

# 3660 instrument: RS232 Protocol

## 1-Introduction:

This paper describes the RS232 interface of the "Win3660 Ver3.00" program. All the RS232 functions are in the "RS232.c" module. This program is written in C language for Windows.

It contains the routines:

- Serial interface initialization
- Message transmission
- Message reception
- Errors handling
- Instrument commands

## 2- Handshaking:

The exchange of data between the computer and the micrologger always begins with the activation of the "RTS" line by the computer. In a 5sec delay, the micrologger should activate the "CTS" line to show that it is ready to receive a message.

After reception of the message, the micrologger executes a control and always sends back a message to the computer. It then clears the "CTS" line.

The message received by the computer is in the global variable Echo[4000]. After reception of the message, the computer clears the "RTS" line.

## 3- Message constitution:

The messages sent by the computer to the micrologger are built as follows:

- First byte: 'T'.
- Second and third bytes: 2 digits that indicate the function number. According to the control, the following bytes will represent a character string or bytes ( integers, floating numbers...). The 4th byte permits to know if it is a string or only bytes.
- 4th byte: 0xFF if the following bytes represent a character string, or the total number of bytes of the message.
- 5th to nth bytes: character string or (n-4) bytes.

## 4- Micrologger controls:

For each control corresponds a routine. These routines send back a byte (char) that indicates if an error happened:

- 0: no error
- 1: Transmission error: "CTS time out"
- 2: Transmission error: Transmission interrupted
- 3: Message received by the computer was not completed
- 4: Error in the Com port choice
- 5: Message received by the micrologger not legible: the micrologger sends back "ERROR0"
- 6: No message received by the micrologger in a 5 sec. delay

The bytes sent back by the micrologger are in the 'Echo' variable. Each byte has a different meaning according to the control.

#### 4.1 Eeprom writing:

Prototype: **char EcritureEeprom ( char Page, char AdressePage, char NbreBytes, char Fonction, void \*Outbuffer)**

Parameters: Page: 1 of the 16 pages of the Eeproms ( each has 256 bytes)

AdressePage: 1 of the 256 bytes in the selected page

NbreBytes: number of bytes to be written in the Eeprom

Function: 0: Simple writing

1: Current parameter writing

2: Temperature parameter writing

3: Pressure parameter writing

4: EEPROM0 configuration writing

5: Time interval between 2 automatic storage

6: High/low alams levels

7: High/low current output levels

8: Thermal cut-off temperature

9: Span gas concentration

14: EEPROM1 configuration writing

\*Outbuffer: Address of the first byte to be written in the Eeprom

Return value: Error number

Transmitted message:

- First byte: 'T'

- 2nd and 3rd bytes: 22

- 4th byte: 8 + NbreBytes

- 5th byte: Page

- 6th byte: AdressePage

- 7th byte: NbreBytes

- 8th byte: Fonction

- 9th byte to (8 + NbreBytes)th byte: bytes written in the Eeprom

Received message:

Echo[0]: Number of byte written in the Eeprom

Echo[1] and the following: bytes to be written in the Eeprom

#### 4.2 Eeprom reading:

Prototype: **char LectureEeprom ( char Page, char AdressePage, char NbreBytes)**

Parameters: Page: 1 of the 16 pages of the 2 Eeproms (each with 8 pages of 256 bytes)

AdressePage: 1 of the 256 bytes in the selected page

NbreBytes: number of bytes to be written in the Eeprom

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 23
- 4th byte: 7
- 5th byte: Page
- 6th byte: AdressePage
- 7th byte: NbreBytes

Received message:

Echo[0]: Number of byte read in the Eeprom

Echo[1] and the following: bytes read in the Eeprom

#### 4.3 RS232 test:

Prototype: **char TestRs232 ( char \*OutBuffer)**

Parameters: \*OutBuffer: Address of the string

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 24
- 4th byte: 0xFF
- 5th byte to nth byte: character string

Received message:

Echo: String received by the micrologger

#### 4.4 Analog voltages reading:

Prototype: **char LectureADC( void)**

Parameters: none

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 25
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo[0]: gas channel range( 0 to 3)

Echo[1] to Echo[4]: Voltage on the gas channel ( floating point number)

Echo[5] to Echo[8]: Voltage on the temperature channel ( floating point number)

Echo[9] to Echo[12]: Voltage on the pressure channel ( floating point number)

#### 4.5 Keyboard test:

Prototype: **char TestClavier( void)**

Parameters: none

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 26
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo[0]: Keyboard byte:     1: MEAS  
                              2: CAL  
                              4: STO  
                              8: UP  
                             16: DOWN  
                             32: MODE

#### 4.6 Display test:

Prototype: **char TestDisplay( float Nombre, char Unites)**

Parameters: Nombre: Number to be displayed ( floating)

Unites:  3- °C or °F  
         4- ppm/ppb  
         5- %/ppm  
         6- Kpa/pa  
         7- bar/mbar  
         8-  
         9- %sat  
         10- ppm  
         11- %

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 27
- 4th byte: 9
- 5th byte to 8th byte: Nombre
- 9th byte: Unites

Received message:

Echo = "OK"

#### 4.7 Measurements reading:

Prototype: **char LectureMesures( void)**

Parameters: none

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 28
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo[0] to Echo[3]: Concentration ( floating point number)

Echo[4] to Echo[7]: Temperature ( floating point number)

Echo[8] to Echo[11]: Pressure ( floating point number)

#### 4.8 Keyboard & display desactivation

Prototype: **char DesactivationInstrument(void)**

Parameters: none

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 29
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo "OK"

#### 4.9 Keyboard & display activation

Prototype: **char activationInstrument(void)**

Parameters: none

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 30
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo "OK"

#### 4.10 Checksum reading

Prototype: **char LectureChecksum(void)**

Parameters: none

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 31
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo[0]: byte that contains the checksum of the user memory (stored data)

#### 4.11 Reset of the Eeprom

Prototype: **char ResetEeprom(char fonction)**

Parameters: Fonction: 0: Parameters memory  
1: User memory

Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 32
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo= "OK"

#### 4.12 Stored data reading

Prototype: **char LectureEchantillons(void)**

Parameters: none  
Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 33
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo: 8 bytes for each sample: 4 bytes for the concentration in EEPROM0 and 4 bytes in EEPROM1 for the date & time. The 4 first bytes in both EEPROMs are for the sample 0 and the 4 last for the sample 499. An empty sample has its 4 bytes in EEPROM0 to 0xFF.

#### 4.13 Date/time reading

Prototype: **char LectureHeure(void)**

Parameters: none  
Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 36
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo: 5 bytes with a special codage

#### 4.14 Date/time writing

Prototype: **char EcritureHeure(void)**

Parameters: none  
Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 37
- 4th byte: 9
- 5<sup>th</sup> to 9<sup>th</sup> byte: date/time

Received message:

Echo= "OK"

#### 4.15 Analog output test

Prototype: **char TestSortieAnalogique(char TestFlag, float Tension)**

Parameters: none  
Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 38
- 4th byte: 9
- 5th byte: TestFlag: 0x01 active test, 0x00 inactive test
- 6<sup>th</sup> to 9<sup>th</sup> byte: Tension: 0.0 to 4095.0mV

Received message:

Echo= "OK"

#### 4.16 Alarm output test

Prototype: **char TestAlarmes(char TestFlag, char Alarme)**

Parameters: none  
Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 39
- 4th byte: 6
- 5th byte: TestFlag: 0x01 active test, 0x00 inactive test
- 6<sup>th</sup> byte: Alarme:   -0: no relay activated  
                          -1: Low alarm relay activated  
                          -2: High alarm relay activated

Received message:

Echo= "OK"

#### 4.17 Sensor current test

Prototype: **char MesureCourantSonde(void)**

Parameters: none  
Return value: Error number

Transmitted message:

- First byte: 'T'
- 2nd and 3rd bytes: 40
- 4th byte: 0xFF
- 5th byte: '\0'

Received message:

Echo: 4 bytes that represent a float value: (a current in  $\mu\text{A}$ )

### 5- Notes

-A **floating** number is coded on 4 bytes. It is necessary to invert the order of these bytes to get a format compatible with Microsoft.

-An **int** number is coded on 2 bytes. It is necessary to invert the order of these bytes to get a format compatible with Microsoft.