

# HF RFID System



## BLUEBOX GEN 2 HF MR COMMUNICATION PROTOCOL

**BLUEBOX**  
RFid System

**Profibus**

## Preface

iDTRONIC GmbH (IDTRONIC) reserves the right to make changes to its products or services or to discontinue any product or service at any time without notice. IDTRONIC provides customer assistance in various technical areas, but does not have full access to data concerning the use and applications of customer's products. Therefore, IDTRONIC assumes no liability and is not responsible for customer applications or product or software design or performance relating to systems or applications incorporating IDTRONIC products. In addition, IDTRONIC assumes no liability and is not responsible for infringement of patents and/or any other intellectual or industrial property rights of third parties, which may result from assistance provided by IDTRONIC. IDTRONIC products are not designed, intended, authorized or warranted to be suitable for life support applications or any other life critical applications that could involve potential risk of death, personal injury or severe property or environmental damage. With the edition of this document, all previous editions become void. Indications made in this manual may be changed without previous notice. Composition of the information in this manual has been done to the best of our knowledge. IDTRONIC does not guarantee the correctness and completeness of the details given in this manual and may not be held liable for damages ensuing from incorrect or incomplete information. Since, despite all our efforts, errors may not be completely avoided, we are always grateful for your useful tips. The installation instructions given in this manual are based on advantageous boundary conditions. IDTRONIC does not give any guarantee promise for perfect function in cross environments. The companies or products mentioned in this document might be brands or brand names of the different suppliers or their subsidiaries in any country. This document may be downloaded onto a computer, stored and duplicated as necessary to support the use of the related IDTRONIC products. Any other type of duplication, circulation or storage on data carriers in any manner not authorized by IDTRONIC represents a violation of the applicable copyright laws and shall be prosecuted.

Safety Instructions / Warning - Read before start-up!

- The device may only be used for the intended purpose designed by the manufacturer. The operation manual should be conveniently kept available at all times for each user.
- Unauthorized changes and the use of spare parts and additional devices that have not been sold or recommended by the manufacturer may cause fire, electric shocks or injuries. Such unauthorized measures shall exclude any liability by the manufacturer.
- The liability-prescriptions of the manufacturer in the issue valid at the time of purchase are valid for the device. The manufacturer shall not be held legally responsible for inaccuracies, errors, or omissions in the manual or automatically set parameters for a device or for an incorrect application of a device.

- Repairs may be executed by the manufacturer only.
- Only qualified personnel should carry out installation, operation, and maintenance procedures.
- Use of the device and its installation must be in accordance with national legal requirements and local electrical codes.
- When working on devices the valid safety regulations must be observed.

Reference table of the devices object of this manual:

Code	Communication Interface	Antenna
5233HM	Profibus	One External Channel

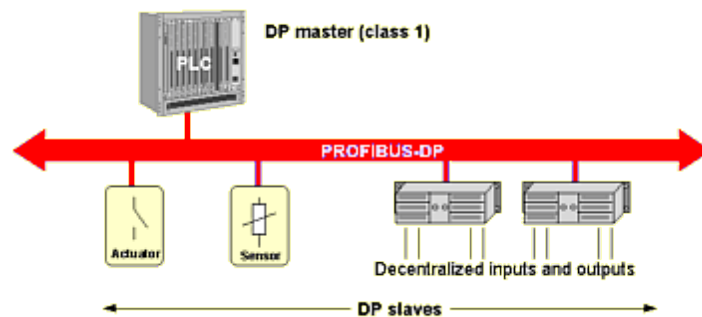
## Table of Contents

Preface .....	2
1 Introduction.....	5
2 Operating Features .....	7
3 Communication Features .....	9
3.1 Introduction .....	9
3.2 Device Reset .....	13
3.3 Device Serial Number Reading.....	14
3.4 FW Version Reading .....	14
3.5 Temperature Reading .....	15
3.6 Date/Time Reading .....	16
3.7 Date/Time Programming .....	16
3.8 Profibus Network Parameters Programming .....	17
3.9 Profibus Network Parameters Reading .....	17
3.10 Operating Parameters Programming .....	18
3.11 Operating Parameters Reading.....	19
3.12 Default parameters programming.....	20
3.13 Digital Output Activation .....	20
3.14 Status Reading .....	20
3.15 RF Deactivation.....	21
3.16 RF Activation .....	21
3.17 Data request .....	22
3.18 Queue data request.....	22
3.19 ISO 15963 Transponders 'Inventory' Command .....	23
3.20 Reading a Data Block of an ISO 15693 Transponder .....	24
3.21 Writing a Data Block of an ISO 15693 Transponder .....	25
3.22 Locking a Data Block of an ISO 15693 Transponder .....	26
3.23 ISO 15963 Transponder 'Get System Info' Command .....	26
3.24 ISO 15963 Transponder 'General Protocol' Command .....	28
A. Supported Transponders.....	30
B. '.GSD' File.....	32

## 1 Introduction

The **BLUEBOX GEN2 INDUSTRIAL HF** hereinafter named **BLUEBOX** is a read/write RFID device for industrial application (item 5233HM) that communicates with a 'host' system (typically a PLC) through a PROFIBUS DP serial line. The **BLUEBOX** acts as a joint through a set of commands between the host system and the rfid tag/s (or transponder/s) present near the antenna/s.

PROFIBUS DP (Dezentrale Pheripherie) was developed for high-speed communication between central controllers (typically PLC) and remote devices (I/O, drives, actuators, sensors, ...).

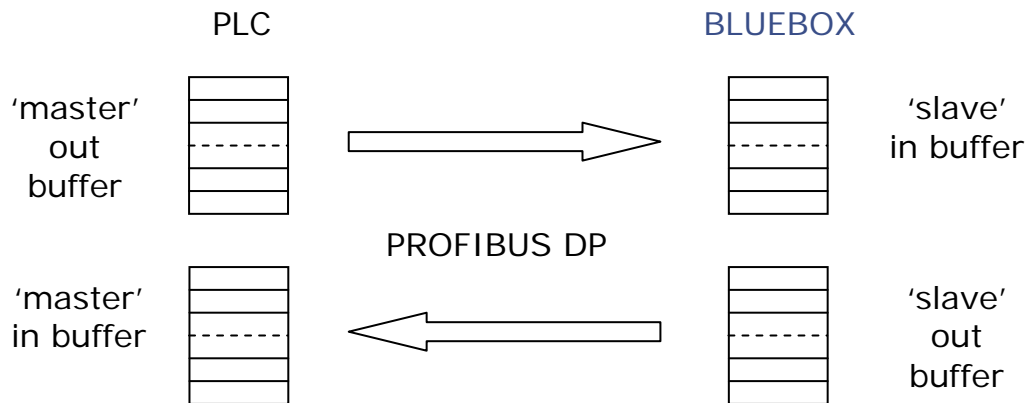


More devices can be connected on the bus, each one has a different address (the address is a configurable parameter). Once configured, the communication with devices is cyclic.

Therefore the **BLUEBOX** is a remote device ('slave') with an input buffer and an output buffer (the length of the buffers is a configurable parameter). Similarly, in order to talk to the **BLUEBOX**, the PLC ('master') has an output buffer (same size of the corresponding input buffer of the **BLUEBOX**) and an input buffer (same size of the corresponding output buffer of the **BLUEBOX**). The input buffer of the **BLUEBOX** can only be modified by the 'master' (PLC) while the output buffer of the **BLUEBOX** can only be modified by the 'slave' that is the **BLUEBOX**. The aim of the cyclic communication is to keep up to date the corresponding buffers at the 'master' side and at the 'slave' side.

At the application level, a specific protocol has been defined for enabling the delivery of control messages from the PLC to the **BLUEBOX** and reply messages from the **BLUEBOX** to the PLC.

For the developers: a .GSD (Generic Station Description) is provided that defines the characteristics of the device and is necessary for the configuration of the profibus network.



An USB connection, working as virtual com, is also available on the **BLUEBOX** and is principally used as service port to configure the operating parameters and to update the firmware of the device, the 'BLUEBOX show' program of the SDK is foreseen to explicate these operations.

Furthermore the **BLUEBOX** is able to handle 2 channels of digital I/O; each channel can be used as output to drive a low side load or as input either driven by a 'pnp' output or by a 'clean' contact. Warning, when the I/O is used as input, do not use it also as output to avoid conflicts! The **BLUEBOX** is available with 1 external RF antenna (item 5233HM).

Compared to the **BLUEBOX GEN2 INDUSTRIAL HF SR** device (item 5233H), the **BLUEBOX GEN2 INDUSTRIAL HF MR** device (item 5233HM) differs for the following reasons:

- increased RF power which leads to higher performance in terms of read/write distance;
- only the ISO 15693 transponders are supported;
- specific 50 ohm antennas with an IP65 industrial TNC connector have to be used;
- ability to set the RF power level and mode of communication transponder/reader (ASK with 1 subcarrier / FSK with 2 subcarriers).

## 2 Operating Features

Supported transponders (or tags) and relative associated functions are described in annexe A.

In 'continuous' mode the **BLUEBOX** is characterized by the coexistence of 2 'parallel' and asynchronous activities: the tag identification consisting of a repetitive inventory of tags through an anticollision loop and the communication with the 'host' system. The 'continuous' identification activity interacts with the communication activity through a buffer that contains the code of the last identified tags or that is empty indicating the absence of tags. Due to synchronization and filtering reasons, the buffer is handled for each tag by a parameter defined as 'hold time' same as 'filter time' defined below (to be set in the range of 0 ... 99 seconds or 0 ... 99 minutes, default value 1 second) and allows to extend 'artificially' the presence of the tag after it leaves the antenna's influence area; this behavior is observable looking at the antenna led status that is 'on' indicating the presence of tag/s and also through the activation of the digital output channel 1 (if its 'automatic' management is enabled by the flag defined in the general parameters). Through the command 'data request' it is possible to get the data contained in the buffer (tag/s ID).

The **BLUEBOX** handles also a 31 elements FIFO queue which is combined with the 'filter time' general parameter (to be set in a range of 0 ... 99 seconds or 0 ... 99 minutes, default value 1 second) that prevents the queue saturation in case of a tag 'continuous' presence. At the end of each inventory session, for each identified tag, the **BLUEBOX** compares it to the list of previous read transponder/s. If the considered transponder do not belong to the list, it is defined as 'new', its code (tag ID) will be inserted in the queue and the filter time will be started. Otherwise (the transponder belong to the list), the **BLUEBOX** verifies if the filter time is expired. In this case (the filter time is expired), the transponder is defined as 'new' and will be processed as described above, otherwise only the filter time will be rearmed. Through the command 'queue data request' and the relative 'ack', it is possible to get the data contained in the queue (tag ID) and unload it.

Every time that a new transponder is identified, the buzzer will be activated for 250ms if its 'automatic' management is enabled by the flag defined in the operating parameters.

In 'continuous' mode the **BLUEBOX** can be configured to obtain the behavior of a 'spontaneous' reader that will send a message on the RS232 serial line and on the Ethernet channel every time that a 'new' transponder is identified.

The **BLUEBOX** allows the execution of 'on request' functions. During the execution of these functions, the 'continuous' identification activity will be suspended temporarily; the involved commands are relative to device configuration and tag read/write specific activities.

If not required, the 'continuous' identification activity can be disabled through a flag defined in the general parameters. In this case, the **BLUEBOX** will only execute the 'on request' commands already defined above.

List of configurable operating parameters:

Parameters	Range / Choice	Default
Hold time	0 ... 99 seconds	1 sec
Filter time	0 ... 99 seconds 0 ... 99 minutes	1 sec
Buzzer	Disabled, enabled	Enabled
Output channel 1	Disabled, enabled	Disabled
'Continuous mode'	Disabled, enabled	Enabled

List of configurable profibus network parameters:

Parameters	Range / Choice	Default
Profibus address	1 ... 126	126
Profibus buffer length	8, 12, 16, 20, 32, 64	16



### 3 Communication Features

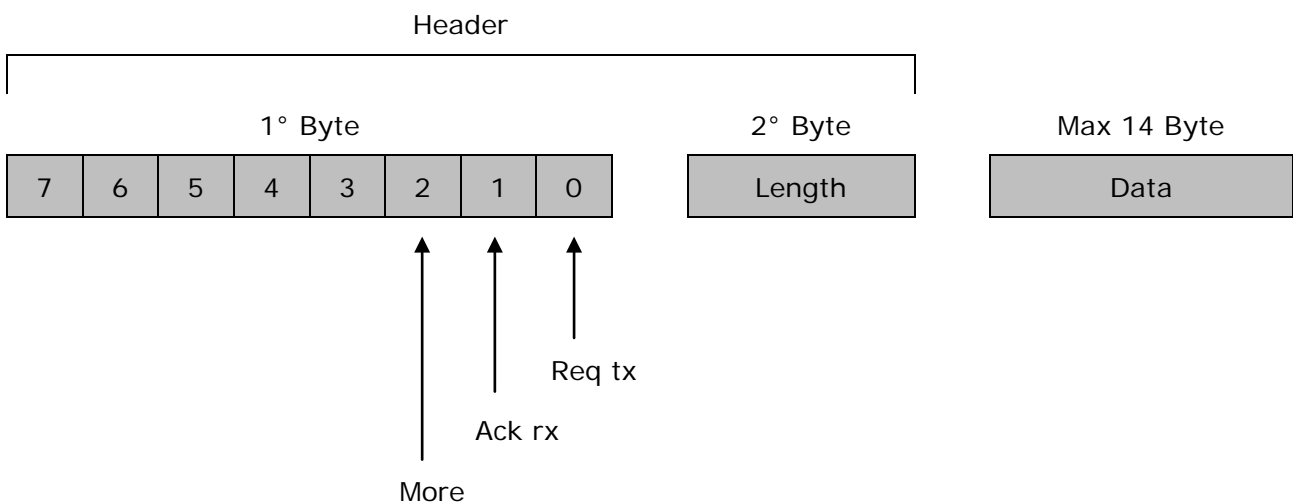
#### 3.1 Introduction

There are two types of data packets used in the communication:

- Outgoing data packets from the 'master' (PLC) to the 'slave' in order to send a command or an answer to the **BLUEBOX**
- Incoming data packets to the 'master' (PLC) from the 'slave' (**BLUEBOX**) carrying for example the answer to a command

In the following example, a buffer length of 16 bytes is supposed.

The outgoing data packets from the 'master' assume the following structure:



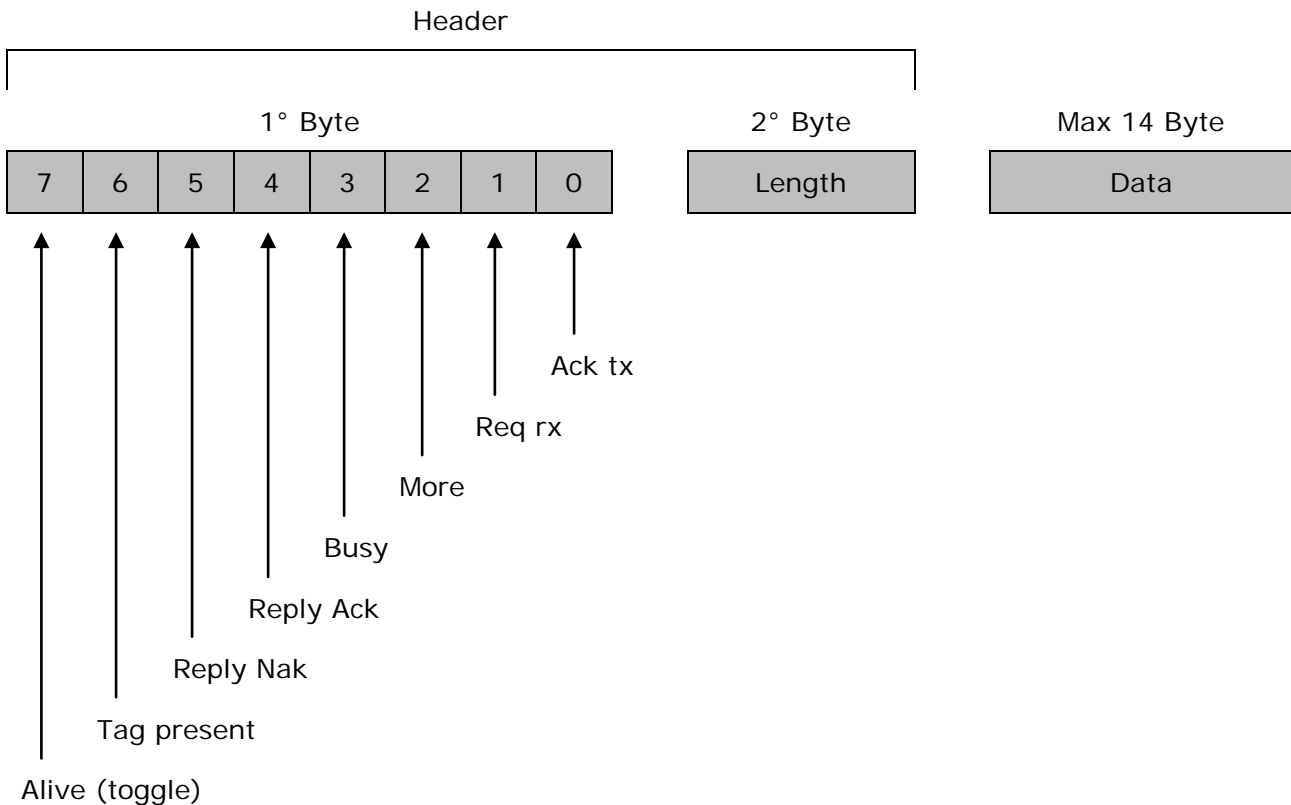
The outgoing data packet consists of a header (first two bytes) and a data buffer of 14 bytes.

The first byte of the header consists of the following flags:

- Bit 7 ... 3: Not used
- Bit 2: More, set to '1' means that the message is composed of several data packets
- Bit 1: Ack rx, reception acknowledge
- Bit 0: Req tx, transmission request

The second byte (Length) of the header specifies the number of data bytes in the data buffer (max 14 data bytes in this example).

The incoming data packets to the 'master' assume the following structure:



Also the incoming data packet consists of a header (first two bytes) and a data buffer of 14 bytes in this example.

The first byte of the header consists of the following flags:

- Bit 7: Alive, toggles every second and means that the 'slave' is running correctly
- Bit 6: Tag present, set to '1' by the 'slave' means that a transponder is present near the antenna/s (detected by the 'continuous' identification activity)
- Bit 5: Reply Nak, set to '1' by the 'slave' means that an error has occurred
- Bit 4: Reply Ack, set to '1' by the 'slave' means that the received command has been processed
- Bit 3: Busy, set to '1' by the 'slave' means that the 'slave' is processing the command message from the 'master'
- Bit 2: More, set to '1' by the 'slave' means that the message is composed of several data packets
- Bit 1: Req rx, reception request (from 'slave')
- Bit 0: Ack tx, transmission acknowledge

The second byte (Length) of the header specifies the number of data bytes in the data buffer (max 14 data bytes).

The messages consist of one or more data packets. If the length of the message is shorter than 14 bytes, the message will be composed of only 1 data packet. If the length of the message is bigger than 14 bytes, the message will be composed of more than 1 data packet; in this case the header of all the transmitted data packets, apart the last one, will present at '1' the flag 'More'

indicating that the message is not completed and another data packet will follow.

The communication between 'master' and 'slave' for a command message take place with the following handshake:

1. The 'master' loads the buffer with the command message and subsequently sets to '1' the flag 'Req tx' to inform the 'slave' that a data packet is ready to be acquired
2. The 'slave' acquires the data packet from the 'master' and confirm the completion of the operation by setting to '1' the flag 'Ack tx'
3. After having received the acknowledgment of the completion of the operation through the flag 'Ack tx' at '1', the 'master' resets to '0' the flag 'Req tx'
4. After having verified that the flag 'Req tx' is reset to '0', also the 'slave' resets to '0' the flag 'Ack tx'
5. During the execution time of the received command, the 'slave' sets to '1' the flag 'Busy' to inform the 'master' that it is not temporarily not available for further communication

In the case of a message length that needs more than one data packet, the previous handshake will be repeated for every data packet until the end of the message.

The answer of the 'slave' to a command message from the 'master' can take place through a full answer message or in a short form depending of the type of command. In the case of a short form answer, it take place through the setting to '1' of the flag 'Reply Ack' or the flag 'Reply Nak' (in function of the result of the execution of the command).

The communication between 'slave' and 'master' for an answer message take place with the following handshake:

1. The 'slave' loads the buffer with the answer message and subsequently sets to '1' the flag 'Req rx' to inform the 'master' that a data packet is ready to be acquired
2. The 'master' acquires the data packet from the 'slave' and confirm the completion of the operation by setting to '1' the flag 'Ack rx'
3. After having received the acknowledgment of the completion of the operation through the flag 'Ack rx' at '1', the 'slave' resets to '0' the flag 'Req rx'
4. After having verified that the flag 'Req rx' is reset to '0', also the 'master' resets to '0' the flag 'Ack rx'

In the case of a message length that needs more than one data packet, the previous handshake will be repeated for every data packet until the end of the message.

Practical example:

The 'master' device requests the inventory of the transponder/s present/s near the antenna; for the example, we suppose that there are 2 ICODE2 transponders having the following hex codes: 0xE0, 0x04, 0x01, 0x00, 0x01, 0x02, 0x03, 0x04 and 0xE0, 0x04, 0x01, 0x10, 0x11, 0x12, 0x13, 0x14.

The bit 7 of the first byte of the header of the incoming buffer of the 'master' device will be indicate as 'T' (toggle) because we suppose that the 'slave' is running correctly.

In this case, the message from the 'master' command is composed of the byte 0x10 (code of the 'inventory' command).

In the following the sequence of the composition of the outgoing and incoming buffer of the 'master' device will be graphically depicted.

For the transmission of this first data package the 'master' loads the second byte of the header (length) and the data bytes of the command (in this case only one) and then sets to '1' the flag 'Req tx' for the 'slave'.

1° Byte								2° Byte	Data	
0	0	0	0	0	0	0	1	01	10	OUT
T	1	0	0	0	0	0	0			IN

The 'slave' confirms the reception of the data packet setting to '1' the flag 'Ack tx'.

1° Byte								2° Byte	Data	
0	0	0	0	0	0	0	1	01	10	OUT
T	1	0	0	0	0	0	1			IN

Consequently the 'master' will reset to '0' the flag 'Req tx' and then also the 'slave' will reset to '0' the flag 'Ack tx'.

1° Byte								2° Byte	Data	
0	0	0	0	0	0	0	0			OUT
T	1	0	0	0	0	0	1			IN

1° Byte								2° Byte	Data	
0	0	0	0	0	0	0	0			OUT
T	1	0	0	0	0	0	0			IN

The 'slave' will set to '1' the flag 'Busy' during the execution of the received command.

1° Byte								2° Byte	Data	
0	0	0	0	0	0	0	0			OUT
T	1	0	0	1	0	0	0			IN

At the end of the execution, the 'slave' will answer to the 'master' sending the readed transponder code.

The message consists of 2 data blocks because the code length is 20 bytes and the data buffer can only support at most 14 bytes.

For the first data packet the 'slave' loads the second byte of the header (length) and the data bytes of the first 14 data bytes of the answer and then sets to '1' the flag 'More' (another data packet will follow) and the flag 'Req rx' for the 'master'.

1° Byte								2° Byte	Data														
0	0	0	0	0	0	0	0																OUT
T	1	0	0	0	1	1	0	OE	10	00	E0	04	01	00	01	02	03	04	E0	04	01	10	IN

The 'master' confirms the reception of the data packet setting to '1' the flag 'Ack rx'.

1° Byte								2° Byte	Data																		OUT IN
0	0	0	0	0	0	1	0																				
T	1	0	0	0	1	1	0	OE	10	00	E0	04	01	00	01	02	03	04	E0	04	01	10					

The handshake will terminate by resetting to '0' the flag 'Req rx' (by the 'slave') and the flag 'Ack rx' (by the 'master').

1° Byte								2° Byte	Data	OUT IN
0	0	0	0	0	0	1	0			
T	1	0	0	0	1	0	0			

1° Byte								2° Byte	Data	OUT IN
0	0	0	0	0	0	0	0			
T	1	0	0	0	1	0	0			

The 'slave' again loads the second byte of the header (length) and the data bytes of the last 4 data bytes of the answer and then resets to '0' the flag 'More' (no other data packet will follow) and the flag 'Req rx' for the 'master'.

1° Byte								2° Byte	Data																		OUT IN
0	0	0	0	0	0	0	0																				
T	1	0	0	0	0	1	0	04	11	12	13	14															

The 'master' confirms the reception of the data packet setting to '1' the flag 'Ack rx'.

1° Byte								2° Byte	Data																		OUT IN
0	0	0	0	0	0	1	0																				
T	1	0	0	0	0	1	0	04	11	12	13	14															

The handshake will terminate by resetting to '0' the flag 'Req rx' (by the 'slave') and the flag 'Ack rx' (by the 'master').

1° Byte								2° Byte	Data	OUT IN
0	0	0	0	0	0	1	0			
T	1	0	0	0	0	0	0			

1° Byte								2° Byte	Data	OUT IN
0	0	0	0	0	0	0	0			
T	1	0	0	0	0	0	0			

### 3.2 Device Reset

This command is used to restart the **BLUEBOX** (the device has the same behaviour like when it is powered up).

Byte	Value	Description	Notes
1	0x30	Command code	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

### 3.3 Device Serial Number Reading

This command is used to get the SN code of the **BLUEBOX** (unique for each device and assigned during the production process), the SN is constituted by 6 bytes.

Byte	Value	Description	Notes
1	0x2A	Command code	
2	0x01	Command code	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x2A	Command code	
2	0x01	Command code	
2+1	0x..	Serial number, 1st byte	The sn is 6 bytes long
2+i	0x..	Serial number, i th byte	i < 6
2+6	0x..	Serial number, 6th byte	i = 6

Note: the SN is a numeric code constituted by 12 digits, the bytes of the SN are BCD-coded and so every byte encodes 2 digits.

### 3.4 FW Version Reading

This command used to get the firmware version loaded on the **BLUEBOX**.

Byte	Value	Description	Notes
1	0x34	Command code	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x34	Command code	
1+1	0x..	Firmware version, 1st ASCII char	The fw version is 16 chars long
1+i	0x..	Firmware version, i th ASCII char	i < 16
1+16	0x..	Firmware version, 16th ASCII char	i = 16

The 16 bytes (2 to 17) are a string of 16 ASCII chars that defines the firmware version. The general form is: '**BB\_TWO\_HM** y.yy '; y.yy gives the firmware version (example **1.16**).

It is also possible to get the firmware version of the HFM reader module mounted in the **BLUEBOX**.

Byte	Value	Description	Notes
1	0x34	Command code	
2	0x01	Module number	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x34	Command code	
2	0x01	Module number	
2+1	0x..	Firmware version, 1st ASCII char	The fw version is 16 chars long
2+i	0x..	Firmware version, i th ASCII char	i < 16
2+16	0x..	Firmware version, 16th ASCII char	i = 16

The 16 bytes (3 to 18) are a string of 16 ASCII chars that defines the firmware version. The general form is: '**MIDRANGE** x.x'; x.xx gives the firmware version (example **1.7**).

### 3.5 Temperature Reading

This command sends back the internal temperature of the **BLUEBOX** measured by the on board temperature sensor.

Byte	Value	Description	Notes
1	0x3A	Command code	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x3A	Command code	
2	0x..	Integer value of the temperature in °C	
3	0x..	Fractional value of the temperature. Bits 7, 6, 5 encode the fractional value in steps of 0.125 °C: 00000000b → .000 °C 00100000b → .125 °C ... 11100000b → .875°C	

### 3.6 Date/Time Reading

This command sends back the date/time of the **BLUEBOX** available on the internal real time clock device.

Byte	Value	Description	Notes
1	0x28	Command code	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x28	Command code	
2	0x..	Year value thousands and hundreds. BCD encoded byte.	
3	0x..	Year value tens and units. BCD encoded byte.	
4	0x..	Month value tens and units. BCD encoded byte.	
5	0x..	Day value tens and units. BCD encoded byte.	
6	0x..	Hour value tens and units. BCD encoded byte.	
7	0x..	Minute value tens and units. BCD encoded byte.	
8	0x..	Second value tens and units. BCD encoded byte.	

### 3.7 Date/Time Programming

This command is used to set the date/time of the **BLUEBOX** in the internal real time clock device.

Byte	Value	Description	Notes
1	0x29	Command code	
2	0x..	Year value thousands and hundreds. BCD encoded byte.	
3	0x..	Year value tens and units. BCD encoded byte.	
4	0x..	Month value tens and units.	



Byte	Value	Description	Notes
		BCD encoded byte.	
5	0x..	Day value tens and units. BCD encoded byte.	
6	0x..	Hour value tens and units. BCD encoded byte.	
7	0x..	Minute value tens and units. BCD encoded byte.	
8	0x..	Second value tens and units. BCD encoded byte.	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

### 3.8 Profibus Network Parameters Programming

This command is used to set the address and the length of the buffers for the Profibus communication of the **BLUEBOX**.

Byte	Value	Description	Notes
1	0x3D	Command code	
2	0x03	Command code	
3	0x..	Address: Decimal 1 ... 126	
4	0x..	Buffers length: <ul style="list-style-type: none"> <li>• 0x00 -&gt; 8 bytes</li> <li>• 0x01 -&gt; 12 bytes</li> <li>• 0x02 -&gt; 16 bytes</li> <li>• 0x03 -&gt; 20 bytes</li> <li>• 0x04 -&gt; 32 bytes</li> <li>• 0x05 -&gt; 64 bytes</li> </ul>	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

**Note:** after the command execution, it is necessary to restart the **BLUEBOX** (command 'Device Reset') in order to apply these new parameters.

### 3.9 Profibus Network Parameters Reading

This command is used to get the values of the Profibus communication parameters (address and length of buffers) of the **BLUEBOX**.

Byte	Value	Description	Notes
------	-------	-------------	-------

Byte	Value	Description	Notes
1	0x3C	Command code	
2	0x03	Command code	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x3C	Command code	
2	0x..	Address: Decimal 1 ... 126	
3	0x..	Buffers length: <ul style="list-style-type: none"> <li>• 0x00 -&gt; 8 bytes</li> <li>• 0x01 -&gt; 12 bytes</li> <li>• 0x02 -&gt; 16 bytes</li> <li>• 0x03 -&gt; 20 bytes</li> <li>• 0x04 -&gt; 32 bytes</li> <li>• 0x05 -&gt; 64 bytes</li> </ul>	
4	0x..	Don't care	
5	0x..	Don't care	
6	0x..	Don't care	
7	0x..	Don't care	
8	0x..	Don't care	

### 3.10 Operating Parameters Programming

This command is used to set the operating parameters of the **BLUEBOX** that are: hold time (00 ... 99 seconds), filter time (0 ... 99 seconds / 0 ... 99 minutes), flag for 'automatic' buzzer management (disabled, enabled), flag for 'automatic' output 1 management (disabled, enabled) and flag for 'continuous' mode (disabled, enabled).

Byte	Value	Description	Notes
1	0x2F	Command code	
2	0xFF	Don't care value	0xFF suggested value
3	0x48	Don't care value	0x48 suggested value
4	0x20	Don't care value	0x20 suggested value
5	0x01	Don't care value	0x01 suggested value

Byte	Value	Description	Notes
6	0x03	Don't care value	0x03 suggested value
7	0x..	Filter time: <ul style="list-style-type: none"> <li>Decimal 0 ... 99 for time in seconds (0 ... 99 seconds)</li> <li>Decimal 100 ... 199 for time in minutes (0 ... 99 minutes)</li> </ul>	
8	0x..	Flags: single bits are dedicated to disable (0 value) or enable (1 value) functions: <ul style="list-style-type: none"> <li>Bit 7: automatic buzzer management</li> <li>Bit 6: automatic output 1 management</li> <li>Bit 5 ... bit 1: not used</li> <li>Bit 0: to disable 'continuous' mode</li> </ul>	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

**Note:** after the command execution, the **BLUEBOX** resets itself to apply the new parameters.

### 3.11 Operating Parameters Reading

This command is used to get the values of the operating parameters of the **BLUEBOX**.

Byte	Value	Description	Notes
1	0x2A	Command code	
If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the <b>BLUEBOX</b> answers:			
Byte	Value	Description	Notes
1	0x2A	Command code	
2	0x..	Don't care value	
3	0x..	Don't care value	
4	0x..	Don't care value	
5	0x..	Don't care value	
6	0x..	Don't care value	
7	0x..	Filter time: <ul style="list-style-type: none"> <li>Decimal 0 ... 99 for time in seconds (0 ... 99 seconds)</li> <li>Decimal 100 ... 199 for time in minutes (0 ... 99 minutes)</li> </ul>	

Byte	Value	Description	Notes
8	0x..	Flags. Single bits are dedicated to disable (0 value) or enable (1 value) functions: <ul style="list-style-type: none"> <li>• Bit 7: automatic buzzer management</li> <li>• Bit 6: automatic output 1 management</li> <li>• Bit 5 ... bit 1: not used</li> <li>• Bit 0: to disable 'continuous' mode</li> </ul>	

### 3.12 Default parameters programming

This command is used to set the default values of the communication and operating parameters of the **BLUEBOX**.

Byte	Value	Description	Notes
1	0x31	Command code	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

**Note:** after the command execution, the **BLUEBOX** resets itself to apply the new parameters.

### 3.13 Digital Output Activation

This command is used to excite each individual output and to set also the duration in case of impulsive use.

Byte	Value	Description	Notes
1	0x37	Command code	
2	0x..	Output to activate: <ul style="list-style-type: none"> <li>• 0x01 -&gt; Output 1</li> <li>• 0x02 -&gt; Output 2</li> </ul>	
3	0x..	Activation time: <ul style="list-style-type: none"> <li>• 0x01 ... 0x63 (1 ... 99 seconds -&gt; in case of 'impulsive' output activation)</li> <li>• 0x81 -&gt; 'Continuous' activation</li> <li>• 0x80 -&gt; Deactivation</li> </ul>	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

### 3.14 Status Reading

The **BLUEBOX** will answer to this command with a series of information about the current status and particularly about the digital inputs status.

Byte	Value	Description	Notes
------	-------	-------------	-------

1	0x36	Command code	
---	------	--------------	--

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x36	Command code	
2	0x..	<b>BLUEBOX</b> status byte 1. ASCII encoded byte whose bits have the following meaning: <ul style="list-style-type: none"> <li>• Bit 7, 6: Not used</li> <li>• Bit 5: RF 1 status (0=off, 1=on)</li> <li>• Bit 4: 'continuous' mode status (0=disab., 1=enab.)</li> <li>• Bit 3, 2: Not used</li> <li>• Bit 1: Output 2 status (1=activated)</li> <li>• Bit 0: Output 1 status (1=activated)</li> </ul>	
3	0x..	<b>BLUEBOX</b> status byte 2. ASCII encoded byte whose bits have the following meaning: <ul style="list-style-type: none"> <li>• Bit 7...2: Not used</li> <li>• Bit 1: Input 2 status (1=activated)</li> <li>• Bit 0: Input 1 status (1=activated)</li> </ul>	

### 3.15 RF Deactivation

In 'continuous' mode, this command is used to suspend the activity of the RF antennas connected to the **BLUEBOX**; see also 'RF activation' command.

Byte	Value	Description	Notes
1	0x38	Command code	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

### 3.16 RF Activation

In 'continuous' mode, this command is used to resume the activity of the RF antennas connected to the **BLUEBOX**; see also 'RF Deactivation' command.

Byte	Value	Description	Notes
1	0x39	Command code	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command don't 'produce' data).

### 3.17 Data request

This command sends back the code of the eventual transponder that is present in the buffer. When 'continuous' mode is enabled, the reply is immediate because the **BLUEBOX** sends back the data hold in the buffer that is managed by the 'continuous' identification activity; otherwise, the **BLUEBOX** performs readily the identification task under time out protection and sends back the result of the operation.

Byte	Value	Description	Notes
1	0x05	Command code	

If command is not successfully executed, the answer is given in short form through the 'Reply Ack' / 'Reply Nak' flags; otherwise, the **BLUEBOX** answers:

Byte	Value	Description	Notes
1	0x..	Transponder type. See Annexe A for type assignement table.	
1+1	0x..	Transponder UID, 1 st byte	
1+i	0x..	Transponder UID, i th byte	i < 8
1+8	0x..	Transponder UID, 8 th byte	8 = UID length

Note: if no transponder is present near the antenna, the answer message will consist of 1 null byte (0x00).

### 3.18 Queue data request

In 'continuous' mode, when the **BLUEBOX** finds a 'new' transponder, it inserts the code in the FIFO queue. This command sends back the first present code (UID/type of transponder) in the queue.

Byte	Value	Description	Notes
1	0x06	Command code	

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answer:

Byte	Value	Description	Notes
1	0x..	Transponder type. See Annexe A for type assignement table.	
1+1	0x..	Transponder UID, 1 st byte	
1+i	0x..	Transponder UID, i th byte	i < n
1+8	0x..	Transponder UID, 8 th byte	8 = UID length

Note: if the queue is empty, the answer message will consist of 1 null byte (0x00).

To delete the received code from the queue, the 'master' reply to the **BLUEBOX**:

Byte	Value	Description	Notes
1	0x07	Command code	

The answer to this command is given in short form through the 'Reply Ack' / 'Reply Nak' flags (this command doesn't 'produce' data).

### 3.19 ISO 15963 Transponders 'Inventory' Command

This command is used to get the UID code of the identified ISO 15693 transponders that are present near the antenna/s.

Byte	Value	Description	Notes
1	0x10	Command code	

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answers,  
 a) case of absence of transponder:

Byte	Value	Description	Notes
1	0x10	Command code	
2	0x01	Status no transponder	

b) case of presence of transponder/s:

Byte	Value	Description	Notes
1	0x10	Command code	
2	0x00	Status ok → transponder/s is/are present	
2+1	0x..	Transponder 1 UID, 1 st byte	
2+i	0x..	Transponder 1 UID, i th byte	i < 8
2+8	0x..	Transponder 1 UID, 8 th byte	
...	...	...	
...	0x..	Transponder n UID, 1 st byte	
...	0x..	Transponder n UID, j nd byte	j < 8
2+n*8	0x..	Transponder n UID, 8 th byte	

### 3.20 Reading a Data Block of an ISO 15693 Transponder

This command is used to get a data block of a known (UID) ISO 15693 transponder. Note that the number of bytes of a block and the number of blocks depends on the transponder type; for example, the **ICODE2** transponder is organized in blocks of 4 bytes, the **MB89R118** transponder is organized in blocks of 8 bytes, for more details see the specific transponder data sheet.

Byte	Value	Description	Notes
1	0x11	Command code	
1+1	0x..	Transponder UID, 1 st byte	
1+i	0x..	Transponder UID, i th byte	$i < 8$
1+8	0x..	Transponder UID, 8 th byte	
10	0x..	Block number	Max value depends on the specific transponder

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answers,  
 a) if the addressed transponder is not present:

Byte	Value	Description	Notes
1	0x11	Command code	
2	0x01	Status no transponder	

b) if the addressed transponder do not support the requested block or if some error is occurred during the transaction:

Byte	Value	Description	Notes
1	0x11	Command code	
2	0x02	Status block not supported or errors	

c) if the addressed transponder is present:

Byte	Value	Description	Notes
1	0x11	Command code	
2	0x00	Status ok	
2+1	0x..	Data of the block, 1 st byte	
2+i	0x..	Data of the block, i th byte	$i < n$
2+n	0x..	Data of the block, n th byte	n depends on the specific transponder



### 3.21 Writing a Data Block of an ISO 15693 Transponder

This command is used to write a data block of a known (UID) ISO 15693 transponder. Note that the number of bytes of a block depends on the transponder type; for example, the **ICODE2** transponder is organized in blocks of 4 bytes, the **MB89R118** transponder is organized in blocks of 8 bytes, for more details see the specific transponder data sheet.

Byte	Value	Description	Notes
1	0x12	Command code	
1+1	0x..	Transponder UID, 1 st byte	
1+i	0x..	Transponder UID, i th byte	i < 8
1+8	0x..	Transponder UID, 8 th byte	
10	0x..	Block number	Max value depends on the specific transponder
10+1	0x..	Data to be written in the block, 1 st byte	
10+i		Data to be written in the block, i th byte	i < n
10+n		Data to be written in the block, n th byte	n depends on the specific transponder

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answers,  
a) if the addressed transponder is not present:

Byte	Value	Description	Notes
1	0x12	Command code	
2	0x01	Status no transponder	

b) if the addressed transponder do not support the requested block or if some error is occurred during the transaction:

Byte	Value	Description	Notes
1	0x12	Command code	
2	0x02	Status block not supported or errors	

c) if the addressed transponder is present and the block has been correctly written:

Byte	Value	Description	Notes
1	0x12	Command code	
2	0x00	Status ok block successfully written	

### 3.22 Locking a Data Block of an ISO 15693 Transponder

This command is used to lock a data block of a known (UID) ISO 15693 transponder. For more details see the specific transponder data sheet.

Byte	Value	Description	Notes
1	0x13	Command code	
1+1	0x..	Transponder UID, 1 st byte	
1+i	0x..	Transponder UID, i th byte	i < 8
1+8	0x..	Transponder UID, 8 th byte	
10	0x..	Block number	Max value depends on the specific transponder

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answers,

a) if the addressed transponder is not present:

Byte	Value	Description	Notes
1	0x13	Command code	
2	0x01	Status no transponder	

b) if the addressed transponder do not support the requested block or if some error is occurred during the transaction:

Byte	Value	Description	Notes
1	0x13	Command code	
2	0x02	Status block not supported or errors	

c) if the addressed transponder is present and the block has been correctly locked:

Byte	Value	Description	Notes
1	0x13	Command code	
2	0x00	Status ok block successfully locked	

### 3.23 ISO 15963 Transponder 'Get System Info' Command

This command is used to get the system info data block of a known (UID) ISO 15693 transponder. For more details see the specific transponder data sheet.

Byte	Value	Description	Notes
1	0x14	Command code	

Byte	Value	Description	Notes
1+1	0x..	Transponder UID, 1 st byte	
1+i	0x..	Transponder UID, i th byte	$i < 8$
1+8	0x..	Transponder UID, 8 th byte	

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answers,

a) if the addressed transponder is not present:

Byte	Value	Description	Notes
1	0x14	Command code	
2	0x01	Status no transponder	

b) if some error is occurred during the transaction:

Byte	Value	Description	Notes
1	0x14	Command code	
2	0x02	Status error	

c) if the addressed transponder is present:

Byte	Value	Description	Notes
1	0x14	Command code	
2	0x00	Status ok block successfully locked	
3	0x..	Info Flags: single bits are dedicated to specify the presence of the following fields (0 → absent, 1 → present): <ul style="list-style-type: none"> <li>• Bit 7 ... bit 4: not used</li> <li>• Bit 3: IC Reference (1 byte)</li> <li>• Bit 2: Memory Size (2 bytes)</li> <li>• Bit 1: AFI (1 byte)</li> <li>• Bit 0: DSFID (1 byte)</li> </ul>	
4+1	0x..	Transponder UID, 1 st byte	
4+i	0x..	Transponder UID, i th byte	$i < 8$
4+8	0x..	Transponder UID, 8 th byte	
...	0x..	DSFID	Present only if bit 0 of Info Flags is set
...	0x..	AFI	Present only if bit 1 of Info Flags is set

Byte	Value	Description	Notes
...	0x..	Memory Size – Block Size in bytes 0x00 (1 byte) ... 0x1F (32 bytes)	Present only if bit 2 of Info Flags is set
...	0x..	Memory Size – Number of Blocks 0x00 (1 block) ... 0xFF (256 blocks)	Present only if bit 2 of Info Flags is set
...	0x..	IC Reference	Present only if bit 3 of Info Flags is set

### 3.24 ISO 15963 Transponder 'General Protocol' Command

This command allows to send any ISO 15693 general format protocol command (flags field, command code field, parameters fields, application data fields) to a ISO 15693 transponder and to receive, in case of successful operation, the response of the transponder (flag field, parameters fields, data fields). For more details see the specific transponder data sheet and ISO 15693 protocol. If the 'continuous' mode is enabled, it will be suspended by the execution of this command and will be resumed as long as this command is used; it will be resumed automatically when another type of command will be executed.

Byte	Value	Description	Notes
1	0x15	Command code	
1+1	0x..	Data to send to the tag, 1 st byte	
1+i	0x..	Data to send to the tag, i th byte	$i < n$
1+n	0x..	Data to send to the tag, n th byte	n depends on the specific protocol command

If the command is not successfully executed, the answer is given in short form through the 'reply ack' / 'reply nak' flags; otherwise, the **BLUEBOX** answers,  
 a) if the transponder is not present:

Byte	Value	Description	Notes
1	0x15	Command code	
2	0x01	Status no transponder	

b) if some error is occurred during the transaction:

Byte	Value	Description	Notes
1	0x15	Command code	
2	0x02	Status error	

c) if the transponder is present:

Byte	Value	Description	Notes
------	-------	-------------	-------

Byte	Value	Description	Notes
1	0x15	Command code	
2	0x00	Status ok	
2+1	0x..	Data received from the tag, 1 st byte	
2+i	0x..	Data received from the tag, i th byte	$i < n$
2+n	0x..	Data received from the tag, n th byte	n depends on the specific protocol command response

## A. Supported Transponders

Supported transponders by the **BLUEBOX** are:

Manuf.	Part	Standard	Chip	Notes / Functions
NXP	ICODE2	15693	SL2 IC S20	Inventory Read block (4 bytes) Write block (4 bytes) Lock block Get system info
TI	TAG-IT HF-I	15693		Inventory Read block (4 bytes) Write b block (4 bytes) Lock block Get system info
MEM	EM 4035	15693		Inventory Read block (8 bytes) Write block (8 bytes) Lock block Get system info
STM	LRI 64	15693		Inventory Read block (1 byte) Write block (1 byte) Get system info
STM	LRI 512	15693		Inventory Read block (4 bytes) Write block (4 bytes) Lock block Get system info
FUJITSU	MB89R118	15693		Inventory Read block (8 bytes) Write block (8 bytes) Get system info
INFINEON	MY-D 02P	15693	SRF 55V02P	Inventory Read block (4 bytes) Write block (4 bytes) Lock block
INFINEON	MY-D 10P	15693	SRF 55V10P	Inventory Read block (4 bytes) Write block (4 bytes)

Type coding for supported transponders:

Type Code	Part	UID Length
0x20	Generic ISO 15693	8 bytes
0x21	ICODE2	8 bytes
0x22	TAG-IT HF-I	8 bytes
0x23	EM 4035	8 bytes
0x24	LRI 64/512	8 bytes
0x25	MB89R118	8 bytes
0x26	MY-D 02P/10P	8 bytes

## B. '.GSD' File

```

;*****
;***
;***      Filename: HIL_0a12.GSD (c) 2007      ***
;***      GSD file version 1.000 from 10.12.2007 ***
;***
;*****
;
;ATTENTION:
;=====
;Changes in this file can cause configuration or communication problems.
;This file is compatible to the firmware of the device.
;
;Changes
;=====
;10.12.07   V1.000   R. Hornung
;-  created

#Profibus_DP

GSD_Revision      = 5
Vendor_Name       = "Hilscher GmbH"
Model_Name        = "NETX DP/DPS"
Revision          = "Version 1.000"
Ident_Number      = 0x0A12
Protocol_Ident    = 0
Station_Type      = 0
Hardware_Release  = "Version 1.000"
Software_Release  = "Version 2.000"
Implementation_Type = "netX"
9.6_supp          = 1
19.2_supp         = 1
45.45_supp        = 1
93.75_supp        = 1
187.5_supp        = 1
500_supp          = 1
1.5M_supp         = 1
3M_supp           = 1
6M_supp           = 1
12M_supp          = 1
MaxTsdr_9.6       = 60
MaxTsdr_19.2      = 60
MaxTsdr_45.45     = 60
MaxTsdr_93.75     = 60
MaxTsdr_187.5     = 60
MaxTsdr_500       = 100
MaxTsdr_1.5M      = 150
MaxTsdr_3M        = 250
MaxTsdr_6M        = 450
MaxTsdr_12M       = 800
Redundancy        = 0
Repeater_Ctrl_Sig = 2
24V_Pins          = 0
Freeze_Mode_supp  = 1
Sync_Mode_supp    = 1
Auto_Baud_supp    = 1
Set_Slave_Add_supp = 0
Min_Slave_Intervall = 6
Modular_Station   = 1
Max_Module        = 24
Max_Input_Len     = 244
Max_Output_Len    = 244
Max_Data_Len      = 488
Max_Diag_Data_Len = 244
Max_User_Prm_Data_Len = 5

```



```

Slave_Family          = 0

DPV1_Slave            = 1
DPV1_Data_Types       = 0

C1_Read_Write_supp    = 1
C1_Max_Data_Len       = 240
C1_Response_Timeout   = 100

C2_Read_Write_supp    = 1
C2_Max_Count_Channels = 1
C2_Max_Data_Len       = 240
C2_Response_Timeout   = 100
Max_Initiate_PDU_Length = 244

Extra_Alarm_SAP_supp   = 0
Alarm_Sequence_Mode_Count = 32
Alarm_Type_Mode_supp   = 1

Diagnostic_Alarm_supp  = 1
Process_Alarm_supp     = 1
Pull_Plug_Alarm_supp  = 1
Status_Alarm_supp     = 1
Update_Alarm_supp     = 1
Manufacturer_Specific_Alarm_supp = 1

Ident_Maintenance_supp = 1

Bitmap_Device = "CIFXDPSR"
Bitmap_Diag   = "CIFXDPSD"
Bitmap_SF     = "CIFXDPSS"

PrmText = 1
Text(0) = "Disable"
Text(1) = "Enable"
EndPrmText

PrmText = 2
Text(0) = "OFF"
Text(1) = "ON"
EndPrmText

PrmText = 3
Text(0) = "1 alarm of each type"
Text(1) = "2 alarms in total"
Text(2) = "4 alarms in total"
Text(3) = "8 alarms in total"
Text(4) = "12 alarms in total"
Text(5) = "16 alarms in total"
Text(6) = "24 alarms in total"
Text(7) = "32 alarms in total"
EndPrmText

PrmText = 4
Text(0) = "reserved"
EndPrmText

ExtUserPrmData = 1 "DPV1"
Bit(7) 0 0-1
Prm_Text_Ref = 1
EndExtUserPrmData

ExtUserPrmData = 2 "Fail Safe"
Bit(6) 0 0-1
Prm_Text_Ref = 2
EndExtUserPrmData

ExtUserPrmData = 3 "Pull Plug Alarm"
Bit(7) 0 0-1
Prm_Text_Ref = 2

```

```
EndExtUserPrmData
ExtUserPrmData = 4 "Process Alarm"
Bit(6) 0 0-1
Prm_Text_Ref = 2
EndExtUserPrmData
```

```
ExtUserPrmData = 5 "Diagnostic Alarm"
Bit(5) 0 0-1
Prm_Text_Ref = 2
EndExtUserPrmData
```

```
ExtUserPrmData = 6 "Manufacturer Specific Alarm"
Bit(4) 0 0-1
Prm_Text_Ref = 2
EndExtUserPrmData
```

```
ExtUserPrmData = 7 "Status Alarm"
Bit(3) 0 0-1
Prm_Text_Ref = 2
EndExtUserPrmData
```

```
ExtUserPrmData = 8 "Update Alarm"
Bit(2) 0 0-1
Prm_Text_Ref = 2
EndExtUserPrmData
```

```
ExtUserPrmData = 9 "Alarm Mode"
BitArea(0-2) 0 0-7
Prm_Text_Ref = 3
EndExtUserPrmData
```

```
ExtUserPrmData = 10 "reserved"
BitArea(0-2) 0 0-7
Prm_Text_Ref = 4
EndExtUserPrmData
```

```
ExtUserPrmData = 11 "reserved"
BitArea(0-2) 0 0-7
Prm_Text_Ref = 4
EndExtUserPrmData
```

```
Ext_User_Prm_Data_Ref(0) = 1
Ext_User_Prm_Data_Ref(0) = 2
Ext_User_Prm_Data_Ref(1) = 3
Ext_User_Prm_Data_Ref(1) = 4
Ext_User_Prm_Data_Ref(1) = 5
Ext_User_Prm_Data_Ref(1) = 6
Ext_User_Prm_Data_Ref(1) = 7
Ext_User_Prm_Data_Ref(1) = 8
Ext_User_Prm_Data_Ref(2) = 9
Ext_User_Prm_Data_Ref(3) = 10
Ext_User_Prm_Data_Ref(4) = 11
```

```
;*****
;***                blank space                ***
;*****
```

```
Module = "blank space" 0x00
0
EndModule
```

```
;*****
;***                1 Byte Input/Output                ***
;*****
```

```
Module = "1 Byte In" 0x90
1
EndModule
Module = "1 Byte Out" 0xA0
2
EndModule
```

```

;*****
;***          1 Word Input/Output          ***
;*****
Module = "1 Word In" 0xD0
3
EndModule
Module = "1 Word Out" 0xE0
4
EndModule

;*****
;***          2 Byte Input/Output          ***
;*****
Module = "2 Bytes In" 0x91
5
EndModule
Module = "2 Bytes Out" 0xA1
6
EndModule

;*****
;***          2 Word Input/Output          ***
;*****
Module = "2 Words In" 0xD1
7
EndModule
Module = "2 Words Out" 0xE1
8
EndModule

;*****
;***          3 Byte Input/Output          ***
;*****
Module = "3 Bytes In" 0x92
9
EndModule
Module = "3 Bytes Out" 0xA2
10
EndModule

;*****
;***          3 Word Input/Output          ***
;*****
Module = "3 Words In" 0xD2
11
EndModule
Module = "3 Words Out" 0xE2
12
EndModule

;*****
;***          4 Byte Input/Output          ***
;*****
Module = "4 Bytes In" 0x93
13
EndModule
Module = "4 Bytes Out" 0xA3
14
EndModule

;*****
;***          4 Word Input/Output          ***
;*****
Module = "4 Words In" 0xD3
15
EndModule
Module = "4 Words Out" 0xE3
16
EndModule
;*****

```

```
***                               8 Byte Input/Output                               ***
;*****
Module = "8 Bytes In"  0x97
17
EndModule
Module = "8 Bytes Out" 0xA7
18
EndModule

;*****
;***                               8 Word Input/Output                               ***
;*****
Module = "8 Words In"  0xD7
19
EndModule
Module = "8 Words Out" 0xE7
20
EndModule

;*****
;***                               12 Byte Input/Output                               ***
;*****
Module = "12 Bytes In"  0x9B
21
EndModule
Module = "12 Bytes Out" 0xAB
22
EndModule

;*****
;***                               12 Word Input/Output                               ***
;*****
Module = "12 Words In"  0xDB
23
EndModule
Module = "12 Words Out" 0xEB
24
EndModule

;*****
;***                               16 Byte Input/Output                               ***
;*****
Module = "16 Bytes In"  0x9F
25
EndModule
Module = "16 Bytes Out" 0xAF
26
EndModule

;*****
;***                               16 Word Input/Output                               ***
;*****
Module = "16 Words In"  0xDF
27
EndModule
Module = "16 Words Out" 0xEF
28
EndModule

;*****
;***                               20 Byte Input/Output                               ***
;*****
Module = "20 Bytes In"  0x40,0x93
29
EndModule
Module = "20 Bytes Out" 0x80,0x93
30
EndModule
;*****
;***                               20 Word Input/Output                               ***
```

```

;*****
Module = "20 Words In" 0x40,0xD3
31
EndModule
Module = "20 Words Out" 0x80,0xD3
32
EndModule

;*****
;***          32 Byte Input/Output          ***
;*****
Module = "32 Bytes In" 0x40,0x9F
33
EndModule
Module = "32 Bytes Out" 0x80,0x9F
34
EndModule

;*****
;***          32 Word Input/Output          ***
;*****
Module = "32 Words In" 0x40,0xDF
35
EndModule
Module = "32 Words Out" 0x80,0xDF
36
EndModule

;*****
;***          64 Byte Input/Output          ***
;*****
Module = "64 Bytes In" 0x40,0xBF
37
EndModule
Module = "64 Bytes Out" 0x80,0xBF
38
EndModule

;*****
;***          64 Word Input/Output          ***
;*****
Module = "64 Words In" 0x40,0xFF
39
EndModule
Module = "64 Words Out" 0x80,0xFF
40
EndModule

```