Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

Hart Slave Stack C++ 7.6

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.

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.

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

Introduction

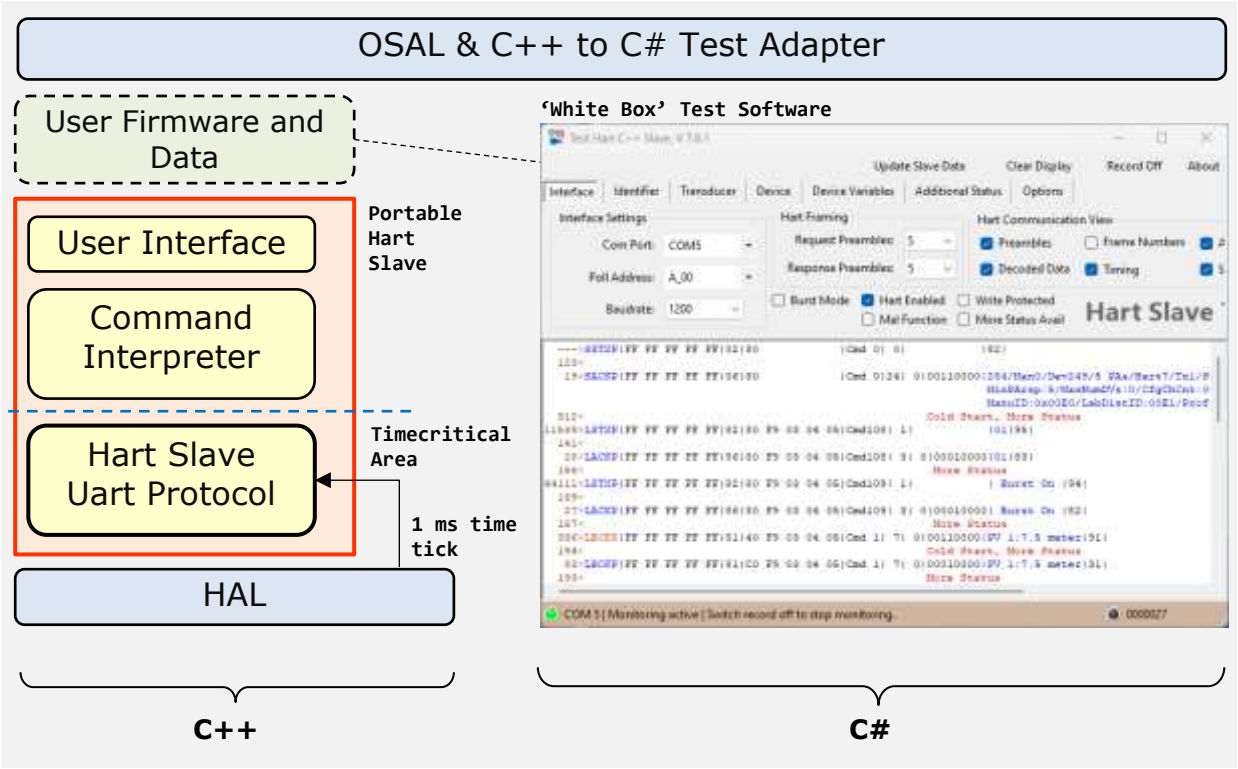
Implemented Commands

#	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.

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

Architecture



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();
```

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

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.

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

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.

Embedded 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

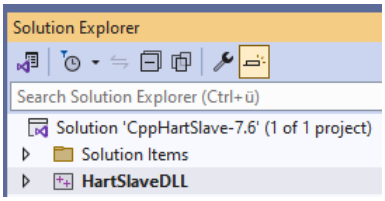
Regarding this issue, I have only defined some formats that makes the scope of a label clearer. It's just to make the code easier to read. This simple type of coding convention can be used in both C++ and C#.

Pascal 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		
Camel case			
PublicVariable	**PublicObject**	**AnyMethod**	
Variable with public or internal scope	Object with public or internal scope	No difference between public and private	

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

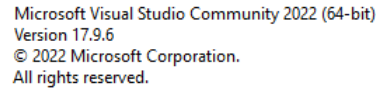
Visual Studio 2022

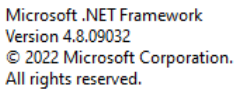
Test Environment



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.

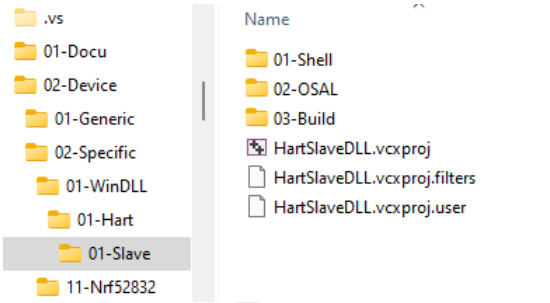
Prerequisites



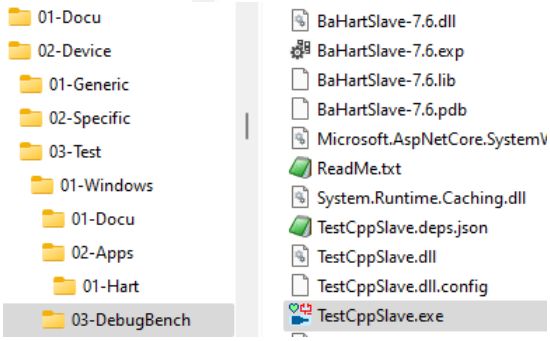


The solution must be opened with VS 2022. However, the community version is sufficient. There are no further requirements.

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.



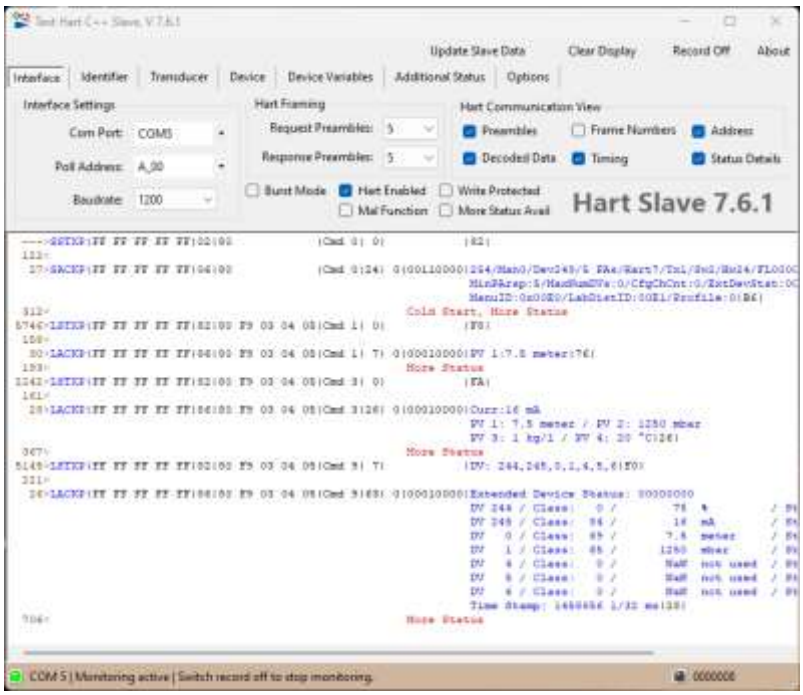
The test software is only be found as executable in the path 03-DebugBech. The executable file TestCppSlave.exe and the simulation DLL BaHartSlave-7.6.dll are both located here.
When you start debugging the executable ist started and loading the dll which is respresenting the slave device.

Getting Started

1. Unzip the file hart-slave-source-code-7.6.1.zip into a directory of your choice.
2. Open the solution 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

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

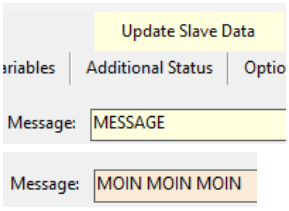
Test Interface



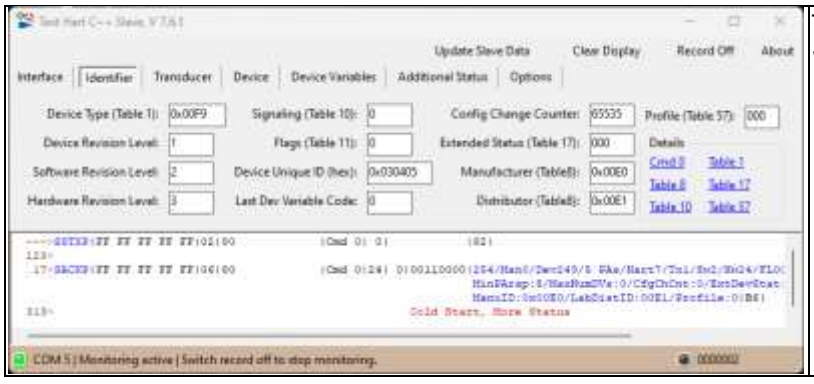
When the executable file is started, the container DLL for the slave is automatically loaded.
The work surface is divided into two halves.
Settings are made in the tab area, while the lower area is reserved for a monitor that shows the communication process.
While the following tabs mostly deal with 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 'Interface'

Data Exchange



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



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

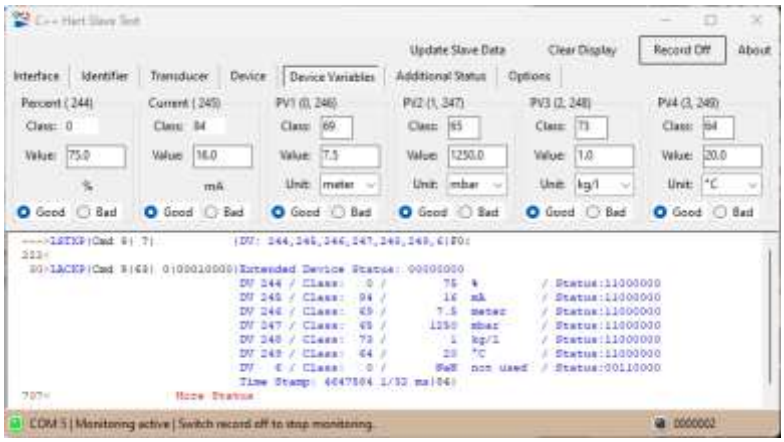
Screenshot 2: The Tab 'Identifier'

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

Screenshot 3: The Tab 'Transducer'

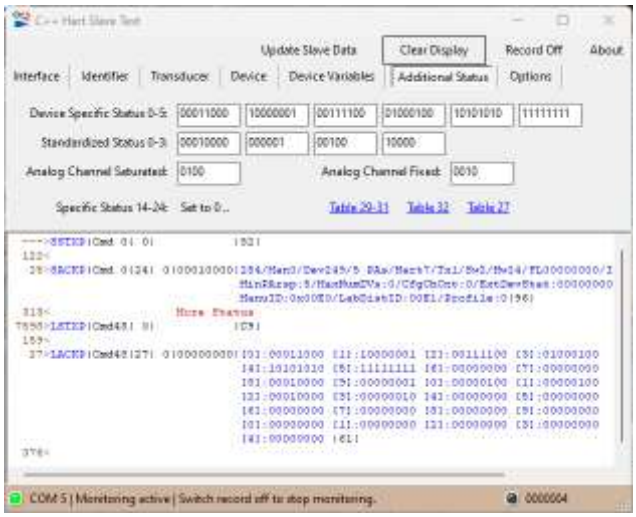
The tab 'Device' mainly deals with data related to the commands 12, 13, 15, 16 and 20.

Screenshot 4: The Tab 'Device'



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.

Screenshot 5: The Tab 'Device Variables'



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.

Screenshot 6: The Tab 'Additional Status'

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://www.borst-automation.com/downloads/hart-slave-source-code-7.6.1.zip

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

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: info@borst-automation.de

Home: https://www.borst-automation.de/

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.