

# Hemisson Modules

## Serial Command Guide

September 5, 2005

[www.roadnarrowsrobotics.com/support/hemisson/supHem-Base.html](http://www.roadnarrowsrobotics.com/support/hemisson/supHem-Base.html)



**RoadNarrows LLC**

## **RoadNarrows LLC**

<http://www.roadnarrowsrobotics.com/>

<mailto:oneway@roadnarrowsrobotics.com>

### NOTICE

The authors of this guide have used their best efforts in preparing and validating the accuracy contained within. The authors and RoadNarrows LLC make no warranty of any kind, expressed or implied, with regard to this guide's accuracy. The authors and RoadNarrows LLC are not liable nor take any responsibility for any errors found in this guide.

The content of this guide is subject to change without notice.

This guide describes serial commands provided by version 1.50RNf of the HemiOS to access and control Hemisson I2C capable modules connected to the robot base. The HemiOs interface changes of version HemOs-1.50RNf from K-Team's firmware version HemOS V1.50 are documented. Please refer to the Hemisson User Manual for all other commands. The primary drivers that differentiate the two versions are 1) support for a robust Hemisson serial port interface and 2) command line support for Hemisson modules. The Hemisson Base 16F877 processor has very little program space, which highly constrains functionality and feature support. Table Base1 summarizes changes and status between the two versions. Serial commands may be sent by a host computer and responses received either 1) via a cabled connection to the Hemisson RS232 serial port or 2) wirelessly via the Hemisson Bluetooth Serial Module.

The HemOs-1.50RNf sources and downloadable Hex file can be found at:  
[www.roadnarrowsrobotics.com/support/hemisson/supHem-Base.html](http://www.roadnarrowsrobotics.com/support/hemisson/supHem-Base.html).

Symbol	Description
←	ASCII Carriage Return CR (0x0D). All commands sent to the Hemisson end with a CR ←.
↓	ASCII Newline NL (0x0A). All responses from the Hemisson are terminated with CR-NL ←↓ pair.
<i>h</i>	ASCII hexadecimal digit: 'a' – 'f', 'A' – 'F', '0' – '9'
<i>d</i>	ASCII decimal digit: '0' – '9'
<i>b</i>	ASCII binary digit: '0', '1'
<i>c</i>	ASCII character
<i>val</i>	Variable ASCII value.
A, B, ...	ASCII literal character 'A', 'B', ... All commands are upper case, single letter non-whitespace, printable characters.
a, b, ...	ASCII literal character 'a', 'b', ... All responses are lower case equivalents to their respective commands.
,	ASCII Comma (0x2C). Command/response field separator.
[ ]	Optional arguments to command or values in the response.
...	Zero or more arguments/values following.

**Table Intro1: Notation**

Each I2C module supports a set of commands specific to that module (see the following sections). To issue a module command, the following command/response syntax is used.

Command: *A,M,C[,cmdarg...]* ←

Response: *a,M,r[,rspval...]* ← ↓

Arguments/Values:

- M* Unique, one-character module identifier (see Table Intro2)
- C* Specific, single-letter, upper case module command.
- cmdarg* Optional command argument. Specific to each command.
- r* Response identifier. Lower case equivalent to command.
- rspval* Optional response value. Specific to each response.

On error responses, the following error response syntax is used.

Error Rsp: `a, M<Eec: elem>`

Values:

`ec` Error code. See Table Intro3 for list of module error codes.  
Format: hh.

`elem` Relevant command element.

The support of of command-line interfaces to the Hemisson modules are summarized in Table Inro2.

Module	Identifier	Notes
BasicStamp II	2	RN HemiOS support not available yet.
Bluetooth Serial	-	Not I2C capable.
<b>General I/O</b>	<b>G</b>	<b>Available.</b>
LCD	D	RN HemiOS support not available yet.
<b>Linear Camera</b>	<b>L</b>	<b>Available.</b>
<b>Text-To-Speech</b>	<b>T</b>	<b>Available.</b>
<b>UltraSonic Sensor</b>	<b>U</b>	<b>Available.</b>
Wireless Color Camera	-	Not I2C capable.

**Table Intro2: Module Summary**

Module specific error codes are explained in Table Intro3.

Error Code	Description
11	Invalid module one-character identifier.
12	Module is not connected.
13	Module support is not enabled in this RN HemiOS version.
14	Module is busy.
15	Invalid command length.
16	Command element is out-of-range.
17	Unknown module command.

**Table Intro3: Module Error Codes**

This section describes HemiOs interface and behavior deviations of version HemOs-1.50RNf from K-Team's firmware version HemOS V1.50. The primary drivers that differentiate the two versions are 1) support for a robust Hemisson serial port interface and 2) command line support for Hemisson modules. The Hemisson Base 16F877 processor has very little program space, which highly constrains functionality and feature support. Table Base1 summarizes changes and status between the two versions.

Change #	Description
C1	Recovery from UART errors has been added.
C2	Significantly more command syntax checking is done.
C3	Error reporting as been streamlined to save memory and for easier parsing for the Host. See Tables Base2 and Intro3.
C4	Some responses to Base commands have been modified. See the commands below.
C5	The hidden Tristate commands '1' and '2' have been removed.
C6	The new module command 'A' has been added.
C7	Command interfaces to Hemisson modules are supported. See Table Intro2 for summary of supported modules.
C8	The built-in Obstacle Avoidance, Line Following, and Dance behaviors have not been included. It is hopeful that this will be added in a future version.
C9	The TV remote capability has been removed because of lack of ROM when the UltraSonicSensor module commands were added.

**Table Base1: Summary of Changes**

On error responses, the following error response syntax is used.

Error Rsp: <Eec: msg>

Values:

*ec* Error code. See Table Base2 for list of base error codes.  
Format: hh.

*msg* Additional context.

Error Code	Description
01	Unknown command.
02	Invalid syntax.
03	Buffer overflow.

**Table Base2: Base Error Codes**



**E (doc) Get motors speed**

---

Command: E ←

Response: e, left, right ←↓

Arguments/Values:

*left* Left motor speed.

Format: -d or 0d

*right* Right motor speed.

Format: -d or 0d

Example 1: Send a command to get current motor speeds.

Command: E ←

Response: e, -5, 5 ← ↓

This was an undocumented command.

**H (chg) Turn beep on/off**

---

Command: H, state ←

Response: h, state ←↓

Arguments/Values:

*state* Beep on/off state.

Format: b

Enum: 0 (off), 1 (on)

Example 1: Send a command to turn the buzzer on.

Command: H, 1 ←

Response: h, 1 ← ↓

The response has been changed to echo back the on/off state.

**J (chg) Scan all I2C addresses for all connected I2C capable modules.**

---

Command: J ←

Response: j[,M,addr,ver[,...]] ← ↓

## Arguments/Values:

*M* Unique, one-character module identifier.  
Format: see Table 2.

*addr* Hexadecimal base I2C address of the module.  
Format: *hh*.

*ver* Hexadecimal version number of the module.  
Format: *hh*.

Example 1: Scan for connected modules. The Hemisson has the Linear Camera , Text-To-Speech and General I/O modules connected.

Command: J ← Response: j , L , C0 , 05 , T , C4 , 06 , G , D0 , 06 ← ↓

For each discovered module, a triplet of response values are sent. The J command is useful to discover which I2C modules are connected to the Hemisson robot base.

**L (chg) Set the LED states**

---

Command: L,ledpower, ledexepgm, ledleft, ledright ←

Response: l,ledpower, ledexepgm, ledleft, ledright ← ↓

## Arguments/Values:

*ledpower* Power LED on/off state.  
Format: b  
Enum: 0 (off), 1 (on)

*ledexepgm* Exec/Pgm LED on/off state.  
Format: b  
Enum: 0 (off), 1 (on)

*ledleft* Left User LED on/off state.  
Format: b  
Enum: 0 (off), 1 (on)

*ledright* Right User LED on/off state.  
Format: b  
Enum: 0 (off), 1 (on)

Example 1: Send a command to turn on the front two user LEDs.

Command: L,0,0,1,1 ← Response: l,0,0,1,1 ← ↓

The response has been changed to echo back the LED states.

Module Identifier: **G**

### **A**      **Read Analog register**

---

Command:  $A, G, A, reg \leftarrow$

Response:  $a, G, a, reg, val \leftarrow \downarrow$

Arguments/Values:

*reg*      Analog register number.  
             Format: *hh*.  
             Range: 00 – 04 hexadecimal (1 – 4)

*val*      Value of the 8-bit register read.  
             Format: *hh*.

Example 1: Read analog register 3, returning the value of 31.

Command:  $A, G, A, 03 \leftarrow$                       Response:  $a, G, a, 03, 1F \leftarrow \downarrow$

Note 1:      The analog register address is  $0x10 + reg$ .

### **R**      **Read digital register**

---

Command:  $A, G, R, reg \leftarrow$

Response:  $a, G, r, reg, val \leftarrow \downarrow$

Arguments/Values:

*reg*      Digital register number.  
             Format: *hh*.  
             Range: 00 – 0B hexadecimal (1 – 11)

*val*      Value of the 1-bit register read.  
             Format: *b*.

Example 1: Read digital register 10, returning the value of 1.

Command:  $A, G, R, 0A \leftarrow$                       Response:  $a, G, r, 0A, 1 \leftarrow \downarrow$

Note 1:      The digital register address is  $0x20 + reg$ .

**W**      **Write digital register**Command:  $A, G, W, reg, val \leftarrow$ Response:  $a, G, w, reg, val \leftarrow \downarrow$ 

Arguments/Values:

*reg*      Digital register number.  
             Format: *hh*.  
             Range: 00 – 0B hexadecimal (1 – 11)

*val*      1-bit value to write.  
             Format: *b*.

Example 1: Write the value 0 to digital register 7.

Command:  $A, G, W, 07, 0 \leftarrow$ Response:  $a, G, w, 07, 0 \leftarrow \downarrow$ Note 1:      The digital register address is  $0x20 + reg$ .**V**      **Get Version number of this module**Command:  $A, G, V \leftarrow$ Response:  $a, G, v, vernum \leftarrow \downarrow$ 

Arguments/Values:

*vernum*    Module version number..  
             Format: *hh*.

Example 1: Get the module's version number.

Command:  $A, G, V \leftarrow$ Response:  $a, G, v, 06 \leftarrow \downarrow$

Module Identifier: L

### **E**      **Set Exposure time**

---

Command: `A,L,E,ms ←`

Response: `a,L,e,ms ← ↓`

Arguments/Values:

`ms`      Exposure time (milliseconds).  
 Format: *hh*.  
 Range: 00 – FF hexadecimal (0 – 255ms)  
 Default: 01

Example 1: Set the exposure time to 2 milliseconds.

Command: `A,L,E,02 ←`      Response: `a,L,e,02 ← ↓`

Set the exposure time (milliseconds) between each frame grab. The camera can refresh up to 100Hz. Exposure times may effect the refresh rate. An exposure time of 0 is undefined.

### **F**      **Get exposure time**

---

Command: `A,L,F ←`

Response: `a,L,f,ms ← ↓`

Arguments/Values:

`ms`      Exposure time (milliseconds).  
 Format: *hh*.  
 Range: 00 – FF hexadecimal (0 – 255ms)

Example 1: Get the current exposure time.

Command: `A,L,F ←`      Response: `a,L,f,01 ← ↓`

**L Set the LED state**

---

Command: `A,L,L,led ←`Response: `a,L,l,led ← ↓`

## Arguments/Values:

`led` Camera LED on/off state.  
 Format: *b*.  
 Enum: 0 (off), 1 (on)

Example 1: Turn the camera's LED on.

Command: `A,L,L,1 ←`Response: `a,L,l,1 ← ↓`

Note 1: After performing a read of [un]thresholded pixel values, the LED is automatically turn off.

**P Read camera Pixel unthresholded values.**

---

Command: `A,L,P,zone ←`Response: `a,L,p,zone ,pxpxpxpx... ← ↓`

## Arguments/Values:

`zone` Linear camera zone.  
 Format: *d*.  
 Enum: 0 (all zones), 1 (left zone), 2 (middle zone), 3 (right zone)

`px` 256-bit gray-scale pixel.  
 Format: *hh*.  
 Range: 00 (black) – FF (white)

Example 1: Read the pixels from the left zone.

Command: `A,L,P,1 ←`Response: `a,L,p,1,1C1D0908... ← ↓`

The linear camera pixels are read from left to right, zones left to right. There are 102 total pixels in 3 zones of 34 pixels. Each pixel contains a 256-bit gray-scale value. The camera's exposure time effects the sensitivity of every pixel.

**Q**      **Read camera thresholded Q pixel values.**

---

Command: `A,L,Q,zone ←`Response: `a,L,q,zone ,pxpxpxpx... ← ↓`

## Arguments/Values:

`zone`      Linear camera zone.Format: `d`.

Enum: 0 (all zones), 1 (left zone), 2 (middle zone), 3 (right zone)

`px`      256-bit black or white pixel.Format: `hh`.

Enum: 00 (black), FF (white)

Example 1: Read the pixels from all zones.

Command: `A,L,Q,0 ←`Response: `a,L,q,0,00FFFF00... ← ↓`

Any pixels with values equal to or higher (lighter) than the current threshold setting will have the value of FF. Otherwise the pixel's value will be 00. The linear camera pixels are read from left to right, zones left to right. There are 102 total pixels in 3 zones of 34 pixels. Each pixel contains a 256-bit gray-scale value. The camera's exposure time effects the sensitivity of every pixel.

**T**      **Set pixel Threshold value**

---

Command: `A,L,T,th ←`Response: `a,L,t,th ← ↓`

## Arguments/Values:

`th`      Threshold value.Format: `hh`.

Range: 01 – FF hexadecimal (1 – 255)

Default: FF

Example 1: Read the current unthresholded pixels, set the threshold appropriately, and then read the thresholded values.

Command: `A,L,P,0 ←`Response: `a,L,p,0,C1DEDFC0... ← ↓`Command: `A,L,T,C0 ←`Response: `a,L,t,C0 ← ↓`Command: `A,L,Q,0 ←`Response: `a,L,q,0,FF0000FF... ← ↓`

L

## Linear Camera Module

L

### **U** Get pixel threshold value

---

Command: A,L,U ←

Response: a,L,u,th ← ↓

Arguments/Values:

*th* Threshold value.  
Format: *hh*.  
Range: 01 – FF hexadecimal (1 – 255)

Example 1: Get the current threshold value.

Command: A,L,U ←

Response: a,L,u,FF ← ↓

L

## Linear Camera Module

L

### **V** Get Version number of this module

---

Command: A,L,V ←

Response: a,L,v,vernum ← ↓

Arguments/Values:

*vernum* Module version number..  
Format: *hh*.

Example 1: Get the module's version number.

Command: A,L,V ←

Response: a,L,v,05 ← ↓

Module Identifier: T

### **C** Speak a **C**anned message

---

Command: A,T,C,msgnum ←

Response: a,T,c,msgnum ← ↓

Arguments/Values:

*msgnum* The number of the pre-stored message.  
 Format: *hh*.  
 Range: 01 – 1E hexadecimal (1 – 30)

Example 1: Play canned message number 1.

Command: A,T,C,01 ←

Response: a,T,c,01 ← ↓

Pre-stored messages are loaded into into the Text-To-Speech module through the module's serial port using the Window's application SP03.exe. Up to 30 messages may be pre-stored.

Note 1: Text messages are converted and stored using the voice pitch and rate values at the time the text is converted. The pitch and rate module commands have no effect on pre-stored messages.

### **G** Set speaker **G**ain (volume)

---

Command: A,T,G,gain ←

Response: a,T,g,gain ← ↓

Arguments/Values:

*gain* Gain (volume) of the speaker.  
 Format: *d*.  
 Range: 0 (loudest) – 7 (quietest)  
 Default: 0

Example 1: Set the volume to the 'ghetto blaster' level.

Command: A,T,G,0 ←

Response: a,T,g,0 ← ↓

Note 1: The speaker has very small range in power. All but loudest settings are virtually inaudible.

### **P** Set Pitch of voice in text-to-speech conversion

---

Command: `A,T,P,pitch ←`

Response: `a,T,p,pitch ← ↓`

Arguments/Values:

*pitch* Voice pitch of the converted text.  
 Format: *d*.  
 Range: 0 (highest) – 7 (lowest)  
 Default: 5

Example 1: "Mickey Mouse" the voice.

Command: `A,T,P,0 ←`

Response: `a,T,p,0 ← ↓`

Note 1: Setting this value has no effect on pre-stored, canned messages.

Note 2: The pitch value 06 is weird

### **Q** Query if Text-To-Speech module is currently speaking

---

Command: `A,T,Q ←`

Response: `a,T,q,state ← ↓`

Arguments/Values:

*state* TTS module is [not] speaking.  
 Format: *b*.  
 Enum: 0 (not speaking), 1 (speaking)

Example 1: Query TTS module until it shuts up.

Command: `A,T,S,blah blah ←`

Response: `a,T,s,blah blah ← ↓`

Command: `A,Q ←`

Response: `a,q,1 ← ↓`

Command: `A,Q ←`

Response: `a,q,0 ← ↓`

## **R**      **Set Rate (speed) of voice in text-to-speech conversion**

---

Command: `A,T,R,rate ←`

Response: `a,T,r,rate ← ↓`

### Arguments/Values:

*rate*      Voice rate (speed) of the converted text.  
 Format: *d*.  
 Range: 0 (slowest) – 3 (fastest)  
 Default: 3

Example 1: Speak Southern, y'all.

Command: `A,T,R,0 ←`

Response: `a,T,r,0 ← ↓`

Note 1: Setting this value has no effect on pre-stored, canned messages.

Note 2: Voice rate settings are almost imperceptible.

## **S**      **Speak a message**

---

Command: `A,T,S,string ←`

Response: `a,T,s,string ← ↓`

### Arguments/Values:

*string*      Text message to speak.  
 Format: ASCII text. See Table T1.  
 Length: 1 – 72 bytes

Example 1: Speak 'H' 'E' 'L' 'L' 'O'.

Command: `A,T,S,HELLO ←`

Response: `a,T,s,HELLO ← ↓`

Example 2: Speak 'Hello Dolly'.

Command: `A,T,S>Hello Dolly ←`

Response: `a,T,s>Hello Dolly ← ↓`

The text will be spoken at the current speaker gain with current voice pitch and rate settings.

Text	Rule
<i>WORD</i>	An all upper case <i>WORD</i> is spoken as an acronym (i.e. each letter is pronounced).
<i>Word</i>	<i>Word</i> is pronounced.
<i>word</i>	<i>word</i> is pronounced.
"', . ? !	Punctuation marks are ignored.

**Table T1: Text-To-Speech Conversion Rules**

**V**      **Get Version number of this module**

---

Command:    A, T, V ←

Response:   a, T, v, *vernum* ← ↓

Arguments/Values:

*vernum*    Module version number..  
              Format: *hh*.

Example 1:    Get the module's version number.

      Command:    A, T, V ←

      Response:   a, T, v, 06 ← ↓

Module Identifier: **U**

The Hemisson UltraSonic Sensor (USS) uses the SRF08 range finder. The SRF08 has a maximum effective range of 6 meters (although 11 meters may be specified). At 11 meters it takes 65 milliseconds to complete a measurement cycle. The USS emits an approximately 55° conical beam (27.5° to the left and right of center). The distance accuracy is between 2-3 cm.

The measurement cycle begins by issuing a measurement command. The USS will emit an ultrasonic pulse and then wait synchronously for the any echoes to return. Up to 17 echoes may be recorded, in ascending order by distance. A maximum distance may be specified less than the default of 11 meters. The measurement cycle will then be correspondingly shorter. Two commands are supported to retrieve the current echo measurements. The retrieved data are in the current units. Units supported are centimeters, inches, or microseconds. An echo value of zero indicates that no echo was detected.

Boot-up Defaults:

Measurement Units: centimeters

Maximum Range: 11008 millimeters (11.008 meters)

See Also: <http://www.robot-electronics.co.uk/htm/srf08tech.shtml>

## A Get All 17 echo measurements

---

Command: `A,U,A ←`  
 Response: `a,U,a,echo00,echo01, ... ,echo10 ← ↓`

Arguments/Values:

`echoxx` Distance of echo `xx`. Values of distance are in the current units.  
 Format: `hhhh`.  
 Range: 0000 – FFFF hexadecimal (0 – 65535)  
 Boot-up default: undetermined (measurement must be taken first)

Example 1: Get all of the echoes from the last issued distance measurement.  
 Command: `A,U,A ←`      Response: `a,U,a,00A2,01FE,... ← ↓`

In example 1, assuming the current units are centimeters, then `echo00 = 00A2` translates to a distances of 162 cm. Likewise `echo01` is 510 centimeters

Note 1: All measurement values are from the last measurement taken (See. the 'M' command).

Note 2: A distance value of 0000 indicates that no echo was detected.

## E Get an Echo measurement

---

Command: `A,U,E,echonum ←`  
 Response: `a,T,e,echonum,echo ← ↓`

Arguments/Values:

`echonum` Number of the echo.  
 Format: `hh`.  
 Range: 00 – 10 hexadecimal (0 – 16)

`echo` Distance of echo `echonum`. Values of distance are in the current units.  
 Format: `hhhh`.  
 Range: 0000 – FFFF hexadecimal (0 – 65535)  
 Boot-up default: undetermined (measurement must be taken first)

Example 1: Get the 3<sup>rd</sup> echo distance.  
 Command: `A,U,E,02 ←`      Response: `a,U,e,02,020B ← ↓`

Note 1: The measurement value is from the last measurement taken (See. the 'M' command).

Note 2: A distance value of 0000 indicates that no echo was detected.

### **L** Get the ambient Light intensity

---

Command: `A,U,L←`

Response: `a,U,l,light ← ↓`

Arguments/Values:

*light* The intensity of the ambient light.  
 Format: *hh*.  
 Range: 00 (darkest) – FF (brightest) hexadecimal (0 - 255)  
 Boot-up default: undetermined (measurement must be taken first)

Example 1: We live in a yellow banana.

Command: `A,U,L ←`

Response: `a,U,l,C9 ← ↓`

Note 1: A value between 0-3 is complete darkness, while values  $\geq 248$  (0xF8) are under bright light.

Note 2: The measurement is made during each ultrasonic measurement (See. the 'M' command).

Note 3: Why this sensor is included with the SRF08 is a mystery.

### **M** Take an ultrasonic Measurement

---

Command: `A,U,M ←`

Response: `a,U,m ← ↓`

By issuing this command, the following operations are performed:

1. *SRF08 power on*
2. *ambient light measurement*
3. *ultrasonic pings*
4. *echo distance measurements*
5. *SRF08 standby*

Note 1: The 'M' command is synchronous. No response is given until the operation sequence above is completed. At most 65 milliseconds will elapse between the command and response at the maximum ranging distance.

**R**      **Set the maximum Range to sound**

---

Command: `A,U,R,range ←`Response: `a,U,r,range← ↓`

## Arguments/Values:

*range*      Maximum range in the current units.  
                  Format: *hhhh*.  
                  Range: 0000 units to FFFF units  
                  Boot-up default: maximum range (11.008 meters)

Example 1: Assuming the current units are inches, then the following example sets the maximum range to 12 feet (144 inches).

Command: `A,U,R,0090 ←`                      Response: `a,U,r,0090 ← ↓`

Note 1: The step size of the SRF08 maximum range setting is 43 millimeters. The minimum distance is 43mm and the maximum is  $43 * 255 + 43 = 11008\text{mm}$ .

Note 2: This command will round up to the nearest 43mm up to the maximum.

**U**      **Set the current Units**

---

Command: `A,U,U,units ←`Response: `a,U,u,units ← ↓`

## Arguments/Values:

*units*      Units of measure and range.  
                  Format: *c*  
                  Enum: C (centimeters), I (inches), U (microseconds)  
                  Boot-up default: centimeters

Example 1: Set units to inches

Command: `A,U,U,I ←`                      Response: `a,U,u,i ← ↓`

Note 1: The new units do not take effect until the next measurement cycle. Any old measurements are still in the old units.

Note 2: The microsecond units give flexibility in distance calculations. The SRF08 assumes the speed of sound is 343 m/s (0.343mm/μsec).

**V**      **Get Version number of this module**

---

Command:    A,U,V ←

Response:    a,U,v,vernum ← ↓

Arguments/Values:

*vernum*    Module version number..  
                Format: *hh*.

Example 1:    Get the module's version number.

    Command:    A,U,V ←

    Response:    a,U,v,05 ← ↓