

**I-CODE RFID
SYSTEM**

Software Development Manual



EHAG
ELECTRONIC HARDWARE AG
Industriestr. 8 CH-8618 Oetwil a/S.
T: +41 43 844 94 00 info@ehag.ch
F: +41 43 844 94 01 www.ehag.ch

Cod.: 130.0004.1 Rev.: 1

September 2000

Índice

1.- OVERVIEW	3
1.1.- DISK CONTENTS AND INSTALLATION	3
2.- DLL FUNCTIONS	4
2.1.- TAG CONTROL FUNCTIONS	4
2.1.1.- READ UNSELECTED	5
2.1.2.- ANTICOLLISION SELECTION	6
2.1.3.- READ	7
2.1.4.- WRITE	8
2.1.5.- HALT	9
2.1.5.- EAS	10
2.1.6.- RESET QUIET MODE	11
2.2.- READER/ENCODER CONTROL FUNCTIONS	12
2.2.1.- OUTPUT PORT CONTROL	13
2.2.2.- INPUT PORT CONTROL	13
2.2.3.- CONFIGURATION	14
2.2.4.- CONFIGURATION INFORMATION	15
2.2.5.- SIGNAL SAMPLES	15
2.2.6.- MENORY READING	16
2.3.- SERIAL COMMUNICATION CONTROL FUNCTIONS	17
2.3.1.- SERIAL PORT OPEN	18
2.3.2.- SERIAL PORT CLOSE	18
2.3.3.- COMMUNICATION PROTOCOL STATUS	19
2.3.4.- GET COMMUNICATION HANDLE	19
2.3.5.- SERIAL PROTOCOL SELECTION	20
2.3.6.- ADDRESS AND MODE SELECTION	21
3.- OPERATING PROCEDURES	22
3.1.- READ UNSELECTED	22
3.2.- READ SELECTED	23
3.3.- WRITE PROCESS	25
APPENDIX A.-FUNCTION DECLARATIONS	27
APPENDIX B.-ERROR AND STATUS CODES FOR COMMUNICATION	28
APPENDIX C.-RIDEL5000 CONFIGURATION COMMANDS	29
APPENDIX D.-RIDEL5000 INFORMATION COMMANDS	30
APPENDIX E.-EEPROM ADDRESSES AND DATA LENGTH	31
APPENDIX F.-RF PROTOCOL ERROR CODES	32
APPENDIX G.-PC SERIAL PORT SETTING CONSTANTS	33
APPENDIX H.-FUNCTION DECLARATIONS AND INVOCATIONS	34

1.- OVERVIEW

This document provides software developers or system integrators, all the necessary information to use the RIDEL5000 DLL control functions.

The process of writing software to control the RIDEL5000 is simplified by the use of a set of functions included in a dynamic link library (DLL) developed by Softrónica, with all the necessary functions to make complex applications for the RIDEL5000 long range reader/encoder.

1.1.- DISK CONTENTS AND INSTALLATION

Two files are included in the supplied disk:

- RFIDPROT.DLL is the dynamic link library. It will have to be installed in the WINDOWS/ SYSTEM directory, or in the same directory as the application file.
- RIDEL_DEMO This directory contains the source code of a BORLAND C++ BUILDER program that make use of the different functions included in the DLL. It also includes the #define statements with the error values and constants.

2.- DLL FUNCTIONS

There are three types of DLL functions:

- Tag control functions
- RIDEL5000 control functions
- Serial communications control functions

They are described in the following sections.

2.1.- TAG CONTROL FUNCTIONS

The tag control functions are functions to support protocols to operate on the tags:

```
BYTE RFM_read_unselected (BYTE hash, BYTE blnr, BYTE nobl, BYTE *resp);
```

```
BYTE RFM_anticoll_select (BYTE hash, BYTE tse, BYTE *resp);
```

```
BYTE RFM_read (BYTE blnr, BYTE nobl, BYTE *resp);
```

```
BYTE RFM_write (BYTE hash, BYTE tse, BYTE blnr, BYTE *acttms, BYTE *data, BYTE *resp);
```

```
BYTE RFM_halt (BYTE hash, BYTE tse, BYTE *acttms, BYTE *resp);
```

```
BYTE RFM_eas (BYTE *resp);
```

```
BYTE RFM_reset_quiet_bit (void);
```

2.1.1.- READ UNSELECTED

Function

```
BYTE RFM_read_unselected (BYTE hash, BYTE tse, BYTE blnr, BYTE nobl, BYTE *resp);
```

Arguments

- hash** Value between 0 and 31. It is used in the algorithm for determining the timeslot used by the tag to answer.
- tse** 2^{tse} =Number of timeslots assigned to the function.
- blnr** Value between 0 and 15. This is the block number. The first block of the tag memory that is to be read.
- nobl** Value between 1 and 16. This is the number of blocks. The reader is going to read **nobl** blocks beginning at **blnr**.
- *resp** Pointer to an array to contain the tag memory contents. The dimension of this array has to be enough to hold the data corresponding to the number of blocks requested and the number of timeslots. Let $2^{tse}=n$, **blnr=m** and **nobl=x**. The read data will be returned in the following format:

resp length:	$4+n*(1+4*x)$
resp[0..1]	Reserved
resp[2]	Nº of bytes in answer (Low Byte)
resp[3]	Nº of bytes in answer (High Byte)
resp[4]	Timeslot 1 status
resp[5]	Timeslot 2 status
:	
resp[4+n-1]	Timeslot n status
resp[4+n..4+n+3]	Block n, timeslot 1
:	
resp[4+n+4*(x-1)..4+n+4*x-1]	Block m+x-1, timeslot1
:	
resp[4+n+4*x(n-1)..4+n+4*x*(n-1)+3]	Block m, timeslot n
:	
resp[4+n+4*(x*n-1)..4+n+4*x*n-1]	Block m+x-1, timeslot n

The status byte for every timeslot contains information of the results of the reading process for the corresponding timeslot. This byte can have the following values:

00	No error
01	No tag
02	CRC Error in the RF interface
03	Collision. 2 or more tags have answered in the same slot
08	Weak collision. There is a collision, but the answer of one tag is clear

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to read all the tags in the field that have not been previously selected. If there is one tag in the field, this function will give the correct results, but if there is more than one tag, it could give collisions, so that a selection+read selected process will be necessary.

The **blnr** and **nobl** parameters are used to describe the part of the tag memory that is going to be read. It will be important to select a number of timeslots 2^{tse} much bigger than the number of tags that are expected to be in the reading range at the same time (it is recommended to use at least double timeslots than expected tags). On the other hand, a too high number of timeslots will increase reading time.

The **hash** value is used, with the serial number of the tag, in the algorithm for determining the timeslot used by the tag to answer. It will be given a value between 0 and 31, and if the function returns collision (provided that the timeslot value is correctly selected), a new call of the function with a different hash value will have to be made.

The **resp[2..3]** bytes indicate the length of the remaining response, that is, the number of bytes returned minus 4.

2.1.2.- ANTICOLLISION SELECTION

Function

```
BYTE RFM_anticoll_select (BYTE hash, BYTE tse, BYTE *resp);
```

Arguments

- hash** Value between 0 and 31 It is used in the algorithm for determining the timeslot used by the tag to answer.
- tse** 2^{tse} =Number of timeslots assigned to the function.
- *resp** Pointer to an array to contain the tag memory contents. The dimension of this array has to be enough to hold the data corresponding to the number of timeslots. The format of the data returned will be:

resp length:	$4+n*8$	$(n=2^{tse})$
resp[0..1]	Reserved	
resp[2]	N° of bytes in answer (Low Byte)	
resp[3]	N° of bytes in answer (High Byte)	
resp[4..11]	Timeslot 1 tag serial number	
resp[12..19]	Timeslot 2 tag serial number	
:		
resp[4+8*(n-1)..4+8*n-1]	Timeslot n tag serial number	

The following responses are not valid serial numbers:

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 .. 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0F
```

They are used to give status information of the timeslot:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01	No tag
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02	CRC Error on the RF interface
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03	Collision
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x08	Weak collision

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to select all the tags present in the reading field with the anti-collision protocol, so that every tag is assigned its own timeslot. This operation has to be done before writing the tags or before making a read operation.

The **hash** value is used, with the serial number of the tag, in the algorithm for determining the timeslot used by the tag to answer. It will be given a value between 0 and 31, and if the function returns collision (provided that the timeslot value is correctly selected) , a new call of the function with a different hash value will have to be made.

When this command is sent, the reader tries to establish a link with all the tags in the field. When a tag answers in one particular timeslot, and the answer is correctly received (there is not any other tag in the same timeslot), the tag is assigned to the timeslot, and is set to QUIET mode, and the rest of the tags will not answer in this timeslot. The process continues until every tag is assigned to its timeslot.

If the timeslot number is properly selected, and the function returns collision status, the function will be called again with a different **hash** value.

The tags will remain selected until they get out of the RF field, and in this condition they can be written or read with the read function, but not with the read_unselected function.

The **resp[2..3]** bytes indicate the length of the remaining response, that is, the number of bytes returned minus 4.

2.1.3.- READ

Function

```
BYTE RFM_read (BYTE blnr, BYTE nobl, BYTE *resp);
```

Arguments

- blnr** Value between 0 and 15. This is the block number. The first block of the tag memory that is to be read.
- nobl** Value between 1 and 16. This is the number of blocks. The reader is going to read **nobl** blocks beginning at **blnr**.
- *resp** Pointer to an array to contain the tag memory contents. The dimension of this array has to be enough to hold the data corresponding to the number of blocks requested and the number of timeslots. Let **blnr=m** and **nobl=x**. The read data will be returned in the following format:

resp length:	$4+n*(1+4*x)$	(n: number of timeslots)
resp[0..1]	Reserved	
resp[2]	Nº of bytes in answer (Low Byte)	
resp[3]	Nº of bytes in answer (High Byte)	
resp[4]	Timeslot 1 status	
resp[5]	Timeslot 2 status	
:		
resp[4+n-1]	Timeslot n status	
resp[4+n..4+n+3]	Block n, timeslot 1	
:		
resp[4+n+4*(x-1)..4+n+4*x-1]	Block m+x-1, timeslot1	
:		
resp[4+n+4*x(n-1)..4+n+4*x*(n-1)+3]	Block m, timeslot n	
:		
resp[4+n+4*(x*n-1)..4+n+4*x*n-1]	Block m+x-1, timeslot n	

The status byte for every timeslot contains information of the results of the reading process for the corresponding timeslot. This byte can have the following values:

00	No error
01	No tag
02	CRC Error in the RF interface
03	Collision. 2 or more tags have answered in the same slot
04	The serial number is not the same as expected in the corresponding timeslot
08	Weak collision. There is a collision but the answer of one tag is clear

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to read all the tags in the field that have been previously selected, by means of a `anticoll_select` operation (see section on `RFM_anticoll_select` function).

The **blnr** and **nobl** parameters are used to describe the part of the tag memory that is going to be read. It will be important to select a number of timeslots 2^{15} (in the select operation) much bigger than the number of tags that are expected to be in the reading range at the same time (it is recommended to use at least double timeslots than expected tags). On the other hand, a too high number of timeslots will increase reading time.

The **resp[2..3]** bytes indicate the length of the remaining response, that is, the number of bytes returned minus 4.

2.1.4.- WRITE

Function

```
BYTE RFM_write (BYTE hash, BYTE tse, BYTE blnr, BYTE *acttms, BYTE *data, BYTE *resp);
```

Arguments

- hash** Value between 0 and 31 It is used in the algorithm for determining the timeslot used by the tag to answer.
- tse** 2^{tse} =Number of timeslots assigned to the function.
- blnr** Value between 0 and 15. This is the block number. The first block of the tag memory that is to be read.
- *acttms** Pointer to an array whose contents determine the timeslots of the tags to be written. This array is encoded in a bitwise format. The LSB of the first element corresponds to timeslot 0, and the MSB to timeslot 7. The LSB of the second element corresponds to timeslot 8, and so on. A 1 in one bit makes the tag of the corresponding timeslot to be written.
- *data** Pointer to an array where the data to write to the block **blnr** is encoded.
- *resp** Pointer to an array to contain the status of the write process for every timeslot. The dimension of this array has to be enough to hold the data corresponding to the number of timeslots. The status will be returned in the following order:

resp length:	4+n	(n= 2^{tse} number of timeslots)
resp[0..1]	Reserved	
resp[2]	Nº of bytes in answer (Low Byte)	
resp[3]	Nº of bytes in answer (High Byte)	
resp[4]	Timeslot 1 status	
resp[5]	Timeslot 2 status	
:		
resp[4+n-1]	Timeslot n status	

The status byte for every timeslot contains information of the results of the writing process for the corresponding timeslot. This byte can have the following values:

0x00	No error
0x01	No tag
0x02	CRC Error in the RF interface
0x03	Collision. 2 or more tags have answered in the same slot
0x04	The serial number is not the same as expected in the corresponding timeslot
0x08	Weak collision. There is a collision but the answer of one tag is clear
0x10	Not Write OK. The corresponding tag has not been properly written

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to write all or part of the tags in the reading field, that have been previously selected with an `anticoll_select` command (see `RFM_anticoll_selec` description in the corresponding section). Only one block will be written. It is important to notice that the 0 and 1 blocks are read only, and that care must be taken to write in block 2, because it can result with part or the whole tag in read only mode.

The **hash** value is used, with the serial number of the tag, in the algorithm for determining the timeslot used by the tag to answer. It will be given a value between 0 and 31, and if the function returns collision (provided that the timeslot value is correctly selected) , a new call of the function with a different hash value will have to be made.

The **blnr** is used to describe the block that is going to be written. It will be important to select a number of timeslots 2^{tse} much bigger than the number of tags that are expected to be in the reader range at the same time (it is recommended to use at least double timeslots than expected tags). The **acttms** parameter will be used to tell the reader which tags need to be written. The **data** parameter will contain 4 bytes corresponding to the contents of the block to be written.

The **resp[2..3]** bytes indicate the length of the remaining response, that is, the number of bytes returned minus 4.

2.1.5.- HALT

Function

```
BYTE RFM_halt (BYTE hash, BYTE tse, BYTE *acttms, BYTE *resp);
```

Arguments

hash Value between 0 and 31 It is used in the algorithm for determining the timeslot used by the tag to answer.

tse 2^{tse} =Number of timeslots assigned to the function.

***acttms** Pointer to an array whose contents determine the timeslots of the tags to be put in Halt state. This array is encoded in a bitwise format. The LSB of the first element corresponds to timeslot 0, and the MSB to timeslot 7. The LSB of the second element corresponds to timeslot 8, and so on. A 1 in one bit makes the tag of the corresponding timeslot to be written.

***resp** Pointer to an array to contain the status of the write process for every timeslot. The dimension of this array has to be enough to hold the data corresponding to the number of timeslots. The status will be returned in the following order:

resp length:	4+n	(n= 2^{tse} number of timeslots)
resp[0..1]	Reserved	
resp[2]	Nº of bytes in answer (Low Byte)	
resp[3]	Nº of bytes in answer (High Byte)	
resp[4]	Timeslot 1 status	
resp[5]	Timeslot 2 status	
:		
resp[4+n-1]	Timeslot n status	

The status byte for every timeslot contains information of the results of the writing process for the corresponding timeslot. This byte can have the following values:

0x00	No error
0x01	No tag
0x02	CRC Error in the RF interface
0x03	Collision. 2 or more tags have answered in the same slot
0x04	The serial number is not the same as expected in the corresponding timeslot
0x08	Weak collision. There is a collision but the answer of one tag is clear
0x20	Not HALT OK. The corresponding tag has not been properly written

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to halt all or part of the tags in the reading field, that have been previously selected with an anticoll_select command (see *RFM_anticoll_selec* description in the corresponding section). In halt mode the tag will no answer further reading or writing commands until the tags are removed from the RF field.

The **hash** value is used, with the serial number of the tag, in the algorithm for determining the timeslot used by the tag to answer. It will be given a value between 0 and 31, and if the function returns collision (provided that the timeslot value is correctly selected) , a new call of the function with a different hash value will have to be made.

It will be important to select a number of timeslots 2^{tse} much bigger than the number of tags that are expected to be in the reader range at the same time (it is recommended to use at least double timeslots than expected tags). The **acttms** parameter will be used to tell the reader which tags need to be halted.

The **resp[2..3]** bytes indicate the length of the remaining response, that is, the number of bytes returned minus 4.

2.1.5.- EAS

Function

```
BYTE RFM_eas (BYTE *resp);
```

Arguments

***resp** Pointer to an array to contain a value (resp[4]) in the range 0-255 corresponding to a percentage of the EAS level received.

resp length:	5	(n=number of timeslots)
resp[0..1]	Reserved	
resp[2]	Nº of bytes in answer (Low Byte)	
resp[3]	Nº of bytes in answer (High Byte)	
resp[4]	EAS value	

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to detect any tag with the EAS (Electronic Article Surveillance) bit active, in the range of the antenna. A value of 0x00 will mean no tag with EAS bit active in the field, while a 0xFF will mean a very strong response from a tag received. A threshold will be used to give EAS alarm when, for example, a value bigger than 0x40 is received.

2.1.6.- RESET QUIET MODE

Function

```
BYTE RFM_reset_quiet_bit (BYTE *resp);
```

Arguments

This functions takes no arguments.

Returned value

The returned values are described in the APPENDIX B.

Description

The tags with the QUIET bit active, are in a sleep state, so they don't answer to any command except the EAS if their corresponding EAS bit is active. To make them respond to other commands (read, write, select), they have to receive a RESET_QUIET.

When the reader sends this command, all the tags in the field with the QUIET mode active will be in active mode again.

It is important to notice the difference between the HALT and QUIET mode. The halt mode disappears when the tag is taken out of the RF field, while the QUIET mode only disappears after this command.

2.2.- READER/ENCODER CONTROL FUNCTIONS

The purpose of these functions is to control the reader operating parameters or configuration.

```
BYTE RFM_set_port (BYTE portbyte);  
  
BYTE RFM_get_port (BYTE *resp);  
  
BYTE RFM_config (BYTE mode, BYTE confbyte);  
  
BYTE RFM_get_info (BYTE mode, BYTE *resp);  
  
BYTE RFM_scope (BYTE bytes, BYTE *resp);  
  
BYTE RFM_read_mem (BYTE *resp);
```

2.2.1.- OUTPUT PORT CONTROL

Function

```
BYTE RFM_set_port (BYTE portbyte);
```

Arguments

portbyte The lower nibble of this byte correspond to the four output ports of the RIDEL5000. Writing a 1 activates the port. Writing a 0 deactivates it.

Returned value

The returned values are described in the APPENDIX B.

Description

The RIDEL5000 incorporates an 4 bits output port whose value is controlled from this function. It is used to handle output peripherals, as lights, alarms, ...

2.2.2.- INPUT PORT CONTROL

Function

```
BYTE RFM_get_port (BYTE *resp);
```

Arguments

***resp** Pointer to an array whose first byte returns the current state of the four input ports.

Returned value

The returned values are described in the APPENDIX B.

Description

The RIDEL5000 incorporates an 4 bits input port whose value is read with this function. It is used to handle input peripherals, as digital sensors or detectors.

2.2.3.- CONFIGURATION

Function

```
void RFM_config (BYTE mode, BYTE confbyte);
```

Arguments

- mode** Value selected from the list of constants of the APPENDIX C. It indicates the parameter of the RIDEL5000 to be adjusted.
- confbyte** Setting for the parameter selected with the **mode** argument.

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to control all the operating parameters of the RIDEL5000, according to the values described in APPENDIX C. There are two special cases:

- CFG_EEPROM
- CFG_DIR_EEPROM

These two values for **mode** are used to save all the configuration parameters to the internal EEPROM of the RIDEL5000. The first value CFG_EEPROM is used to set the address whose parameter is going to be modified or read, according to the APPENDIX E information. Once the address is set, it is possible to modify the contents using the CFG_DIR_EEPROM as the **mode**.

There are some parameters with more than one byte. The data length for each parameter is described in the second column of APPENDIX E. In those cases, it will be necessary to make a new pair of calls to the function, increasing the address by one. Lets see an example:

To change the EE_TEMPERATURE_1 parameter (2 bytes), follow those steps:

- Call *RFM_config* with CFG_DIR_EEPROM as the **mode**. The **confbyte** argument will be taken from the table on the APPENDIX E, in the column Dir EEPROM, and will be EE_TEMPERATURE_1, that is 24.
- Call *RFM_config*, with CFG_EEPROM as the **mode**, and the new value for the parameter as **cnfbyte**. This will save the first byte of the parameter.
- A second call to *RFM_config* with CFG_DIR_EEPROM as **mode** and 1+EE_TEMPERATURE_1, that is, 25, will be made.
- Call *RFM_config*, with CFG_EEPROM as the **mode**, and the new value for the second byte of the parameter as **cnfbyte**. This will save the second byte of the parameter.
- A Call to *RFM_get_info* (see next section), with CFG_EEPROM as the **mode**, will retrieve the value for the address selected with the last CFG_DIR_EEPROM command.

2.2.4.- CONFIGURATION INFORMATION

Function

```
void RFM_get_info (BYTE mode, BYTE *resp);
```

Arguments

- mode** Value selected from the list of constants of the APPENDIX D. It indicates the parameter of the RIDEL5000 to be retrieved.
- *resp** Pointer to an array to hold the data returned.

resp[0..1]	Reserved
resp[2]	Nº of bytes in answer (Low Byte)
resp[3]	Nº of bytes in answer (High Byte)
resp[4..]	Returned data

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to retrieve the values of the operating parameters of the RIDEL5000. The **mode** argument is used in the same manner as in the *RFM_config* command, but with the modes described in APPENDIX D.

2.2.5.- SIGNAL SAMPLES

Function

```
void RFM_scope (BYTE bytes, BYTE *resp);
```

Arguments

- bytes** Number of signal samples requested.
- *resp** Pointer to an array to hold the data returned, corresponding to the signal samples. The data returned will start at byte 4 of the response, and each data is represented in 12 bit format, so 2 bytes will be needed for one data. The format is described next:

resp length: $(resp[2]+resp[3]*256)/2$

The C function to calculate the real value for every sample is:

```
RL=(resp[2]+resp[3]*256)/2;           //Response length
for (int i=0;i<RL;i++)
  AD_value[i]=((resp((i*2)+4)*256+resp((i*2)+5))>>1)&0x0FFF;
```

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to return a set of samples corresponding to the analog signal received from the tag. This feature allows to implement a real time virtual oscilloscope display in the PC, what may be very interesting in the process of installation to see the reading range, or the noise present in the environment.

The RX signal is sampled 8 times each bit, with 12 bits resolution. The **bytes** parameter, is used to control the number of samples requested, from 1 to 255.

2.2.6.- MEMORY READING

Function

```
BYTE RFM_read_mem (BYTE *resp);
```

Arguments

***resp** Pointer to an array to contain the tag memory contents. The dimension of this array has to be enough to hold the data corresponding to the number of blocks requested and the number of timeslots. Let **tse=n**, **blnr=m** and **nobl=x**. The read data will be returned in the following format:

resp length:	4+(64*n)	n=tag number
resp[2]	L del numero de bytes	
resp[3]	H del numero de bytes	
resp[4..]	Tag data	

The status byte for every timeslot contains information of the results of the reading process for the corresponding timeslot. This byte can have the following values:

00	No error
01	No tag
02	CRC Error in the RF interface
03	Collision. 2 or more tags have answered in the same slot
08	Weak collision. There is a collision, but the answer from one tag is clear

Returned value

The returned values are described in the APPENDIX B.

Description

This function is used to ask the reader for the memory contents of a set of labels stored in the RIDEL5000 memory when working in stand-alone mode. In this mode, the RIDEL5000 stores all the tags correctly read in its internal memory.

All the corresponding data can be then retrieved with this command in the same format as a normal reading operation.

The **resp[2..3]** bytes indicate the length of the remaining response, that is, the number of bytes returned minus 4.

2.3.- SERIAL COMMUNICATION CONTROL FUNCTIONS

The purpose of these functions is to control the PC serial port.

```
bool OpenComm (int commport,DWORD baudrate,BYTE parity,BYTE databits,BYTE stopbits);  
void CloseComm (void);  
BYTE get_prot_state (void);  
HANDLE get_comm_handle (void);  
void SetProtocol (BYTE protocol);  
void SetAddrMd (WORD addr,BYTE commod);
```

2.3.1.- SERIAL PORT OPEN

Function

```
bool OpenComm (int commport, DWORD baudrate, BYTE parity, BYTE databits, BYTE stopbits);
```

Arguments

commport	PC serial port used to control the RIDEL5000.
baudrate	Operating baudrate.
parity	Parity control
databits	Number of bits in every byte
stopbits	Number of stop bits

The constant definition for these parameters are on APPENDIX G.

Returned value

The function returns true if the comm open operation is succesful, and false if it fails.

Description

This function is used to open the PC port that is going to be used to control the RIDEL5000. It shall be used before any other function call.

2.3.2.- SERIAL PORT CLOSE

Function

```
void CloseComm (void);
```

Arguments

This function takes no arguments.

Returned value

This function doesn't return any value.

Description

This function is used to close the communication serial port before terminating the application.

2.3.3.- COMMUNICATION PROTOCOL STATUS

Function

```
BYTE get_prot_state (void);
```

Arguments

This function takes no arguments.

Returned value

This function returns a value from the definitions in APPENDIX B (ERRORS).

Description

This function returns a value with the current status or error code of the communication port or the RIDEL5000 protocol.

When a function returns a value different thanb OK (0x00), then a call to *get_prot_state*, will give more information about the type of error.

2.3.4.- GET COMMUNICATION HANDLE

Function

```
HANDLE get_comm_handle (void);
```

Arguments

This function takes no arguments.

Returned value

This function returns a handle to the communication device.

Description

This function returns a value with the handle to the communication device. This handle should not be used unless it is necessary to include low level handling of the communication port.

2.3.5.- SERIAL PROTOCOL SELECTION

Function

```
void SetProtocol (BYTE protocol);
```

Arguments

protocol Protocol selected for communication

This parameter can take the following values:

00	Philips
01	Softrónica ASCII
02	Softrónica NET
03	TagWorld ASCII
04	Philips NET

See RIDEL5000 Technical Manual for a description of the different protocols.

Returned value

This function doesn't return any value.

Description

This function is used to set the serial protocol used to control the RIDEL5000. The PC will use the value from the **protocol** parameter to make all the subsequent communications. So any call to the functions of the DLL will use the new protocol for the communication with the RIDEL5000.

The RIDEL5000 protocol configured in the EE_PROTOCOL address (see *RFM_config* command description), will have to be the same as the one selected in the PC by means of this command.

2.3.6.- ADDRESS AND MODE SELECTION

Function

```
void SetAddressMd (WORD addr, BYTE commod);
```

Arguments

addr Address of the RIDEL5000 to communicate for the following commands. 0 for all (broadcast)

commod Mode of communication.

This parameter can take the following values:

'E'	Normal mode
'C'	Only command execution. There is no answer from the RIDEL5000
'P'	Polling for answer for the last command. No execution.

Returned value

This function doesn't return any value.

Description

This function is used to set the address and mode for addressing one or all the RIDEL5000 in a RS485 configuration. The PC will use the **addr** value to address one of several RIDEL5000 in a RS485 configuration.

The parameter here selected will have to match the parameters configured in the RIDEL5000, in the EE_485_ID_H and EE_485_ID_L constants (high and low parts)(see *RFM_config* command description).

The **commod** parameter is used to select between normal or 'E' mode (command-response) or a delayed-response mode.

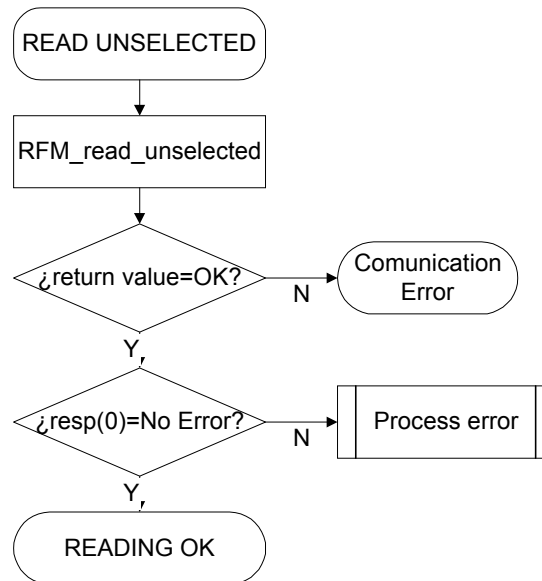
In this last mode, the PC sends a broadcast message in «only command execution» mode (addr=0,commod='C'), for all the RIDEL5000 in the communication chain. The readers will not answer. They begin execution of the command. The application program will wait for the necessary time for the command to be finished, and then will begin a polling operation for every reader in the chain, in mode «polling» (addr=device,commod='P'), so as to get the response of the last executed command.

This mode of operation is used for the execution of «slow» commands. All the devices shall make the slow operation at once, and they will answer in a faster way. If, the normal mode is used, the execution time for all the devices will add, so that the whole process will be much slower.

3.- OPERATING PROCEDURES

This chapter is used to give some guidelines about the procedures used to perform the different operations to the tags. Flow charts are given for all the basic operating procedures, like reading or writing single tags, or handling the anticollision protocol to read or write several tags simultaneously.

3.1.- READ UNSELECTED



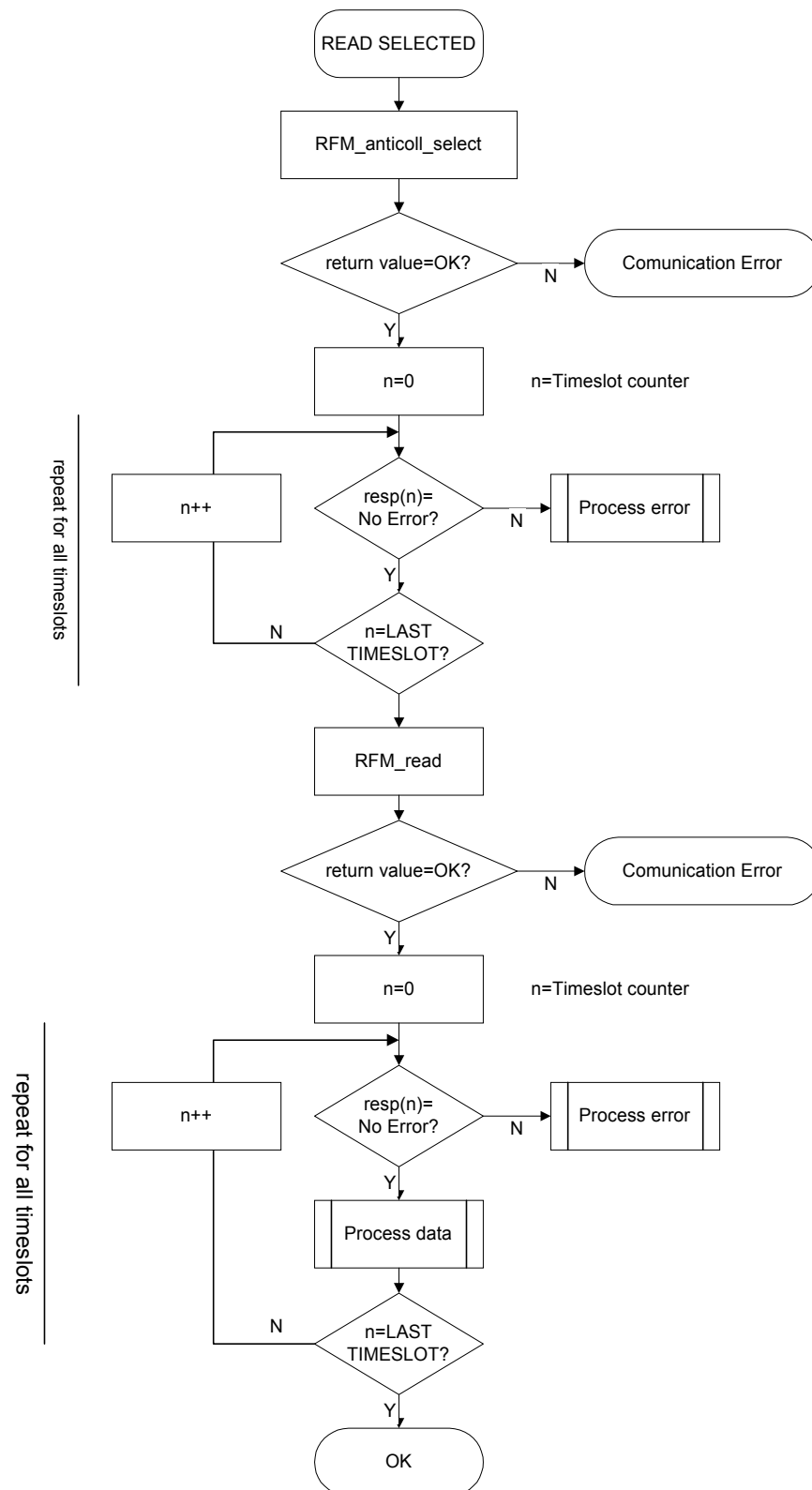
In the read unselected procedure, the read_unselected function is used. It will work properly if only one tag is present in the reading field. If there is more than one tag, it can give collision status. This condition can be checked in the first data of the response.

The communication error condition will be checked in the value returned by the function. If there is not communication error, the status of each timeslot will be checked looking for tags correctly read.

The process error routine could consist of repeating the reading process with a different hash value, if the error is caused by collision, or going to some selection & read selected process.

3.2.- READ SELECTED

This is the usual way to read a set of tags in the reading field.



In this example, an anticollision selection and a read is made for a number of tags in the field. First, the **tse** parameter must be selected, so that the number of timeslots 2^{tse} at least duplicate the number of tags expected in the field.

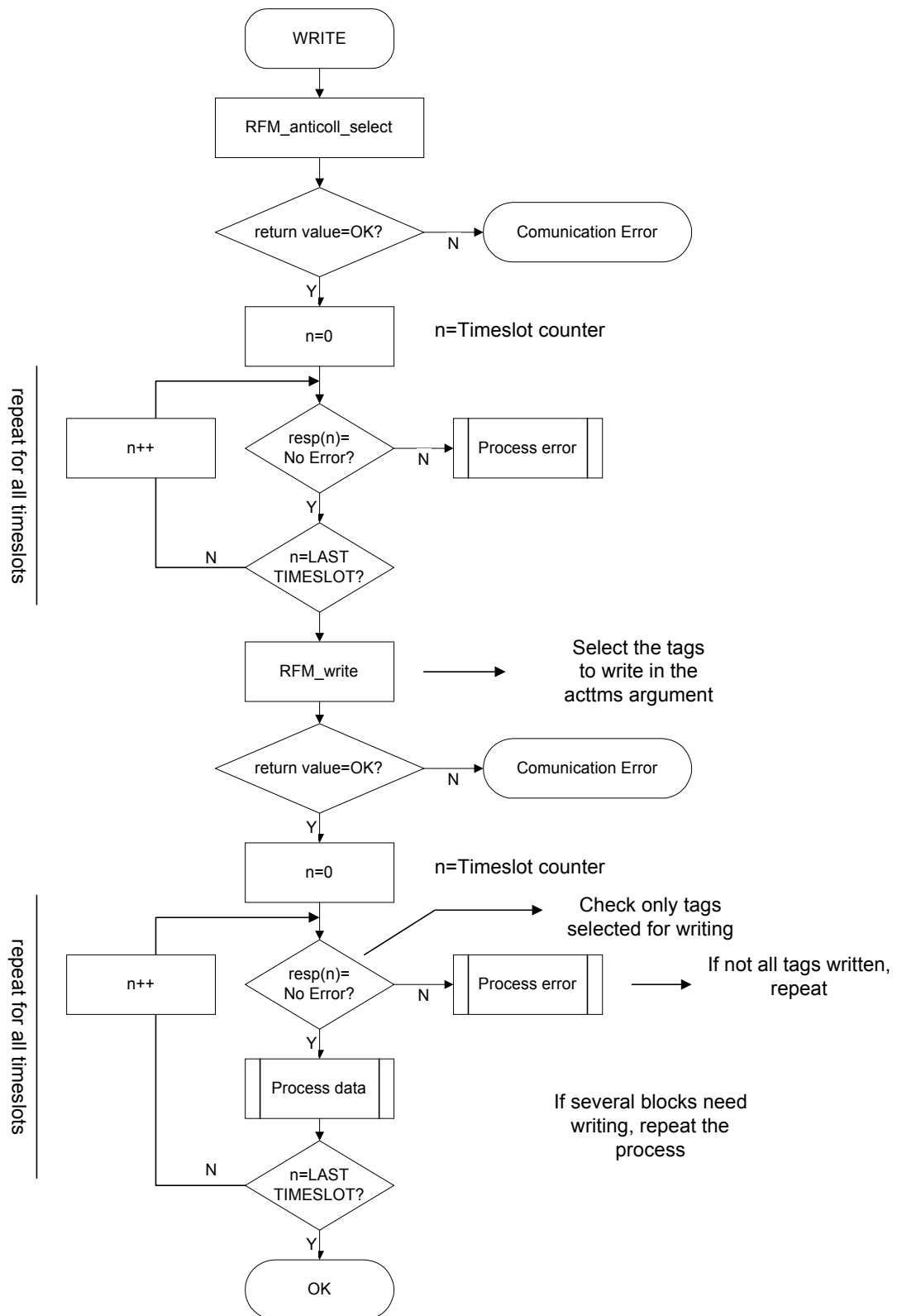
The *RFM_anticoll_select* function will return a code that will be checked for communication errors. Then the values returned for every timeslot (*resp(0)..resp(n)*) will be checked for the presence of tags or the error conditions. The serial number of the tags will also be returned in the rest of the *resp* array.

The *RFM_read* function will be called then to retrieve the information of the blocks selected by the **blnr** and **nobl** arguments. The communication error, and the tag responses will be then checked for every timeslot.

The error processing routine will have into account the type of error detected. If there is a collision problem, for instance, a new call to the function with different hash value will be issued, or a new **tse** value will be selected, in case the tags present are more than expected.

3.3.- WRITE PROCESS

The process for writing one or several tags in the field is described here.



In this example, an anticollision selection and a write is made for a number of tags in the field. First, the **tse** parameter (2^{tse} =number of timeslots) must be selected, at least to duplicate the number of tags expected in the field.

The *RFM_anticoll_select* function will return a code that will be checked for communication errors. Then the values returned for every timeslot (*resp(0)..resp(n)*) will be checked for the presence of tags or the error conditions. The serial number of the tags will also be returned in the rest of the *resp* array.

The *RFM_write* function will be called then to write one block of information to the tag memory. The arguments to the function will be selected to write all or part of the tags in the field.

Once the communication error is checked in the value returned by the function, it will be necessary to review the state returned for every tag that had to be written, in the **resp** array, so that if any of the tags is not written, the function should be called again.

The process should be repeated for all the blocks that need writing.

It is important to notice that this is a time consuming process, so if speed is a requirement, the number of blocks to write should be kept to a minimum. It is necessary to consider this requirement when designing the memory contents of the tag. **All the bits to be written in a real time situation shall be kept in the same block or blocks.**

The error processing routine will check the type of error, and process it in accordance. If there are **collision** or **weak collision** conditions, a new call to the function will be made with a new hash value. If there are **Not Write OK** conditions, a new call will be made also, but just for the tags not previously written.

APPENDIX A.-FUNCTION DECLARATIONS

```
BYTE RFM_read_unselected (BYTE hash, BYTE blnr, BYTE nobl, BYTE *resp);  
  
BYTE RFM_anticoll_select (BYTE hash, BYTE tse, BYTE *resp);  
  
BYTE RFM_read (BYTE blnr, BYTE nobl, BYTE *resp);  
  
BYTE RFM_write (BYTE hash, BYTE tse, BYTE blnr, BYTE *acttms, BYTE *data, BYTE *resp);  
  
BYTE RFM_halt (BYTE hash, BYTE tse, BYTE *acttms, BYTE *resp);  
  
BYTE RFM_eas (BYTE *resp);  
  
BYTE RFM_reset_quiet_bit (void);  
  
BYTE RFM_set_port (BYTE portbyte);  
  
BYTE RFM_get_port (BYTE *resp);  
  
BYTE RFM_config (BYTE mode, BYTE confbyte);  
  
BYTE RFM_get_info (BYTE mode, BYTE *resp);  
  
BYTE RFM_scope (BYTE bytes, BYTE *resp);  
  
BYTE RFM_read_mem (BYTE *resp);  
  
bool OpenComm (int commport, DWORD baudrate, BYTE parity, BYTE databits, BYTE stopbits);  
  
void CloseComm (void);  
  
HANDLE get_comm_handle (void);  
  
BYTE get_prot_state (void);  
  
void SetProtocol (BYTE protocol);  
  
void SetAddrMd (WORD addr, BYTE commod);
```

APPENDIX B.-ERROR AND STATUS CODES FOR COMMUNICATION

STATUS

OK	0x00
ST_WAITING_COMAND	0x01
ST_SENDING_COMAND	0x02
ST_WAITING_STX	0x03
ST_WAITING_DATA	0x04
ST_COMAND_OK	0x05

ERRORS

ERR_COMM_PORT	0x06
ERR_COMM	0x07
ERR_PROTOCOL_1	0x08
ERR_PROTOCOL_2	0x09
ERR_PROTOCOL_3	0x0A
ERR_PROTOCOL_4	0x0B
ERR_PROTOCOL_5	0x0C
ERR_CRC	0x0D
ERR_TIME_OUT	0x0E

APPENDIX C.-RIDEL5000 CONFIGURATION COMMANDS

CFG_RF_PAUSE_INIT	0
CFG_INIT	1
CFG_DEFAULT	2
CFG_EAS_LEVEL	3
CFG_EAS_LENGTH	4
CFG_PULSE_OFFSET	5
CFG_PULSE_LENGTH	6
CFG_RF_OFF_ON	7
CFG_FAST_MODE	8
CFG_FAMILY_CODE	9
CFG_APPLICATION_ID	10
CFG_MOD_DEPTH	11
CFG_RF_POWER	12
CFG_FAST_OFFSET	13
CFG_FAST_LENGTH	14
CFG_START_OFFSET	15
CFG_MODULACION_TEST	16
CFG_TUNE_VOLTAGE	17
CFG_AGC	18
CFG_EEPROM	19
CFG_DIR_EEPROM	20
CFG_RXFILTER	22
CFG_RF_VALUE	23
CFG_SLICER	24
CFG_RF_ON	25
CFG_RF_OFF	26
CFG_CSTX1	27
CFG_CPTX1	28
CFG_CSTX2	29
CFG_CPTX2	30
#define CFG_CSTX	27
#define CFG_CPTX	28
#define CFG_CSRX	29
#define CFG_CPRX	30
CFG_MODE	31
CFG_MEM_INDEX	32
CFG_TX_TUNER	33
CFG_TIME_1	34
CFG_TIME_2	35
CFG_TIME_3	36
CFG_TIME_4	37
CFG_SER_OUT_0	38
CFG_SER_OUT_1	39
CFG_SER_OUT_2	40
CFG_SER_OUT_3	41

The description of these parameters and the default values for them can be seen in the RIDEL5000 technical manual.

APPENDIX D.-RIDEL5000 INFORMATION COMMANDS

CRM_GET_TIMESLOTS	0
CRM_GET_VERSION	1
CRM_GET_NOISE_LEVEL	2
CRM_GET_MOD_LEVEL	3
CRM_GET_SYS_VARS	4
CRM_GET_MEAS	5
CRM_GET_EEPROM	6
CRM_GET_AD	7
CRM_GET_RAM	8

APPENDIX E.-EEPROM ADDRESSES AND DATA LENGTH

PARAMETER	DIREEPROM	DATALENGTH
EE_PROTOCOL	11	1 BYTE
EE_485_ID_H	12	1 BYTES
EE_485_ID_L	13	1 BYTES
EE_TUNE_VOLTAGE	14	2 BYTES
EE_AGC	16	2 BYTES
EE_POWER	18	2 BYTES
EE_MOD_INDEX	20	2 BYTES
EE_REFLECTED_AL	22	2 BYTES
EE_TEMPERATURE_1	24	2 BYTES
EE_TEMPERATURE_2	26	2 BYTES
EE_MODE	28	1 BYTE
EE_MODE_1	29	1 BYTE
EE_STAR_OFF	30	1 BYTE
EE_PULSE_OFF	31	1 BYTE
EE_PULSE_LEN	32	1 BYTE
EE_FAST_OFF	33	1 BYTE
EE_FAST_LEN	34	1 BYTE
EE_EAS_LEVEL	35	1 BYTE
EE_EAS_LEN	36	1 BYTE
EE_CAL_TIME	37	2 BYTE
EE_UMBRAL	39	2 BYTES
EE_BAUDRATE	41	2 BYTES
EE_MASK_IO_EAS	43	1 BYTE
EE_MASK_IO_RD	44	1 BYTE

APPENDIX F.-RF PROTOCOL ERROR CODES

Error code	Description	
0	OK	No error
1	No tag found	There is no tag in the timeslot
2	CRC Error	CRC communication error
3	Collision	Collision in the timeslot
4	Wrong serial number	The serial number is not the one expected in the timeslot
8	Weak collision	There is a collision but one response is clear
16	No WRITE OK	The tag is not correctly written
32	No HALT OK	The tag is not correctly halted

APPENDIX G.-PC SERIAL PORT SETTING CONSTANTS

Baud Rate

BAUD_075	0x00000001
BAUD_110	0x00000002
BAUD_134_5	0x00000004
BAUD_150	0x00000008
BAUD_300	0x00000010
BAUD_600	0x00000020
BAUD_1200	0x00000040
BAUD_1800	0x00000080
BAUD_2400	0x00000100
BAUD_4800	0x00000200
BAUD_7200	0x00000400
BAUD_9600	0x00000800
BAUD_14400	0x00001000
BAUD_19200	0x00002000
BAUD_38400	0x00004000
BAUD_56K	0x00008000
BAUD_115200	0x00020000
BAUD_57600	0x00040000

Data Bits

DATABITS_8	0x0008
------------	--------

Stop Bits

STOPBITS_10	0x0001
-------------	--------

Stop Bits

PARITY_NONE	0x0100
PARITY_ODD	0x0200
PARITY_EVEN	0x0400
PARITY_MARK	0x0800
PARITY_SPACE	0x1000

APPENDIX H.-FUNCTION DECLARATIONS AND INVOCATIONS

PROGRAM EXAMPLE FOR BORLAND C++BUILDER

Function declaration

```
typedef BYTE (*RFM_READ) (BYTE blnr, BYTE nobl, BYTE *resp);  
RFM_READ RFM_read;
```

Function inicialization

```
// handle to DLL  
hinstLib = LoadLibrary("rfidprot");  
// if handle is valid get function address.  
if(hinstLib != NULL)  
{  
    RFM_read = (RFM_READ) GetProcAddress(hinstLib, "_RFM_read");  
}  
else  
{  
    Application->MessageBox("Could not open rfidprot.DLL",NULL,MB_OK);  
    Application->Terminate();  
}
```

Function invocation

```
...  
if(RFM_read((BYTE)iblock,(BYTE)nblocks,SerData)!=ST_COMAND_OK)  
...  

```

PROGRAM EXAMPLE FOR MICROSOFT VISUAL BASIC 6.0

Function declaration

```
Private Declare Function RQ Lib "rfidprot" Alias "_RFM_reset_quiet_bit" () As Byte
```

Función call(button event handler). Labell will hold the result information

```
Private Sub Command1_Click()  
  
    Labell.Caption = RQ()  
  
End Sub
```

