Hart Slave Stack C++ 8.0

Technical Data Sheet

C++ Source Code for an Embedded Firmware Module with the following Properties

- No external dynamic memory management. The amount of reserved RAM remains constant.
- The number of objects is determined at compile time and startup.
- No operating system is required to integrate the software. Timer and serial interrupts are enough.
- Simple asynchronous user interface to encapsulate the time-critical part.
- Implements the Uart protocol and Hart Ip.

The implementation is based on the Hart Documents in:
HART Communication Protocol Specification, HCF_SPEC-13, FCG TS20013 Revision 7.09, Release Date: 06 January 2023

Details for the Hart Protocol are provided via the following link: https://www.fieldcommgroup.org/technologies/hart.

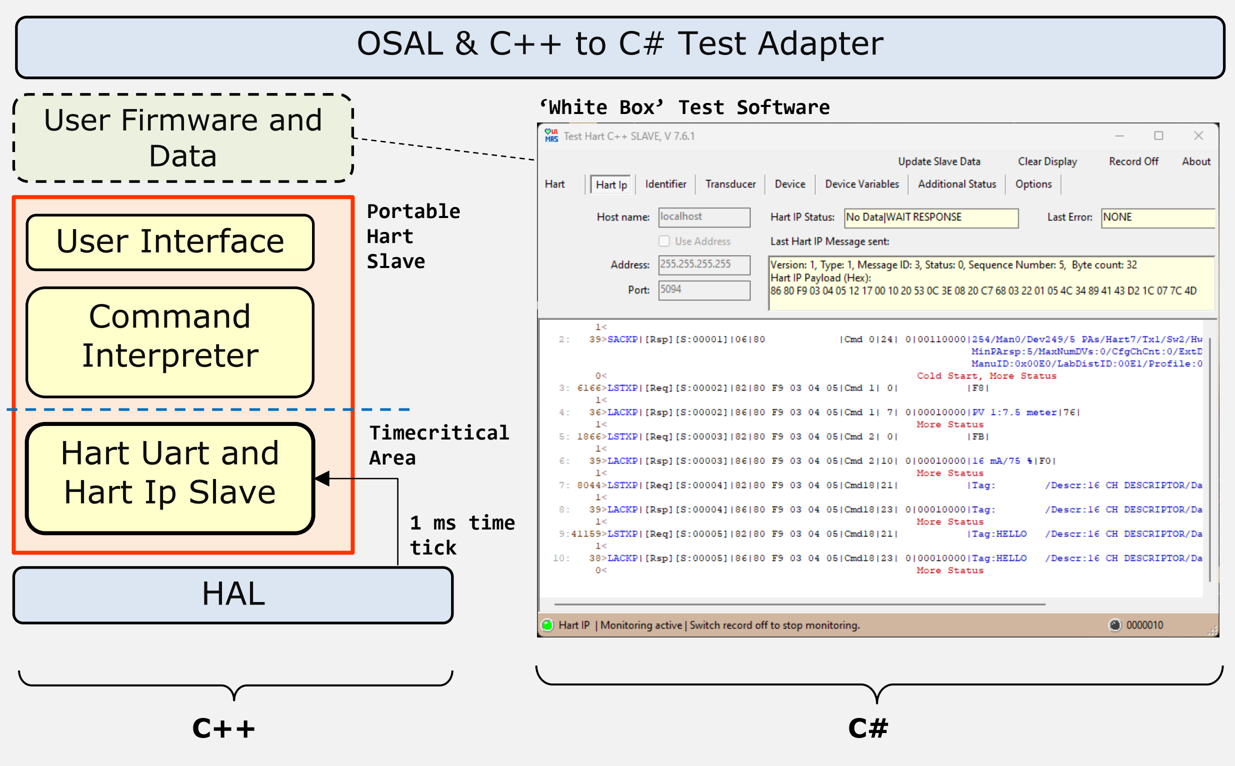
Introduction

Implemented Commands

#	Description	Remarks
Universal Commands		
0	Read Unique Identifier	See also Command Summary Specification
1	Read Primary Variable	
2	Read Loop Current And Percent Of Range	
3	Read Dynamic Variables And Loop Current	
6	Write Polling Address	
7	Read Loop Configuration	
8	Read Dynamic Variable Classifications	Currently only the standard device variables are supported: 244, 245, 246, 247, 248, 249, 0, 1, 2, 3.
9	Read Device Variables with Status	
11	Read Unique Identifier Associated With Tag	
12	Read Message	
13	Read Tag, Descriptor, Date	
14	Read Primary Variable Transducer Information	
15	Read Device Information	
16	Read Final Assembly Number	
17	Write Message	
18	Write Tag, Descriptor, Date	
19	Write Final Assembly Number	
20	Read Long Tag	
21	Read Unique Identifier Associated With Long Tag	
22	Write Long Tag	
38	Reset Configuration Change Flag	
48	Read Additional Device Status	The slave module saves a copy of the last additional device status sent for each master and compares it with the bitstream provided by the user application.
Common Practice Commands		
33	Read Device Variables	
34	Write Primary Variable Damping Value	
35	Write Primary Variable Range Values	
49	Write Primary Variable Transducer Serial Number	
54	Read Device Variable Information	Currently only the standard device variables are supported: 244, 245, 246, 247, 248, 249, 0, 1, 2, 3.
108	Write Burst Mode Command Number	Commands 1, 2, 3 and 9 are currently accepted. Burst messages are not (yet) supported.
109	Burst Mode Control	
512	Read Country Code	
513	Write Country Code	

I consider the now implemented set of commands to be the minimum that must be available in a Hart slave. However, I also recommend making all important functions of a slave accessible via universal and common practice commands and not using user-specific commands. In this case it is not necessary to provide a device description. This saves development time and development costs.

Architecture



Host name: localhost

□ Use Address

Address: 255.255.255.255

Port: 5094

Hart IP Status: No Data|WAIT RESPONSE

Last Error: NONE

Last Hart IP Message sent:

Version: 1, Type: 1, Message ID: 3, Status: 0, Sequence Number: 5, Byte count: 32
Hart IP Payload (Hex):
86 80 F9 03 04 05 12 17 00 10 20 53 0C 3E 08 20 C7 68 03 22 01 05 4C 34 89 41 43 D2 1C 07 7C 4D

```
     1<
 2:   39>SACKP|[Rsp][S:00001]|06|80             |Cmd 0|24| 0|00110000|254/Man0/Dev249/5 PAs/Hart7/Tx1/Sw2/Hw
                                                                      MinPArsp:5/MaxNumDVs:0/CfgChCnt:0/ExtD
                                                                      ManuID:0x00E0/LabDistID:00E1/Profile:0
     0<                                                         Cold Start, More Status
 3: 6166>LSTXP|[Req][S:00002]|82|80 F9 03 04 05|Cmd 1| 0|           |F8|
     1<
 4:   36>LACKP|[Rsp][S:00002]|86|80 F9 03 04 05|Cmd 1| 7| 0|00010000|PV 1:7.5 meter|76|
     1<                                                         More Status
 5: 1866>LSTXP|[Req][S:00003]|82|80 F9 03 04 05|Cmd 2| 0|           |FB|
     1<
 6:   39>LACKP|[Rsp][S:00003]|86|80 F9 03 04 05|Cmd 2|10| 0|00010000|16 mA/75 %|F0|
     1<                                                         More Status
 7: 8044>LSTXP|[Req][S:00004]|82|80 F9 03 04 05|Cmd18|21|           |Tag:        /Descr:16 CH DESCRIPTOR/Da
     1<
 8:   39>LACKP|[Rsp][S:00004]|86|80 F9 03 04 05|Cmd18|23| 0|00010000|Tag:        /Descr:16 CH DESCRIPTOR/Da
     1<                                                         More Status
 9:41159>LSTXP|[Req][S:00005]|82|80 F9 03 04 05|Cmd18|21|           |Tag:HELLO   /Descr:16 CH DESCRIPTOR/Da
     1<
10:   38>LACKP|[Rsp][S:00005]|86|80 F9 03 04 05|Cmd18|23| 0|00010000|Tag:HELLO   /Descr:16 CH DESCRIPTOR/Da
     0<                                                         More Status
```

Hart IP | Monitoring active | Switch record off to stop monitoring.   000010

The package Portable Hart Slave includes all sources needed to represent the slave part of the Hart protocol. The package is written in standard C++ and does not use any direct connection to a system environment. Data link layer, application layer (command interpreter) and network management of the Hart protocol are implemented. The connection to the outside occurs via three interfaces: The User Interface, a Time Trigger and the HAL to the Uart interface.

I used the C# environment to debug the Hart slave code during development. In fact, it is not(!) a simulation that is used here. The firmware is simply embedded in a Windows environment that allows the code to run in real time(!). In this way, all functions of the implementation can be analyzed in detail. The analysis of the temporal processes takes place in the range of milliseconds.

The C# software (White Box Test) was developed to create a transparent user interface for visualizing the data and communication processes. Visual Studio 2022 and .NET 6.0 were used to keep the programming effort within limits.

The command interpreter is triggered from the C# environment, but this happens within a 'real' thread and not within a worker thread from .NET:

```
CommandInterpreter = new Thread(ExecuteCommandInterpreter);
CommandInterpreter.Priority = ThreadPriority.Highest;
CommandInterpreter.Start();
```

and in endless loop of the thread:

```
result = (EN_Bool)HartSlaveDLL.BAHASL_WasCommandReceived();
if (result == EN_Bool.TRUE8)
{
    // Simulate typical application
    Thread.Sleep(20);
    command = HartSlaveDLL.BAHASL_ExecuteCommandInterpreter();
```

Hart Slave C++ Code

User Interface

Public Functions

The following functions are realized in the module HartS_UartIface.cpp in the class `CUartSlave`. In the DLL interface for the test client the function names are preceeded by `BAHASL_`.

Declaration	Description
Operation	

`EN_Bool OpenChannel(`
`    TY_Word port_number_,`
`    EN_CommType type_);`
	The function allocates the selected com port if possible and starts its own working thread for accessing the Hart services. The port_number_ is limited to the range of 1 .. 254. The selected communication type (type_) should be UART in this version of the paket. The function returns TRUE8 if successful.
	In the present implementation only a single channel is possible. Thus no channel handle is required.

`void CloseChannel();`	It is required to call this function at least when the application is terminating.
Data Interface	

`void GetConstDataHart(`
`    TY_ConstDataHart* const_data_);`
	Copies constant data from the Hart slave area to the test application area.

`void SetConstDataHart(`
`    TY_ConstDataHart* const_data_);`
	Copies constant data from the application area to the Hart slave area.

`void GetDynDataHart(`
`    TY_DynDataHart* dyn_data_);`
	Copies dynamic data from the Hart slave area to the test application area.

`void SetDynDataHart(`
`    TY_DynDataHart* dyn_data_);`
	Copies dynamic data from the application area to the Hart slave area.

`void GetStatDataHart(`
`    TY_StatDataHart* stat_data_);`
	Copies static data from the Hart slave area to the test application area.

`void SetStatDataHart (`
`    TY_StatDataHart* stat_data_);`
	Copies static data from the application area to the Hart slave area.

Command Interpreter	
`EN_Bool WasCommandReceived();`	The function returns FB_Bool::TRUE8 if the Hart protocol has recently (a few milliseconds ago) received a command.
`TY_Word ExecuteCommandInterpreter();`	This function calls the command interpreter in the slave to process any new data. If the command was recognized and executed, the function returns the number of the command. If this was not the case, the value 0xffff is returned.
Encoding	

`void PutInt8(`
`    TY_Byte  data_,`
`    TY_Byte  offset_,`
`    TY_Byte* data_ref_);`
	Insert an integer 8 into the byte array buffer pointed to by data_ref_ starting at the position offset_.

`void PutInt16(`
`    TY_Word   data_,`
`    TY_Byte   offset_,`
`    TY_Byte*  data_ref_,`
`    EN_Endian endian_);`
	Insert an integer 16 into the byte array buffer pointed to by data_ref_ starting at the position offset_. Start with the most significant byte if endian is MSB_FIRST(0), which is the Hart standard.

void PutInt24(
    TY_DWord data_,
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Insert an integer 24 into the byte array buffer pointed to by data_ref_ starting at the position offset_. Start with the most significant byte if endian is MSB_FIRST(0), which is the Hart standard.

void PutInt32(
    TY_DWord data_,
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Insert an integer 32 into the byte array buffer pointed to by data_ref_ starting at the position offset_. Start with the most significant byte if endian is MSB_FIRST(0), which is the Hart standard.

void PutInt64(
    TY_DWord data_,
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Insert an integer 64 into the byte array buffer pointed to by data_ref_ starting at the position offset_. Start with the most significant byte if endian is MSB_FIRST(0), which is the Hart standard.

void PutFloat(
    TY_Float data_,
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Insert a single precision IEEE 754 float value into the byte array buffer pointed to by data_ref_ starting at the position offset. Start with the most significant byte if endian is MSB_FIRST(0), which is the Hart standard.

void PutDFloat(
    TY_DFloat data_,
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Insert a double precision IEEE 754 float value into the byte array buffer pointed to by dataRef starting at the position offset. Start with the most significant byte if endian is MSB_FIRST(0), which is the Hart standard.

void PutPackedASCII(
    TY_Byte* asc_string_ref_,
    TY_Byte asc_string_len_,
    TY_Byte offset_,
    TY_Byte* data_ref_);
	Insert a string (asc_string_ref_) of the length of asc_string_len_ in packed ASCII format into the byte array buffer pointed to by data_ref_ starting at the position offset_. It is recommented that asc_string_len_ is an ordinary multiple of 4.

void PutOctets(
    TY_Byte* stream_ref_,
    TY_Byte stream_len_,
    TY_Byte offset_,
    TY_Byte* data_ref_);
	Copy a number of stream_len_ bytes into the byte array buffer pointed to by data_ref_ starting at the position offset_.

void PutString(
    TY_Byte* string_ref_,
    TY_Byte string_max_len_,
    TY_Byte offset_,
    TY_Byte* data_ref_);
	Copy a string from string_ref_ to data_ref_. The actual number of characters stored cannot be greater than string_max_len_. If the string contains a null, the last character saved is a null character if this does not exceed the string_max_len_ limit.

Decoding	

TY_Byte PickInt8(
    TY_Byte offset_,
    TY_Byte* data_ref_);
	Return the value of the byte in the byte array buffer pointed to by data_ref_ from the position offset_.

TY_Word PickInt16(
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Return the value of the integer 16 from the byte array buffer pointed to by data_ref_ from the position offset_. Assume that the most significant byte is the first if endian is MSB_FIRST(0), which is the Hart standard.

TY_DWord PickInt24(
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Return the value of the integer 24 from the byte array buffer pointed to by dtaRef at the position offset. Assume that the most significant byte is the first if endian is MSB_FIRST(0), which is the Hart standard.

TY_DWord PickInt32(
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Return the value of the integer 32 from the byte array buffer pointed to by data_ref_ from the position offset_. Assume that the most significant byte is the first if endian is MSB_FIRST(0), which is the Hart standard.

TY_UInt64 PickInt64(
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Return the value of the integer 64 from the byte array buffer pointed to by data_ref_ from the position offset_. Assume that the most significant byte is the first if endian is MSB_FIRST(0), which is the Hart standard.

TY_Float PickFloat(
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Return the value of the single precision IEEE754 number from the byte array buffer pointed to by data_ref_ from the position offset_. Assume that the most significant byte is the first if endian is MSB_FIRST(0), which is the Hart standard.

TY_DFloat PickDFloat(
    TY_Byte   offset_,
    TY_Byte* data_ref_,
    EN_Endian endian_);
	Return the value of the double precision IEEE754 number from the byte array buffer pointed to by data_ref_ from the position offset_. Assume that the most significant byte is the first if endian is MSB_FIRST(0), which is the Hart standard.

void PickPackedASCII(
    TY_Byte* string_ref_,
    TY_Byte string_len_,
    TY_Byte offset_,
    TY_Byte* data_ref_);
	Generate a string and copy it to the buffer pointed to by sb. The final string should have the length string_len. The packedASCII source is a set of bytes in the byte array buffer pointed to by data_ref_, starting at index offset_.
	Note: The string length has to by a multiple of 4 while the number of packedASCII bytes is a multiple of 3.

void PickOctets(
    TY_Byte* stream_ref_,
    TY_Byte stream_len_,
    TY_Byte offset_,
    TY_Byte* data_ref_);
	Copy a number (numOctets) of bytes from the byte array buffer pointed to by dataSource to the user buffer pointed to by dataDestination.

void PickString(
    TY_Byte* string_ref_,
    TY_Byte string_max_len_,
    TY_Byte offset_,
    TY_Byte* data_ref_);
	The function reads a string from a buffer (data_ref_) starting at index offset_ and stores the characters in string_ref_. The string buffer is read from until a null character appears or string_max_len_ is reached. If possible, the null character is also saved.

Internal	
void FastCyclicHandler(TY_Word time_ms_);	Although this function is not accessible to the test client, it is required for the operation of the Hart protocol. The function must be called by a separate task approximately every millisecond to enable timing in the communication. The time_ms parameter indicates how many milliseconds have passed since the last call. Usually this should be a value of 1 in most cases.

Data Interface

The data interface provides three different types of data that can be written or read by the user. A structure is provided for each data type, which can be found in the file WbHartS_Structures.h.

Constant data does not change. In most systems it is stored in flash memory and cannot be written.

Dynamic data is data that can always change. This includes measured values and status information.

Static data is used to configure a device. It is usually changed by external access. Whenever static data is changed, the configuration change flag must be set in Hart and the configuration change counter in Hart must be incremented.

Coding Considerations

Microcontrollers which are used today for HART devices are at least 16 Bit microcontrollers. Otherwise the complexity of the measurement and number of parameters could not be managed.

☞ Low amount of memory.

The amount of memory is always critical because software kind of behaves like an ideal gas. It uses to fill the given space. Nevertheless, the coding of the Hart Slave was done as carefully as possible regarding the amount of flash memory and RAM.

☞ The user needs source code.

The Hart Protocol requires a strict timing specially for burst mode support and the primary and secondary master time slots. To provide the optimum transparency to the user to allow all kinds of debugging and to give the opportunity to optimize code in critical sections, the Hart Slave Firmware is not realized as a library but delivered as source code.

Hardware Abstraction

☞ *OSAL is including the HAL.*

A Hardware Abstraction Layer is needed to design the interface of a software component independent from the hardware platform. In this very small interface of the Hart master a distinction of HAL and OSAL was not made. Therefore only an Operating System Abstraction Layer is defined which is covering all the needs of an appropriate HAL.

List of Files

Category	Name	Description
02-Code		
01-Common	`OSAL.h`	The Operating System Abstraction Layer is the top header. This is where the central connection to the respective hardware or software platform takes place. The header OSAL.h can only exist once, while a special implementation (OSAL.cpp) exists for each specific hardware or software.
	`HartCoding.cpp/h`	This module combines functions that carry out the encoding and decoding of communication primitives and data objects.
	`HartFrame.cpp/h`	The hart frame is a construct used to collect all information which is needed to encode and decode data of so called service primitives like responses and requests, which are finally octet streams.
	`HartLib.h`	Some classes for the definition of HART constants.
02-Code\01-Common		
01-Interface	`HartSlaveIface.cpp/h`	This is where the actual interface of the master implementation is located, which would also have to be integrated into an embedded system. The version with the DLL is only intended for testing under Windows. You can find a detailed description of the provided functions in the 'Public Functions' chapter.
	`WbHartS_Structures.h`	This file contains structures which are accessed at the outer interface as well as in some modules in the master kernel.
	`WbHartS_TypeDefs.h`	This file contains type definitions which are used in all modules in the Hart master kernel.
	`WbHartUser.h`	Limits applied by the user of the hart master software.
	`HartDevice.cpp/h`	This module is nearly empty and subject to be removed.
02-AppLayer	`HartChannel.cpp/h`	The channel manages a communication interface and the associated propperties. The channel also uses services to conduct Hart commands.
	`HartBurst.cpp/h`	Handling of the burst mode from the perspective of the application.
	`AnyCommandIntp.cpp/h`	Any command interpreter for common practice and user commands.
03-Layer7	`HartService.cpp/h`	In simple terms, a service executes a Hart command by passing a request to Layer2 of the Hart protocol. In doing so, it returns a handle to the caller, with which the calling program can check the status. A service is only considered completed when the caller has read the response (e.g. FetchConfirmation).
	`HartData.cpp/h`	Data defined for the Hart commands.
	`UniCommandIntp.cpp/h`	Universal commands interpreter.

04-Layer2 01-Uart 02-HartIp	**HSuartLayer2.cpp/h and HSipLayer2.cpp/h**	This module implements the entire state machine of the Hart communication protocol (CHartSM) including the state machines for sending (CTxSM) and receiving (CRxSM) bytes.
	HSuartMacPort.h and HSipMacPort.h	The interface to the MAC port is relatively narrow and can be defined generically. However, the implementation depends on the hardware and software environment. That's why there is only a header at this point, while the files HMuartMacPort.cpp and HMipMacPort.cpp can be found in the OSAL area.
	HSuartProtocol.cpp/h and HSipProtocol.cpp/h	This protocol layer controls the UART interface on the lower level and calls the higher status machines when necessary (events). After this call, a ToDo Part occurs, which in turn affects the Uart or HartIp interface.
	Monitor.h	The same applies to the Monitor function as to the MacPort. At this point only the interface can be defined. The implementation takes place in the specific part.
02-Code\02-Specific\01-WinDLL		
01-Shell	**BaHartSlave.cpp/h**	The implementation for the calls to the Windows DLL is located here. In practice, it is just a shell through which the functions in the CUartMaster module are called.
02-OSAL 01-Uart 02-HartIp	**HSuartMacPort.cpp and HSipMacPort.cpp**	The Execute method is called directly by the fast cyclic handler. This basically drives all status machines in the Hart implementation. Here too, the method is divided into an Event handler and a ToDo handler.
	Monitor.cpp	On the one hand, there are methods that are mapped to the interface of the Windows DLL. In addition, there are a number of functions that are included with the kernel functions. Since this module is so small overall, the methods were not implemented in two different files.
	OSAL.cpp	The Operating System Abstraction Layer maps general functions to the operating system.
	WinSystem.cpp/h	The OSAL concept cannot be applied to all functions that are required. These functions were implemented in the code of this module.

System Requirements

It is difficult to estimate the system requirements for targets based on different micro controllers and different development environments. The following is therefore giving a very rough scenario for the target system estimated resources.

Item	Requirement/Size	Comment
RAM	**32k**	Depends very much on the addressing structure of the controller and the used compiler and linker.
ROM (Flash)	**100k**	
Timing	**1-2 ms Timer interrupt**	2 ms is the minimum requirement, 1 ms would be much better.
	50 ms cyclic call from task level	This is needed to run the command interpreter.
I/O	**UART and Hart MODEM Rx and Tx functions**	Carrier detection would be helpful but is not required.

System	**Simple math +-*/
	memcpy()
	memset()
	memcmp()**
		Only a few standard library functions are required. There is no special need for multi tasking, messaging or semaphores.

	1 ms timing resolution	

Table 1: Embedded System Requirements

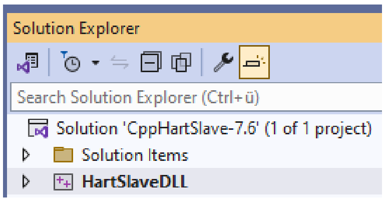
Coding Conventions

Regarding this issue, I have only defined some formats that makes the scope of a label clearer. It's just to make the code easier to read. This simple type of coding convention can be used in both C++ and C#.

Snake Case			
`local_variable`	**`function_param_`**	**`m_member_var`**	**`mo_member_object`**
Variable with local scope	A function parameter has a tailing underscore	Basic type private member variable	Complex object member
`s_member_var`	**`so_member_object`**		
Basic type static private member variable	Complex static object member		
Pascal Case			
`PublicVariable`	**`PublicObject`**	**`AnyMethod`**	
Variable with public or internal scope	Object with public or internal scope	No difference between public and private	

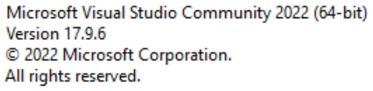
Visual Studio 2022

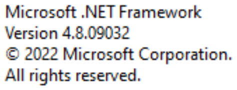
Test Environment


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'CppHartSlave-7.6' (1 of 1 project)
Solution Items
HartSlaveDLL

There are only one project in this solution. The C++ Hart Slave is encapsulated in the HartSalveDLL project.
The solution is directly in the path on which you copied the package to.

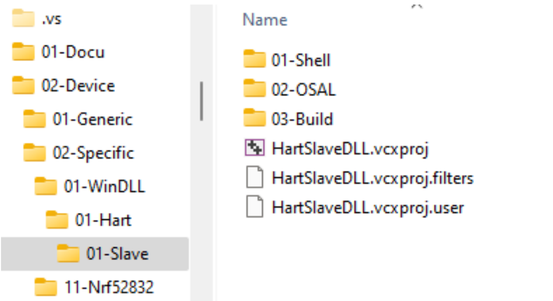
Prerequisites


Microsoft Visual Studio Community 2022 (64-bit)
Version 17.9.6
© 2022 Microsoft Corporation.
All rights reserved.

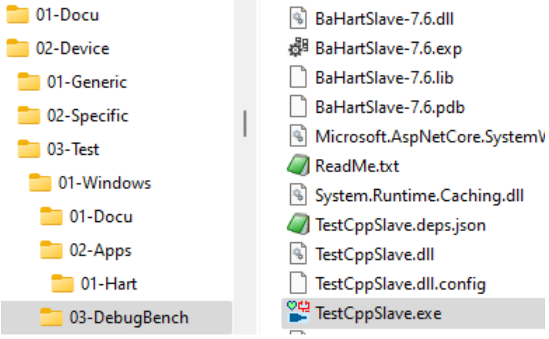

Microsoft .NET Framework
Version 4.8.09032
© 2022 Microsoft Corporation.
All rights reserved.

The solution must be opened with VS 2022. However, the community version is sufficient. There are no further requirements.

Development Directory Structure

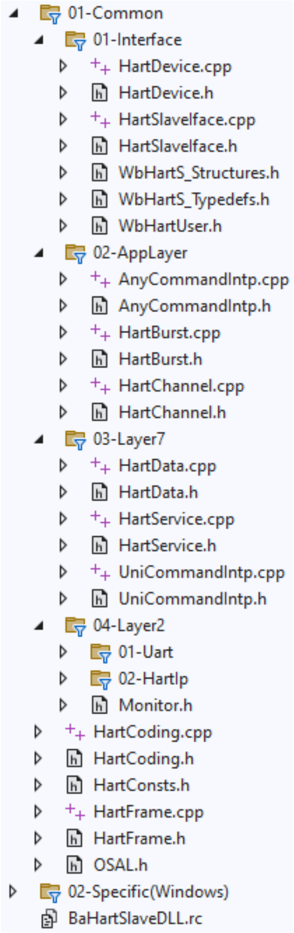


The project for the Hart Slave in C++ can be found in the directory:
.\02-Device\02-Specific\01-WinDLL\01-Hart\01-Slave.
However, most of the C++ sources used are located in the directory .\02-Device\01-Generic\ and its subdirectories.



The test software is only be found as executable in the path 03-DebugBech. The executable file TestCppSlave.exe and the simulation DLL BaHartSlave-7.6.dll are both located here. When you start debugging the executable ist started and loading the dll which is respresenting the slave device.

Project Structure



```
01-Common
    01-Interface
        HartDevice.cpp
        HartDevice.h
        HartSlaveIface.cpp
        HartSlaveIface.h
        WbHartS_Structures.h
        WbHartS_Typedefs.h
        WbHartUser.h
    02-AppLayer
        AnyCommandIntp.cpp
        AnyCommandIntp.h
        HartBurst.cpp
        HartBurst.h
        HartChannel.cpp
        HartChannel.h
    03-Layer7
        HartData.cpp
        HartData.h
        HartService.cpp
        HartService.h
        UniCommandIntp.cpp
        UniCommandIntp.h
    04-Layer2
        01-Uart
        02-HartIp
        Monitor.h
    HartCoding.cpp
    HartCoding.h
    HartConsts.h
    HartFrame.cpp
    HartFrame.h
    OSAL.h
02-Specific(Windows)
BaHartSlaveDLL.rc
```

The project structure is very similar to the directory structure. Here too there is a strict distinction between generic area and specific area.

The specific contents of the files are described in more detail in the list below.

In contrast to the last published documentation, there is one significant difference. The data link layer is divided into the areas Uart and HartIp. The same applies to the Mac port in the OSAL directory.

Getting Started

1. Unzip the file hart-master-slave-c++-demo-7.6.1.zip into a directory of your choice. For getting the required password please send an e-mail to: Hart@walter-borst.de.
2. Open the solution .\03-Slave\CppHartSlave-7.6.sln with Visual Studio 2022. It has to be 2022. Other versions are not supported yet. Unless you have 2022 not installed on your computer. You can download it from microsoft: https://visualstudio.microsoft.com/de/downloads/.
3. The community version is sufficient enough and free of charge.
4. Perform a 'Build All'.
5. Start debugging and investigate the source code

Test Interface

The Windows test adapter is a software developed in C#. This test adapter uses a Windows DLL in which the Hart Master is embedded. The DLL implements the HART Protocol, whose firmware was written in C++ for real time requirements.

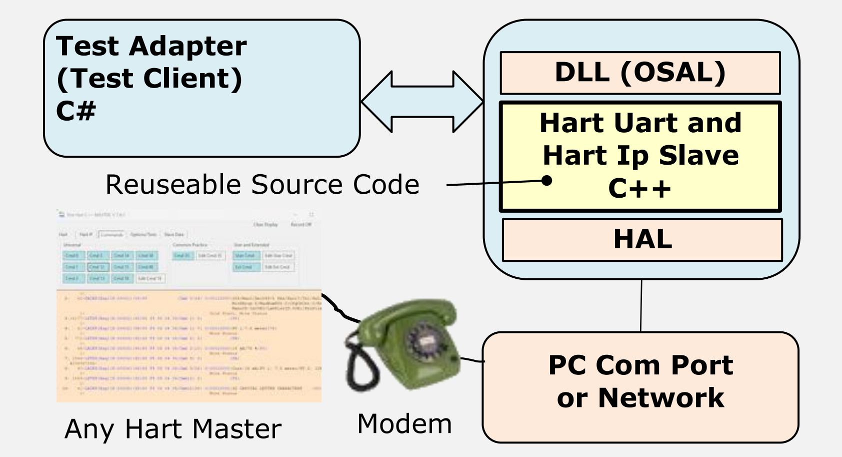
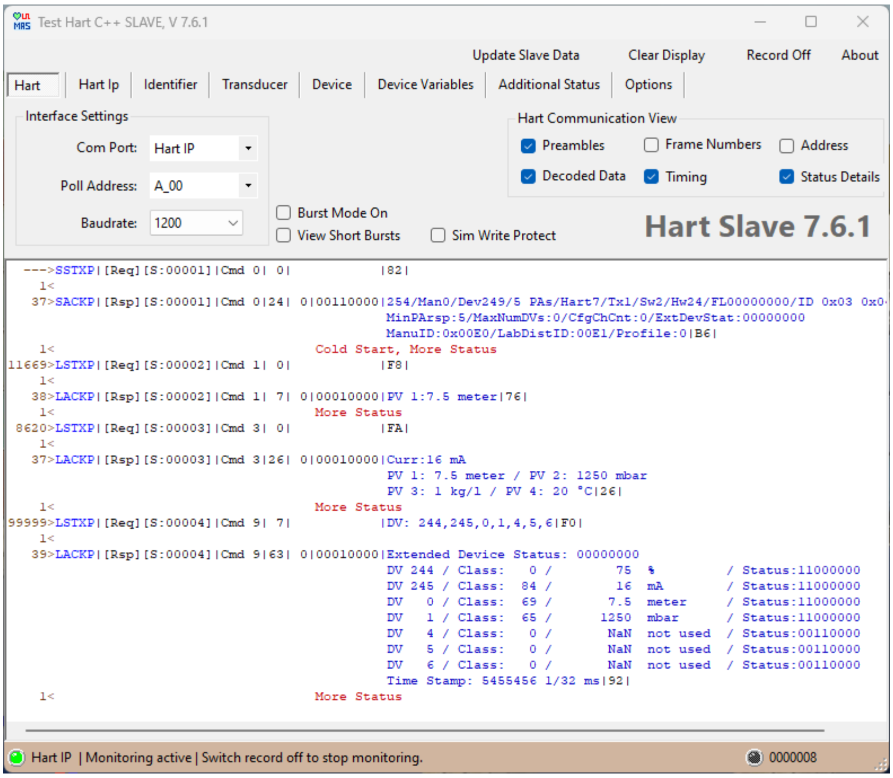


Figure 1: Architecture of the Test Environment

The executable file for the test adapter is located at the following location:

`.\03-Slave\03-Test\01-Windows\03-DebugBench\BaTestHartSlave.exe`



Interface Settings

Com Port: Hart IP

Poll Address: A_00

Baudrate: 1200

□ Burst Mode On

□ View Short Bursts

□ Sim Write Protect

Hart Communication View

☑ Preambles □ Frame Numbers □ Address

☑ Decoded Data ☑ Timing ☑ Status Details

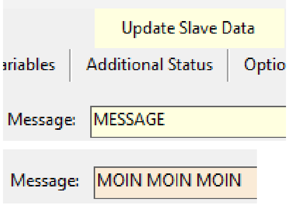
```
  --->SSTXP|[Req][S:00001]|Cmd 0| 0|          |82|
    1<
   37>SACKP|[Rsp][S:00001]|Cmd 0|24| 0|00110000|254/Man0/Dev249/5 PAs/Hart7/Tx1/Sw2/Hw24/FL00000000/ID 0x03 0x0
                                               MinPArsp:5/MaxNumDVs:0/CfgChCnt:0/ExtDevStat:00000000
                                               ManuID:0x00E0/LabDistID:00E1/Profile:0|B6|
    1<                                 Cold Start, More Status
11669>LSTXP|[Req][S:00002]|Cmd 1| 0|          |F8|
    1<
   38>LACKP|[Rsp][S:00002]|Cmd 1| 7| 0|00010000|PV 1:7.5 meter|76|
    1<                                 More Status
 8620>LSTXP|[Req][S:00003]|Cmd 3| 0|          |FA|
    1<
   37>LACKP|[Rsp][S:00003]|Cmd 3|26| 0|00010000|Curr:16 mA
                                               PV 1: 7.5 meter / PV 2: 1250 mbar
                                               PV 3: 1 kg/l / PV 4: 20 °C|26|
    1<                                 More Status
99999>LSTXP|[Req][S:00004]|Cmd 9| 7|          |DV: 244,245,0,1,4,5,6|F0|
    1<
   39>LACKP|[Rsp][S:00004]|Cmd 9|63| 0|00010000|Extended Device Status: 00000000
                                               DV 244 / Class:   0 /        75  %        / Status:11000000
                                               DV 245 / Class:  84 /        16  mA       / Status:11000000
                                               DV   0 / Class:  69 /       7.5  meter    / Status:11000000
                                               DV   1 / Class:  65 /      1250  mbar     / Status:11000000
                                               DV   4 / Class:   0 /       NaN  not used / Status:00110000
                                               DV   5 / Class:   0 /       NaN  not used / Status:00110000
                                               DV   6 / Class:   0 /       NaN  not used / Status:00110000
                                               Time Stamp: 5455456 1/32 ms|92|
    1<                                 More Status
```

● Hart IP | Monitoring active | Switch record off to stop monitoring. 0000008

When the executable file is started, the container DLL for the slave is automatically loaded. The work surface is divided into two halves. Settings are made in the tab area, while the lower area is reserved for a monitor that shows the communication process. While the following tabs mostly deal with the slave data, the inputs in the interface have a fairly direct effect on the running software. For example, it is possible to activate burst mode without having to use the Hart command 109.

Screenshot 1: The Tab 'Hart'

Data Exchange

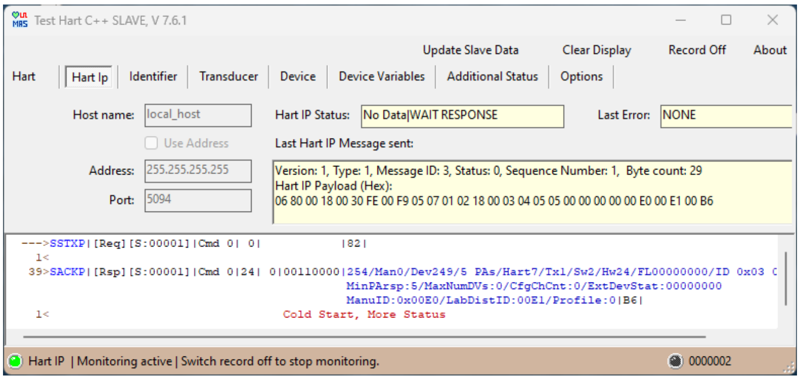


Message: MESSAGE

Message: MOIN MOIN MOIN

The following tabs deal with the transmitter data. If this data is edited, this is indicated by a yellow color. The menu button also turns yellow and must be clicked for the change to take effect in the slave.

If a parameter is changed by a master connected to the slave, this change appears in the display and the parameter in question is colored red



Host name: local_host

Hart IP Status: No Data|WAIT RESPONSE

Last Error: NONE

□ Use Address

Last Hart IP Message sent:

Address: 255.255.255.255

Port: 5094

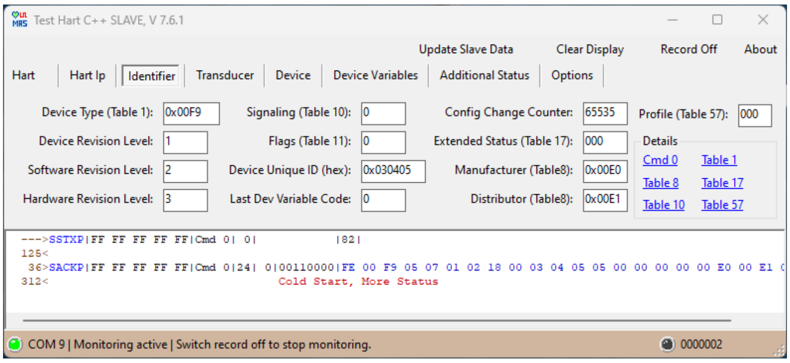
Version: 1, Type: 1, Message ID: 3, Status: 0, Sequence Number: 1, Byte count: 29
Hart IP Payload (Hex):
06 80 00 18 00 30 FE 00 F9 05 07 01 02 18 00 03 04 05 05 00 00 00 00 00 E0 00 E1 00 B6

```
 --->SSTXP|[Req][S:00001]|Cmd  0|  0|           |82|
   1<
  39>SACKP|[Rsp][S:00001]|Cmd  0|24|  0|00110000|254/Man0/Dev249/5 PAs/Hart7/Tx1/Sw2/Hw24/FL00000000/ID 0x03 C
                                                 MinPArsp:5/MaxNumDVs:0/CfgChCnt:0/ExtDevStat:00000000
                                                 ManuID:0x00E0/LabDistID:00E1/Profile:0|B6|
   1<                                    Cold Start, More Status
```

Hart IP | Monitoring active | Switch record off to stop monitoring. 0000002

If Hart IP is used, additional parameters are needed to connect to the slave. However, currently the demo version works on localhost.

Screenshot 2: The Tab 'Hart Ip'



Device Type (Table 1): 0x00F9

Signaling (Table 10): 0

Config Change Counter: 65535

Profile (Table 57): 000

Device Revision Level: 1

Flags (Table 11): 0

Extended Status (Table 17): 000

Software Revision Level: 2

Device Unique ID (hex): 0x030405

Manufacturer (Table8): 0x00E0

Hardware Revision Level: 3

Last Dev Variable Code: 0

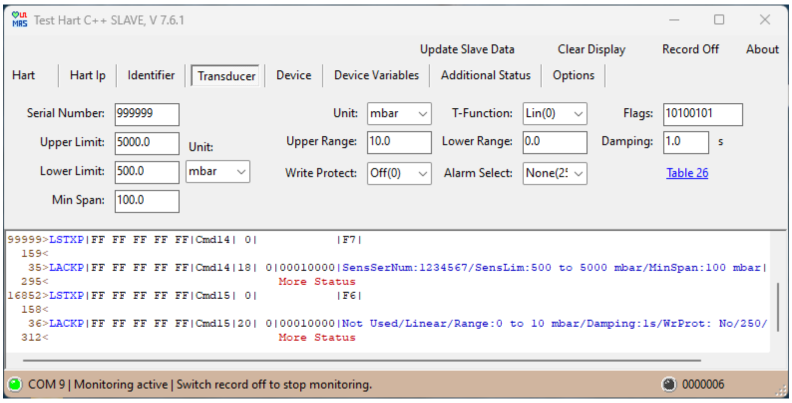
Distributor (Table8): 0x00E1

```
 --->SSTXP|FF FF FF FF FF|Cmd  0|  0|           |82|
 125<
  36>SACKP|FF FF FF FF FF|Cmd  0|24|  0|00110000|FE 00 F9 05 07 01 02 18 00 03 04 05 05 00 00 00 00 00 E0 00 E1 C
 312<                                    Cold Start, More Status
```

COM 9 | Monitoring active | Switch record off to stop monitoring. 0000002

The tab 'Identifier' mainly deals with data related to command 0.

Screenshot 3: The Tab 'Identifier'



Serial Number: 999999

Unit: mbar

T-Function: Lin(0)

Flags: 10100101

Upper Limit: 5000.0

Unit: mbar

Upper Range: 10.0

Lower Range: 0.0

Damping: 1.0 s

Lower Limit: 500.0

Write Protect: Off(0)

Alarm Select: None(2!

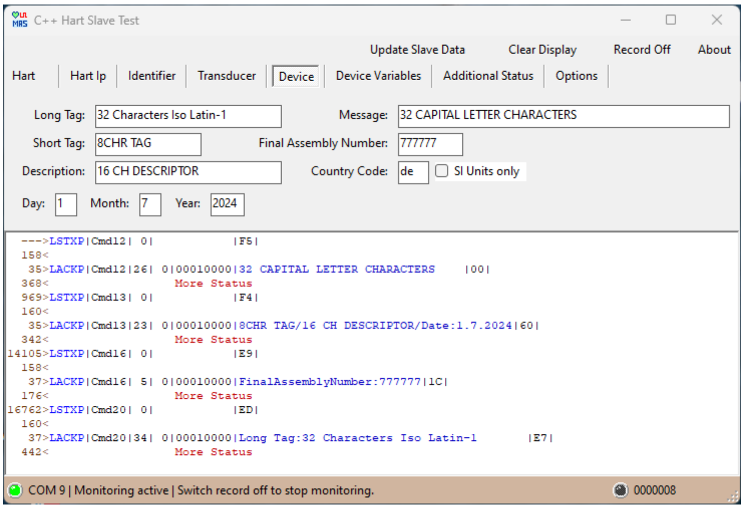
Min Span: 100.0

```
99999>LSTXP|FF FF FF FF FF|Cmd14|  0|           |F7|
  159<
   35>LACKP|FF FF FF FF FF|Cmd14|18|  0|00010000|SensSerNum:1234567/SensLim:500 to 5000 mbar/MinSpan:100 mbar|
  295<                                 More Status
16852>LSTXP|FF FF FF FF FF|Cmd15|  0|           |F6|
  158<
   36>LACKP|FF FF FF FF FF|Cmd15|20|  0|00010000|Not Used/Linear/Range:0 to 10 mbar/Damping:1s/WrProt: No/250/
  312<                                 More Status
```

COM 9 | Monitoring active | Switch record off to stop monitoring. 0000006

The tab 'Transducer' mainly deals with data related to the commands 14 and 15.

Screenshot 4: The Tab 'Transducer'



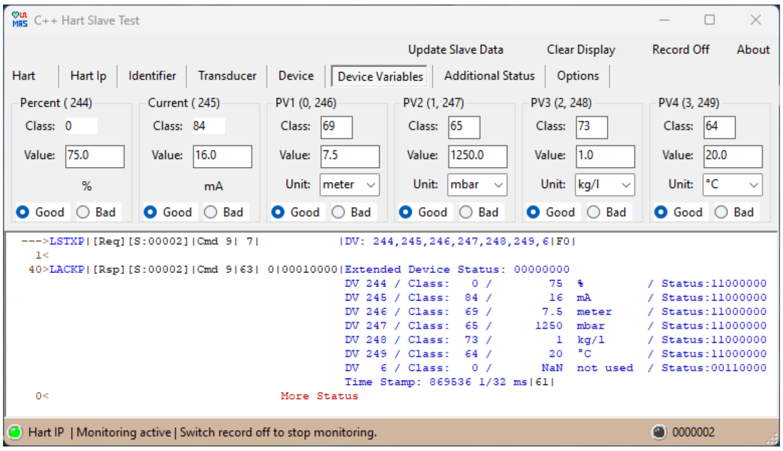
Long Tag: 32 Characters Iso Latin-1
Message: 32 CAPITAL LETTER CHARACTERS
Short Tag: 8CHR TAG
Final Assembly Number: 777777
Description: 16 CH DESCRIPTOR
Country Code: de ☐ SI Units only
Day: 1 Month: 7 Year: 2024

```
  --->LSTXP|Cmd12| 0|             |F5|
  158<
   35>LACKP|Cmd12|26| 0|00010000|32 CAPITAL LETTER CHARACTERS    |00|
  368<                      More Status
  969>LSTXP|Cmd13| 0|             |F4|
  160<
   35>LACKP|Cmd13|23| 0|00010000|8CHR TAG/16 CH DESCRIPTOR/Date:1.7.2024|60|
  342<                      More Status
14105>LSTXP|Cmd16| 0|             |E9|
  158<
   37>LACKP|Cmd16| 5| 0|00010000|FinalAssemblyNumber:777777|1C|
  176<                      More Status
16762>LSTXP|Cmd20| 0|             |ED|
  160<
   37>LACKP|Cmd20|34| 0|00010000|Long Tag:32 Characters Iso Latin-1       |E7|
  442<                      More Status
```

COM 9 | Monitoring active | Switch record off to stop monitoring. 0000008

Screenshot 5: The Tab 'Device'

The tab 'Device' mainly deals with data related to the commands 12, 13, 16 and 20.



Percent (244)	Current (245)	PV1 (0, 246)	PV2 (1, 247)	PV3 (2, 248)	PV4 (3, 249)
Class: 0	Class: 84	Class: 69	Class: 65	Class: 73	Class: 64
Value: 75.0	Value: 16.0	Value: 7.5	Value: 1250.0	Value: 1.0	Value: 20.0
%	mA	Unit: meter	Unit: mbar	Unit: kg/l	Unit: °C
● Good ○ Bad	● Good ○ Bad	● Good ○ Bad	● Good ○ Bad	● Good ○ Bad	● Good ○ Bad

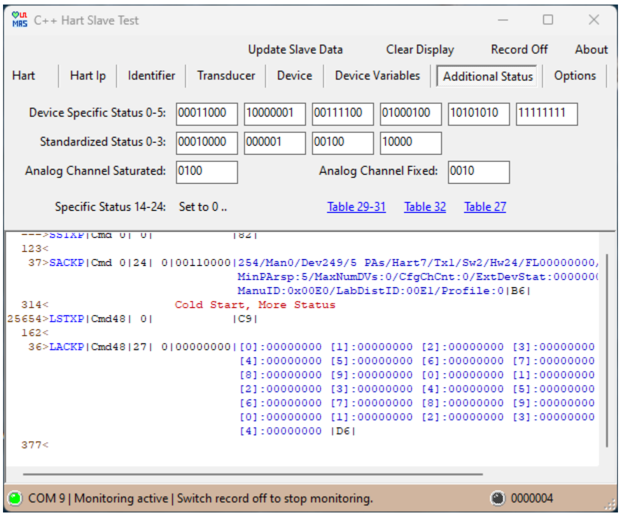
```
  --->LSTXP|[Req][S:00002]|Cmd 9| 7|          |DV: 244,245,246,247,248,249,6|F0|
    1<
   40>LACKP|[Rsp][S:00002]|Cmd 9|63| 0|00010000|Extended Device Status: 00000000
                                               DV 244 / Class:  0 /     75  %        / Status:11000000
                                               DV 245 / Class: 84 /     16  mA       / Status:11000000
                                               DV 246 / Class: 69 /    7.5  meter    / Status:11000000
                                               DV 247 / Class: 65 /   1250  mbar     / Status:11000000
                                               DV 248 / Class: 73 /      1  kg/l     / Status:11000000
                                               DV 249 / Class: 64 /     20  °C       / Status:11000000
                                               DV   6 / Class:  0 /    NaN  not used / Status:00110000
                                               Time Stamp: 869536 1/32 ms|61|
    0<                                    More Status
```

Hart IP | Monitoring active | Switch record off to stop monitoring. 0000002

Screenshot 6: The Tab 'Device Variables'

The 'Device Variables' tab provides access to the data needed to implement device variables. Currently, only device variable codes in the range 244-249 and 0..3 are accepted. These are the only required device variables.

Of course, further variables for the user are possible at any time.



Device Specific Status 0-5: 00011000 10000001 00111100 01000100 10101010 11111111
Standardized Status 0-3: 00010000 000001 00100 10000
Analog Channel Saturated: 0100
Analog Channel Fixed: 0010
Specific Status 14-24: Set to 0 ..

```
  --->SSTXP|Cmd 0| 0|             |82|
  123<
   37>SACKP|Cmd 0|24| 0|00110000|254/Man0/Dev249/5 PAs/Hart7/Tx1/Sw2/Hw24/FL00000000,
                                 MinPArsp:5/MaxNumDVs:0/CfgChCnt:0/ExtDevStat:000000(
                                 ManuID:0x00E0/LabDistID:00E1/Profile:0|B6|
  314<                  Cold Start, More Status
25654>LSTXP|Cmd48| 0|             |C9|
  162<
   36>LACKP|Cmd48|27| 0|00000000|[0]:00000000 [1]:00000000 [2]:00000000 [3]:00000000
                                 [4]:00000000 [5]:00000000 [6]:00000000 [7]:00000000
                                 [8]:00000000 [9]:00000000 [0]:00000000 [1]:00000000
                                 [2]:00000000 [3]:00000000 [4]:00000000 [5]:00000000
                                 [6]:00000000 [7]:00000000 [8]:00000000 [9]:00000000
                                 [0]:00000000 [1]:00000000 [2]:00000000 [3]:00000000
                                 [4]:00000000 |D6|
  377<
```

COM 9 | Monitoring active | Switch record off to stop monitoring. 0000004

Screenshot 7: The Tab 'Additional Status'

This is about command 48. As already mentioned elsewhere, the slave manages the responses to the two masters separately and stores which response it has sent to a master. If something changes in the additional status, the software knows which master it affects because it can compare it with the copies.

Appendix

Internet Links

Specification Documents	
HART Specifications	FieldComm Group
MODEMs	
RS 232 Modem	Microflex
USB Modem	Endress + Hauser
Viator USB Modem	Pepperl+Fuchs
Ethernet-APL	
Advanced Physical Layer	FieldComm Group
Ethernet - To the Field	Ethernet APL Organisation
HART-IP Developer Kit	FieldComm Group

Download Location

The software package described in this document can be downloaded via the following link:

https://github.com/BorstAutomation/Hart-Master-Slave-8.0.git

Legal Issues

Conformity

This software package was developed to the best of my knowledge and my belief. The basis is the specifications of the Hart Communication Foundation in version 7.9.

However, it cannot be guaranteed that the software included in this package meets the HCF specifications in all required respects.

It is only possible to prove the conformity of this software after the user has integrated the software into his device and commissions HCF or a certified company to carry out this test. Under no circumstances am I, Walter Borst, responsible for carrying out such tests. Nor am I responsible for correcting any deficiencies resulting from such a test.

Copyright

Kapitaen-Alexander-Strasse 39, 27472 Cuxhaven, GERMANY

Fon: +49 (0)4721 6985100, Fax: +49 (0)4721 6985102

E-Mail: Office@walter-borst.de

Home: https://walter-borst.de/hart-communication-software.html

No Warranty

Walter Borst expressly disclaims any warranty for the software package. This software package and related documents are provided "As Is".

By using this software package, the user agrees that no event shall Borst Automation or Walter Borst make responsible or liable for damages whatsoever. This includes, without limitation, damages for loss of business profits, loss due to business interruption, loss of business information, or any other pecuniary loss, arising out of the use of or the inability to use this software package.