CURRENT_DEVICE_OUTPUT  = 0x23, /* Current, device output side   */
   ABP_DI_EVENT_VOLTAGE                = 0x30, /* Voltage                       */
   ABP_DI_EVENT_MAINS_VOLTAGE          = 0x31, /* Mains Voltage                 */
   ABP_DI_EVENT_VOLTAGE_INSIDE_DEVICE  = 0x32, /* Voltage inside the device     */
   ABP_DI_EVENT_OUTPUT_VOLTAGE         = 0x33, /* Output Voltage                */
   ABP_DI_EVENT_TEMPERATURE            = 0x40, /* Temperature                   */
   ABP_DI_EVENT_AMBIENT_TEMPERATURE    = 0x41, /* Ambient Temperature           */
   ABP_DI_EVENT_DEVICE_TEMPERATURE     = 0x42, /* Device Temperature            */
   ABP_DI_EVENT_DEVICE_HARDWARE        = 0x50, /* Device Hardware               */
   ABP_DI_EVENT_DEVICE_SOFTWARE        = 0x60, /* Device Software               */
   ABP_DI_EVENT_INTERNAL_SOFTWARE      = 0x61, /* Internal Software             */
   ABP_DI_EVENT_USER_SOFTWARE          = 0x62, /* User Software                 */
   ABP_DI_EVENT_DATA_SET               = 0x63, /* Data Set                      */
   ABP_DI_EVENT_ADDITIONAL_MODULES     = 0x70, /* Additional Modules            */
   ABP_DI_EVENT_MONITORING             = 0x80, /* Monitoring                    */
   ABP_DI_EVENT_COMMUNICATION          = 0x81, /* Communication                 */
   ABP_DI_EVENT_PROTOCOL_ERROR         = 0x82, /* Protocol Error                */
   ABP_DI_EVENT_EXTERNAL_ERROR         = 0x90, /* External Error                */
   ABP_DI_EVENT_ADDITIONAL_FUNCTIONS   = 0xF0, /* Additional Functions          */
   ABP_DI_EVENT_NW_SPECIFIC            = 0xFF  /* Network specific              */
}
ABP_DiEventCodeType;
```

7 Readout of Diagnostics in STEP7 V.5.5

A diagnostics event is created in the Anybus CompactCom 40 module. Start the diagnostics mode in Step 7 in going online by clicking on the button “1” in the picture. Then double-click at the left upper corner of the Anybus slave device on the button “2” represented in the picture by the square box attached to the PROFIBUS DP master.

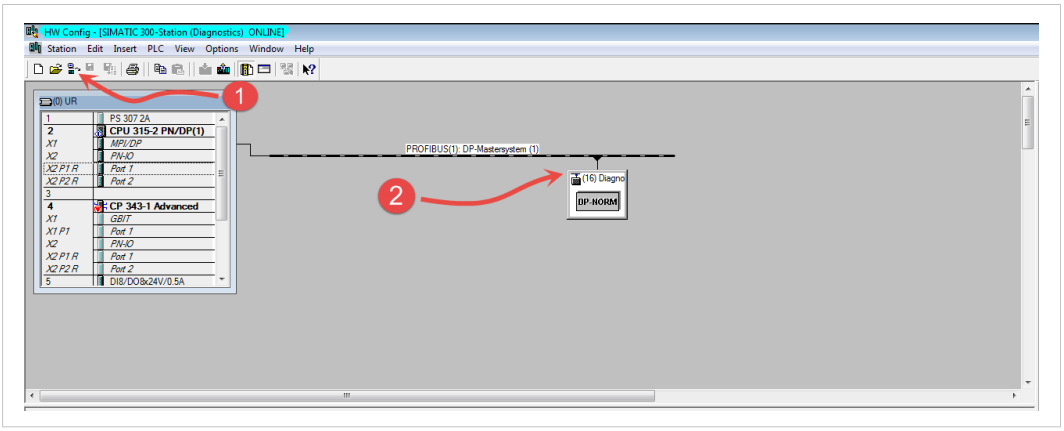


Fig. 2 Step 7 Hardware Diagnostics

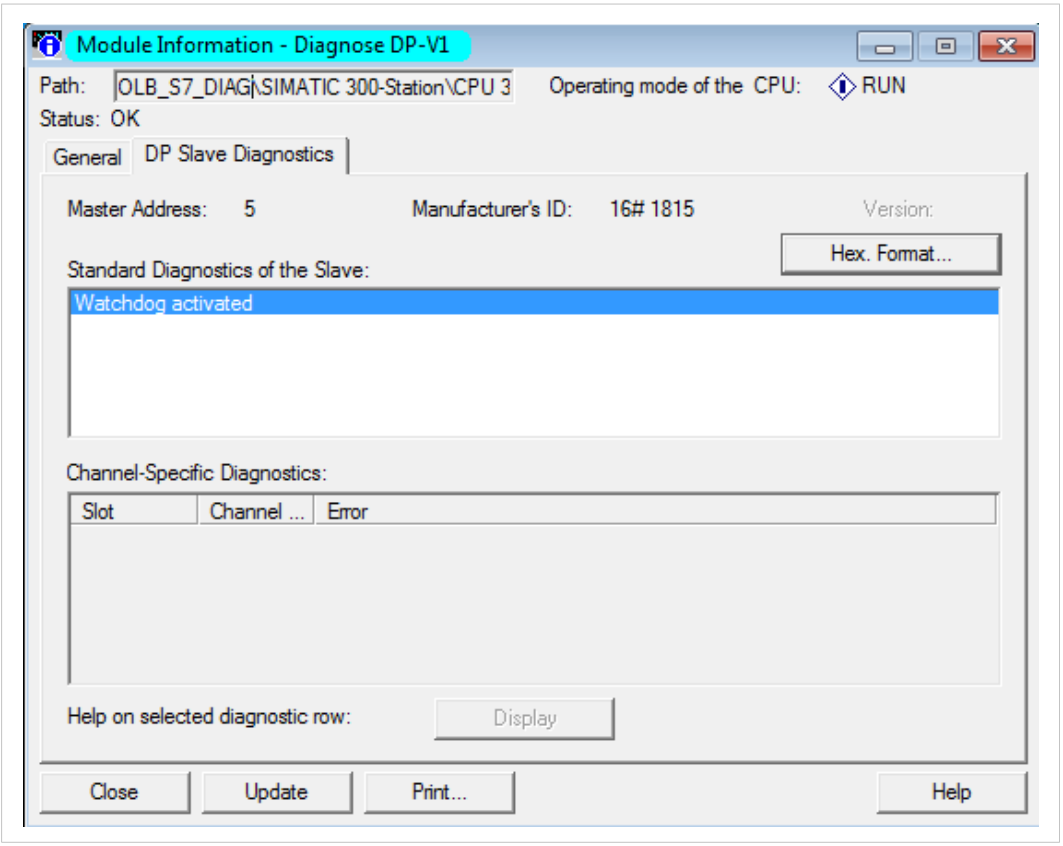


Fig. 3 Diagnostics Window

The diagnostics window of the Anybus CompactCom 40 device will be displayed as shown in figure 4 and available diagnostic information can be read out. The integrated GSD file contains an area where the user can assign to each diagnostic event created a text.

See also the application note *How to configure an Anybus PROFIBUS Slave module with a Siemens Step 7 PLC* showing how available diagnostics can be read out in Step 7.

7.1 Severity: Minor Recoverable and Major Recoverable

The picture presents five active diagnostic events defined as ***minor recoverable***. The diagnostics window below shows the diagnostic information of an Anybus CompactCom 40 device.

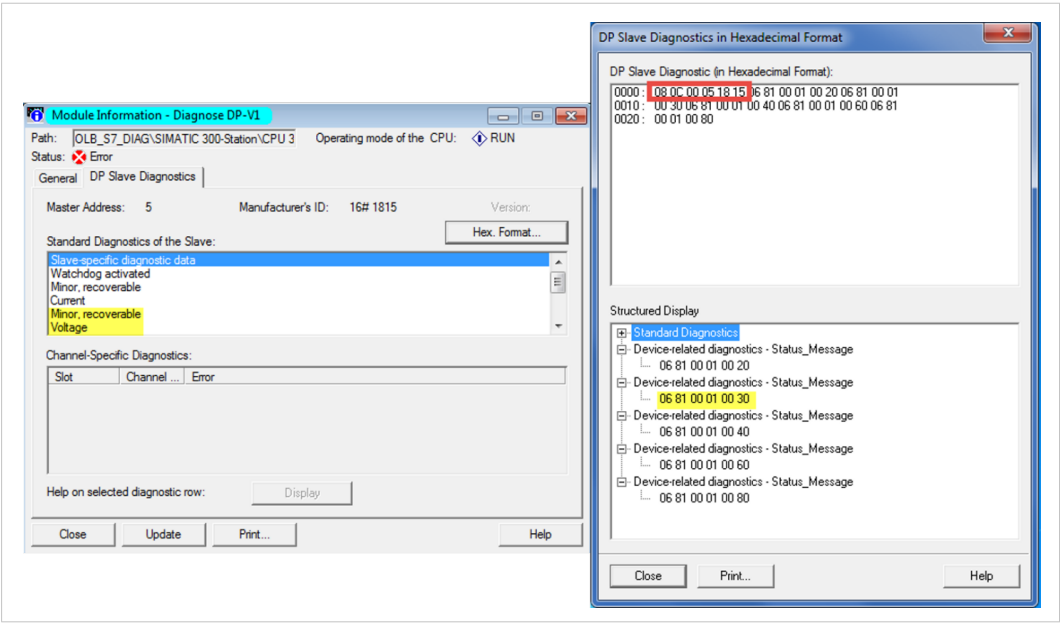


Fig. 4 Minor Recoverable

On the left side of the picture for each active diagnostic event the corresponding text is displayed. The diagnostic window provides an alternative for reading out diagnostics by means of the button *Hex. Format*. When clicking on it the full diagnostic telegram is displayed in hexadecimal format as shown on the picture on the right.

The diagnostic telegram is divided into two parts: the standard diagnostics and the extended diagnostics.

The standard diagnostics contains always 6 octets as shown on the right picture in the red rectangle (“08 0C 00 05 18 15”). The first three octets (“08 0C 00”) named “Status 1”, “Status 2” and “Status 3” contain status info. The fourth octet (“05”) indicates the master address that we can also see on the left picture. The fifth (“18”) and sixth octet (“15”) are used for ident number of the slave. In octet 1 (0x08) of the standard diagnostics bit 3 informs that this diagnostics contains extended diagnostics.

We will have a deeper look to the extended data now.

The extended diagnostics (e.g. “06 81 00 01 00 30”) fill the diagnostic telegram with additional blocks from the seventh octet on.

According to *Status, p. 13*, these bytes have the following meaning:

Octet 1, header	06h	
	Defining type and length of the diagnostic block	
	00**000110**b defines the block length including the header (=> 6)	
	00000110b defines Device Related Diagnostics	***00*** stands for Device Related Diagnostics
		01 stands for Identifier Related Diagnostics
		10 stands for Channel Related Diagnostics
Octet 2, Status_Type	81h (Status_Message)	
Octet 3, Slot_Number	00h (event happened in slot 0)	
Octet 4, Status_Specifier	01h (event is *coming*)	
Octet 5...6, Status_Data_Description	00h 30h (user specific diag data)	
	00h: severity (“**Minor, recoverable**”)	
	30h: event code (“**Voltage**”)	

The picture below shows two active diagnostic events defined as ***major recoverable***. We proceed as for *minor recoverable* to evaluate the diagnostic telegram. The **difference** is in the *user specific diag data* by the severity (Octet 5) and the event code (Octet 6):

Octet 5...6, Status_Data_Description	20h 20h (user specific diag data)
	20h: severity (“**Major, recoverable**”)

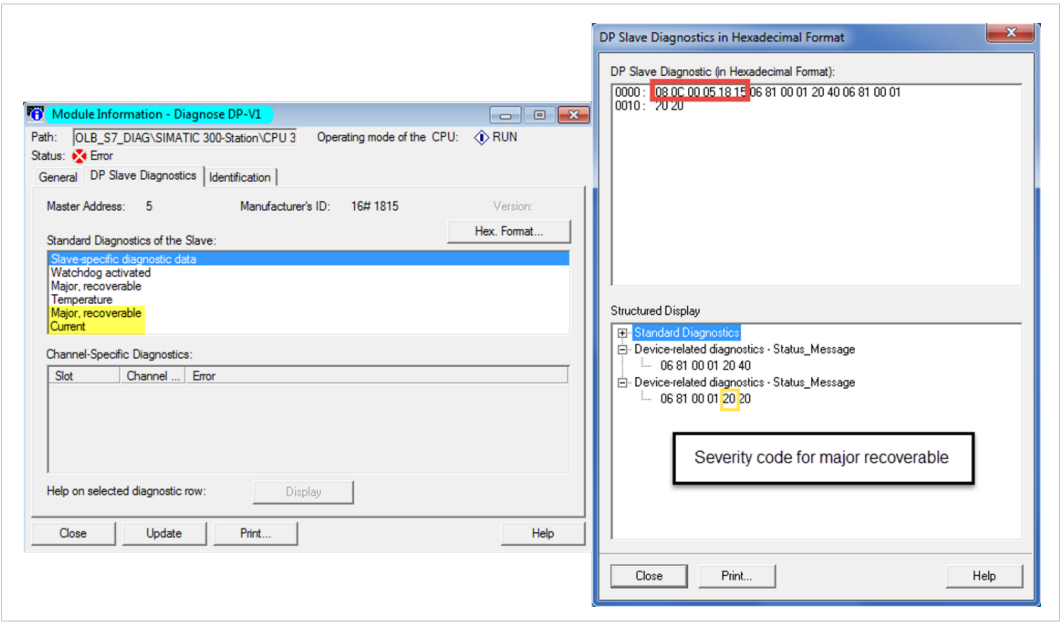


Fig. 5 Major Recoverable Diagnostic Messages

7.2 Severity: Minor Unrecoverable

Figure 6 shows an active diagnostic event defined as *minor unrecoverable*. In the extended diagnostic telegram we can read out from the *user specific diag data* the information about the severity of the event.

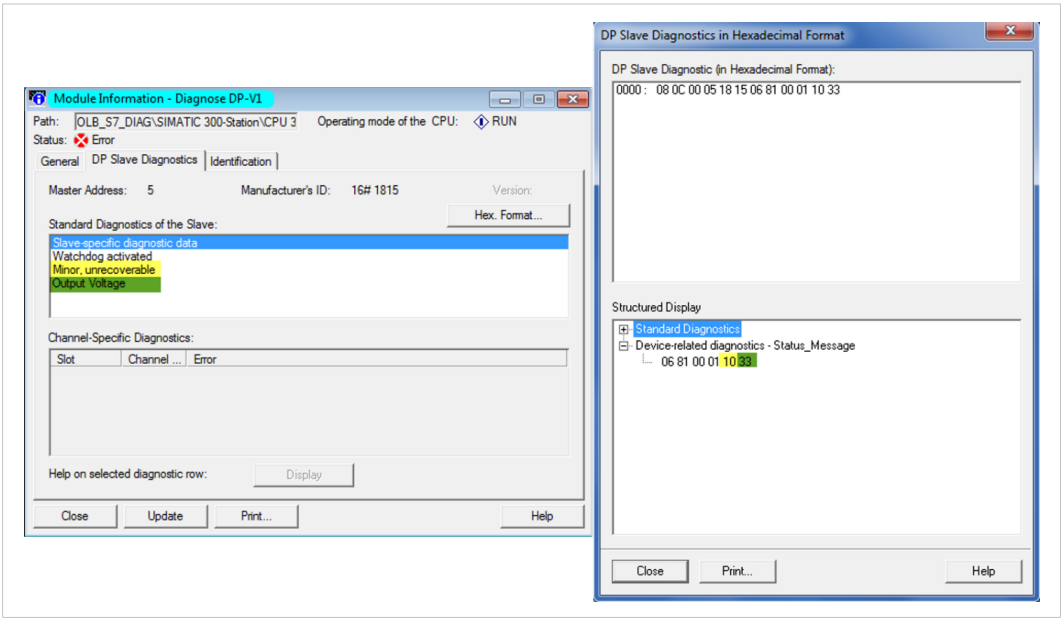


Fig. 6 Minor Unrecoverable Diagnostic Status Message

Octet 5 (0x10) represents the severity code which - in this case - means we are dealing with a minor unrecoverable event with event code (33h, “Output Voltage”). This diagnostic event cannot be deleted. A reset of the CompactCom will delete this diagnostic event.

7.3 Severity: Major Unrecoverable

The creation of a *major unrecoverable* diagnostic event causes the Anybus CompactCom 40 device to enter EXCEPTION state. This results into a disconnection of the device from the network. It is not possible to report any diagnostics. This is confirmed by the picture below. The diagnostics window indicates: the device is not available and since the device being disconnected from bus the diagnostics window of the slave will not display diagnostic events.

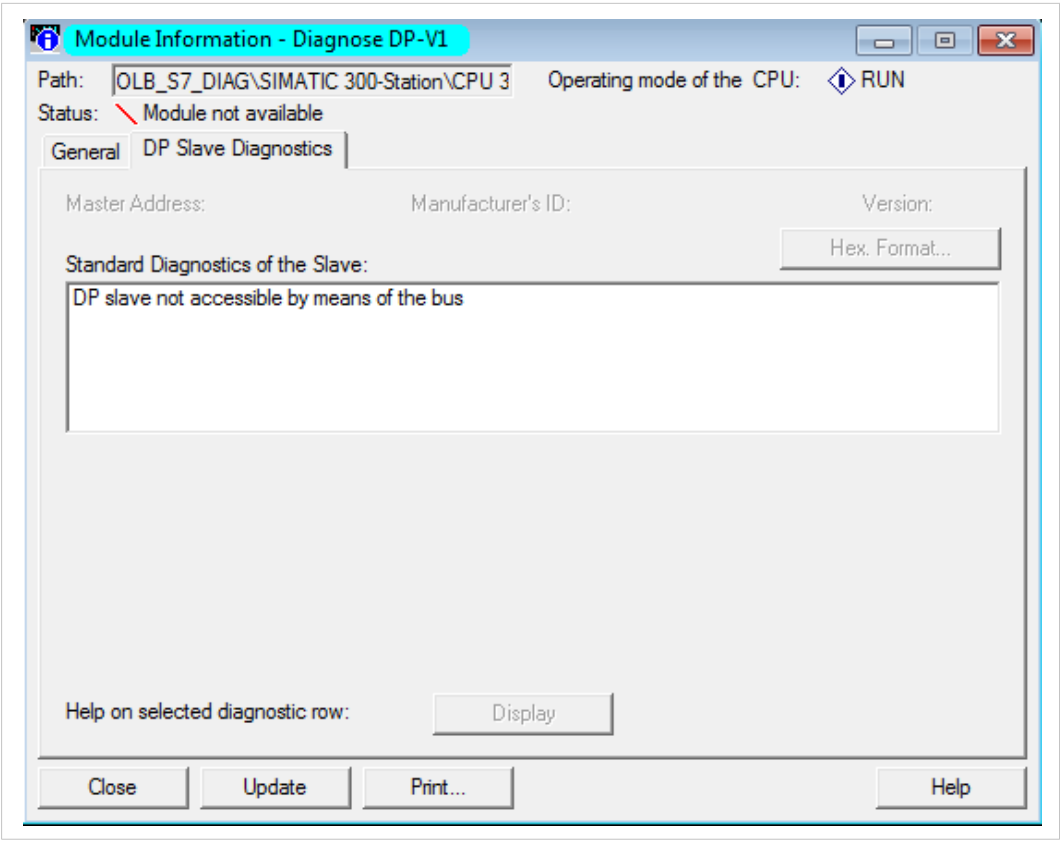


Fig. 7 Disconnection from Network

7.4 Severity: Minor Latching and Major Latching

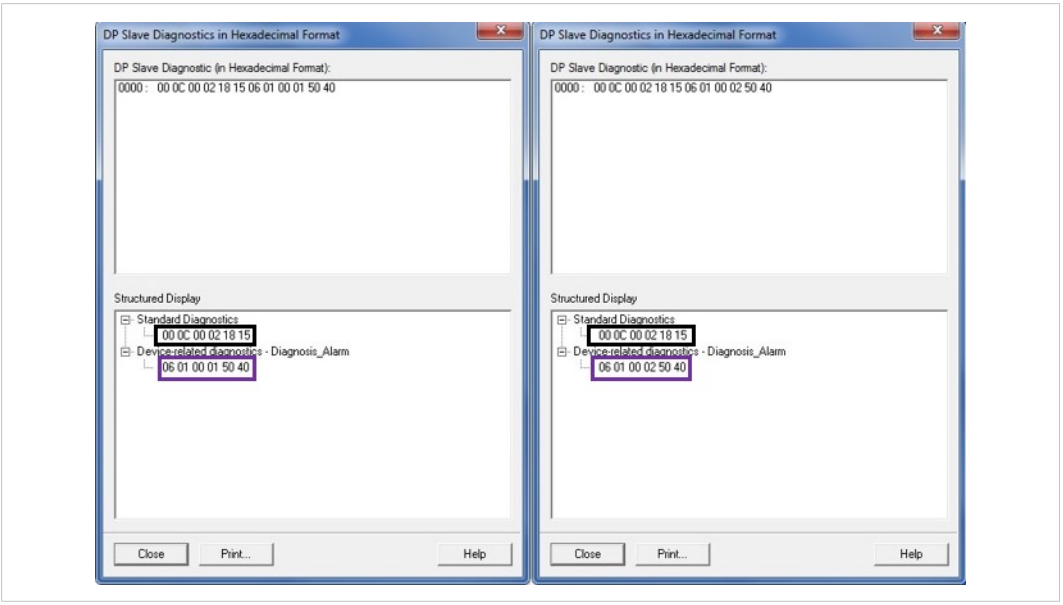


Fig. 8 Minor Latching Diagnostic Message

The picture above presents an **active** *minor latching* diagnostic event on the left side of the picture and a **deleted** *minor latching* diagnostic event on the right side of the picture. The diagnostic telegram is also divided in standard diagnostics and extended diagnostics.

The standard diagnostics contains 6 octets like the other types of diagnostic events. In Octet 1 (0x00) of the standard diagnostics **bit 3** is not set**. This means that latching events will be handled as alarms whereas non-latching events will be handled as status messages**. *This is the main difference between latching and not latching diagnostic events*.

Octet 1, header	06h (Device Related Diagnostics)
Octet 2, Status_Type	**01h (Diagnostics_Alarm)**
Octet 3, Slot_Number	00h (event happened in slot 0)
Octet 4, Status_Specifier	01h (event is *coming*)
	02h (event is going)
Octet 5...6, Status_Data_Description	50h 40h (user specific diag data)
	50h: severity (“**Minor, recoverable latching**”)
	40h: event code (“**Temperature**”)

HMS Industrial Networks AB
Box 4126
300 04 Halmstad, Sweden

info@hms.se

SCM-1202–025 1.1.2608 / 2016-12-01 09:28