# Hemisson Modules

## Serial Command Guide

**September 15th 2004**


**RoadNarrows LLC**

**RoadNarrows LLC**
http://www.roadnarrowsrobotics.com/
mailto:oneway@roadnarrowsrobotics.com

# Hemisson Modules Introduction



This guide describes serial commands provided by version 1.50RNc of the HemiOS to access and control Hemisson I2C capable modules connected to the robot base. 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.

| 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. |
| *h* | ASCII hexidecimal digit: 'a' – 'f', 'A' – 'F', '0' – '9' |
| *d* | ASCII decimal digit: '0' – '9' |
| *b* | ASCII binary digit: '0', '1' |
| *val* | 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 following arguments. |

***Table 1: 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 2) |
| *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. |

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

***Table 2: Module Summary***

The J command is useful to discover which I2C modules are connected to the Hemisson robot base.

**J** **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.<br>Format: see Table 2. |
| *addr* | Hexidecimal base I2C address of the module.<br>Format: *hh*. |
| *ver* | Hexidecimal version number of the module.<br>Format: *hh*. |

For each discovered module, a triplet of response values are sent.

# G General I/O Module G

Module Identifier: **G**

## A Read Analog register

Command: `A,G,A,`*`reg`* ←
Response: `a,G,a,`*reg, val* ← ↓

Arguments/Values:

| | |
|---|---|
| *reg* | Analog register number.<br>Format: *hh*.<br>Range: 00 – 04 hexidecimal (1 – 4) |
| *val* | Value of the 8-bit register read.<br>Format: *hh*. |

Example 1: Read analog register 3, returning the value of 31.
Command: `A,G,A,03` ← Response: `a,G,a,03,1F` ← ↓

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

## R Read digital register

Command: `A,G,R,`*`reg`* ←
Response: `a,G,r,`*reg, val* ← ↓

Arguments/Values:

| | |
|---|---|
| *reg* | Digital register number.<br>Format: *hh*.<br>Range: 00 – 0B hexidecimal (1 – 11) |
| *val* | Value of the 1-bit register read.<br>Format: *b*. |

Example 1: Read digital register 10, returning the value of 1.
Command: `A,G,R,0A` ← Response: `a,G,r,0A,1` ← ↓

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

## W Write digital register

Command: A,G,W,*reg*,*val* ←
Response: a,G,w,*reg*, *val* ← ↓

Arguments/Values:
*reg* Digital register number.
Format: *hh*.
Range: 00 – 0B hexidecimal (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 ← Response: a,G,w,07,0 ← ↓

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

## V Get Version number of this module

Command: A,G,V ←
Response: a,G,v,*vernum* ← ↓

Arguments/Values:
*vernum* Module version number..
Format: *hh*.

Example 1: Get the module's version number.
Command: A,G,V ← Response: a,G,v,06 ← ↓

# T Text-To-Speech Module T

Module Identifier: **T**

## C Speak a Canned 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 hexidecimal (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 Gain (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 vitually 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 wierd.

## Q Query if Text-To-Speach 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' '0'.
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 |
|---|---|
| *WORD* | An all upper case *WORD* is spoken as an acronym (i.e. each letter is pronounced). |
| *Word* | *Word* is pronounced. |
| *word* | *word* 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 ← ↓`