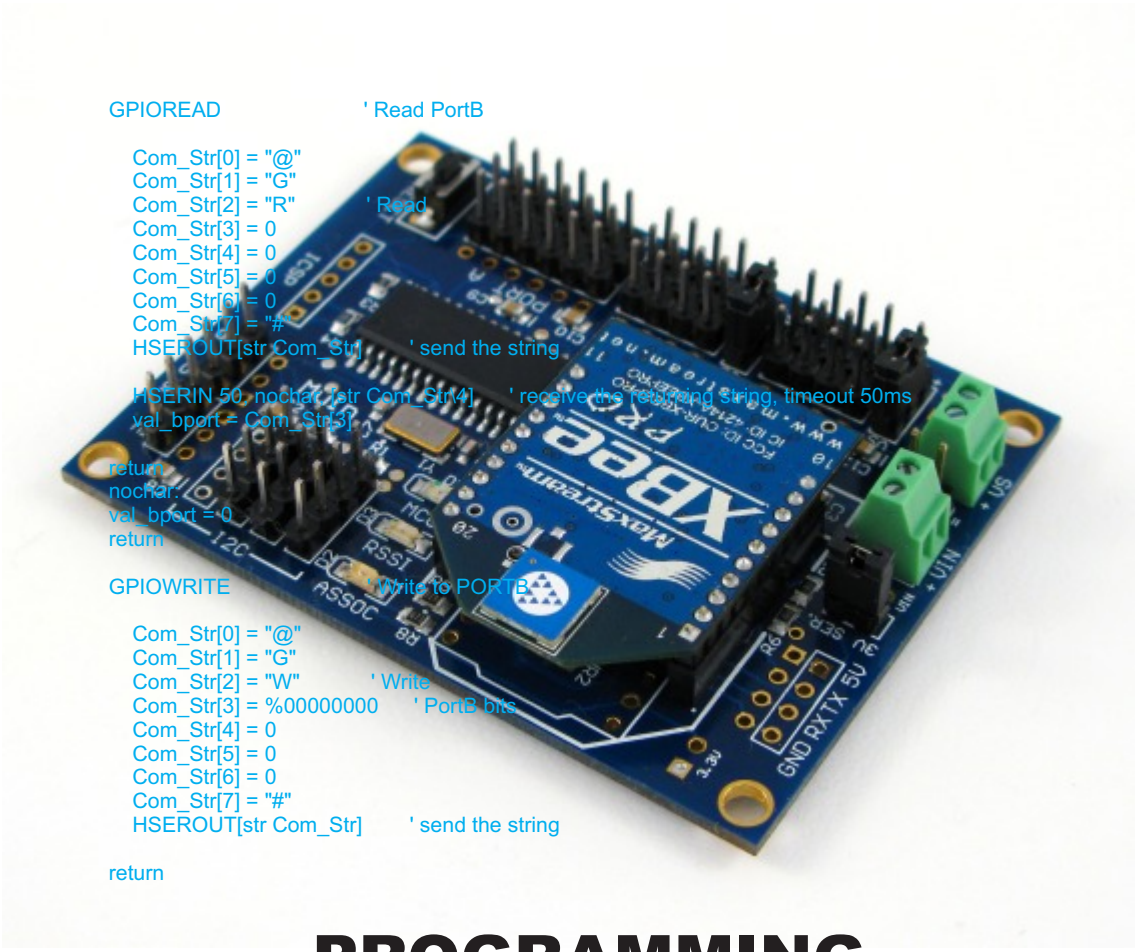
MuIn – Multi Interface Board

990.005

V1.4



PROGRAMMING MANUAL V1.0

This software is provided as is, Droids Sas do not give any warranty and will not be responsible for any damage the use of this software could cause.
We do not provide any direct software support. For any question or help, go to this forum: www.robot-italy.net we do not answer to any email related to firmware support, thanks.

The software is Open Source, you can use it to learn, study, test, etc. You cannot use this software on devices not produced from Droids Sas for commercial purposes.

We will be happy to receive and publish any modifications to our source to share with other Users and make the MuIn an ever better device. Feel free to send your firmware and programs and share it with us!

www.droids.it

COMMAND LIST

Command Structure:

The command must be 8 byte in lenght. It must start with @ and must end with # the unused byte must be 0 (zero).
If after the command start @ the MuIn does not receive a # at the 8th byte, it ignores the command and start waiting for another command.

@							#

PWM

@	P	N	Hb	Lb	0	0	#

Note: At startup the PWM is at the default frequency you set with the GUI.

n = PWM1 =1 - PWM2=2
Hb = High byte of PWM value
Lb = Low byte of PWM value
The PWM value is a 10bit value from 0 to 1024 where 512 is 50%

@	P	R	n	S	0	0	#

PWM-enable pins:

n = 1=RC0 - 2=RC5
S = 0 or 1 this is the value of the bit (0=low - 1=high)

PicBasic Example

```
Pwm_duty    var word    '10bit -  value of PWM duty cycle
Com_str     var byte[8]

PWM_Change:

   Com_Str[0] = "@"
   Com_Str[1] = "P"
   Com_Str[2] = "1"                  '  PWM Channel 1
   Com_Str[3] = Pwm_duty >> 8  ' shift 8 bit to the right (High byte)
   Com_Str[4] = Pwm_duty          ' Low byte
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       'send the string

return

PWM_Dir                    ' enable pin

   Com_Str[0] = "@"
   Com_Str[1] = "P"
   Com_Str[2] = "R"
   Com_Str[3] = "1"          ' enable pin 1 = RC0
   Com_Str[4] = "1"          ' High (value = 1)
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       ' send the string

return
```

GPIO

@	G	R	0	0	0	0	#

This command reads from PortB and returns a string like this:

@ G n #

n is the value of the port. All the bit used as Servo are 0 (zero)

@	G	W	n	0	0	0	#

This command write to PortB

n is the value to write to the Port

PicBasic Example

```
GPIOb        var byte
Com_str      var byte[8]

GPIOREAD:                    ' Read PortB

   Com_Str[0] = "@"
   Com_Str[1] = "G"
   Com_Str[2] = "R"          ' Read
   Com_Str[3] = 0
   Com_Str[4] = 0
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       ' send the string

   HSERIN 100, nochar, [str Com_Str\4]     ' receive the returning string, timeout 100ms

   GPIOb = Com_Str[2]

return

nochar:
GPIOb = 0
return

GPIOWRITE:                   ' Write to PORTB

   Com_Str[0] = "@"
   Com_Str[1] = "G"
   Com_Str[2] = "W"              ' Write
   Com_Str[3] = %11110000      ' PortB bits
   Com_Str[4] = 0
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       ' send the string

return
```

Notes:

ADC

@	A	0	0	0	0	0	#

This command reads the values of all the five ADC channels.

The MuIn after reading, returns a string like this:

@ A 1234 1234 1234 1234 1234 #

1234 is an ASCII numeric value between 0000 and 1024 this is the value of each ADC channel from ch0 to ch4.
Latency time between the request and the answer is about 100us

PicBasic Example

```
Com_Str     var byte[24]
ADC1        var word
ADC2        var word
ADC3        var word
ADC4        var word
ADC5        var word

ReadADC:                    ' ADC read

   Com_Str[0] = "@"
   Com_Str[1] = "A"
   Com_Str[2] = 0
   Com_Str[3] = 0
   Com_Str[4] = 0
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"

   HSEROUT[str Com_Str]       ' send the string

return

 HSERIN 50, nochar, [str Com_Str\23]    ' receive the ADC string, 23 byte in total, timeout 50 ms

 ‘ string conversion routines  .....

return

nochar:
   ADC1 = 0                    ' receive timeout
   ADC2 = 0                    ' all values = 0
   ADC3 = 0
   ADC4 = 0
   ADC5 = 0
return
```

Notes:

CMPS03

Compass Module

@	C	1	x	0	0	0	#

Tip: The ID number of Devantech devices can be easily changed using the GUI.

x = device address

The MuIn after reading, returns a string like this:

@ C 1234 #

PicBasic Example

```
Com_Str var byte[8]
Heading var word          ‘ contains the Direction value, a value from 0 to 3599

CMPS03:                   ' I2C Compass from Devantech Ltd

   Com_Str[0] = "@"
   Com_Str[1] = "C"
   Com_Str[2] = 1
   Com_Str[3] = $C0        '  Device I2C Address hexadecimal value
   Com_Str[4] = 0
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#”

   HSEROUT[str Com_Str\8]      ' send the string

   HSERIN 150, nochar, [str Com_Str\7]     ' receive the direction string

‘ string conversion routines  .....

return

nochar:
   Heading = 0;
```

SERVO

@	S	n	Hb	Lb	0	0	#

Note: Port B can be configured, with the GUI interface, to works as GPIO or SERVO.To use this command you must configure Port B as SERVO .

n = Servo number (1...8)
Hb = High byte of Pulse duration
Lb = Low byte of Pulse duration
The Pulse duration value is from 500 to 2500 x us (+/-90°) with a resolution of 1us

PicBasic Example

```
ServoPs    var word ‘Servo Position (500...1500...2500)
Com_str     var byte[8]

MoveServo                 ' Servo move

   Com_Str[0] = "@"
   Com_Str[1] = "S"
   Com_Str[2] = "1"                ' Servo number (1...8)
   Com_Str[3] = ServoPs >> 8   ' shift 8 bit to the right (High byte)
   Com_Str[4] = ServoPs          ' low byte
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#”

   HSEROUT[str Com_Str\8]      ' Send the string

return
```

SRFxx

I2C Ultrasonic Sensor

@	F	y	x	0	0	0	#

Tip: The ID number of Devantech devices can be easily changed using the GUI.

Send Ping

y = 1
x = device address

Send read request (60ms delay)

y = 0
x = device address

Answer

@ 1234 #

where 1234 is the range, an ASCII string value

PicBasic Example

```
Com_Str    var word[8]
Range      var word


SRFping                    ' Ping

   Com_Str[0] = "@"
   Com_Str[1] = "F"
   Com_Str[2] = "1"          ' Ping
   Com_Str[3] = $E0          ' I2C address, hexadecimal value
   Com_Str[4] = 0
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       ' send the string

return

SRFread                    'Read

   Com_Str[0] = "@"
   Com_Str[1] = "F"
   Com_Str[2] = "0"          ' Read
   Com_Str[3] = $E0          ' I2C address, hexadecimal value
   Com_Str[4] = 0
   Com_Str[5] = 0
   Com_Str[6] = 0
   Com_Str[7] = "#"
   HSEROUT[str Com_Str\8]       ' send the string

   HSERIN 150, nochar, [str Com_Str\8]     'receive the Range string

' string conversion routines  .....

return

nochar:
   Range = 0
return
```

Notes:

MD22
Dual Motor Driver

@	M	y	x	n	0	0	#

Speed control

y = R = motor 1 - L = motor 2
x = device address
n = speed 0 to 255 (128 = stop)

@	M	M	x	n	0	0	#

Mode Register

x = device address
n = 0 = normal - 2 = L becomes the turn value and R controls both motors speed

PicBasic Example

```
RM_Str   var byte[8]
LM_Str    var byte[8]

‘right motor:

   RM_Str[0] = "@"
   RM_Str[1] = "M”
   RM_Str[2] = "R"         ' Motor 1
   RM_Str[3] = $B0         ' MD22 address
   RM_Str[4] = 200         ' Speed 200
   RM_Str[5] = 0
   RM_Str[6] = 0
   RM_Str[7] = "#”

HSEROUT[str RM_Str\8]       ' Send the string

            pause 1

‘left motor:

   LM_Str[0] = "@"
   LM_Str[1] = "M”
   LM_Str[2] = "L"         ' Motor 2
   LM_Str[3] = $B0         ' MD22 address
   LM_Str[4] = 200         ' Speed 200
   LM_Str[5] = 0
   LM_Str[6] = 0
   LM_Str[7] = "#”

   HSEROUT[str LM_Str\8]       ' Send the string

return
```

Notes:

I2C WRITE

@	I	x	n	P1	P2	P3	#

x = device address
n = number of bytes to write (1 to 3) (if more bytes needed, send more commands)
P1...P3 = parameters to send to the device, if not used must be 0 (zero)

PicBasic Example

```
Snd_Str   var byte[8]

I2C Write:

   Com_Str[0] = "@"
   Com_Str[1] = "I"
   Com_Str[2] = 2            'Bytes to send
   Com_Str[3] = $E0          'I2C Device address
   Com_Str[4] = $0           'memory address
   Com_Str[5] = $51          'command
   Com_Str[6] = 0            'not used
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       ' Send the string

return
```

I2C READ

@	L	x	n	P1	P2	0	#

x = device address
n = number of bytes to receive (1 to 16)
P1, P2 = specify the memory location in EEPROM like I2C devices, if not used must be zero

The MuIn after reading, returns a string like this:

@ I x P0... Pn #

PicBasic Example

```
Snd_Str   var byte[8]

I2C Write:

   Com_Str[0] = "@"
   Com_Str[1] = "L"
   Com_Str[2] = $E0          'I2C Device address
   Com_Str[3] = 2            'bytes to read
   Com_Str[4] = 2            'first memory address
   Com_Str[5] = 0            'not used
   Com_Str[6] = 0            'not used
   Com_Str[7] = "#"

   HSEROUT[str Com_Str\8]       ' Send the string

return
```

Notes: