

# VRbot Communication Protocol

**Revision 1.6**

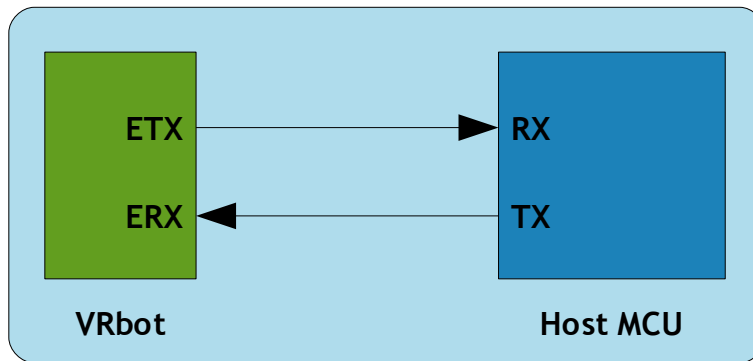
## Table of contents

Protocol and Interface Basics.....	2	"M\$_BA ) \$RA* .....	,
Arguments Mapping.....	3	&status \$etails.....	.
ARG_MIN.....	3	&*&_MA&#.....	.
ARG_MAX.....	3	&*&_" ! ) N*.....	.
ARG_Z R ! .....	3	&*&_A / A# N.....	.
ARG_A " #.....	3	&*&_\$A*A.....	.
"ommand \$etails.....%		&*&_ RR ! R.....	.
"M\$_BR A#.....%		&*&_IN(A ' I\$.....	.
"M\$_&' P.....%		&*&_*IM ! ) *.....	.
"M\$_#N ! B.....%		&*&_IN* RR.....	0
"M\$_' ( '.....%		&*&_&) " " &&.....	0
"M\$_' ANG ) AG .....		&*&_R &) '*.....	0
"M\$_*IM ! ) *.....+		&*&_IMI ' AR.....	0
"M\$_R " ! G_&I.....+		&*&_ ! ) * _ ! 1_M M.....	0
"M\$_*RAIN_&\$.....+		&*&_I\$.....	0
"M\$_GR ! ) P_&\$.....+		"ommunication 2amples.....	3
"M\$_) NGR ! ) P_&\$.....+		456Recommended 7a8e up procedure9.....	3
"M\$_R " ! G_&\$ .....		426Recommended setup procedure9.....	3
"M\$_ RA& _&\$ .....		436Recognition of a :uilt;in &I command9.....	3
"M\$_NAM _&\$ .....		4%6Adding a ne 7 &\$ command9.....	5<
"M\$_" ! ) N*_&\$ .....		4+6*raining an &\$ command9.....	5<
"M\$_\$ ) MP_&\$ .....		4,6Read used command groups9.....	55
"M\$_MA&#_&\$ .....		4.6Read =o7 man> commands in a group9....	55
"M\$_R & *A ' ' .....		406Read a user defined command9.....	52
"M\$_I\$ .....		Built;in "ommand &ets.....	53
"M\$_\$ ' A- .....		error codes.....	5%

# Protocol and Interface Basics

The communication interface (UART) module uses a standard UART interface compatible with 3.3V and 5V logic levels according to the pinning voltage (V<sub>CC</sub>).

A typical connection to an MCU is as follows:



The initial configuration at power on is 3,8400 baud, 8 data bits, No parity, 1 stop bit. The baud rate can be changed later to operate in the range 3,8400 to 55,200 baud.

The communication protocol only uses printable ASCII characters which can be divided into 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

A command sent on the TX line with Aero or more additional arguments is received as an answer on the RX line in the form of a status followed by Aero or more arguments.

There is a minimum delay before each byte sent out from the UART module to the RX line that is initially set to 2ms and can be selected later in the ranges < 3ms, 5ms, 10ms, 20ms, 50ms, 100ms, 200ms, 500ms, 1s. That accounts for slower or faster host systems and therefore suitable also for software-based serial communication.

The communication is asynchronous and each byte of the reply to a command is acknowledged by the host to receive additional status data using the space character. The reply is aborted if another 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 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 0 to 35. These are encoded using most significant characters in the range of uppercase letters. These are some useful constants to handle arguments easily.

ARG_MIN	
'@' 0<=6	Minimum argument value 0;56

ARG_MAX	
'`' 4,<=6	Maximum argument value 4C356

ARG_ZERO	
'A' 0%5=6	Zero argument value 4<6

ARG_ACK	
' ' 42<=6	Read more status arguments

Defining these constants defined in our 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.

A&"II	'@'	'A'	'B'	'C'	...	'Y'	'Z'	'^'	'['	'\'	']'	'_'	'`'
D X	0<=6	0%5=6	0%2=6	0%3=6	...	4+3=6	4+A=6	4+B=6	4+"=6	4+\$=6	4+ =6	4+1=6	4,<=6
(alue	;5	<	5	2	...	2%	2+	2,	2.	20	23	3<	35

## Command Details

Format of command strings accepted : > t=e module. Please note t=at numeric arguments of command reBuests are mapped to upper;case letters 4see a:o?e section6.

### *CMD BREAK*

'b' 4,2=6	A:ort recognition in progress if an> or do not=ing  Known issues: In firm7are I\$ < an> ot=er c=aracter recei?ed during recognition 7ill pre?ent t=is command from stopping recognition! t=at 7ill continue until timeout or ot=er recognition results.
E pected replies: &*&_& ) " " &&! &*&_IN* RR	

### *CMD SLEEP*

's' 4.3=6	Go to t=e specified po7er;do7n mode
[1]	&leep mode 4<;069  < G 7a8e on recei?ed c=aracter onl> 5 G 7a8e on 7=istle or recei?ed c=aracter 2 G 7a8e on loud sound or recei?ed c=aracter 3;+ G 7a8e on dou:le clap 47 it= ?ar>ing sensiti?it>6 or recei?ed c=aracter ,;0 G 7a8e on triple clap 47 it= ?ar>ing sensiti?it>6 or recei?ed c=aracter
E pected replies: &*&_& ) " " &&	

### *CMD KNOB*

'k' 4,B=6	&et &l 8no: to specified le?el
[1]	"onfidence t=res=old le?el 4<;%69  <G loosest9more ?alid results 2G t>pical ?alue 4default6 %G tig=test9fe7er ?alid results  N! * 9 8no: is ignored for trigger 7ords
E pected replies: &*&_& ) " " &&	

### *CMD LEVEL*

'v' 4.,=6	&et &\$ le?el
[1]	&trictness control setting 45;+69  5 G eas> 2 G default + G =ard  A =ig=er setting 7ill result in more recognition errors.
E pected replies: &*&_& ) " " &&	

### *CMD LANGUAGE*

'l' 4,"=6	&et &l language
[1]	'anguage 4< G nglis= 5 G Italian 2 G Fapanese 3 G German6
E pected replies: &*&_& ) " " &&	

#### *CMD TIMEOUT*

'o' 4,1=6	&et recognition timeout
[1]	*imeout 4;5 G default 5;35 G seconds6
Expected replies: &* &_& ) " " &&	

#### *CMD RECOG SI*

'i' 4,3=6	Activate &l recognition from specified 7 ordset
[1]	/ ordset inde2 4<;36
Expected replies: &* &_&IMI 'AR &* &_*IM ! ) * &* &_ RR ! R	

#### *CMD TRAIN SD*

't' 4,%=6	*rain specified &\$E& ( command
[1]	Group inde2 4< G trigger 5;5+ G generic 5 , G pass 7 ord6
[2]	"ommand position 4<;356
Expected replies: &* &_& ) " " && &* &_R & ) ' * &* &_&IMI 'AR &* &_*IM ! ) * &* &_ RR ! R	

#### *CMD GROUP SD*

'g' 4,.=6	Insert ne7 &\$E& ( command
[1]	Group inde2 4< G trigger 5;5+ G generic 5 , G pass 7 ord6
[2]	Position 4<;356
Expected replies: &* &_& ) " " && &* &_! ) * _ ! 1_M M	

#### *CMD UNGROUP SD*

'u' 4,+=6	Remove &\$E& ( command
[1]	Group inde2 4< G trigger 5;5+ G generic 5 , G pass 7 ord6
[2]	Position 4<;356
Expected replies: &* &_& ) " " &&	

#### *CMD RECOG SD*

'd' 4,%=6	Activate &\$E& ( recognition
[1]	Group inde2 4< G trigger 5;5+ G generic 5 , G pass 7 ord6
Expected replies: &* &_R & ) ' * &* &_&IMI 'AR &* &_*IM ! ) * &* &_ RR ! R	

#### *CMD ERASE SD*

'e' 4,+=6	rase training of &\$E& ( command
[1]	Group inde2 4< G trigger 5;5+ G generic 5 , G pass 7 ord6
[2]	"ommand position 4<;356
Expected replies: &* &_& ) " " &&	

**CMD\_NAME SD**

'n' 4, =6	'a:el &\$E& ( command
[1]	Group inde2 4< G trigger 5;5+ G generic 5, G pass7 ord6
[2]	"ommand position 4<;356
[3]	'engt= of la:el 4<;356
[4-n]	*e2t for la:el 4A&" ll c=aracters from 'A' to ' ` '6

Expected replies: &\*&\_&) " " &&

**CMD\_COUNT SD**

'c' 4,3=6	ReBuest count of &\$E& ( commands in t=e specified group
[1]	Group inde2 4< G trigger 5;5+ G generic 5, G pass7 ord6

Expected replies: &\*&\_" ! ) N\*

**CMD\_DUMP SD**

'p' 4.<=6	Read &\$E& ( command data 4la:el and training6
[1]	Group inde2 4< G trigger 5;5+ G generic 5, G pass7 ord6
[2]	"ommand position 4<;356

Expected replies: &\*&\_ \$A\*A

**CMD\_MASK SD**

'm' 4, \$=6	ReBuest : it;mas8 of non;empt> groups
-------------	---------------------------------------

Expected replies: &\*&\_MA&#

**CMD\_RESETALL**

'r' 4.2=6	Reset all commands and groups
'R' 4+2=6	"onfirmation c=aracter

Expected replies: &\*&\_&) " " &&

**CMD\_ID**

'x' 4.0=6	ReBuest firm7are identification
-----------	---------------------------------

Expected replies: &\*&\_I\$

**CMD\_DELAY**

'y' 4.3=6	&et transmit dela>
[1]	*ime 4<;5< G <;5< ms 55;53 G 2<;5<< ms 2<;20 G 2<<;5<<< ms6

Expected replies: &\*&\_&) " " &&

**CMD\_BAUDRATE**

'a' 4,5=6	&et communication :aud;rate
[1]	&peed mode 45 G 55+2<<< 2 G +. , <<< 3 G 30%<<< , G 532<<< 52 G 3, <<6

Expected replies: &\*&\_&) " " &&

## !tatus Details

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

STS_MASK	
'k' 4,B=6	Mask of non-empty groups
[1-8]	%;: it values that form 32;: it mask ' &B first
In repl" to: "M\$_MA&#_&\$	

STS_COUNT	
'c' 4,3=6	Count of commands
[1]	Integer 4<;356
In repl" to: "M\$_" ! ) N*_&\$	

STS_AWAKEN	
'w' 4..=6	/ awake; up 4: awake from power; down mode
In repl" to: An> character after power on or sleep mode	

STS_DATA	
'd' 4,%=6	Provide command data
[1]	*training information 4;5empty> 5; , G training count C0 G &\$E& ( conflict C5 , G &I conflict Known issues: In firmware I\$ < command creation/deletion might cause other empty commands training count to change to .. *reat count values of ;5< or . as empty training markers. Never train commands more than 2 or 3 times.
[2]	"conflicting command position 4<;35 only> meaningful when trained
[3]	'length of label 4<;356
[4-n]	*each of label 4A&"    characters from 'A' to ' ` '6
In repl" to: "M\$_\$ ) MP_&\$	

STS_ERROR	
'e' 4,+ =6	Signal recognition error
[1-2]	*70 %;: it values that form 0;: it error code 40<= G N ! *A other wise see 1luent " =ip error codes
In repl" to: "M\$_R " !G_&I "M\$_R " !G_&\$ "M\$_*RAIN_&\$	

STS_INVALID	
'v' 4., =6	Invalid command or argument
In repl" to: An> invalid command or argument	

STS_TIMEOUT	
't' 4.%=6	*timeout expired
In repl" to: "M\$_R " !G_&I "M\$_R " !G_&\$ "M\$_*RAIN_&\$	

STS INTERRUPT	
'i' 4,3=6	Interrupted recognition
In repl" to: "M\$_BR A# 7=ile in training or recognition	

STS SUCCESS	
'o' 4,1=6	! # or no errors status
In repl" to: "M\$_BR A# "M\$_\$ 'A- "M\$_BA) \$RA* "M\$_*IM ! ) * "M\$_#N!B "M\$_' ( ' "M\$_'ANG)AG "M\$_&' P "M\$_GR!)P_&\$ "M\$_)NGR!)P_&\$ "M\$_ RA& _&\$ "M\$_NAM _&\$ "M\$_R & *A' '	

STS RESULT	
'r' 4.2=6	Recognised &\$E& ( command or *raining similar to &\$E& ( command
[1]	"ommand position 4<;356
In repl" to: "M\$_R " !G_&\$ "M\$_*RAIN_&\$	

STS SIMILAR	
's' 4.3=6	Recognised &I 7ord or *raining similar to &I 7ord
[1]	/ ord inde2 4<;356
In repl" to: "M\$_R " !G_&I "M\$_R " !G_&\$ "M\$_*RAIN_&\$	

STS OUT OF MEM	
'm' 4, \$=6	Memor> full error
In repl" to: "M\$_GR! )P_&\$	

STS ID	
'x' 4.0=6	Pro?ide firm7are identification
[1]	(ersion identifier 4<6
In repl" to: "M\$_I\$	



## Communication Examples

These are some examples of actual command and status strings exchanged between a host (Robot module) and a program and the expected program flow. These are pseudo-code sequences.

The pseudo-instruction `SEND` transmits the specified character to the module. The `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 those conditions.

Lines beginning with a `#` and a space 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 follow all the characters sent and received are written explicitly in order to clarify the communication protocol detailed in the previous sections.

**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'
```

**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
```

**Recognition of a built-in 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

```

## Adding a new !D 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

```

## ,raining an !D 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

```

#-% 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') * 24
SEND ' '
mask = mask + (RECEIVE - 'A') * 28
...
SEND ' '
mask = mask + (RECEIVE - 'A') * 224

```

#.% Read /ow man" 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'
```

R

or a completely trained command)

```
SEND ' '
```

# Built-in Command Sets

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

		Language			
		<	5	2	3
Trigger Wordset	Command Index	English / #4 ! %	Italian	Japanese	German

<	<	robot	robot	ロボット	roboter
---	---	-------	-------	------	---------

5	<	action	azione	アクション	action
	5	move	vai	ススメ	gehe
	2	turn	gira	マガレ	ende
	3	run	corri	ハシレ	lauf
	%	look	guarda	ミロ	schau
	+	attack	attacca	コーゲキ	attacke
	,	stop	fermo	トマレ	=alt
	.	hello	ciao	こんにちは	=allo

2	<	left	a sinistra	ヒダリ	nach links
	5	right	a destra	ミギ	nach rechts
	2	up	in alto	ウエ	=inauf
	3	down	in basso	シタ	=inunter
	%	forward	avanti	マエ	vorwärts
	+	backward	indietro	ウシロ	rückwärts

3	<	Aero	Aero	ゼロ	null
	5	one	uno	いち	eins
	2	two	due	ニ	zwei
	3	three	tre	サン	drei
	%	four	quattro	ヨン	vier
	+	five	cinque	ゴ	fünf
	,	six	sei	ロク	sechs
	.	seven	sette	ナナ	sieben
	0	eight	otto	ハチ	acht
	3	nine	nove	キュー	neun
	5<	ten	dieci	ジュー	zehn

## Error codes

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

<3=	RR_\$A*A"! '_*!!_N!l& -	too noisy
<%=	RR_\$A*A"! '_*!!_&!1*	speech too soft
<+=	RR_\$A*A"! '_*!!_ '! ) \$	speech too loud
< ,=	RR_\$A*A"! '_*!!_&! !N	speech too soon
< .=	RR_\$A*A"! '_*!!_ "D!PP -	too many segments or too complex
55=	RR_R " !G_1AI'	recognition failed
52=	RR_R " !G_ '! / _ " !N1	recognition result doubtful
53=	RR_R " !G_MI\$_ " !N1	recognition result may be
5%=	RR_R " !G_BA\$_* MP'A*	invalid &\$E& ( command stored in memory)
5.=	RR_R " !G_\$ )RA*! !N	bad pattern durations
0<=	RR_N!*_A_ / !R\$	recognized word is not in vocabulary

\*= the first group of codes <3= J < .=6 are due to errors in the quality of speaking to the (Robot or disturbances in the acquired audio signal that may depend on the surrounding environment.

\*= the second group 455= J 53=6 indicate an insufficient score of the recognized word from lowest to highest. Acceptance of lower score results may be allowed by lowering the Knowledge or Knowledge settings respectively for built-in and custom commands (see "M\$\_#N!B and "M\$\_' ( '6.

A third group of codes 45%= J 5.=6 reports errors in the stored commands that may be due to memory corruption. We suggest you check power levels and connections then erase all the commands in the fault group and train them again.

\*= the last code 40<=6 means that a word has been recognized that is not in the specified built-in sets. \*=is is due to our 8-Bit Independent recognition words and should be ignored.