VRbot Communication Protocol

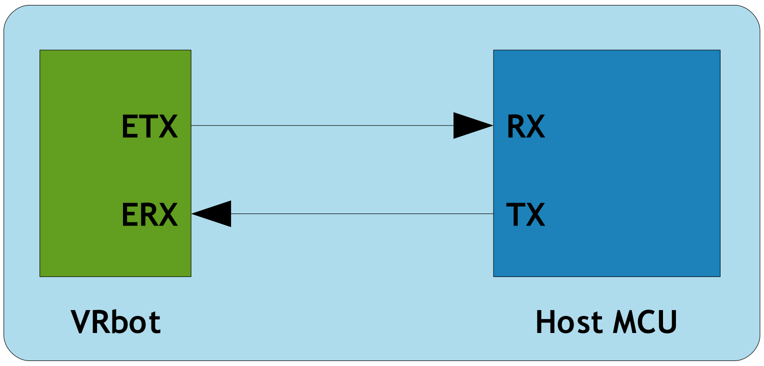
Revision 1.6

Table of contents

Protocol and Interface Basics

Communication with the VRbot module uses a standard UART interface compatible with 3.3-5V TTL logical levels, according to the powering voltage VCC.

A typical connection to an MCU-based host:



The initial configuration at power on is 9600 baud, 8 bit data, No parity, 1 bit stop. The baud rate can be changed later to operate in the range 9600 - 115200 baud.

The communication protocol only uses printable ASCII characters, which can be divided in two main groups:

- Command and status characters, respectively on the TX and RX lines, chosen among lower-case letters
- Command arguments or status details, again on the TX and RX lines, spanning the range of capital letters

Each command sent on the TX line, with zero or more additional argument bytes, receives an answer on the RX line in the form of a status byte followed by zero or more arguments.

There is a minimum delay before each byte sent out from the VRbot module to the RX line, that is initially set to 20 ms and can be selected later in the ranges 0 - 9 ms, 10 - 90 ms, 100 ms - 1 s. That accounts for slower or faster host systems and therefore suitable also for software-based serial communication (bit-banging).

The communication is host-driven and each byte of the reply to a command has to be acknowledged by the host to receive additional status data, using the *space* character. The reply is aborted if any other character is received and so there is no need to read all the bytes of a reply if not required.

Invalid combinations of commands or arguments are signaled by a specific status byte, that the host should be prepared to receive if the communication fails. Also a reasonable timeout should be used to recover from unexpected failures.

If the host does not send all the required arguments of a command, the command is ignored by the module, without further notification, and the host can start sending another command.

The module automatically goes to lowest power sleep mode after power on. To initiate communication, send any character to wake-up the module.

Arguments Mapping

Command or status messages sent over the serial link may have one or more numerical arguments in the range -1 to 31, which are encoded using mostly characters in the range of uppercase letters. These are some useful constants to handle arguments easily:

ARG_MIN	
`'@'` (40h)	Minimum argument value (-1)

ARG_MAX	
``'`'`` (60h)	Maximum argument value (+31)

ARG_ZERO	
`'A'` (41h)	Zero argument value (0)

ARG_ACK	
`' '` (20h)	Read more status arguments

Having those constants defined in your code, can simplify validity checks and the encoding/decoding process. For example (in pseudo-code):

```
# encode value 5
FIVE = 5 + ARG_ZERO

# decode value 5
FIVE - ARG_ZERO = 5

# validity check
IF ARG < ARG_MIN OR ARG > ARG_MAX THEN ERROR
```

Just to make things clearer, here is a table showing how the argument mapping works:

ASCII	'@'	'A'	'B'	'C'	...	'Y'	'Z'	'^'	'['	'\'	']'	'_'	'`'
HEX	(40h)	(41h)	(42h)	(43h)	...	(59h)	(5Ah)	(5Bh)	(5Ch)	(5Dh)	(5Eh)	(5Fh)	(60h)
Value	-1	0	1	2	...	24	25	26	27	28	29	30	31

Command Details

Format of command strings accepted by the module. Please note that numeric arguments of command requests are mapped to upper-case letters (see above section).

CMD_BREAK	
`'b'` (62h)	Abort recognition in progress if any or do nothing **Known issues:** In firmware ID 0, any other character received during recognition will prevent this command from stopping recognition, that will continue until timeout or other recognition results.
Expected replies: STS_SUCCESS, STS_INTERR	

CMD_SLEEP	
`'s'` (73h)	Go to the specified power-down mode
`[1]`	Sleep mode (0-8): 0 = wake on received character only 1 = wake on whistle or received character 2 = wake on loud sound or received character 3-5 = wake on double clap (with varying sensitivity) or received character 6-8 = wake on triple clap (with varying sensitivity) or received character
Expected replies: STS_SUCCESS	

CMD_KNOB	
`'k'` (6Bh)	Set SI knob to specified level
`[1]`	Confidence threshold level (0-4): 0= loosest:more valid results 2= typical value (default) 4= tightest:fewer valid results NOTE: knob is ignored for trigger words
Expected replies: STS_SUCCESS	

CMD_LEVEL	
`'v'` (76h)	Set SD level
`[1]`	Strictness control setting (1-5): 1 = easy, 2 = default, 5 = hard A higher setting will result in more recognition errors.
Expected replies: STS_SUCCESS	

CMD_LANGUAGE	
`'l'` (6Ch)	Set SI language
`[1]`	Language (0 = English, 1 = Italian, 2 = Japanese, 3 = German)
Expected replies: STS_SUCCESS	

CMD_TIMEOUT	
`'o'` (6Fh)	Set recognition timeout
`[1]`	Timeout (-1 = default, 0 = infinite, 1-31 = seconds)
Expected replies: STS_SUCCESS	

CMD_RECOG_SI	
`'i'` (69h)	Activate SI recognition from specified wordset
`[1]`	Wordset index (0-3)
Expected replies: STS_SIMILAR, STS_TIMEOUT, STS_ERROR	

CMD_TRAIN_SD	
`'t'` (74h)	Train specified SD/SV command
`[1]`	Group index (0 = trigger, 1-15 = generic, 16 = password)
`[2]`	Command position (0-31)
Expected replies: STS_SUCCESS, STS_RESULT, STS_SIMILAR, STS_TIMEOUT, STS_ERROR	

CMD_GROUP_SD	
`'g'` (67h)	Insert new SD/SV command
`[1]`	Group index (0 = trigger, 1-15 = generic, 16 = password)
`[2]`	Position (0-31)
Expected replies: STS_SUCCESS, STS_OUT_OF_MEM	

CMD_UNGROUP_SD	
`'u'` (75h)	Remove SD/SV command
`[1]`	Group index (0 = trigger, 1-15 = generic, 16 = password)
`[2]`	Position (0-31)
Expected replies: STS_SUCCESS	

CMD_RECOG_SD	
`'d'` (64h)	Activate SD/SV recognition
`[1]`	Group index (0 = trigger, 1-15 = generic, 16 = password)
Expected replies: STS_RESULT, STS_SIMILAR, STS_TIMEOUT, STS_ERROR	

CMD_ERASE_SD	
`'e'` (65h)	Erase training of SD/SV command
`[1]`	Group index (0 = trigger, 1-15 = generic, 16 = password)
`[2]`	Command position (0-31)
Expected replies: STS_SUCCESS	

CMD_NAME_SD	
'n' (6Eh)	Label SD/SV command
[1]	Group index (0 = trigger, 1-15 = generic, 16 = password)
[2]	Command position (0-31)
[3]	Length of label (0-31)
[4-n]	Text for label (ASCII characters from 'A' to '`')
Expected replies: STS_SUCCESS	

CMD_COUNT_SD	
'c' (63h)	Request count of SD/SV commands in the specified group
[1]	Group index (0 = trigger, 1-15 = generic, 16 = password)
Expected replies: STS_COUNT	

CMD_DUMP_SD	
'p' (70h)	Read SD/SV command data (label and training)
[1]	Group index (0 = trigger, 1-15 = generic, 16 = password)
[2]	Command position (0-31)
Expected replies: STS_DATA	

CMD_MASK_SD	
'm' (6Dh)	Request bit-mask of non-empty groups
Expected replies: STS_MASK	

CMD_RESETALL	
'r' (72h)	Reset all commands and groups
'R' (52h)	Confirmation character
Expected replies: STS_SUCCESS	

CMD_ID	
'x' (78h)	Request firmware identification
Expected replies: STS_ID	

CMD_DELAY	
'y' (79h)	Set transmit delay
[1]	Time (0-10 = 0-10 ms, 11-19 = 20-100 ms, 20-28 = 200-1000 ms)
Expected replies: STS_SUCCESS	

CMD_BAUDRATE	
'a' (61h)	Set communication baud-rate
[1]	Speed mode (1 = 115200, 2 = 57600, 3 = 38400, 6 = 19200, 12 = 9600)
Expected replies: STS_SUCCESS	

Status Details

Replies to commands follow this format. Please note that numeric arguments of status replies are mapped to upper-case letters (see the related section).

STS_MASK	
`'k'` (6Bh)	Mask of non-empty groups
`[1-8]`	4-bit values that form 32-bit mask, LSB first
In reply to: CMD_MASK_SD	

STS_COUNT	
`'c'` (63h)	Count of commands
`[1]`	Integer (0-31)
In reply to: CMD_COUNT_SD	

STS_AWAKEN	
`'w'` (77h)	Wake-up (back from power-down mode)
In reply to: Any character after power on or sleep mode	

STS_DATA	
`'d'` (64h)	Provide command data
`[1]`	Training information (-1=empty, 1-6 = training count, +8 = SD/SV conflict, +16 = SI conflict) **Known issues:** In firmware ID 0, command creation/deletion might cause other empty commands training count to change to 7. Treat count values of -1, 0 or 7 as empty training markers. Never train commands more than 2 or 3 times.
`[2]`	Conflicting command position (0-31, only meaningful when trained)
`[3]`	Length of label (0-31)
`[4-n]`	Text of label (ASCII characters from `'A'` to ``'`'``)
In reply to: CMD_DUMP_SD	

STS_ERROR	
`'e'` (65h)	Signal recognition error
`[1-2]`	Two 4-bit values that form 8-bit error code (80h = NOTA, otherwise see FluentChip error codes)
In reply to: CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD	

STS_INVALID	
`'v'` (76h)	Invalid command or argument
In reply to: Any invalid command or argument	

STS_TIMEOUT	
`'t'` (74h)	Timeout expired
In reply to: CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD	

STS_INTERR	
`'i'` (69h)	Interrupted recognition
In reply to: CMD_BREAK while in training or recognition	

STS_SUCCESS	
`'o'` (6Fh)	OK or no errors status
In reply to: CMD_BREAK, CMD_DELAY, CMD_BAUDRATE, CMD_TIMEOUT, CMD_KNOB, CMD_LEVEL, CMD_LANGUAGE, CMD_SLEEP, CMD_GROUP_SD, CMD_UNGROUP_SD, CMD_ERASE_SD, CMD_NAME_SD, CMD_RESETALL	

STS_RESULT	
`'r'` (72h)	Recognised SD/SV command or Training similar to SD/SV command
`[1]`	Command position (0-31)
In reply to: CMD_RECOG_SD, CMD_TRAIN_SD	

STS_SIMILAR	
`'s'` (73h)	Recognised SI word or Training similar to SI word
`[1]`	Word index (0-31)
In reply to: CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD	

STS_OUT_OF_MEM	
`'m'` (6Dh)	Memory full error
In reply to: CMD_GROUP_SD	

STS_ID	
`'x'` (78h)	Provide firmware identification
`[1]`	Version identifier (0)
In reply to: CMD_ID	

Communication Examples

These are some examples of actual command and status strings exchanged with the VRbot module by host programs and the expected program flow with pseudo-code sequences.

The pseudo-instruction `SEND` transmits the specified character to the module, while `RECEIVE` waits for a reply character (a timeout is not explicitly handled for simple commands, but should be always implemented if possible).

Also, the `OK` and `ERROR` routines are not explicitly defined, since they are host and programming language dependent, but appropriate code should be written to handle both conditions.

Lines beginning with a `#` (sharp) character are comments.

Please note that in a real programming language it would be best to define some constants for the command and status characters, as well as for mapping numeric arguments, that would be used throughout the program, to minimize the chance of repetition errors and clarify the meaning of the code.

See the header file *protocol.h* for sample definitions that can be used in a C language environment.

Here below all the characters sent and received are written explicitly in order to clarify the communication protocol detailed in the previous sections.

(1) Recommended wake up procedure:

```
# wake up or interrupt recognition or do nothing
# (use a timeout or max repetition count)
DO
    SEND 'b'
LOOP UNTIL RECEIVE = 'o'
```

(2) Recommended setup procedure:

```
# ask firmware id
SEND 'x'
IF NOT RECEIVE = 'x' THEN ERROR

# send ack and read status (expecting id=0)
SEND ' '
IF RECEIVE = 'A' THEN OK ELSE ERROR

# set language for SI recognition (Japanese)
SEND 'l'
SEND 'C'
IF RECEIVE = 'o' THEN OK ELSE ERROR

# set timeout (5 seconds)
SEND 'o'
SEND 'F'
IF RECEIVE = 'o' THEN OK ELSE ERROR
```

(3) Recognition of a built-in SI command:

```
# start recognition in wordset 1
SEND 'i'
SEND 'B'
# wait for reply:
# (if 5s timeout has been set, wait for max 6s then abort
#  otherwise trigger recognition could never end)
result = RECEIVE

IF result = 's' THEN
    # successful recognition, ack and read result
    SEND ' '
    command = RECEIVE – 'A'
    # perform actions according to command
ELSE IF result = 't' THEN
    # timed out, no word spoken
ELSE IF result = 'e' THEN
    # error code, ack and read which one
    SEND ' '
    error = (RECEIVE – 'A') * 16
    SEND ' '
    error = error + (RECEIVE – 'A')
    # perform actions according to error
ELSE
    # invalid request or reply
    ERROR
END IF
```

(4) Adding a new SD command:

```
# insert command 0 in group 3
SEND 'g'
SEND 'D'
SEND 'A'
IF RECEIVE = 'o' THEN OK ELSE ERROR

# set command label to “ARDUINO_2009”
SEND 'g'
SEND 'D'
SEND 'A'
SEND 'M'  # name length (12 characters)
SEND 'A'
SEND 'R'
SEND 'D'
SEND 'U'
SEND 'I'
SEND 'N'
SEND 'O'
SEND '_'
# encode each digit with a ^ prefix
# followed by the digit mapped to upper case letters
SEND '^'
SEND 'C'
SEND '^'
SEND 'A'
SEND '^'
SEND 'A'
SEND '^'
SEND 'J'
IF RECEIVE = 'o' THEN OK ELSE ERROR
```

(5) Training an SD command:

```
# repeat the whole training procedure twice for best results
# train command 0 in group 3
SEND 't'
SEND 'D'
SEND 'A'
# wait for reply:
#  (default timeout is 3s, wait for max 1s more then abort)
result = RECEIVE

IF RECEIVE = 'o' THEN
    # training successful
    OK
ELSE IF result = 'r' THEN
    # training saved, but spoken command is similar to
    # another SD command, read which one
    SEND ' '
    command = RECEIVE - 'A'
    # may notify user and erase training or keep it
ELSE IF result = 's' THEN
    # training saved, but spoken command is similar to
    # another SI command (always trigger, may skip reading)
    SEND ' '
    command = RECEIVE - 'A'
    # may notify user and erase training or keep it
ELSE IF result = 't' THEN
    # timed out, no word spoken or heard
ELSE IF result = 'e' THEN
    # error code, ack and read which one
    SEND ' '
    error = (RECEIVE - 'A') * 16
    SEND ' '
    error = error + (RECEIVE - 'A')
    # perform actions according to error
ELSE
    # invalid request or reply
    ERROR
END IF
```

(6) Read used command groups:

```
# request mask of groups in use
SEND 'm'
IF NOT RECEIVE = 'k' THEN ERROR
# read mask to 32 bits variable
# in 8 chunks of 4 bits each
SEND ' '
mask = (RECEIVE - 'A')
SEND ' '
mask = mask + (RECEIVE - 'A') * $2^{4}$
SEND ' '
mask = mask + (RECEIVE - 'A') * $2^{8}$
...
SEND ' '
mask = mask + (RECEIVE - 'A') * $2^{24}$
```

(7) Read how many commands in a group:

```
# request command count of group 3
SEND 'c'
SEND 'D'
IF NOT RECEIVE = 'c' THEN ERROR
# ack and read count
SEND ' '
count = RECEIVE - 'A'
```

(8) Read a user defined command:

```
# dump command 0 in group 3
SEND 'p'
SEND 'D'
SEND 'A'
IF NOT RECEIVE = 'd' THEN ERROR
# read command data
SEND ' '
training = RECEIVE - 'A'
# extract training count (2 for a completely trained command)
tr_count = training AND 7
# extract flags for conflicts (SD or SI)
tr_flags = training AND 24
# read index of conflicting command (same group) if any
SEND ' '
conflict = RECEIVE - 'A'
# read label length
SEND ' '
length =  RECEIVE - 'A'
# read label text
FOR i = 0 TO length - 1
 SEND ' '
 label[i] = RECEIVE
 # decode digits
 IF label[i] = '^' THEN
 SEND ' '
 label[i] = RECEIVE - 'A' + '0'
 END IF
NEXT
```

Built-in Command Sets

In the tables below a list of all built-in commands for each supported language, along with group index (trigger or wordset), command index and language identifier to use with the communication protocol.

		Language			
		0	1	2	3
Trigger/Wordset	**Command Index**	**English (US)**	**Italian**	**Japanese**	**German**
0	0	robot	robot	ロボット	roboter
1	0	action	azione	アクション	aktion
	1	move	vai	ススメ	gehe
	2	turn	gira	マガレ	wende
	3	run	corri	ハシレ	lauf
	4	look	guarda	ミロ	schau
	5	attack	attacca	コーゲキ	attacke
	6	stop	fermo	トマレ	halt
	7	hello	ciao	こんにちわ	hallo
2	0	left	a sinistra	ヒダリ	nach_links
	1	right	a destra	ミギ	nach_rechts
	2	up	in alto	ウエ	hinauf
	3	down	in basso	シタ	hinunter
	4	forward	avanti	マエ	vorwärts
	5	backward	indietro	ウシロ	rückwärts
3	0	zero	zero	ゼロ	null
	1	one	uno	いち	eins
	2	two	due	ニ	zwei
	3	three	tre	サン	drei
	4	four	quattro	ヨン	vier
	5	five	cinque	ゴ	fünf
	6	six	sei	ロク	sechs
	7	seven	sette	ナナ	sieben
	8	eight	otto	ハち	acht
	9	nine	nove	クュー	neun
	10	ten	dieci	ジュー	zehn

Error codes

In the table below a list of some (the most useful) error codes that may be returned by training or recognition commands.

03h	ERR_DATACOL_TOO_NOISY	too noisy
04h	ERR_DATACOL_TOO_SOFT	spoke too soft
05h	ERR_DATACOL_TOO_LOUD	spoke too loud
06h	ERR_DATACOL_TOO_SOON	spoke too soon
07h	ERR_DATACOL_TOO_CHOPPY	too many segments/too complex
11h	ERR_RECOG_FAIL	recognition failed
12h	ERR_RECOG_LOW_CONF	recognition result doubtful
13h	ERR_RECOG_MID_CONF	recognition result maybe
14h	ERR_RECOG_BAD_TEMPLATE	invalid SD/SV command stored in memory
17h	ERR_RECOG_DURATION	bad pattern durations
80h	ERR_NOT_A_WORD	recognized word is not in vocabulary

The first group of codes (03h – 07h) are due to errors in the way of speaking to the VRbot or disturbances in the acquired audio signal, that may depend on the surrounding environment.

The second group (11h – 13h) indicate an insufficient score of the recognized word (from lowest to highest). Acceptance of lower score results may be allowed by lowering the “knob” or “level” settings, respectively for built-in and custom commands (see CMD_KNOB and CMD_LEVEL).

A third group of codes (14h – 17h) reports errors in the stored commands, that may be due to memory corruption. We suggest you check power level and connections, then erase all the commands in the faulty group and train them again.

The last code (80h) means that a word has been recognized that is not in the specified built-in sets. This is due to how Speaker Independent recognition works and should be ignored.