



# User Manual for Remote Receiver Board

Document Revision: 1, 5/28/2009



## Contents

Introduction.....	3
What you will need.....	3
Features.....	3
Typical Applications.....	3
1 Quick Start, Use with AI-01 Remote Control.....	7
2 Quick Start, RRB to RRB mode.....	8
3 Serial Interface With A Computer.....	9
3.1 Command 1: Servo motor control.....	9
3.2 Command 2: Digital I/O set/get.....	10
3.3 Command 3: Get A/D values.....	11
3.4 Command 4: Set RF configuration.....	11
3.5 Command 5: Send command to another RRB via RF.....	12
3.6 Error codes for serial interface.....	13
4 Emulating an AI-01 remote, 10 byte packet format definition.....	13
5 ASCII character usage in serial commands.....	14
6 Advanced Users, Reprogramming the RRB.....	15

## Introduction

The Remote Receiver Board (RRB) is a simple solution for the user that wants to quickly implement a solution into their robot or similar project using the Active Innovations AI-01 remote control.

## What you will need

You must connect an RF module available from Sparkfun ([www.sparkfun.com](http://www.sparkfun.com)) to complete the RF connection. Modules available from Sparkfun that will work:

- Olimex nRF24L01 Transceiver Straight (catalog number WRL-08283)
- Olimex nRF24L01 Transceiver Curved (WRL-08250)
- Sparkfun nRF24L01 Module with antenna (WRL-00691)

The Sparkfun module with antenna is least expensive but requires you to solder onto the board. The Sparkfun module also has slightly lower range indoors with the RRB. The Olimex modules plug into a header with ribbon cables (no soldering required).

*If you want to use the serial feature* to communicate via a computer, you will need a Sparkfun USB to AI adapter (part number DEV-08473) since the serial interface on the RRB is TTL level. The adapter gives RS-232 voltage level capability.

## Features

A brief overview of features available on the RRB

- Easy RF link using Nordic nRF24L01 (requires module available from Sparkfun)
- 8 digital I/O lines (0-3.3V) accessible via 0.1" spacing pins
- 2 A/D lines (10 bit resolution) accessible via 0.1" spacing pins
- Easy to configure new address and frequency of RF link via serial
- Serial interface to drive on-board digital I/Os, A/D and servo control (requires USB/Serial Adapter available from Sparkfun)
- On-board supplies of 5V and 3.3V, Input 6-9V DC
- 2 Ready to plug in 3 pin servo motor headers (power is direct DC input power to servos)

## Typical Applications

Some ways the RRB was designed to be used are listed below.

- 1) Use with an AI-01 remote to build a simple robot
  - Drive 2 servos as diff drive from joystick commands
  - 7 I/O line control from button press on remote
  - Receive and serially output data from remote
  - 100' range
- 2) Use with another RRB for bi-directional link robotics projects
  - RRB1 is a "command center" board, driven by serial link with PC
  - RRB2 is slave bot board
  - RRB1 commands 2. 2 responds and can send back data via RF
  - Digital I/O: both have 8 I/O lines that be set as input or output
  - A/D capable: collect A/D from 2 lines and send data via link

Servo motor control: easy to use control of servo on an RRB via RF link

Digital inputs and 2 A/D line data are sent back automatically

Both RRBs output serially (9600 baud) data received via RF

Both RRBs can be commanded serially to set digital I/Os, get A/D, and drive servos

- 3) Advanced users can reprogram the RRB (open source code available) to operate as desired. The RF link functions out of the box.

**Electrical and functional specifications:**

<b>Parameter</b>	<b>Detail</b>	<b>Notes</b>
Input power	6-9 (absolute max 12) VDC	
Output power	3.3 and 5V, max 1A out	
RF link range	100' open air/50' indoors	
Servos	40 possible position commands. If standard servo rotates from 0-180 degrees in 4.5 degree increments.	Note that servos have 3 pin interface of ground, power and signal provided on board. The power pin is received directly from the PWR connector. Driving servos with more than 9V is not recommended.
Digital I/O	8 lines, 0-3.3 Volts	Can be configured for in or out via serial or RF commands
A/D	2 10 bit A/D lines are provided by default via a 3 pin interface that provides power, gnd, and input pin. Range is 0-3.3V.	18F2321 has available other lines for A/D (requires reprogramming of board)
Serial interface	9600 baud, 8 data bits, 1 start bit, 1 stop bit, no flow control. TTL level only, requires Sparkfun adapter board for RS-232 level.	NOTE: Be sure to disable any feature to send line ends with line feeds on serial program such as Hyperterminal as it make it difficult to switch between RF link and serial command.
Data timeout failsafe, motors off	3 seconds	If no command either serial or RF is received, motors are shut off. Digital I/O lines stay as they were last set.

**Table 1: Electrical and functional specifications for the RRB**

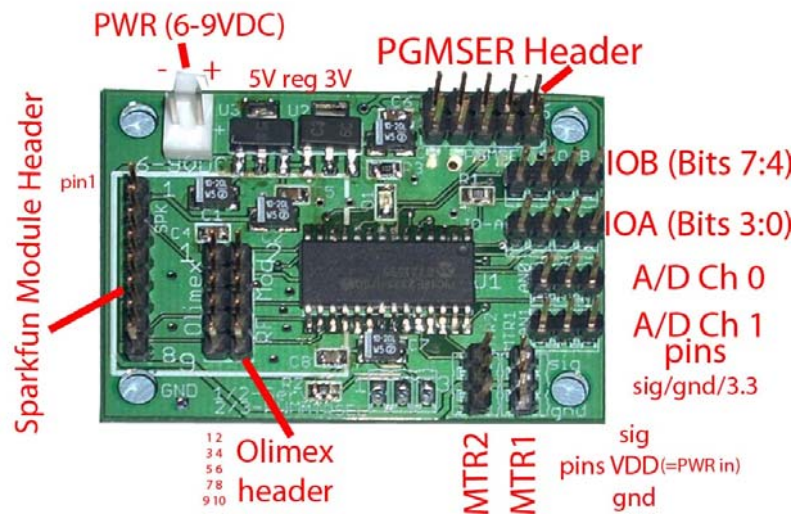


Figure 1: RRB with pinouts



Figure 2: Connection with Olimex Modules



Figure 3: RRB with Sparkfun nRF module soldered

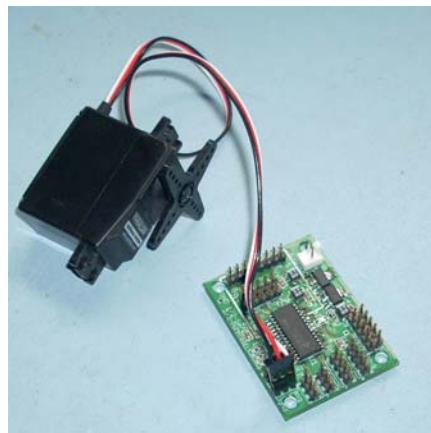


Figure 4: Connection with servo

## 1 Quick Start, Use with AI-01 Remote Control

- 1) Connect a nRF24L01 module to the RRB.
  - a) Either Olimex modules can be plugged into the header marked Olimex RF Mod using the supplied ribbon cable. Cable should extend across the board as shown in figure 2.
  - b) The Sparkfun module must be soldered onto the 8 pin header marked SPK as shown in figure 3.
- 2) Connect a source of DC power to the RRB PWR connector. Source is to be from 6-9V DC. A 9V battery will work, but bear in mind that if you are going to run the servo motors that it will drain fairly quickly (20 minutes or less).
- 3) You should see yellow LED come on. If not, check connections from steps 1 and 2.
- 4) Turn on remote control. Make sure it is set to channel A1 (red LED, and yellow LED 1). This is the default setting out of the box. See AI-01 robot manual for direction on how to change remote channels. Alternatively, you can use serial interface on the RRB to change the RRB channel to match your remote's channel setting (see section 3.4).
- 5) Joystick motion is mixed on the RRB and turned into a differential control signal between MTR1 (left) and MTR2 (right). If you connect continuous rotation servo motors to these two headers as shown in figure 4 (white signal line to inside of board), you will have electronics for a basic driving robot.
- 6) There are 8 digital I/O lines on the RRB, 7 of which will respond to the remote button presses (press gives 3.3V out, released gives 0V out) as shown in table 1 below:

Remote Button	Affects I/O line
B	0
T	1
S	2
H	3
Down	4
Up	5
Joystick button	6

**Table 2: Digital I/O lines and response to AI-01 remote button presses**

## 2 Quick Start, RRB to RRB mode

- 1) Connect a nRF24L01 module to the RRB.
  - c) Either Olimex modules can be plugged into the header marked Olimex RF Mod using the supplied ribbon cable. Cable should extend across the board as shown in figure 2.
  - d) The Sparkfun module must be soldered onto the 8 pin header marked SPK as shown in figure 3.
- 2) Connect the USB to AI serial adapter to the header marked PGMSER as shown in figure 5. Note that adapter's cable goes AWAY from board, not across.

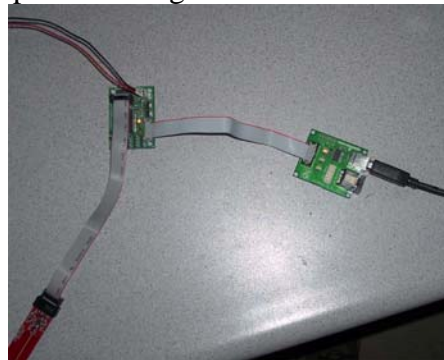


Figure 5: USB to AI adapter connection

- 3) Install drivers for the USB to AI adapter before connecting the adapter to your computer. (<http://www.ftdichip.com/FTDrivers.htm>).
- 4) Connect USB to AI adapter to your computer via USB cable.
- 5) Start a serial terminal program with settings 9600, 8 data bits, 1 start bit, 1 stop bit, no flow control. COM port will depend on how the USB to AI adapter board is installed. Typically, it will be port 4 or 5. Be sure that any option to “append line ends to line feeds” is disabled as it will make it difficult to return to receiving RF signal after transmitting by serial command.
- 6) Connect a source of DC power to the RRB PWR connector. Source is to be from 6-9V DC. A 9V battery will work, but bear in mind that if you are going to run the servo motors that it will drain fairly quickly (20 minutes or less).
- 7) You should see yellow LED come on and the message in your terminal program

```
<Open>Active Innovations RRB, Code Rev. 1.0</Open>
<RFpu addr="2" freq="10">
```

This shows the revision of code operating in the RRB and the channel setting for the RRB. By default out of the box, it is set to address 2, frequency 2.010 GHz.

This is the default setting for an AI-01 remote control.

- 8) You can now use the RRB to send RF signals, commanding another RRB to drive its servo motors, or get A/D or digital input information and send back to you, or to drive outputs. The other RRB will also output serially what it receives if you

wanted to use the 2<sup>nd</sup> RRB as a component in a robot. See next section, Serial Interface With A Computer for more information on the serial interface.

### 3 Serial Interface With A Computer

The RRB comes with software that has a serial interface built in, allowing a user to ‘see’ incoming RF data, to send customized RF signals out to command another RRB, receive data back from the 2<sup>nd</sup> RRB, and finally to directly command an RRB by serial interface to operate the RRB’s inputs and outputs.

Serial Commands: Every command has 3 parts, start sequence bytes (the characters &@), the serial payload, and the end sequence bytes (%\*). There are 5 commands available, listed in Table 2 below.

Command	Purpose	Payload Format	Example
1	Servo motor speed control (including off)	7 bytes, [cmd]:[2 byte mtr1 spd]:[2 byte mtr2 spd]	&@1:30:50%*
2	Digital I/O, set outputs, get inputs	5 bytes, [cmd]d[1 byte dir mask]v[1 byte output values]	&@2d7v)%*
3	Get A/D values	1 byte, [cmd]	&@3%*
4	Configure RF, sets address and frequency	8 bytes, [cmd]a[2 byte addr]f[3 byte freq]	&@4a02f010%*
5	Send custom 10 byte packet via RF	11 bytes, [cmd] [x] (don’t care) [x] > [user ID, arbitrary] [2 byte mtr1 speed] [2 byte mtr2 speed] [1 byte DI/O direction bit mask] [1 byte dig output bit values] [x] [x]	&@5xx>1(F27xx%*

**Table 3: Serial commands**

#### 3.1 Command 1: Servo motor control

Used to drive the servos plugged into MTR1 and MTR2 headers. If you are using as a differential drive system, MTR1=left, MTR2=right. See figure 1 for detail on pinout of the RRB. Note: RRB software has a failsafe mode where if no commands are received by serial or RF for 3 seconds, the motors are shut down.

The two speed commands are in 2 character decimal format, specifying a number between 30 and 70 where each increment is equal to about a 5 us pulse. Servo motors take an input signal of 50 Hz (period=20 ms), and positive pulse between 0.5 and 2.5 ms. The width of the pulse determines the position of rotation of the motor (for unmodified servo motors, for modified continuous rotation the pulse determines direction and to some extent speed). The pulse width is defined by

$$PW = [\text{speed value} - 20] * 50 \text{ us}$$

So a value of 30 (decimal) would be  $(30-20) * 50 \text{ us} = 500 \text{ us} = 0.5 \text{ ms}$ . A value of 50 decimal gives about 1.5 ms which is normally a center position on most servos. You will have to experiment with input values to get exact positioning. Example: `&@1:50:30%*` should rotate mtr1 clockwise at max speed and mtr2 counter clockwise at about medium speed.

**TO STOP THE MOTORS** send a value of 20 ( $[20-20]*50\text{us} = 0 \text{ us pulse.}$ ) or send no commands for 3 seconds.

*Note: The subtract 20 decimal portion of the equation is to allow for sending decimal values 0-20 over serial. Decimal 10 and 13 particularly are problematic since in ASCII these signify end of a line of text in serial. In the serial command mode this does not matter as you send 2 ASCII bytes that are concatenated into a value, but when sending a serial command to transmit by RF (command 5), this value is a single byte. Both serial and send by RF commands are kept the same for ease of use.*

Return string from command 1: When you send a command 1 (motor speed set), the RRB sends back via serial the string

`<Mtr>:[mtr1 speed sent]:[mtr2 speed sent]</Mtr>`

where the decimal 2 char values that you sent are simply given back in the spaces indicated as 2 character decimal values. These values are just read back as acknowledgement. This is NOT a feedback mechanism where the actual speeds are returned to you.

### **3.2 Command 2: Digital I/O set/get.**

Used to set what I/O lines are outputs/inputs and to set bit values that will be driven out on lines that are designated as outputs. The values read on inputs are returned immediately after. See figure 1 for detail on pinout.

The direction mask byte defines the bit data direction. Bits that are 1 are input, 0 are outputs. The value byte determines bit values to be driven out onto output bits. Input bit values are ignored. Example: `&@2d7v)%*` would set data direction to the bit value of char '7' which is hex value 0x37, so I/O lines 7, 6 and 3 would be outputs, while I/O lines 5, 4, 2, 1, 0 would be inputs. The value byte is char ')' which is hex 0x41, so I/O line 7 would have a logic 0, 6 would get a logic 1, and 3 would get a logic 1. The input lines I/O 5,4,2,1,0 would be read and

returned in the return string. For instance if I/O lines 5=Hi, 4=Hi, 2=Hi, 1=Lo, and 0=Hi the value byte would be 0x00110101. Bits set to output state are always shown as 0 in the return byte.

Return string of command 2. Format of return string is

```
<Dio>d="[dir mask byte]" v="[value]"</Dio>
```

Where the direction mask byte is the first value where – occurs, and the [value] found on input lines is returned. Output bits are always 0, but input lines' states are reflected in the returned value byte.

### **3.3 Command 3: Get A/D values.**

Used to get the A/D channel values found on the headers marked AD0 and AD1. See figure 1 for detail on pinout of the RRB.

There is no additional information in the serial payload for this command. Just send the char 3 with the start and end sequence bytes and read the return string. RRB reads in 10 bit resolution on both channels by default.

Return string of command 3. Format is:

```
<An>[4 char hex for value of AD1] : [4 char hex for value of AD0]</An>
```

So, if the voltage on AD0 was 0 volts and AD1 was 3.3V the output would be

```
<An>03FF:0000</An>
```

### **3.4 Command 4: Set RF configuration.**

Used to set the address and frequency of the RRB. Address values are valid from 1 to 99 inclusive (0 and 100+ are invalid). Freq valid values are 1 to 125 (0 and 126+ are invalid). Both are sent as decimal values. Address must be 2 decimal characters, and freq must be 3 decimal characters. So to set address = 3 and freq to 12 you would send '03' and '012'.

Address value sets the address of the RRB which is an arbitrary number to help designate unique channel referenced RRBs. However, it is worth noting that if you wish to receive signals from an AI-01 remote control that only the values '02', '04', '06', '08', '10' and '12' are possible. These correspond to robot number 1, 2, 3, 4, 5, and 6.

The frequency value is

$$F = 2400 + [\text{decimal freq value}] \text{ MHz}$$

So sending a value of '10' sets freq to 2410 MHz. The AI-01 remote control only operates at settings of '010' and '120' (Team A and B respectively).

Return string for command 4 format is

<RfCfg>Addr="[2 byte address]" Freq="[3 byte freq]"</RfCfg>

### **3.5 Command 5: Send command to another RRB via RF.**

This command is used to command another RRB to set its servo motor speeds, set outputs, get inputs and get A/D values. It gives you the ability to send a single RF packet that does what all of the above serial commands can do singly. The AI-01 remote sends a 10 byte data packet about every 60 ms. The RRB uses 10 bytes also in order to be compatible with the AI-01 remote but the format is different for RRB allowing bi-directional communication and allowing you to set servo motor speeds directly rather than mixing the joystick's values.

All bytes sent and received are in hex format.

Format of the RRB transmit packet is

Byte number	Meaning
0	Is always address of the RRB regardless of what is specified in the command string
1	Is always the the freq of the RRB regardless of what is specified in the command string
2	Data direction byte. You must send the character '>' here. One note, if you send 0xC0 the receiving RRB will interpret the incoming 10 byte packet as being sent by a remote resulting in the receiving RRB performing a differential drive mix operation on the motor bytes and output of buttons to the I/O lines as shown in the quick start section for remote use. No digital or A/D input will happen if you send 0xC0 in this byte. This is also how you could control an AI-01 robot using an RRB.
3	Not used. This can be any byte value.
4	Mtr1 speed. Same as the serial command 1(motor control). Valid values are 30 to 70 decimal for speed, 20 to stop.
5	Mtr2 speed. Same as Mtr1 speed above.
6	I/O line data direction mask. Same as that used in serial command 2
7	Don't care for transmit. See receive packet table 4 below.
8	Don't care for transmit. See receive packet table 4 below.
9	Don't care for transmit. See receive packet table 4 below.

**Table 4: RF 10 byte packet format, Transmit**

Format of the RRB receive packet is

Byte number	Meaning
0	Is always address of the RRB
1	Is always the the freq of the RRB.
2	Data direction byte. This will be the ASCII character '<'
3	ASCII character 'x'
4	Mtr1 speed specified in the transmitted command. No feedback is implied. The responding RRB just sends what it received for this value.
5	Mtr2 speed sent, as above for Mtr1.
6	I/O line data direction mask. Again, what was received is just sent back as

	acknowledgement.
7	I/O line input bit values when receiving
8	AD1 value read on the receiving RRB in 8 bits
9	AD0 value read on the receiving RRB in 8 bits

**Table 5: RF 10 byte packet format, Receive**

### 3.6 Error codes for serial interface

The RRB will acknowledge any incoming data on serial interface. The return string for each command shows the acknowledgement message if the command is formatted properly, but if not messages will be returned as shown in Table 6. These status messages will be sent for every serial transaction.

Message	Meaning
<RxSt>[value]<RxSt>	Receive status, negative numbers are ERRORS: 0 input received OK, no errors -1 start sequence (&@) not received -2 start found but no end sequence (%*) -3 no payload found or other serial length problem
<CmdSt>[value]<CmdSt>	Serial command status, negative numbers are ERRORS: 0 serial command received OK, no errors -20 serial command unknown (was not 1-5) -21 motor cmd received, but format wrong -22 digital I/O cmd received, but format wrong -23 get A/D cmd received, but format wrong -24 RF config received, but format wrong -25 send RF cmd received, format wrong (not 10 bytes) -26 RF configuration values invalid

## 4 Emulating an AI-01 remote, 10 byte packet format definition

If you wish to send a 10 byte command packet where an RRB is emulating a remote control, use the format as defined in table 7 below. The only real use of doing so would be to make use of the 'mixing' algorithm for motor speed control, where the joystick x and y values are mixed to create a differential drive signal for motors. Since you can manually send signals for each motor, this is not likely a mode that will see much use for RRB to RRB communication, but it is included if desired. However **if you want to use an RRB to control an AI-01 robot this is the format that you must use.**

Byte number	Meaning
0	Address. Valid values are 0 to 127. Remote will only send 2,4,6,8,10,12
1	Frequency. Values are decimal 10 or 125. Remote will only send 10 or 120.
2	Data direction byte. Value must be 0xC0.
3	Mode and Bot is - Mode is upper nybble, Bot is lower nybble. Shows state of remote with use of the Switch and Hike buttons which are displayed on the red and green LEDs. By default, remote is in Huddle

	mode and Bot A (red LED is on an steady). If Switch button is pressed Bot = B. Pressing Switch sets Bot back to A. Play/Huddle mode is toggled by pressing Hike button. If in huddle LED is steady, play it flashes. If Mode = Play, nybble is 0xA If Mode = Huddle, nybble is 0x5 If Bot = A (red LED on), nybble is 0x1 Bot = B (Green LED) nybble is 0xE Example: Remote is set for RobotA, Play mode then this byte = 0xA1
4	Joystick X axis value (decimal 0-255), center nominal is 127
5	Joystick Y axis value (decimal 0-255), center nominal is 127
6	Button states – this byte shows the state of all the buttons. Each button is represented by a bit where 0 is button released and 1 is pressed Bit Button 7 Reserved 6 Joystick button 5 Up 4 Down 3 Hike 2 Switch 1 Throw 0 Burst
7	Throw state and speed byte. Bit 4 = Throw/notThrow, Bit 3:0 – speed/power (values ranging over decimal 8,9,10,11,12 or 13 depending on how long the button is held down). Note: Robot translates these into numbers into PWM settings for the throw motor where 8 is lowest setting at about 60% duty cycle. 13 is full duty cycle.
8	Not used, you can send anything here
9	Not used, you can send anything here

**Table 7: Format for remote send 10 byte packet**

## 5 ASCII character usage in serial commands

Table 8 is a quick reference for ASCII characters. A handy feature that can be used in Windows Hyperterminal is the ability to send out any of the 256 possible RS-232 (ASCII) characters by using the numeric keypad. Make sure the "Num Lock" key is pressed. To do this, hold down the ALT key and enter the desired numbers. For instance, hold down the ALT key and enter the 1, 2, or 3 numbers needed (i.e.: 0 to 255), and then release the ALT key to send the single ASCII character.

Note: When dealing with Hex code the procedure varies. For example Hex 80 can't be sent by simply entering 80 on the numeric keyboard. If you enter 80 (with ALT) on the keypad, you'll send a "P" (which is Hex 50). Hex 80 is actually Decimal 128, and 128 should have been entered.

Numerical characters generated with the ALT key need to be specified in decimal values 0 through 255. For these complex strings, it would be a good idea to generate your keyboard actions on paper before typing them. Also keep in mind many devices will timeout and throw away part of your command string if you take too long to enter a command.

## ASCII Table and Description

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'

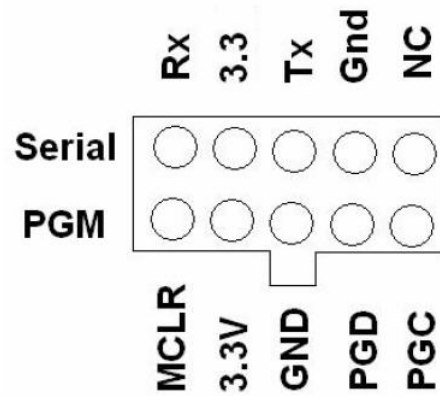
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040		Space	64	40	100		@	96	60	140		
1	1	001	<b>SOH</b> (start of heading)	33	21	041	!	!	65	41	101		A	97	61	141		a
2	2	002	<b>STX</b> (start of text)	34	22	042	"	"	66	42	102		B	98	62	142		b
3	3	003	<b>ETX</b> (end of text)	35	23	043	#	#	67	43	103		C	99	63	143		c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	\$	\$	68	44	104		D	100	64	144		d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	%	%	69	45	105		E	101	65	145		e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&	&	70	46	106		F	102	66	146		f
7	7	007	<b>BEL</b> (bell)	39	27	047	'	'	71	47	107		G	103	67	147		g
8	8	010	<b>BS</b> (backspace)	40	28	050	(	(	72	48	110		H	104	68	150		h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	)	)	73	49	111		I	105	69	151		i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	*	*	74	4A	112		J	106	6A	152		j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	+	+	75	4B	113		K	107	6B	153		k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	,	,	76	4C	114		L	108	6C	154		l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	-	-	77	4D	115		M	109	6D	155		m
14	E	016	<b>SO</b> (shift out)	46	2E	056	.	.	78	4E	116		N	110	6E	156		n
15	F	017	<b>SI</b> (shift in)	47	2F	057	/	/	79	4F	117		O	111	6F	157		o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	0	0	80	50	120		P	112	70	160		p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	1	1	81	51	121		Q	113	71	161		q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	2	2	82	52	122		R	114	72	162		r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	3	3	83	53	123		S	115	73	163		s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	4	4	84	54	124		T	116	74	164		t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	5	5	85	55	125		U	117	75	165		u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	6	6	86	56	126		V	118	76	166		v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	7	7	87	57	127		W	119	77	167		w
24	18	030	<b>CAN</b> (cancel)	56	38	070	8	8	88	58	130		X	120	78	170		x
25	19	031	<b>EM</b> (end of medium)	57	39	071	9	9	89	59	131		Y	121	79	171		y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	:	:	90	5A	132		Z	122	7A	172		z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	;	;	91	5B	133		[	123	7B	173		{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	<	<	92	5C	134		\	124	7C	174		
29	1D	035	<b>GS</b> (group separator)	61	3D	075	=	=	93	5D	135		]	125	7D	175		}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	>	>	94	5E	136		^	126	7E	176		~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	?	?	95	5F	137		_	127	7F	177		DEL

Table 8: ASCII Character Table from [www.asciitable.com](http://www.asciitable.com)

## 6 Advanced Users, Reprogramming the RRB

For those that wish to reprogram the RRB to suit their own purposes, the source code for the RRB is available for download on the Active Innovations site. The source code was written using the CCS compiler for PIC18 series parts (PCH). The code is thoroughly commented. To reprogram the RRB you will need a Microchip ICD (2 or 3). The PICKit3 will also work but will require you to fashion an adapter cable.

The programming interface to the RRB is the same used for serial, the PGM SER port. Pin out for this port is shown below in figure 6. Note that you must turn the board so that the PWR port is at lower right corner for this diagram to match exactly as shown.



**Figure 6: Pinout of PGM SER (inverted orientation)**

Use of the CCS compiler, MPLAB IDE and programming concepts is beyond the scope of this manual, and no support is provided for these topics. Reference to the AI-01 Hackers Manual can be useful in getting introductions to these topics.