micromagic systems

Animatronic & Puppet control systems for Film & Television

p.Brain-HexEngine
Packet Interface Protocol Guide V1.2
Updated 17/04/09

Contents

Description

The hexapod can be controlled using either a simple control interface "SIM CONTROL" as described in the *HexEngine* configuration guide document, or with the Packet Interface Protocol (PIP) which is described in this document. The PIP gives enhanced control features as well as access to the simple control interface commands.

Note: Examples of packet data in this document are either byte names within square bracket [], or hexadecimal values. For the purpose of clarity each value or bracketed name is separated with a comma, the comma's are not sent as part of the packet.

Packet Interface Protocol (PIP)

Packet Format
The packet interface commands are sent in packets of data, each packet must conform to the following protocol:

[Header Byte],[Packet Count],[n Bytes of Data.....],[Check Sum]

Header Byte = 0x7e
Packet Count = Data bytes excluding Header Byte, Packet Count & Check Sum
Check Sum = 0xff – (8 bit sum of all data bytes)

PIP Modes 0 & 1

There are two PIP modes defined by the "PIP" command in the HexEngine configuration menu:

Mode	Pros	Cons
0 = Simple	Simple Programming, Packet length fixed by data bytes.	More chance of packet corruption in segmented data streams.
1 = Escape	More robust, less chance of packet corruption.	More complex programming, packet length can vary depending on data content.

In mode 0, packets are sent as above, as long as the data is not too segmented, this protocol will usually suffice for most applications, also in mode 1, data packets are a fixed length which makes packet reception coding easier. Mode 1 is a much more robust protocol which uses escape codes, with escape codes, packets can vary in length according to the data transmitted, however, the header byte 0x7e can never bee repeated within the data packet, which gives a much better communications protocol. See below for further details on mode 1 protocol.

PIP Mode 1 - Illegal Hex Codes (Escape codes)

There are also two illegal codes which have to be checked for within the packet, this is so that the header byte cannot be duplicated within the packet data and be misread as a packet start. To do this each byte in the packet apart from the header byte is checked against the two illegal hexadecimal codes:

0x7e = Packet Header
0x7d = Escape Code

If either of the two illegal codes are found, they must be replaced with the Escape Code (0x7d) followed by the illegal code exclusively ored with 0x20.

Illegal code example
If a packet consisted of 3 bytes of data, and for the purpose of this example one of these bytes was the escape code '0x7d' the packet would look like this:

0x7e,0x03,0x01,0x7d,0x02,0x7f

The second data byte is 0x7d, and must be Escaped as follows:

Illegal code: 0x7d
Replaced with: 0x7d,0x5d (0x5d = 0x7d xor 0x20)

So our illegal packet would become:

0x7e,0x03,0x01,0x7d,0x5d,0x02,0x7f

Note that even though there is an additional byte in the packet, the packet count and checksum do not change, they are calculated before illegal codes are Escaped.

PIP Handshaking, ACK, NACK & BUSY

Upon reception of a PIP packet, a return PIP packet is sent with one byte of data corresponding to an ACK (Acknowledge) or NACK (Not Acknowledged).

ACK = 'k' (PIP = 0x7e,0x01,0x6b,0x94)
NACK = '?' (PIP = 0x7e,0x01,0x3f,0xc0)
BUSY = 'b' (PIP = 0x7e,0x01,0x62,0x9d)

ACK's and NACK's are only sent for properly formatted PIP packets, if a packet is sent with a bad checksum or incorrect protocol, no reply is sent. If a PIP packet is sent with correct formatting, but the data command is unrecognised, a NACK is sent in reply. The BUSY reply is for commands 'V', 'N', 'v' & 'n'. This response indicates that the packet was received, but the relative command is still busy.

Using “SIM CONTOL” commands in PIP packets

As previously mentioned, all the “SIM CONTROL” commands can be sent using the PIP. All that is different is that the commands need to be sent within a packet. Example, to send the simple control command to wake the hexapod ('+') using the packet interface the following would be sent

0x7e,0x01,0x2b,0xd4

To sleep the hexapod using the ('-') command:

0x7e,0x01,0x2d,0xd2

Table of "SIM CONTROL" commands

CHAR	Description	CHAR	Description
+	Power up hexapod	1	Wave gait 1 (slowest)
-	Power down hexapod	2	Wave gait 2
SPACE	Stop hexapod	3	Wave gait 3 (In my opinion, the best!)
!	Emergency stop hexapod (Shuts off servos instantly)	4	Tripod gait (fastest)
w	Walk forward	5	On Road gait (fast, fluid)
s	Walk backwards	6	Off Road gait (slower, better ground clearance)
a	Turn Left	7	Decrease leg transfer speed by 0.1 seconds
d	Turn Right	8	Increase leg transfer speed by 0.1 seconds
q	Crab Left	9	Reset leg transfer speed to power on default (DLT)
e	Crab Right	r	Reset legs to neutral position
b	Switch on full 3D balance mode	c	Switch off full 3D balance mode
		ESC	Return to main menu

Table of "PIP CONTROL" commands

The following commands are only available in PIP mode, some require additional bytes of data following the extended command, some have returned PIP packets.

Command	Description	PIP Data Bytes	Engine Version
'M'	Proportional walking control	3	1.0
'B'	Body proportional Rotation & Translation	6	1.0
'A'	Auxiliary servo position	12	1.0
'V'	Body Translate & Rotate to position (timed motion move)	8	1.0
'N'	Auxiliary Servo move to position (timed motion move)	14	1.0
'E'	Stop 'V' & 'N' Motion moves immediately	0	1.0
'v'	Poll body translate & rotate motion	0	1.0
'n'	Poll servo move motion	0	1.0
'I'	External I2C Data Write	4 to 36	1.0
'i'	External I2C Data Read	3	1.0
'O'	External Digital I/O Write	1	1.0
'o'	External Digital I/O Read	0	1.0
'p'	External 12bit Analogue Capture	0	1.0
'H'	Head PAN & TILT position	2	1.0
&	Query PIP mode	0	1.2
{	Set PIP mode = 0:SIMPLE	0	1.2
}	Set PIP mode = 1:ESCAPED	0	1.2
J	Set the body rotation offset X,Y,Z	3	1.2

"PIP Control" commands explained

'M'
This command is ideal for joystick like control of the hexapod and is designed to be repeatedly sent to the HexEngine at around 10hz to 20hz. The three data bytes of this command are:

byte 0:	Left/Right crab stride	(X Translate)
byte 1:	Forward/Backward stride	(Y Translate)
byte 2:	Left/Right Turning Stride	(Z Rotate)

Each byte is a 8 bit signed, -128 to 127.
byte 1 example: 0 = stop, -128 = full reverse, 127 = full forward.

'B'
This command is ideal for joystick like control of the hexapod and is designed to be repeatedly sent to the HexEngine at around 10hz to 20hz. The 6 data bytes of this command are:

byte 0:	Body Rotate X
byte 1:	Body Rotate Y
byte 2:	Body Rotate Z
byte 3:	Body Translate X
byte 4:	Body Translate Y
byte 5:	Body Translate Z

Each byte is a 8 bit signed, -128 to 127.
byte 0 example: -128 = full -X rotate, 127 = full +X rotate

'A'
This command is ideal for joystick like control of the hexapod and is designed to be repeatedly sent to the HexEngine at around 10hz to 20hz. The twelve data bytes are arranged as 6 unsigned word values with the high byte of each word first. Each word can be between 500 and 2500 which corresponds to micro seconds of PWM (pulse width modulation). PWM values out side the HexEngine range will be clamped.

byte 0,1:	Auxiliary Servo 1 (MSB, LSB)
byte 2,3:	Auxiliary Servo 2
byte 4,5:	Auxiliary Servo 3
byte 6,7:	Auxiliary Servo 4
byte 8,9:	Auxiliary Servo 5
byte 10,11:	Auxiliary Servo 6

'V'
This command is used to move the body rotation & translation from its current position, to a new position in a defined amount of time. The move will ease in at the start and ease out at the end. The 8 data bytes of this command are:

byte 0:	Body Rotate X
byte 1:	Body Rotate Y
byte 2:	Body Rotate Z
byte 3:	Body Translate X
byte 4:	Body Translate Y
byte 5:	Body Translate Z
byte 6,7:	Frame Count (MSB, LSB)

The Frame Count defines how long the move will take in frames, each frame is 20 milliseconds. The minimum frame count is 10 (200 mS), and the maximum 500 (10 S). If a move is already in progress, the command will be ignored and a busy ACK is returned.

'N'
This command is used to move the auxiliary servos from their current position, to a new position in a defined amount of time. The move will ease in at the start and ease out at the end. The first twelve data bytes are arranged as 6 unsigned word values with the high byte of each word first. Each word can be between 500 and 2500 which corresponds to micro seconds of PWM (pulse width modulation). PWM values out side the HexEngine range will be clamped. The 14 data bytes of this command are:

byte 0,1:	Auxiliary Servo 1 (MSB, LSB)
byte 2,3:	Auxiliary Servo 2
byte 4,5:	Auxiliary Servo 3
byte 6,7:	Auxiliary Servo 4
byte 8,9:	Auxiliary Servo 5
byte 10,11:	Auxiliary Servo 6
byte 12,13:	Frame Count (MSB, LSB)

The Frame Count defines how long the move will take in frames, each frame is 20 milliseconds. The minimum frame count is 10 (200 mS), and the maximum 500 (10 S). If a move is already in progress, the command will be ignored and a busy ACK is returned.

'E'
This command will stop any current 'V' or 'N' motion move currently running.

'v'
This command will return the status of the Body Rotate & Translate motion move with either: BUSY or ACK (Not Busy)

'n'
This command will return the status of the Auxiliary Servo motion move with either: BUSY or ACK (Not Busy)

'I'
I2C Data write. This command allows you to send I2C data to an external device connected to the p.Brain-ds24 SDA, SCL lines. With this command it is possible to interface to I2C memory devices such as the 24xx series eeproms, or ultrasonic range finders such as the SRF08. The command can send up to 32 bytes of data to an I2C device. The data bytes of this command are:

byte 0:	I2C Address
byte 1:	I2C Data Count & I2C Options
byte 2:	I2C Register Address
byte 3:	I2C Data 0
byte 4:	I2C Data 1
...	
byte 35:	I2C Data 31

I2C Data Count (byte 1) can be up to 32 bytes. The upper three bits of byte 1 are reserved for I2C options, see table below. O*nly send as many data bytes as defined by I2C Data Count (byte 1).*

Byte 1 Option Bit	Description
Bit 7 = 0	I2C Low speed 100Khz
Bit 7 = 1	I2C High speed 400Khz
Bit 6 = 0	Each byte is written as a separate I2C data write, with the I2C Register Address incremented after each write.
Bit 6 = 1	The I2C address (byte 0) is written, followed by the I2C Register Address (byte 2, this could be a command byte), followed by the desired amount of data defined by I2C Data Count (byte 1)

Example PIP data bytes for starting a micro second range PING on an SRF08 device.

['O'],[I2C Address],[I2C Data Count],[I2C Register],[I2C Data]
0x4f,0xe0,0x01,0x00,0x52

'i'
I2C Data Read. This command allows you to read I2C data from an external device connected to the p.Brain-ds24 SDA, SCL lines. With this command it is possible to interface to I2C memory devices such as the 24xx series eeproms, or ultrasonic range finders such as the SRF08. The command can read up to 32 bytes of data from an I2C device. The three data bytes of this command are:

byte 0: I2C Address
byte 1: I2C Data Count & I2C Bus Speed
byte 2: I2C Register Address

I2C Data Count (byte 1) can be up to 32 bytes, the MSB bit 7 of byte 1 controls the I2C bus speed, 1 = 400 Khz, 0 = 100 Khz.

This command will read (I2C Data Count) bytes starting from the (I2C Register Address), after each read the I2C register Address is incremented. Data is stored in a buffer, once all data bytes have been read, the data is returned in a PIP packet of (I2C Data Count + 1) in length. The first byte of the returned PIP will be 'I', followed by the I2C data bytes.

Example PIP data bytes to read the PING data on an SRF08 device.

['I'],[I2C Address],[I2C Data Count],[I2C Register]
0x49,0xe0,0x03,0x01

Retuned PIP data from HexEngine:
['I'],[Light Sensor],[Echo High Byte],[Echo Low Byte]

Click here for further information on the SRF08 range finder.

'O'
Digital I/O Write. This command writes one byte of data to the external digital I/O port on the p.Brain-ds24. If using a p.Brain-SMB mother board, this is mapped to CN19. Only pins which have been assigned to digital I/O and configured as outputs are effected. There are no current limit resistors on these output pins of the dsPIC33, so care should be taken not to short the pins in digital output mode. Each output can source and sink 4 mA. The data byte for this command is as follows:

byte 0: 8 bits of digital I/O data.

o
Digital I/O Read. This command reads the current state of the external digital I/O port on the p.Brain-ds24. If using a p.Brain-SMB mother board, this is mapped to CN19. Only pins which have been assigned to digital I/O using the "ADC" & "DIO" commands within the HexEngine config and configured as inputs can be driven by the user. Pins configured as outputs will return the current output state. When the HexEngine receives this command, the port byte is read and returned in a PIP packet. The first byte of the PIP data will be 'o' followed by the 8 bits of the port state as one byte:

Returned PIP data:
byte 0: 'o'
byte 1: Port State (8 bits)

'p'
Analogue Input Read. This command returns the current analogue voltage applied to the eight analogue input pins of the external port on the p.Brain-ds24. If using a p.Brain-SMB mother board, this is mapped to CN19. Only pins that have been configured as analogue input pins using the "ADC" command within the HexEngine configuration are captured. When the HexEngine receives this command, a PIP packet is returned containing the analogue input levels. The 17 data bytes returned are as follows:

Returned PIP Data:
byte 0: 'p'
byte 1: Analogue Input 0 High Byte
byte 2: Analogue Input 0 Low Byte
byte 3: Analogue Input 1 High Byte
byte 4: Analogue Input 1 Low Byte
byte 5: Analogue Input 2 High Byte
byte 6: Analogue Input 2 Low Byte
byte 7: Analogue Input 3 High Byte
byte 8: Analogue Input 3 Low Byte
byte 9: Analogue Input 4 High Byte
byte 10: Analogue Input 4 Low Byte
byte 11: Analogue Input 5 High Byte
byte 12: Analogue Input 5 Low Byte
byte 13: Analogue Input 6 High Byte
byte 14: Analogue Input 6 Low Byte
byte 15: Analogue Input 7 High Byte
byte 16: Analogue Input 7 Low Byte

'H'
This command is ideal for joystick like control of the PAN/TILT head. The command requires that the PAN/TILT servo limits have been configured using the PAM,PA+,PA-,TIM,TI+,TI- commands within the CONFIG menu. Each of the data bytes used to control head position are scaled to the pre-set servo limits. The two data bytes of this command are:

byte 0: Head Left/Right (Pan)
byte 1: Head Up/Down (Tilt)

Each byte is a 8 bit signed, -128 to 127.

'&'
This command will return the current PIP mode either 0 or 1 (Simple/ Escaped)

Returned PIP Data:
byte 0: '&'
byte 1: PIP Mode 0 or 1

'{'
Sets PIP mode = 0:SIMPLE

'}'
Sets PIP mode = 1:ESCAPED

'J'
This command defines the body rotation offset point in X, Y & Z. Each data byte is in mm. The rotation offset is reset to 0,0,0 upon boot. The 3 data bytes of this command are:

byte 0: Body Rotate Offset X
byte 1: Body Rotate Offset Y
byte 2: Body Rotate Offset Z

Each byte is a 8 bit signed, -128 to 127.
For example, to set the rotation point to the head of the MSR-H01 hexapod, X = 0, Y = 118, Z = 0.

C Code example to create a PIP Packet

The following code demonstrates how to create a HexEngine PIP packet. The code assumes you can open, close & send a byte on a serial communications port on the target platform

```
// **********************************************************************
// INCLUDE
// **********************************************************************

#include        <stdio.h>
#include        <stdlib.h>
#include        <conio.h>

// **********************************************************************
// DEFINITIONS
// **********************************************************************

#define CMD_PIP_HEADER 0x7e
#define CMD_PIP_ESCAPE 0x7d
#define CMD_PIP_XOR    0x20


// **********************************************************************
// FUNCTION DEFINITIONS
// **********************************************************************

//      SENDS ONE BYTE TO SERIAL PORT
void    SendByte( char lC );
//      OPENS SERIAL PORT
void    OpenComms( void );
//      CLOSES SERIAL PORT
void    CloseComms( void );
//      SENS PIP PACKET
void    SendCMDPIPPacket( char *pPtr, char pLen );
//      SENDS BYTE CHECKS FOR ILLEGAL CODES
void    SendByteCheckCodes( char lC );


// **********************************************************************
// MAIN CODE
// **********************************************************************


void    main( void )
{
char    lBuffer[10];

//      OPEN COMS DEVICE (User defined communications device)
OpenComms();

//      SEND HEXENGINE WAKE COMMAND
lBuffer[0] = '+';
SendCMDPIPPacket( lBuffer, 1 );

//      Delay 10 Seconds
delay(10000);

//      SEND HEXENGINE SLEEP COMMAND
lBuffer[0] = '-';
SendCMDPIPPacket( lBuffer, 1);

//      CLOSE COMMS DEVICE
CloseComms();

//      END
}
```

```
// ***********************************************************************
// Send Command PIP Packet
//
// Input:
// pPtr = char pointer to data buffer
// pLen = length of data buffer
//
// Output: none
// ***********************************************************************

void    SendCMDPIPPacket( char *pPtr, char pLen )
{
char    lC,lCs,lB;

//      SEND PIP_HEADER
SendByte( CMD_PIP_HEADER );

//      PACKET COUNT NEEDS ESCAPE CODE CHECKING!
SendByteCheckCodes( pLen );

//      SETUP CHECK SUM
lCs = 0;

//      SEND DATA WITH ESCAPE CODES IF NECESSARY.
//      CALCULATE CHECKSUM WITHOUT ESCAPE CODES.
for( lC = 0; lC < pLen; lC++ )
        {
        //      lB = NEXT BYTE OF DATA
        lB = *pPtr;
        //      CHECK FOR ILLEGAL CODE & SEND
        SendByteCheckCodes( lB );
        //      ADD TO CHECK SUM
        lCs += lB;
        //      INCREMENT POINTER
        pPtr++;
        }

//      CALCULATE CHECKSUM
lCs = 0xff - lCs;

//      CHECK FOR ILLEGAL CODE + SEND CHECKSUM
SendByteCheckCodes( lCs );

}


// ***************************************************************************
// CHECK COMMAND ILLEGAL CODE
//
// INPUT: lC = Byte to be sent
//
// OUTPUT: none
// ***************************************************************************

void    SendByteCheckCodes( char lC )
{

//      IF SENDING PIP MODE 0 PACKETS, REMOVE FROM HERE..
if( lC == CMD_PIP_HEADER || lC == CMD_PIP_ESCAPE )
        {
        SendByte( CMD_PIP_ESCAPE );
        SendByte( lC ^ CMD_PIP_XOR );
        }
else
//      TO HERE..
        {
        SendByte( lC );
        }


}
```

Basic Stamp Code example to create a PIP Packet

The following code demonstrates how to create a HexEngine PIP packet. The code assumes you can open, close & send a byte on a serial communications port on the target platform

```
' =========================================================================
'
' File:    HE_2_BS.BS2
' Author   Matt Denton
' Date:    26 AUG 2008
'
' {$STAMP BS2p}
' {$PBASIC 2.5}

' -----[ I/O Definitions ]-------------------------------------------------
LED     PIN 0
SER_OUT PIN 5

' -----[ Constants ]-------------------------------------------------------
CMD_PIP_ESCAPE CON         $7d
CMD_PIP_HEADER CON         $7e
CMD_PIP_XOR    CON         $20

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T1200       CON     813
    T2400       CON     396
    T4800       CON     188
    T9600       CON     84
    T19K2       CON     32
    T38K4       CON     6
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T4800       CON     500
    T9600       CON     240
    T19K2       CON     110
    T38K4       CON     45
    T57K6       CON     23
    T115K2      CON     2
  #CASE BS2PX
    T1200       CON     3313
    T2400       CON     1646
    T4800       CON     813
    T9600       CON     396
    T19K2       CON     188
    T38K4       CON     84
#ENDSELECT

COM_SevenBit        CON     $2000
COM_Inverted        CON     $4000
COM_Open            CON     $8000

Baud            CON     T115K2

' -----[ Variables ]-------------------------------------------------------
PIP_Byte  VAR Byte
PIP_Len   VAR Byte
PIP_Temp  VAR Byte
PIP_Buff  VAR Byte(8)
PIP_Cs    VAR Byte

Temp      VAR Byte


' -----[ Initialization ]--------------------------------------------------
Setup:
HIGH SER_OUT


' -----[ Program Code ]----------------------------------------------------
```

```
Main:

  ' WAKE THE HEXENGINE
  PIP_Buff(0) = "+"
  PIP_Len = 1
  GOSUB Send_Pip_Packet

  PAUSE 5000

  ' SETUP BUFFER FOR WALK FORWARD
  PIP_Buff(0) = "w"
  PIP_Len = 1

  ' TRANSMIT WALK FORWARD FOR 10 SECONDS
  FOR Temp = 1 TO 100
    GOSUB Send_Pip_Packet
    PAUSE 100
  NEXT

  ' STOP THE HEXENGINE
  PIP_Buff(0) = " "
  PIP_Len = 1
  GOSUB Send_Pip_Packet

  ' SLEEP THE HEXENGINE
  PIP_Buff(0) = "-"
  PIP_Len = 1
  GOSUB Send_Pip_Packet

  END

' -----[ Subroutines ]----------------------------------------------------
' **********************************************************************
' SENDS PIP PACKET STORED IN PIP_BUFF OF PIP_LEN LENGTH
'
Send_PIP_Packet:
  ' SEND HEADER
  SEROUT SER_OUT, Baud, [CMD_PIP_HEADER]
  ' SEND PIP LENGHT
  PIP_Byte = PIP_Len
  GOSUB Send_Byte_Check_Codes
  ' SETUP CHECK SUM
  PIP_Cs = 0
  ' SEND PACKET
  FOR PIP_Temp = 0 TO PIP_Len-1
    PIP_Byte = PIP_Buff( PIP_Temp )
    PIP_Cs = PIP_Cs + PIP_Byte
    GOSUB Send_Byte_Check_Codes
  NEXT
  ' SEND CHECK SUM
  PIP_Byte = $ff - PIP_Cs
  GOSUB Send_Byte_Check_Codes

  RETURN

' **********************************************************************
' SENDS PIP BYTE TO SERIAL PORT AND CHECKS PIP CODES
'
Send_Byte_Check_Codes:
  ' IF SENDING PIP MODE 0 SIMPLE PACKETS, REMOVE FROM HERE..
  ' CHECK CODE AGAINST ILLEGAL CODES
  IF( PIP_Byte = CMD_PIP_HEADER | PIP_Byte = CMD_PIP_ESCAPE ) THEN
    ' SEND ESCAPED CODE
    SEROUT SER_OUT, Baud, [CMD_PIP_ESCAPE, PIP_Byte ^ CMD_PIP_XOR]
  ELSE
  ' TO HERE..
    ' SEND NORMAL CODE
    SEROUT SER_OUT, Baud, [PIP_Byte]
  ENDIF
  RETURN
```

Legal

Please read fully before purchasing any merchandise from micromagic systems ltd.

PRODUCTS

In no event shall micromagic systems be liable for any claim for incidental, consequential damages, or any injuries sustained due to the use of or improper use of products and / or kits purchased out of or in connection thereof with the manufacture, sale, delivery or use of any product in this catalogue or web site. All micromagic systems products purchased should NOT be used for medical, life-saving, life-support, or any applications that could cause injury, dangerous / hazardous situations or consequential damages resulting from the use of the mechanical, hardware or software products sold or represented by micromagic systems.

All products sold by micromagic systems are for Self Learning Experiences, and for Safe Entertainment.

Prices and Specifications

Note

Product specifications, prices listed and availability of items in our web site and in our printed catalogue are subject to change without notice. All prices shown in our web site and in our printed catalogue are believed accurate at time of publication, but subject to change without any notice. We will always advise you of the new price increase and seek your approval before processing any orders you place.

General Product Terms and Conditions

Call micromagic systems before returning any items.

PRODUCT RETURNS POLICY

All sales are final.
New products are warranted for 30 days. Any return for repair or replacement must be pre-authorized by micromagic systems and under no circumstance will returns be accepted unless so authorized. Return Products under warranty must be pre-approved by MMS and sent via certified mail, prepaid and insured, for your protection. (Please note we cannot refund shipping fees). All electronic kit sales are final. Due to the fact that components of the kit may be damaged during assembly we do not accept returns or refunds on any electronic kits

If you receive damaged merchandise, you must contact micromagic systems within 2 days of receipt of your original order. Specify clearly the reason for your refusal. We will exchange returned merchandise for same new merchandise, or for the item sterling amount within 7 days once we receive the returned damaged items from you. Proof of mailing is advised, as we cannot be held responsible for loss of the returned merchandise in mail transit. All return postage is non-refundable. The merchandise, including packing and wrapping material, being returned should be in the same condition as when you received them. Please contact us via e-mail at **matt@micromagicsysstems.com**. Defective merchandise will be replaced (No cash will be refunded). We reserve the right to refuse to replace any merchandise, which our micromagic systems technicians determine to be damaged by the user, or through inappropriate use of that merchandise.

WARRANTY POLICY

We guarantee all products except electronic kits to be free of defects in workmanship and material for 30 days from the purchase, delivery date. We will repair or replace non-electronic kits (No cash will be refunded), at our option providing there is no evidence of customer misuse or alteration to that product item.

micromagic systems carries a limited 30 day warranty on most all items, some items carry an additional number of warranty days or special restrictions. If you want specific warranty information about a product contact micromagic systems to obtain that information.

We are not able to offer any refunds or accept returns for the following items and products: Electronic Kits.

CANCELLATION POLICY

Please be aware that if you cancel an order you may be responsible for restocking fees and / or shipping charges, including charges for return shipping. Cancelled orders are subject to a 25% or £10.00 minimum restocking fee. Orders cancelled within 24 hours of order placement will not be subject to restocking fees however this does not apply to orders with Express Shipping and Handling.