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 Master Stack C++ 8.0E

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 to avoid memory leaks.
- 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 Services

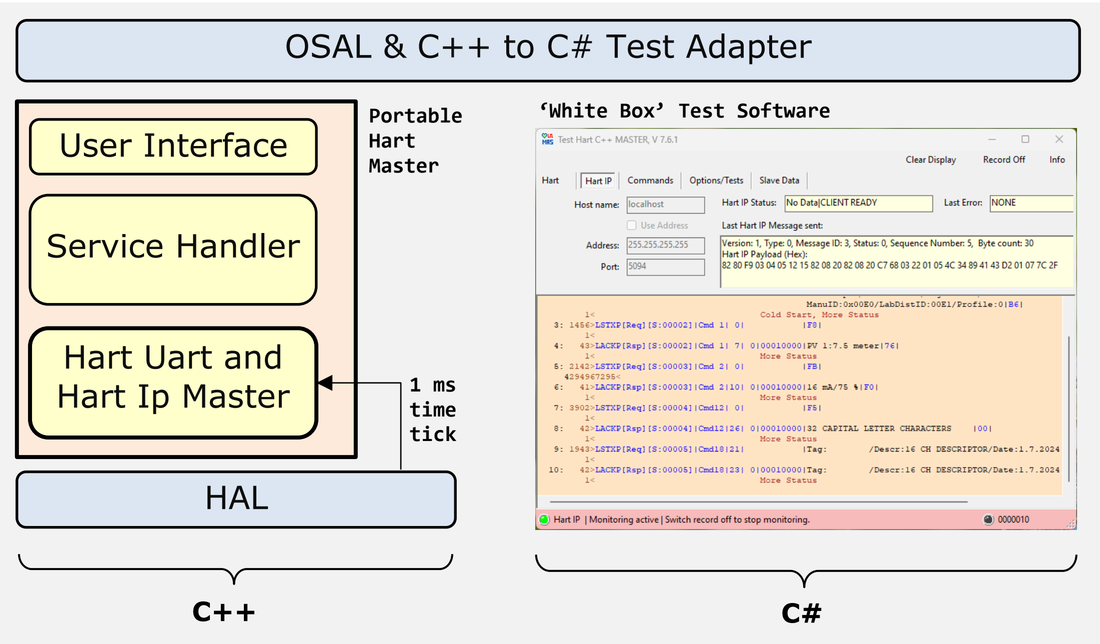
While a Hart slave typically executes Hart commands and sends cyclic data in the form of bursts, a Hart master usually has the task of retrieving Hart commands and processing the slave's responses. We are not talking about specific commands here, but rather about services that are there for all commands.

Two types of services are implemented for this purpose. With a blocking service, the called function waits until a response is received.

If a service is executed non-blockingly, the called function returns a handle that the calling program can use later to check whether the service has finished.

The cyclic data is a special feature, because it is sent by the slave on the hard level without a request. The handling of cyclic data has not yet been implemented in the existing code, but can be quickly retrofitted for customers who license the firmware.

Architecture



The package Portable Hart Master includes all modules needed to represent the master 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 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.

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/

In order to make the source code of the Hart Kernel visible, a simulation of an application is available. The simulation is carried out by integrating the Hart Kernel into a Windows library and controlling it via a 'Test Adapter'. This makes it possible to actually inspect the source code of the implementation at runtime using Visual Studio 2022.

Hart Master C++ Code

User Interface

Public Functions

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

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.	
`void GetConfiguration(` `    TY_Configuration* config_);`	The function copies the configuration data to a data structure provided by the caller.	
`void SetConfiguration(` `    TY_Configuration* config_);`	The function is setting all details required for the configuration. The data is passed in a structure provided by the caller.	
Connection Services		

`SRV_Handle ConnectByAddr(`
`    TY_Byte address_,`
`    EN_Wait qos_,`
`    TY_Byte num_retries_);`
	Use command 0 with short address to get the connection information.	

	address_	0 .. 63
	qos_	NO_WAIT(0), WAIT(1)
	num_retries_	0 .. 10
	The function returns a service handle if successful or INVALID_SRV_HANDLE if there was an error.	

`SRV_Handle ConnectByUniqueID(`
`    TY_Byte* data_ref_,`
`    EN_Wait  qos_,`
`    TY_Byte  num_retries_);`
	Use command 0 with long address to get the connection information.	

	data_ref_	Pointer to a five byte array with the unique identifier
	qos_	NO_WAIT(0), WAIT(1)
	num_retries_	0 .. 10
	The function returns a service handle if successful or INVALID_SRV_HANDLE if there was an error. Note: The function is not yet implemented.	

`SRV_Handle ConnectByShortTag(`
`    TY_Byte* data_ref_,`
`    EN_Wait  qos_,`
`    TY_Byte  num_retries_);`
	Use command 11 with global address to get the connection information.
	data_ref_: Pointer to the byte array of a length of 6 packed ASCII bytes
	qos_: NO_WAIT(0), WAIT(1)
	numRetries: 0 .. 10
	The function returns a service handle if successful or INVALID_SRV_HANDLE if there was an error.
	Note: The function is not yet implemented.

`SRV_Handle ConnectByLongTag(`
`    TY_Byte* data_ref_,`
`    EN_Wait  qos_,`
`    TY_Byte  num_retries_);`
	Use command 21 with global address to get the connection information.
	data_ref_: Pointer to the 32 byte ISO Latin 1 string with the long tag name
	qos_: NO_WAIT(0), WAIT(1)
	num_retries_: 0 .. 10
	The function returns a service handle if successful or INVALID_SRV_HANDLE if there was an error.
	Note: The function is not yet implemented.

`void FetchConnection(`
`    SRV_Handle handle_,`
`    TY_Connection* connection_);`
	Fills a structure provided by the caller with the connection information. hSrv is the service handle which was returned by one of the connection functions.
	Note: After a call of this function the driver is deleting the service. hSrv is no longer valid after calling FetchConnection once.

Communication Services	

`SRV_Handle LaunchCommand(`
`    TY_Byte  command_,`
`    EN_Wait  qos_,`
`    TY_Byte* data_ref_,`
`    TY_Byte  data_len_,`
`    TY_Byte* bytes_of_unique_id_);`
	Send a command in the range 0..255.
	command_: Hart command (0..255) to be sent with the request
	qos_: NO_WAIT(0), WAIT(1)
	data_ref_: Pointer to a native byte array which is sent as payload data
	data_len_: Length of the byte array
	bytes_of_unique_id_: Five byte unique identifier of the addressed device
	The function returns a service handle if successful or INVALID_SRV_HANDLE if there was an error.
	Do command can be used for the support of most of the Hart services including all user specific commands.

`SRV_Handle LaunchExtCommand(`
`    TY_Word  command_,`
`    EN_Wait  qos_,`
`    TY_Byte* data_ref_,`
`    TY_Byte  data_len_,`
`    TY_Byte* bytes_of_unique_id_);`
	Send a command in the range 0..65535.
	command_: Extended Hart command **(0..65535**) to be sent with the request
	qos_: NO_WAIT(0), WAIT(1)
	data_ref_: Pointer to a native byte array which is sent as payload data
	data_len_: Length of the byte array
	bytes_of_unique_id_: Five byte unique identifier of the addressed device
	The function returns a service handle if successful or INVALID_SRV_HANDLE if there was an error.
	The extended command in Hart 6/7 is an extension which is using the byte command 31 to carry a larger command within the data area. Therefore this function was introduced more or less for the convenience of the HartDLL user. The function is automatically taking care of the correct usage of command 31.
	Note: The function is not yet implemented.

`EN_Bool IsServiceCompleted(` `    SRV_Handle service_);`	Returns TRUE8 if the service (service_) was completed.

`void FetchConfirmation(`
`    SRV_Handle service_,`
`    TY_Confirmation* conf_data_);`
	Fills a structure provided by the caller with the service results information such as the response codes and the response data (if any).

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.

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/

`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.

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/

`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_);`	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.

Data Interface

Unlike a slave, a Hart master does not have a predefined data structure whose elements are used to supply the commands.

But the Hart master still needs a few data for settings and options. Therefore, some function calls pass data structures that are specific to the call.

The definition of these structures can be found in the file WbHartM_Structures.h in the directory .\02-Code\01-Common\01-Interface.

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 Master was done as carefully as possible regarding the amount of flash memory and RAM. This is not necessarily required with a Linux computer like Raspberry Pi, but there are also simpler embedded systems.

☞ 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 Master 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.

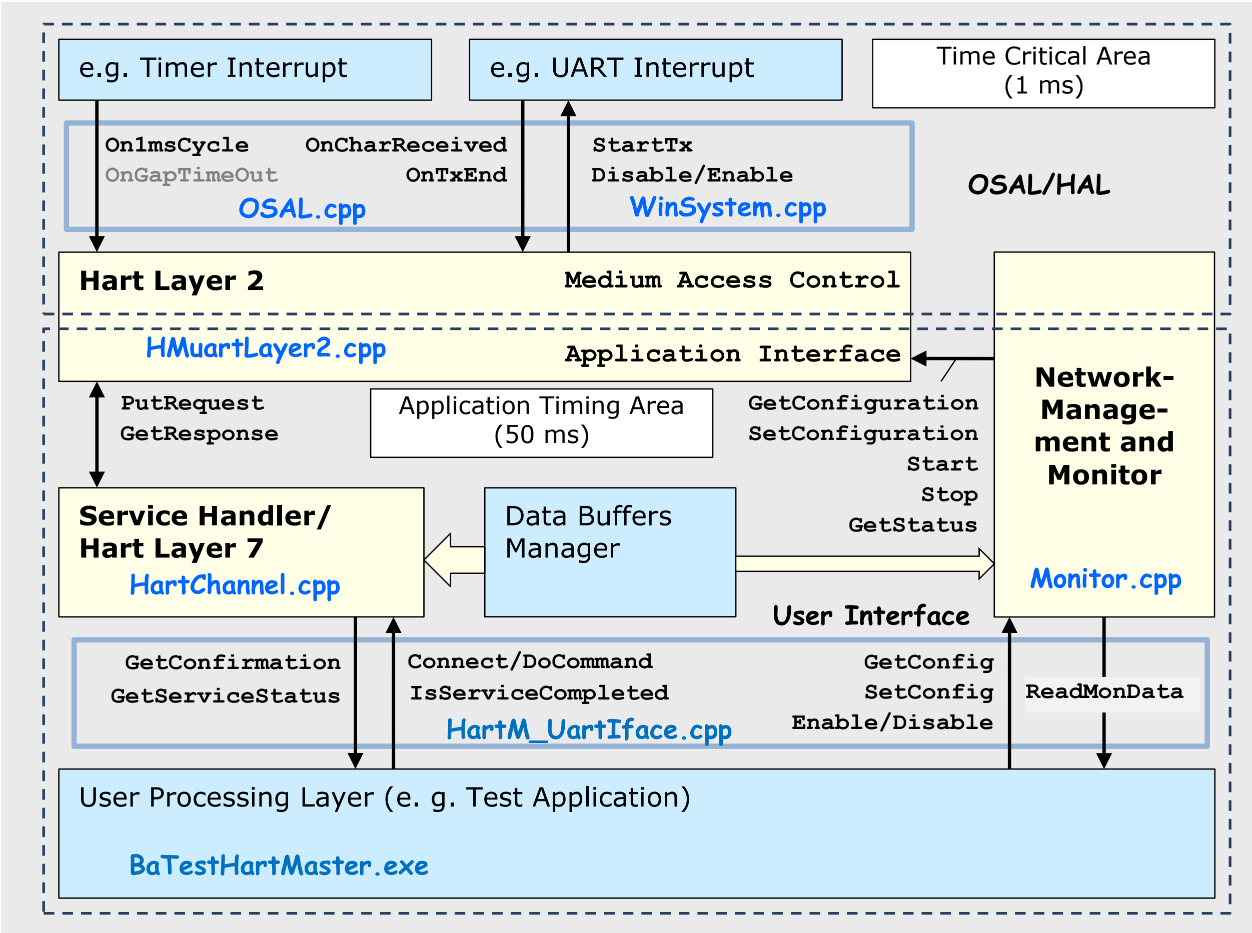


Figure 1: Internal Module Architecture for the Uart Interface

The software is mainly divided into two areas. One is the time critical part, which is needed to meet the requirements of the time controlled dual master protocol of Hart. The other is the area were the application software is working, which is far less time critical.

☞ *Find the source code in the figure above.*

In the diagram above, for some modules I have clearly marked in which file you can find their implementation. This detail is particularly important to me so that you don't think that the diagram is pure theory. The file names are marked with blue color.

☞ *The software architecture is optimized for systems with a few resources.*

The figure above is clearly showing also two user interfaces. There is a user interface (OSAL/HAL) which is connecting the Hart Master software to a timer control interrupt and a UART interrupt which are used for the 'fast' service procedures. Most

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/

of the Hart protocol functionalities are solved in the timer part, which may run on interrupt level. There are arguments for and against this kind of implementation but you ever end up at a point that the incoming frame has to be processed as quickly as possible. So why not spending a few microseconds more once the program has already reached the interrupt level. The Hart protocol is not very complex but it needs to be processed fast enough to catch a precise timing.

The load produced by the implementation is not very high. Because the communication runs with a speed of 1200 bit/s usually there is nothing to do in the 1 ms cycle than to keep track of the timing. Only every 10 ms - if a frame is coming in - a character has to be processed. The processing is done in an incremental way thus not implying the execution of too much instructions.

The split between the time critical area and the user application is done within the Data Link Layer and the so called Network Management. However, the user have not to take any special on these separations except the provision of a few OSAL services for Locking out other tasks. There is an 'atomic' lock out level which has to lock out the interrupts of the Data Link Layer as well as concurrent processes. The other level is 'critical section' which is locking out concurrent processes. More details are described in another chapter of this document.

☞ The top level user interface is at least platform independent.

The interface to the user's application is located on top of the User Data Processing Layer (User Processing Layer). The functions that are made available in this interface are implemented in the file HartM_UartIface.cpp and are described in detail in the 'Public Functions' chapter.

There are a few data objects which are required for Hart protocol and which may be set by the user or an external Hart master. These are such as the tag name and the address. If e.g. the address of the Hart slave is changed through the network the Network-Management will call the user layer to store the data in the NV-memory. If the address is changed through the local HMI of the Hart device, the user layer calls Network-Management of Hart to advise the Data Link Layer protocol to work with the new address. The function used for this setting is SetConfiguration.

In the above figure the parts of the Hart Master are shown in yellow color while the user parts are marked with blue.

☞ The Data Link Layer is an independent piece of software.

The figure is also showing a set of functions between the command executor the network management and the data link layer. These functions may be used if the developer decides to use only the data link layer by providing its own command executor and network management.

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/

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	**HartMasterIface.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.
	WbHartM_Structures.h	This file contains structures which are accessed at the outer interface as well as in some modules in the master kernel.
	WbHartM_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.
02-AppLayer	**HartChannel.cpp/h**	The channel manages a communication interface and the associated propperties. The channel also uses services to conduct Hart 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).
04-Layer2 01-Uart 02-HartIp	**HMuartLayer2.cpp/h and HMipLayer2.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.
	HMuartMacPort.h and HMipMacPort.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.
	HMuartProtocol.cpp/h and HMipProtocol.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	**BaHartMaster.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	**HMuartMacPort.cpp and HMipMacPort.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.

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/

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	`64k`	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#.

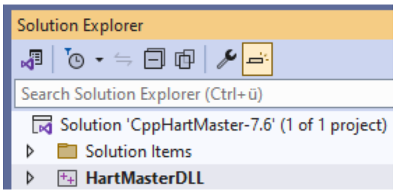
Snake Case			
local_variable	**function_param_**	**m_member_var**	**mo_member_object**
Variable with local scope	A function parameter has a tailing underscore	Basic type private member variable	Complex object member
s_member_var	**so_member_object**		
Basic type static private member variable	Complex 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	

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 is only one project in this solution. The C++ Hart Master is encapsulated in the HartMasterDLL project. The solution is:
.02-Master\CppHartMaster-7.6.sln

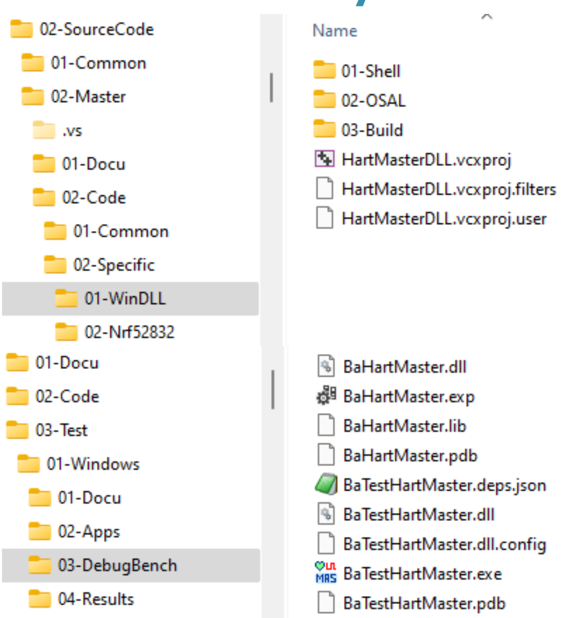
Prerequisites

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.

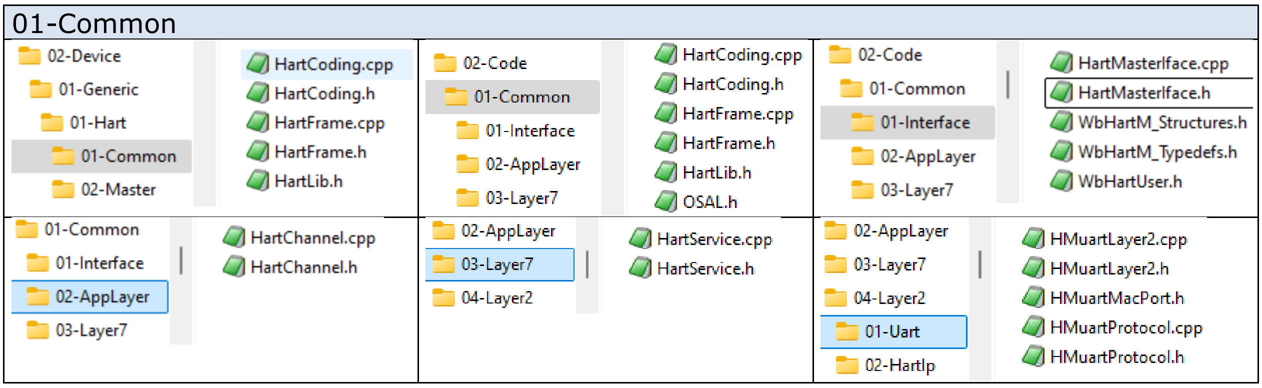
Directory Structure



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

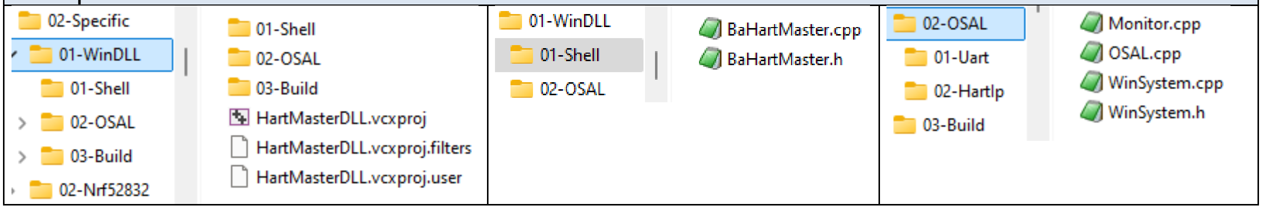
The test software is only be found as executable in the path 03-DebugBech. The executable file BaTestHartMaster.exe and the simulation DLL BaHartMaster.dll are both located here.
When you start debugging the executable ist started and loading the dll which is respresenting the master device.

The following table shows how the individual files are distributed among the directories.

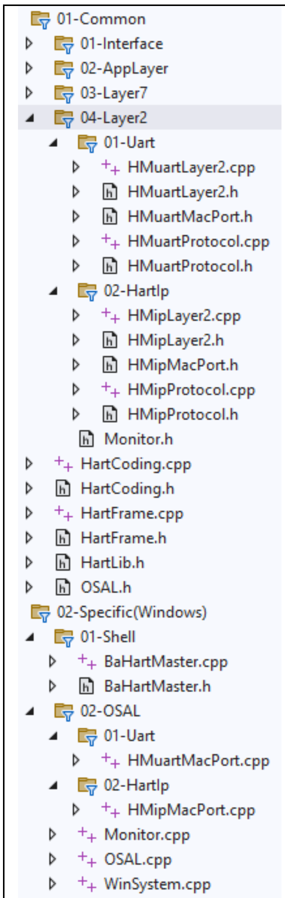


Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon:+49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

02-Specific



Project Structure



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: HartTools@borst-automation.de.
2. 2. Open the solution .\02-Master\CppHartMaster-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

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.

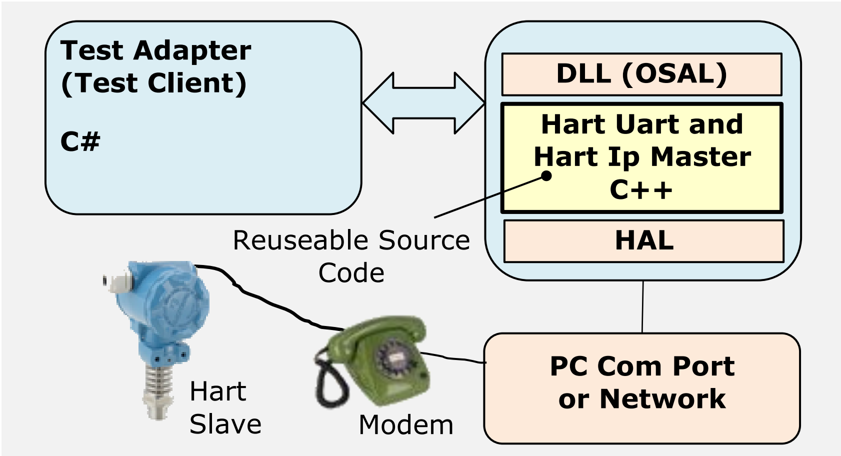
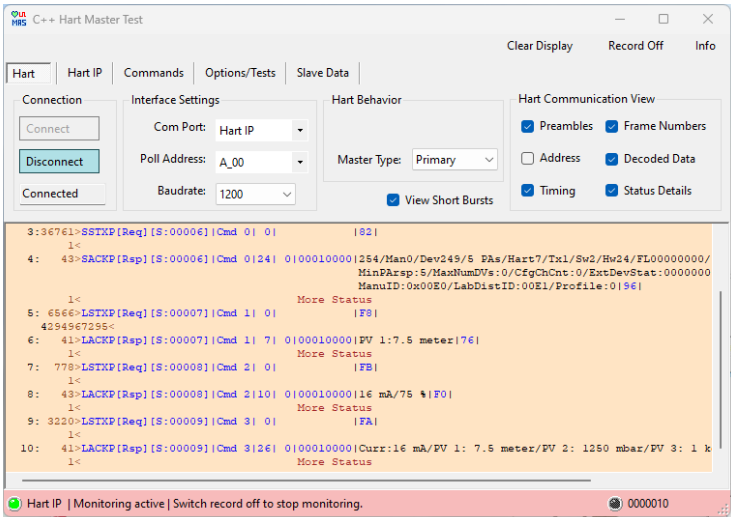


Figure 2: Architecture of the Test Environment

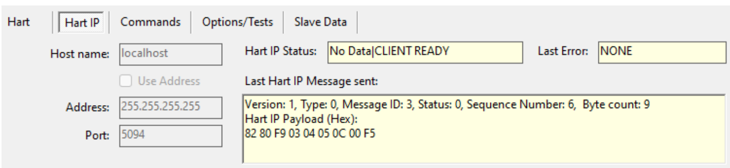
The executable file for the test adapter is located at the following location:

`.\02-Master\03-Test\01-Windows\03-DebugBench\BaTestHartMaster.exe`



When the executable file is started, the Simulations DLL for the master is automatically loaded.
The work surface is divided into two halves. Settings are made or commands are given in the upper area, while the lower area is reserved for a monitor that shows the communication process.
Some basic settings are possible in the Hart tab and a connection can be established with the connected slave.

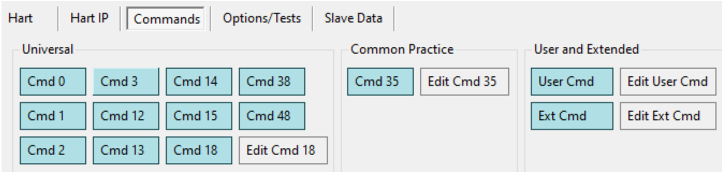
Screenshot 1: The Tab 'Hart'



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

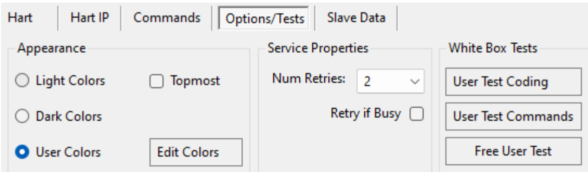
Screenshot 2: The Tab 'Hart Ip'

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/



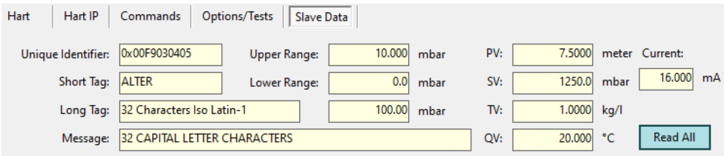
Hart commands are configured and executed in the Commands tab.

Screenshot 3: The Tab 'Commands'



The 'Options/Test' tab contains additional settings and allows the execution of simple tests. Since you have the source code, you can easily modify the tests or add new ones.

Screenshot 4: The Tab 'Options/Tests'



The 'Slave Data' tab is intended to read and display the data of a connected slave.

Screenshot 5: The Tab 'Slave Data'

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-8.0E.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: 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.