Title: Common Components HART 7.2 / 5.9 Subsystem

References: In ´HART Field Communications Protocol Specification Revision 7.2´ listed Specifications for HART Revision 7.2, Test Spec. (including test cases, test method, test result, …), etc.

Review Document

Distribution Software Archive

Author Stefan Tabelander Date: 2012-06-21

Review 2012-06-28 Andreas Stelter

Remarks:

None

Contents

1 Context

This Document is a representation of analysis, planning, implementation and decision-making. It consists of design information for the subsystem <HART>. One of its major goals is to support reuse of the described subsystem.

2 Data Sheet

This chapter gives an overview about all important facts of the subsystem. It can be used by a developer who would like to reuse this subsystem.

Category	*Item*	*Description*
Development	Version / Status	01.00.01 / Released
	Known Bugs	None
	Planned Improvements	-
HW-Platform	Type	M16C/63
	Clocking	> 1 MHz
SW-Development Environment	Compiler	IAR M16C/C++ Compiler Version 3.40.5.40066, no optimization
	Operating System	Segger for M16C emBOS 3.28G
	Framework	Tested in version 2.3.0
	Case / Code-Generation Tool	HART Command generation tool ´HARTGen´. Version 1.0.0 Build 14
Required	Operating System	-

Resources	HW	**LAYER1**: Not part of the common subsystem. Example implementation uses an external MSP430 softmodem. The softmodem claims the following resources: UART2 (Softmodem data exchange) Timer A4 (Softmodem clock source) P6.4 (RTS request to send) P7.2 (Carrier detect) P7.5 (Reset softmodem) **LAYER2**: Timer B0 (Burst timer) Timer B1 (gap timer) **LAYER7**: No special resources necessary
	RAM	1263 Bytes (NEARDATA)
	ROM	13349 Bytes (HUGECODE) 3242 Bytes (HART_COMMAND_TABLE_DATA ➔ Example) 6087 Bytes (CONST+FARCONST)
	Execution Time	**LAYER2**: Average: 400µs; max.: ~1.5ms (1MHz) **LAYER7**: Depends on executed command.
	Special HW	**LAYER1**: External hardware (e.g. filter) are necessary for the soft-modem. Layer 1 is not part of this distribution package
Standards	Safety	For this subsystem the following safety issues are required: - PCLint level 3 - ABB Coding Conventions for embedded software, V1.8 - Code Review - Additional safety measures necessary for SIL approvals (e.g. HARTCanAccess(…)
	Other	
Documentation	Requirements / Use Cases	**HART 7.2 Protocol Specifications**

	Public Interface Description	hart.h, hart_idx.h **LAYER2**: layer2.h **LAYER7**: layer7.h
	Test Specification	HCF_KIT-192 (HTEST-L) version 1.5 **LAYER1**: HCF_TEST-2, Revision 2.2, FSK Physical Layer Test Specification **LAYER2**: HCF_TEST-1, Revision 2.1, Slave Data Link Layer, Test Specification. **LAYER7**: HCF_TEST-3 Slave Universal Command, Test Specification, HCF_TEST-4 Slave Common Practice Command, Test Specification
Special		

3 Detailed Description

3.1 Subsystem integration

The following instructions shall serve as a help when integrating the HART 7 subsystem in a new system environment.

In order to visualize the interfaces the HART 7 subsystem assumes on a target device, a dedicated "Application" subsystem was introduced. This subsystem accommodates those parameters and functions as dummy implementations that shall be replaced by the specific and corresponding equivalent in the target device software. The "Application" (dummy) subsystem shall not be used on the final target device.

In order to help find those places that need to be modified, a special comment was introduced. The developer simply needs to search for
// @@adjust
The occurrences are assumed to be self-explaining and contain hardware dependencies, places where "Application" functions or objects are used or places where the developer is supposed to remove or add specific code.

Please note the current implementation is based on a soft modem solution using an MSP430. The soft modem accommodates the complete Layer 1. Other implementations may not have the soft modem. E.g. the soft modem may reside in the main micro-controller or a special HART ASIC is used. In these cases the developer is supposed to modify the subsystem in such a way, that a dedicated/custom Layer 1 will be added.

Note, that the UART receive- and transmit interrupt hardware for HART communication must be initialized in such a way that the interrupt is initiated in case the --- transmit buffer is empty! ---. This must be done in order to comply with the in HART specified timing (e.g. for burst, gap times, etc.) If the UART is not initialized this way the timers will not be started at the correct locations in the code as the interrupt is generated at a different time.

The HART subsystem provides means to add a second Layer 2. A template for the interface is provided and the interaction with the Layer 7 is available. Still the implementation of the second Layer 2 lies in the responsibility of the device developer. This is for the reason that the Second Layer 2 is not specified and different devices have different/diverting requirements for it. In principle the developer may reuse the Layer2 and modify (reduce) it to his/her needs.

3.2 Static Modelling

The static design of the HART Layer 7 is shown in the following diagram. HART as a subsystem inherits from T_UNIT. The parameters to be accessed via T_UNIT interface are created here. Two special T_DATA_OBJ have been defined in order to accommodate the address setting and configuration changed flag reading and writing. Due to the fact, that they are specific to this subsystem, they have not been made public.

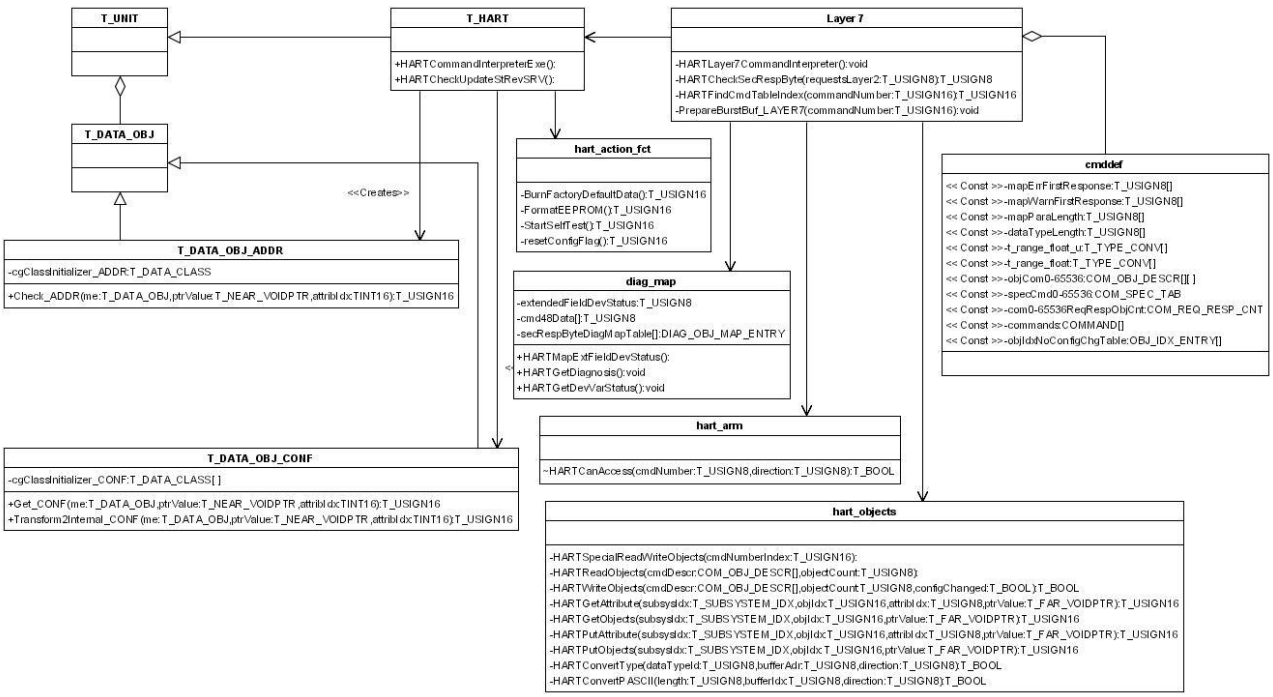
Some functions are defined in **hart_action_fct.c** and are referenced in the dedicated constructor inside the object list. Layer 7 is only one part of this subsystem. All functions that belong to the layer 7 themselves reside in **layer7.c**, **hart_objects.c** and **hart_arm.c**.

The class aggregates **cmddef.c**. This file mainly contains the command definitions and the command table that are referenced from the layer 7. Also it contains the table ´objIdxNoConfigChgTable´ with subsystem objects of the write commands which don´t increment the configuration change counter (see also chapter ´HARTCheckUpdateStRevSRV´).

Most of this file is generated using the HART generation tool ´HARTGen´:

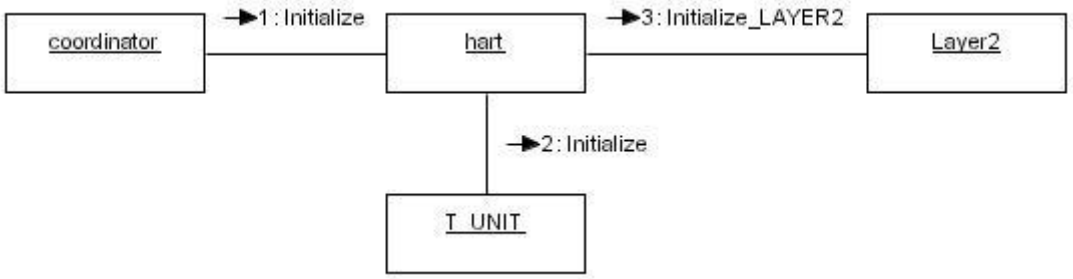
a) The table ´commands´ (struct COMMAND) lists all HART commands implemented in the device.
b) Normal HART commands (Action commands with 0 request data bytes, read commands with 0 request data bytes and write commands with same data length/meaning in request and response) needs only an array of descriptions (struct COM_OBJ_DESCR) of all command objects.
c) Special HART commands (e. g. different request (not equal 0) and response data byte length, data field with different meaning in dependence of request slot code value, …) need a more complex command description (struct COM_SPEC_TAB).
   One array with object descriptions of the request and one array with object descriptions of the response are needed (struct COM_OBJ_DESCR). Every array contains descriptions of the maximum request/response length. Arrays of a slot command (a slot selects the objects of the data field) contain default object descriptions.
   One table contains the information how many request/response objects must be handled in dependence of the received data bytes number (struct COM_REQ_RESP_CNT).
   A slot command needs the information which objects belong to which slot code. This information is written in a table with object descriptions of every slot code (struct COM_OBJ_DESCR).
d) The table ´objIdxNoConfigChgTable´ contains subsystem objects of the write commands which don´t increment the configuration change counter.

Apart from generated code, the first response mapping from T_UNIT return codes (mapErrFirstResponse for errors, mapWarnFirstResponse for warnings) and descriptors for special data structs (t_range_flt_u) are located in this file.

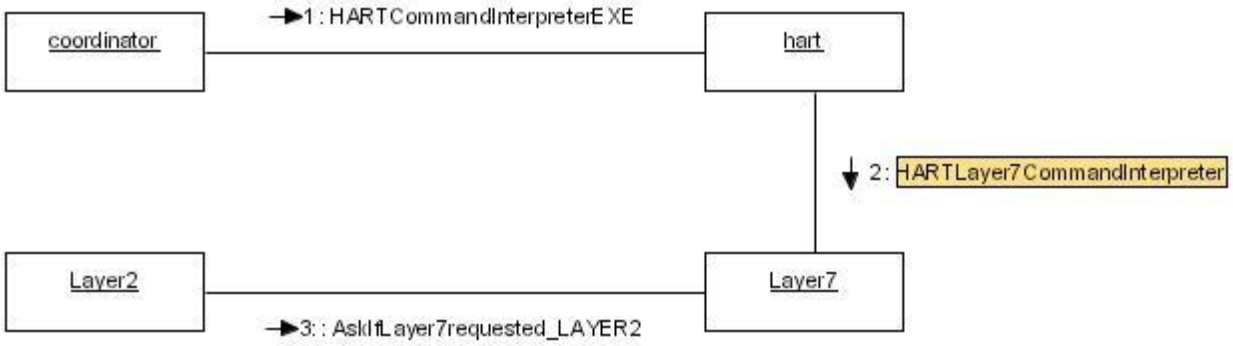


3.3 Dynamic Modelling

The collaboration diagrams below show the dynamic interaction of the classes that are referenced by the HART Layer7.



Collaboration diagram for hart.Initialize()



Collaboration diagram for hart.HARTCommandInterpreterEXE()

3.4 Class Design

3.4.1 Diagnosis/status mapping (Class DIAG_MAP)

Especially for HART the overall diagnosis information of a device is transmitted or offered in some ways:

- The field device status (second response byte) in a command response
- Extended field device status in command #0, #9, #48 (HART revision > 5)
- Device variable status in command #9 (HART revision > 5)

The storage inside the subsystems is not HART encoded so a mapping is assumed to be required.

3.4.1.1 Field device status

A direct mappable field device status bit is ´Malfunction´. The bits ´Loop Current fixed´ and ´Loop Current Saturated´ are mapped from the current out subsystem. The bits ´Primary variable out of limits´ and ´Non primary variable out of limits´ are derived from the device variable status limit bits. The other bits are handled internally.

3.4.1.2 Update of the field device status (Method HARTCheckSecRespByte)

Param1: TUSIGN8 requestsLayer2 = Flags which Layer 2 requests Layer 7.
Return: TUSIGN8 second response byte.

This function is called from Command interpreter after a read or write command has been placed. The function returns the bit-coded second response byte according to the table below.

Second Byte	Description	Mapping
Bit 7	Field Device Malfunction	HARTGetDiagnosis
Bit 6	Configuration Changed	HARTGetObjects (see also chapter ´T_DATA_OBJ_CONF´)
Bit 5	Cold Start	Self determined in dependence which Master (primary/secondary) from which Layer 2 send the request.
Bit 4	More Status Available	Self determined if actual command #48 data changed.
Bit 3	Primary Variable Analog Output Fixed	CURRENT_OUT_IDX,CURR_IDX_diagnosis -> diagnosis if (diagnosis & CURR_DIAG_FIXEDMODE)
Bit 2	Primary Variable Analog Output Saturated **Only if not in multidrop mode!**	CURRENT_OUT_IDX,CURR_IDX_diagnosis -> diagnosis if (diagnosis & (CURR_DIAG_SPANLIMIT_LO | CURR_DIAG_SPANLIMIT_HI))
Bit 1	Non-Primary Variable Out of Limits	HARTGetDiagnosis
Bit 0	Primary Variable Out of Limits	HARTGetDiagnosis

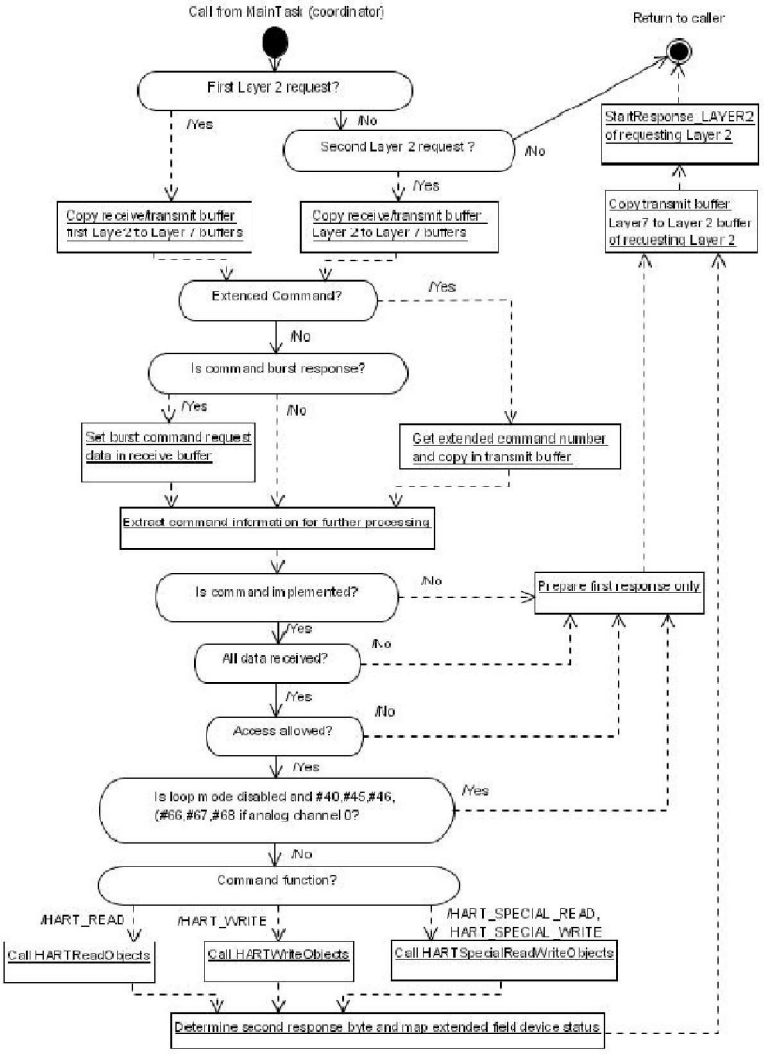
Due to the simplicity of this function, no diagram has been prepared.

3.5 Detailed Design

3.5.1 HARTLayer7CommandInterpreter, HARTCommandInterpreterEXE

Param: void
Return: void

Command interpreter implementation. The function **HARTCommandInterpreterEXE** is called from the coordinator. It calls the Layer 7 function **HARTLayer7CommandInterpreter**. This function will return immediately if no request has been placed (AskIfLayer7requested_LAYER2(), AskIfLayer7requested_LAYER2_2()). In case a command has been received, the command interpreter does the following activities:

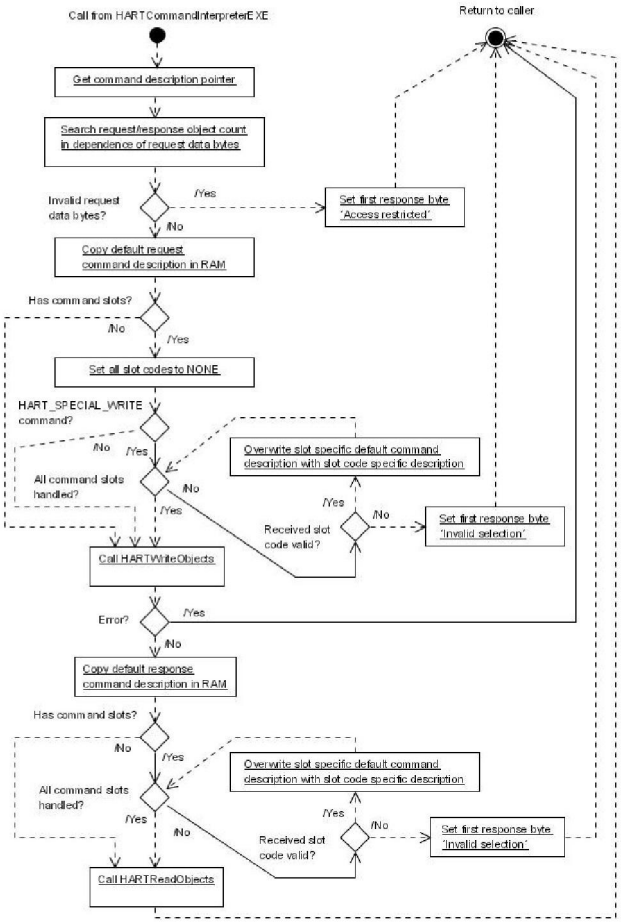


If both Layer 2 have a request at the same time the first layer 2 request will be handled first. The next calling of **HARTCommandInterpreterEXE** handles the second layer 2 request (if no further request from the first layer 2). This mechanism allows both Layer 2 to be used in principle at the same time. Note that the timing (especially in burst mode) may not comply with HART test criteria. In this case enough computing power must be provided (HART schedule more frequently, clock rate higher, etc.)

3.5.2 HARTSpecialReadWriteObjects

Param: TUSIGN16 commandNumberIndex = Index of command in command table.
Return: void

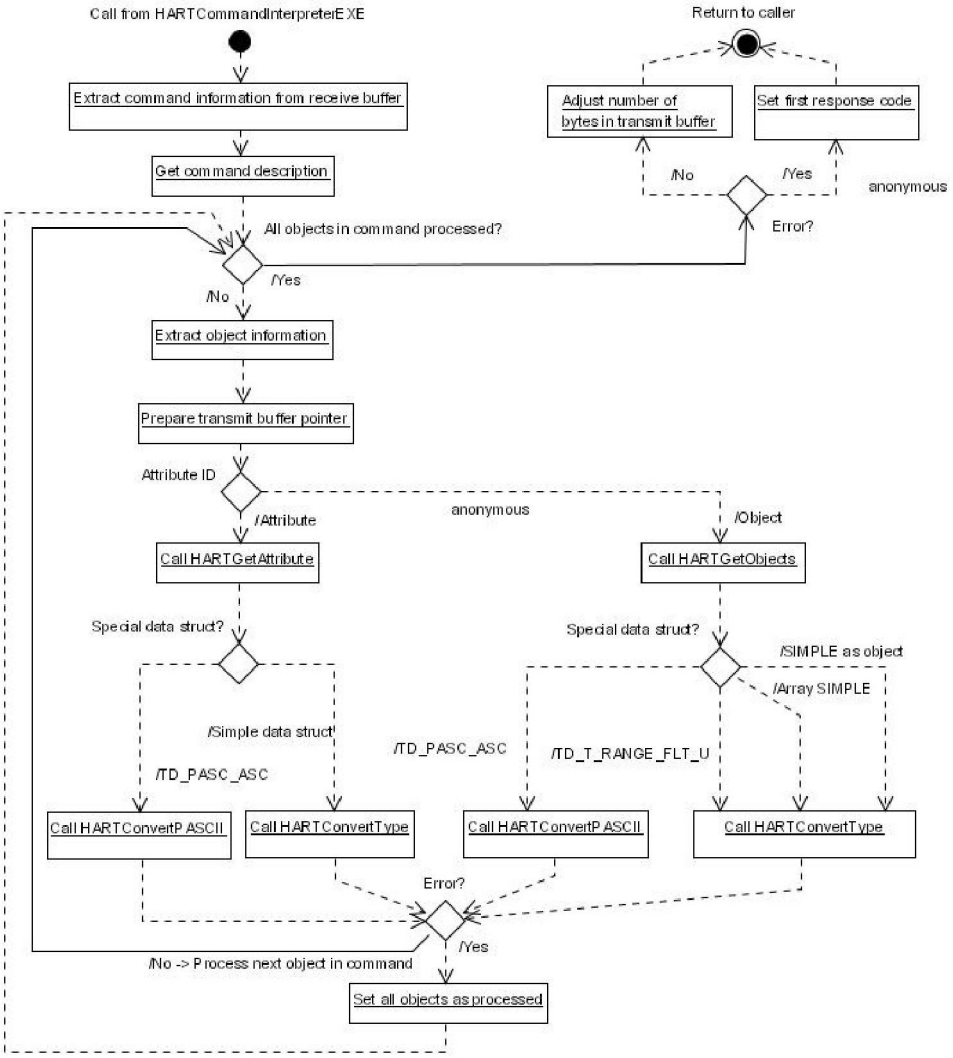
Handles special HART commands where the request/response data area has not always the same length and/or meaning. This is a function that may be extended by the developer to his/her needs. The response codes must be chosen according to this enhancement.
Response code HART_RESP_ERR_ACCESS_RESTRICTED is set if a wrong number of request data bytes is received.
Response code HART_RESP_ERR_INVALID_SELECTION is set if a wrong slot code is received.



3.5.3 HARTReadObjects

Param1: COM_OBJ_DESCR cmdDescr = Pointer to command structure.
Param2: TUSIGN8 objectCount = Number of objects in response.
Return: void

Reads all parameters in a command. The function distinguishes between Attributes and Objects. Arrays are only supported as SIMPLE data types and must be handled as Objects in the command description. The function provides a special handling for data structs that must be enhanced if new types are added.



3.5.4 HARTGetAttribute

Param1:	T_SUBSYSTEM_IDX	subsysIdx = Subsystem index
Param2:	TUSIGN16	objIdx = Object index
Param3:	TUSIGN8	attribIdx = Attribute index.
Param4:	void	*ptrValue = Value pointer
Return:	TUSIGN16	T_DATA_OBJ error codes

This function is called when a specific attribute is to be read. The function is given the identification of the subsystem, the object index and the attribute index. The caller will find the desired value using the value pointer.

The T_UNIT interface function “GetAttribute” will be called. The response coming from this call will be returned to the caller of this function.

Due to the simplicity of this function, no diagram has been prepared.

3.5.5 HARTGetObjects

Param1:	T_SUBSYSTEM_IDX	subsysIdx = Subsystem index
Param2:	TUSIGN16	objIdx = Object index
Param3:	void	*ptrValue = Value pointer
Return:	TUSIGN16	T_DATA_OBJ error codes

This function is called when an object is to be read. The function is given the identification of the subsystem and the object index. The caller will find the desired value using the value pointer.

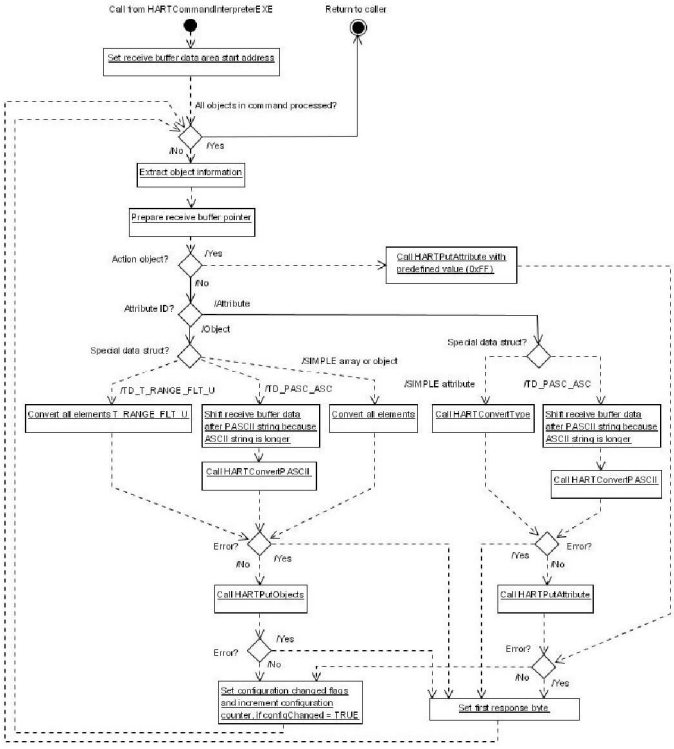
The T_UNIT interface function “GetAttribute” will be called. The response coming from this call will be returned to the caller of this function.

Due to the simplicity of this function, no diagram has been prepared.

3.5.6 HARTWriteObjects

Param1: COM_OBJ_DESCR cmdDescr = Pointer to command structure.
Param2: TUSIGN8 objectCount = Number of objects in response.
Param3: TBOOL configChanged = Indicates if command changes config or not.
Return: TBOOL eTRUE = one or more errors occured, eFALSE = no error occurred.

Writes all parameters in a command. The function distinguishes between Attributes and Objects. Arrays are only supported as SIMPLE data types and must be handled as Objects in the command description. The function provides a special handling for data structs that must be enhanced if new types are added.



Also writing a object of a command is not possible all other writeable objects are stored. This is not conform to the HART specification but accepted, because using the PUT function of the framework.

3.5.7 HARTPutAttribute

Param1: T_SUBSYSTEM_IDX subsysIdx = Subsystem index.
Param2: TUSIGN16 objIdx = Object index.
Param3: TUSIGN8 attribIdx = Attribute index.
Param4: void *ptrValue = Value pointer.
Return: TUSIGN16 T_DATA_OBJ error codes.

This function is called when a specific attribute is to be written. The function is given the identification of the subsystem, the object index and the attribute index. The value intended to be written must have been stored before at the location of value pointer.

The T_UNIT interface function “WriteAttribute” will be called. The response coming from this call will be returned to the caller of this function.
If the response ist OK the function ´HARTGlobalStRevUpdate´ to increment all static revision counters of the other subsystems is called.

Due to the simplicity of this function, no diagram has been prepared.

3.5.8 HARTPutObjects

Param1:	T_SUBSYSTEM_IDX	subsysIdx = Subsystem index.
Param2:	TUSIGN16	objIdx = Object index.
Param3:	void	*ptrValue = Value pointer .
Return:	TUSIGN16	T_DATA_OBJ error codes

This function is called when an object is to be written. The function is given the identification of the subsystem and the object index. The value intended to be written must have been stored before at the location of value pointer.

The T_UNIT interface function “WriteObject” will be called. The response coming from this call will be returned to the caller of this function.
If the response ist OK the function ´HARTGlobalStRevUpdate´ to increment all static revision counters of the other subsystems is called.

Due to the simplicity of this function, no diagram has been prepared.

3.5.9 HARTConvertPASCII

Param1:	TUSIGN8	length = HART commando length of data to be converted
Param2:	TUSIGN8	bufferIdx = index in transmit or receive buffer where to read/write the data
Param3:	TUSIGN8	direction = read or write
Return:	TBOOL	eTRUE = Conversion ok, eFALSE = conversion not possible

This function converts packed ASCII data to ASCII data (if write) or vice versa (if read). In read direction, the transmit buffer provides the source for the data to be converted. In write direction the receive buffer provides the source. Conversion will be done inside the addressed buffer itself at a specific position indicated by the parameter bufferIdx. The conversion is done if the packed ASCII string length is multiple of 3 bytes. Because 3 packed ASCII characters result to 4 ASCII characters there must be enough free space in the buffer for converting the string.

Due to the simplicity of this function, no diagram has been prepared.

3.5.10 HARTConvertType

Param1:	TUSIGN8	dataTypeId = type of data to be converted
Param2:	TUSIGN8	bufferIdx = index in transmit or receive buffer where to read/write the data
Param3:	TUSIGN8	direction = read or write
Return:	TBOOL	eTRUE = Conversion ok, eFALSE = conversion not possible

This function converts data of two or four bytes length. The conversion changes the byte order according to the direction. In read direction, the transmit buffer provides the source for the data to be converted. In write

direction the receive buffer provides the source. Conversion will be done inside the addressed buffer itself at a specific position indicated by the parameter bufferIdx. The data type ID provides the length of data to be converted.

The conversion format is big endian to little endian and vice versa.

Due to the simplicity of this function, no diagram has been prepared.

3.5.11 HARTFindCmdTableIndex

Param1: TUSIGN16 cmdNumber = command number.
Return: TUSIGN16 Index in command table or invalid index if not found.

Searches for the index of the command in the ´commands´ table in cmddef.c. An invalid index will be returned if the command is not implemented.

Due to the simplicity of this function, no diagram has been prepared.

3.5.12 PrepareBurstBuf_LAYER7

Param1: TUSIGN16 cmdNumber = command number.
Return: void.

If Command #9 or #33 are the Burst commands, this function writes the burst slot codes (previously written with command #107) as request data bytes in the receive buffer. Only used burst slots will be requested. With this information the command interpreter generates the burst response.

Due to the simplicity of this function, no diagram has been prepared.

3.5.13 Put_HART

Param1: TUSIGN16 objIdx = object index.
Param2: TINT16 attribIdx = attribute index.
Param3: void *ptrValue = pointer to object value.
Return: TUSIGN16 T_UNIT error code.

This function calls the standard T_UNIT PUT function. A special handling is done if the burst slot codes with command #107 are written. In this case the burst slots #1 - #3 are set to not used if burst slot #0 is written. This is necessary because not all burst slots must be written with command #107.

Due to the simplicity of this function, no diagram has been prepared.

3.5.14 resetConfigFlag

Param1: void
Return: TUSIGN16 T_UNIT error code.

This function will be called in response to command #38. It resets the configuration changed flag of the requesting master. (see also chapter ´T_DATA_OBJ_CONF´)

Due to the simplicity of this function, no diagram has been prepared.

3.5.15 StartSelfTest

Param1: void
Return: TUSIGN16 Allways OK.

This function will be called in response to command #41. As the diagnostic informations are determined in a background task, no explicit function call is needed in order to get such information out of #48. For this reason this functions simply returns OK with every call.

Due to the simplicity of this function, no diagram has been prepared.

3.5.16 HARTCanAccess

Param1: TUSIGN16 cmdNumber = Command number presently read or written to.
Param2: TUSIGN8 direction = read or write.
Return: TUSIGN8 RESP_OK, ACCESS_RESTRICTED, WRITE_PROTECTED

The HART write protection is the only saftety critical part in the layer 7. Access rights are encapsulated in a separate file (hart_arm.c). The function ´HARTCanAccess´ return RESP_OK if access is allowed and WRITE_PROTECTED if access is denied.

This function must be modified to the needs of the target device. Safety relevant measures must be repeated with the target device.

3.6 Design Decisions and Limitations

None.

3.7 Hardware Dependencies

See chapter ´Data Sheet.´

3.8 Data Object Description

3.8.1 T_DATA_OBJ_ADDR

The polling address is an unsigned 8 integer with valid range of 0…63 (0…15 if HART 5 Master sends command #6). If a HART 5 Master sends only the polling address with command #6 an address > 0 sets the current of the CurrentOut subsystem to a fixed value, while the address 0 enables the measurement dependent current.

3.8.2 T_DATA_OBJ_CONF

The configurationFlags object contains the actual state of the HART primary master/secondary master configuration changed flags of both Layer 2.

In a command response the belonging actual configuration changed flag state of the master is set in the second response byte.
All flags are set if one object of a command with set configuration flag attribut in the command description (in cmddef.c) is successful written.
The belonging configuration changed flag is reset if a master sends command #38. The configuration changed flag state of the other masters are not influenced.

3.9 Error Handling

Invalid received HART frames (e. g. communication errors defined by HART specification) will be detected by Layer 2 or Layer 7. The error response is generated inside the HART subsystem and send to the requesting master.

See also LAYER 1 and LAYER 2 design descriptions.

4 Revision Chart

Rev.	Description of Version/Changes	Primary Author(s)	Date
1.0	**Release**	**Stefan Tabelander**	**2011-10-24**
1.1	**Update after release**	**Stefan Tabelander**	**2012-06-28**