EasyVR

www.veear.eu

Table of Contents

Product Description

EasyVR is the second generation version of the successful VRbot Module. It is a multi-purpose speech recognition module designed to easily add versatile, robust and cost effective speech recognition capabilities to virtually any application.

The EasyVR module can be used with any host with an UART interface powered at 3.3V – 5V, such as PIC and Arduino boards. Some application examples include home automation, such as voice controlled light switches, locks or beds, or adding "hearing" to the most popular robots on the market.
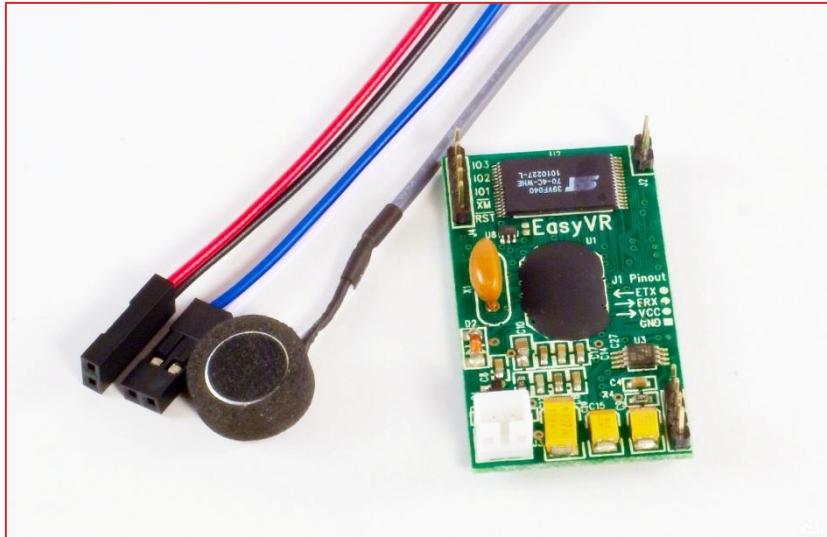
EasyVR features

- A host of built-in Speaker Independent (SI) commands for ready to run basic controls, in the followings languages:
  - English (US)
  - Italian
  - German
  - French
  - Spanish
  - Japanese
- Supports up to 32 user-defined Speaker Dependent (SD) triggers or commands as well as Voice Passwords. SD custom commands can be spoken in ANY language.
- Easy-to-use and simple Graphical User Interface to program Voice Commands and audio.
- Module can be used with any host with an UART interface (powered at 3.3V - 5V)
- Simple and robust documented serial protocol to access and program through the host board
- 3 GPIO lines (IO1, IO2, IO3) that can be controlled by new protocol commands.
- PWM audio output that supports 8 ohm speakers.
- Sound playback feature.

Technical specifications

Physical dimensions and pin assignment
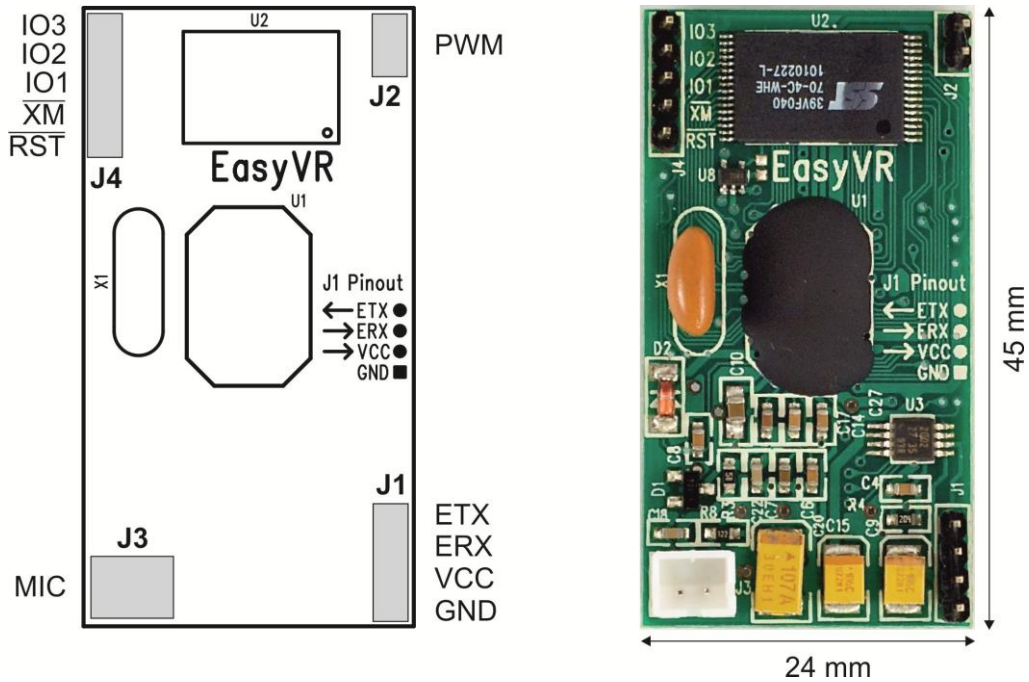
Connector	Number	Name	Type	Description
J1	1	GND	-	Ground
	2	VCC	I	Voltage DC input
	3	ERX	I	Serial Data Receive (TTL level)
	4	ETX	O	Serial Data Transmit (TTL level)
J2	1-2	PWM	O	Differential audio output (can directly drive 8Ω speaker)
J3	1	MIC_RET	-	Microphone reference ground
	2	MIC_IN	I	Microphone input signal
J4	1	/RST	I	Active low asynchronous reset (internal 100K pull-up)
	2	/XM	I	Boot select (internal 1K pull-down)
	3	IO1	I/O	General purpose I/O (**3.0 VDC** TTL level)
	4	IO2	I/O	General purpose I/O (**3.0 VDC** TTL level)
	5	IO3	I/O	General purpose I/O (**3.0 VDC** TTL level)

Note: the GPIO (J4.3, J4.4, and J4.5) are at nominal 3.0VDC level. Do not connect 5VDC directly to these pins!

Recommended Operating Conditions

Symbol	Parameter	Min	Typ	Max	Unit
VCC	Voltage DC Input	3.3	5.0	5.5	V
Ta	Ambient Operating Temperature Range	0	25	70	°C
ERX	Serial Port Receive Data	0	-	VCC	V
ETX	Serial Port Transmit Data	0	-	VCC	V

Electrical Characteristics

These are applicable to J4 pins only, including IO1-3, /XM and /RST.

Symbol	Parameter		Min	Typ	Max	Unit
V_{IH}	Input High Voltage		2.4	3.0	3.3	V
V_{IL}	Input Low Voltage		-0.1	0.0	0.75	V
I_{IL}	Input Leakage Current ($0 < V_{IO} < 3V$, Hi-Z Input)			<1	10	µA
R_{PU}	Pull-up Resistance	Strong		10		kΩ
		Weak		200		kΩ
V_{OH}	Output High Voltage (I_{OH} = -5 mA)		2.4			V
V_{OL}	Output Low Voltage (I_{OL} = 8 mA)				0.6	V

Power Supply Requirements

Symbol	Parameter	Min	Typ	Max	Unit
I_{Sleep}	Sleep current		< 1		mA
I_{Oper}	Operating current		12		mA
$I_{Speaker}$	Audio playback current (with 8ohm speaker)		180		mA (RMS)

Serial Interface

The EasyVR is a "slave" module communicating via an asynchronous serial interface (commonly known as UART interface), with the following features:
- Baud Rate: **9600** (default), 19200, 38700, 57600, 115200
- Frame: **8** Data bits, **N**o parity, **1** Stop bit

The receiver input data line is ERX, while the transmitter output data line is ETX. No handshake lines are used.

Example of a serial data frame representing character "A" (decimal 65 or hexadecimal 41):
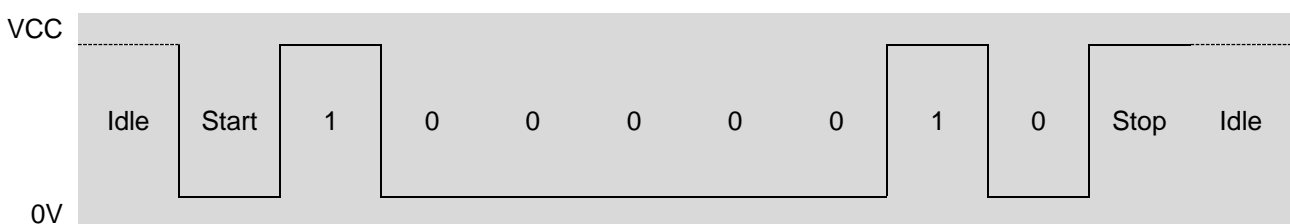
See also chapter COMMUNICATION PROTOCOL later on this manual for communication details.

Microphone

The microphone provided with the EasyVR module is an omnidirectional electret condenser microphone (Horn EM9745P-382):

- Sensitivity -38dB (0dB=1V/Pa @1KHz)
- Load Impedance 2.2K
- Operating Voltage 3V
- Almost flat frequency response in range 100Hz – 20kHz

If you use a microphone with different specifications the recognition accuracy may be adversely affected. No other kind of microphone is supported by the EasyVR.

> **Note:** Vocal commands should be given from about 60cm from the microphone, but you can try at greater distances by talking louder.

Please note that improper acoustic positioning of the microphone will reduce recognition accuracy. Many mechanical arrangements are possible for the microphone element, and some will work better than others. When mounting the microphone in the final device, keep in mind the following guidelines:

1. **Flush Mounting** - The microphone element should be positioned as close to the mounting surface as possible and should be fully seated in the plastic housing. There must be no airspace between the microphone element and the housing. Having such an airspace can lead to acoustic resonance, which can reduce recognition accuracy.

GOOD   BAD

cavity
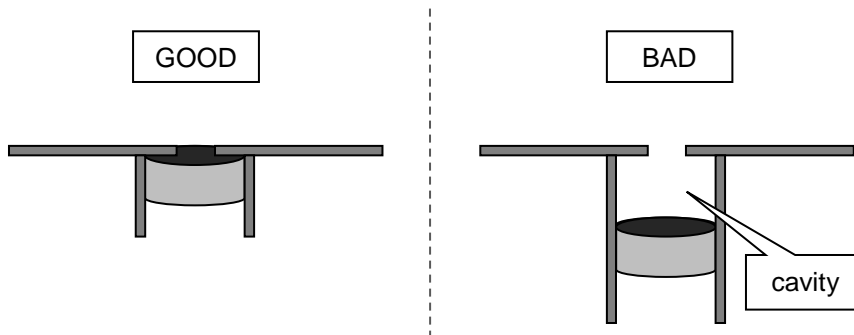
2. **No Obstructions, Large Hole** - The area in front of the microphone element must be kept clear of obstructions to avoid interference with recognition. The diameter of the hole in the housing in front of the microphone should be at least 5 mm. Any necessary plastic surface in front of the microphone should be as thin as possible, being no more than 0.7 mm, if possible.

clear area

internal diaphragm
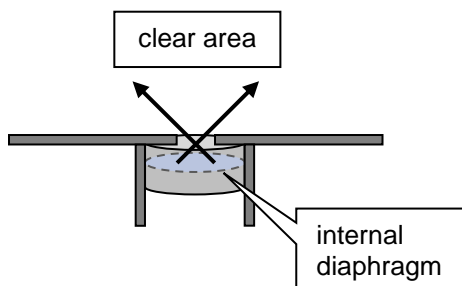
3. **Insulation** - The microphone should be acoustically isolated from the housing if possible. This can be accomplished by surrounding the microphone element with a spongy material such as rubber or foam. The provided microphone has this kind of insulating foam. The purpose is to prevent auditory noises produced by handling or jarring the device from being "picked up" by the microphone. Such extraneous noises can reduce recognition accuracy.

| GOOD | | BAD |
| absorbent material | | fastened directly |
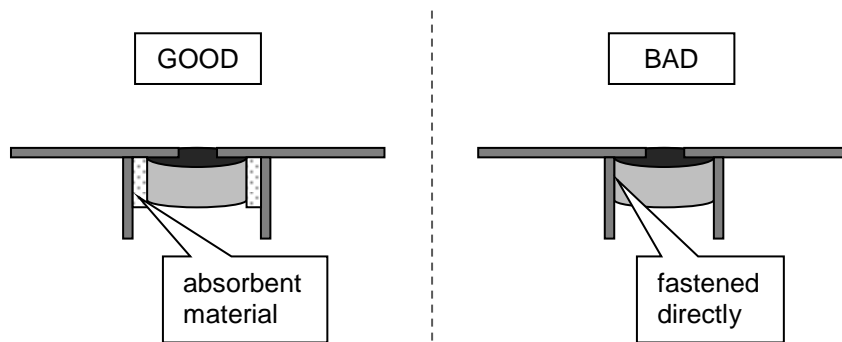
4. **Distance** - If the microphone is moved from 15 cm to 30 cm from the speaker's mouth, the signal power decreases by a factor of four. The difference between a loud and a soft voice can also be more than a factor of four. Although the internal preamplifier of the EasyVR compensates for a wide dynamic range of input signal strength, if its range is exceeded, the user application can provide feedback to the speaker about the voice volume (see appendix ERROR CODES).

Audio Output

The EasyVR audio output interface is capable of directly driving an 8Ω speaker. It could also be connected to an external audio amplifier to drive lower impedance loudspeakers.

> **Note:** Connecting speakers with a lower impedance directly to the module may permanently damage the EasyVR audio output or the whole module.

It is possible to connect higher impedance loads such as headphones, provided that you scale down the output power according to the speaker ratings, for example using a series resistor. The exact resistor value depends on the headphone power ratings and the desired output volume (usually in the order of 10kΩ).

> **Note:** Connecting headphone speakers directly to the EasyVR audio output may damage your hearing.

General Purpose I/O

Since the EasyVR communication interface takes two pins of the host controller, a few spare I/O pins are provided, that can be controlled with the communication protocol, to get those pins back for basic tasks, such as lighting an LED.

The three I/O pins IO1–IO3 are connected directly to the embedded microcontroller on the EasyVR module, so they are referenced to the internal 3.0V regulated power supply. If you need to interface to circuits using a different supply, there is a number of solutions you can adopt. Some of these are outlined below (here IOn indicates any one of the three I/O pins of the EasyVR).

Use a pin as an output

All the I/O pins are inputs with weak internal pull-up after power on. You must explicitly configure a pin before you can use it as an output (see the example code USE GENERAL PURPOSE I/O PINS).
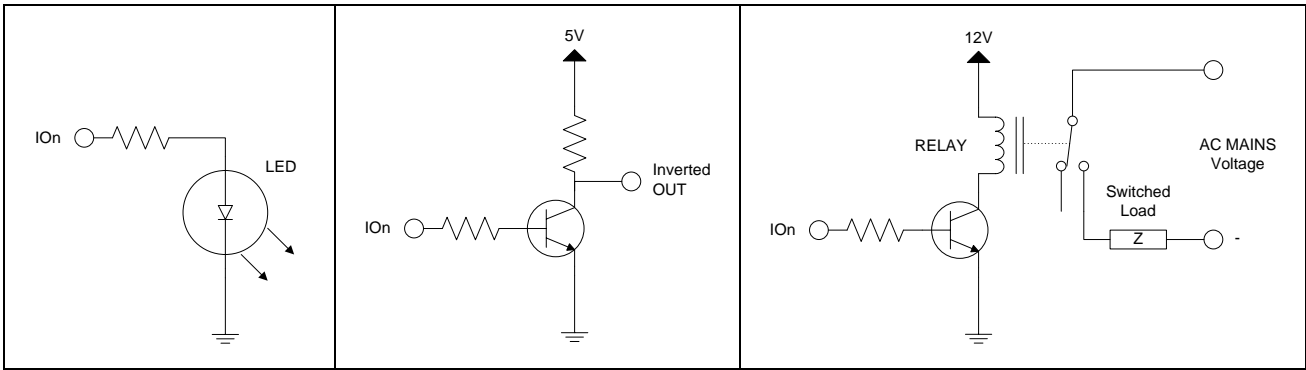
I/O pin directly driving a low-current LED

I/O pin connected to high impedance 5V circuit (such as MCU input pin)

I/O pin switching a load on a high voltage line using a 12V relay

The exact components values in these circuits may vary. You need to calculate required values for your application and choice of components. For example, resistor value for the LED circuit can be calculated approximately as:

$$R_{LED} = \frac{V_{OH} - V_{LED}}{I_{OH}}$$

Where V_{LED} is the LED forward voltage, as reported on the LED datasheet, at the driving current I_{OH} (see section ELECTRICAL CHARACTERISTICS). Let's assume a typical low-current LED has a V_F=1.8V at 5mA, the resistor value is:

$$R_{LED} = \frac{2.4 - 1.8}{0.005} = 120 \; Ohm$$

Now stay on the safe side and choose a slightly larger resistor, such as 150 Ohm.

If you want to drive higher current LEDs, you need a circuit like the second one, where you put the LED between the output resistor and the collector of the NPN transistor.

Use a pin as an input

All the I/O pins are inputs with weak internal pull-up after power on or reset. You may also configure the pin to have a strong pull-up or no pull-up at all (see the example code USE GENERAL PURPOSE I/O PINS).
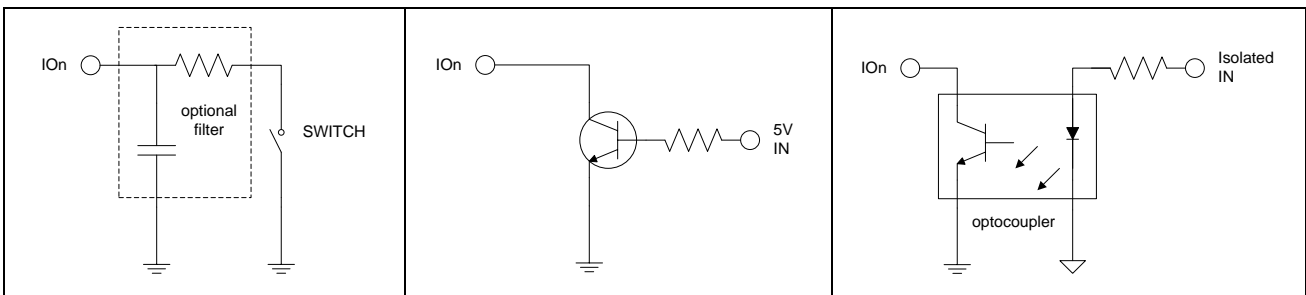
I/O pin connected to a switch (or switching sensor)

I/O pin connected 5V source (such as MCU output pin)

I/O pin with isolated input (for safety circuits)

All these circuits assume the EasyVR pin has been configured with an internal pull-up (passive components value can be adjusted to account for weak or strong pull-up).

Disabling the internal pull-up could be used to put the pin in high-impedance state, for example to simulate a tri-state or open-drain output port.

Again, you should refer to the manufacturer's datasheet when interfacing any external components and to calculate required resistors values or other passive components.

Flash Update

The EasyVR module includes a bootloader that allows to update the firmware and to download new sound tables to the on-board memory.

The boot mode is activated by keeping the **/XM** signal to a high logical level at power on or reset. This can be easily done with a jumper taking the signal to a suitable pull-up resistor.

To download a firmware update or a sound table to the EasyVR, power on the module with the jumper closed. For normal operation, just leave the jumper open. Do not change the jumper position while the module is already powered on. It is safe to change **/XM** level while the module is reset (**/RST** low).
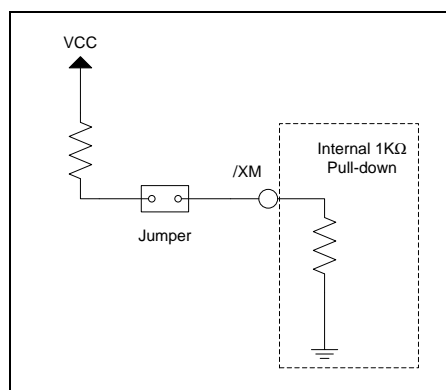
Boot mode selection circuit

The pull-up resistor value to use depends on the VCC power supply voltage. For the voltage of the **/XM** pin when the jumper is closed (short) the following relation holds (note you have a voltage divider circuit):

$$V_{XM} = \frac{1K}{R + 1K} \cdot V_{CC}$$

Now if you want **/XM** to be at 3V (logic high) and solving for *R*, you get:

$$R_{PU} = \frac{V_{CC}}{3} \cdot 1000 - 1000$$

That makes 100Ω for 3.3V and around 680Ω for 5V power supplies. Other kinds of circuit are possible, that is just an example and one of the simplest to realize.

To learn how to download new sound tables to your EasyVR module, have a look at the section DOWNLOAD A NEW SOUND TABLE.

Communication Protocol

Basics

Communication with the EasyVR module uses a standard UART interface compatible with 3.3-5V TTL/CMOS 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:
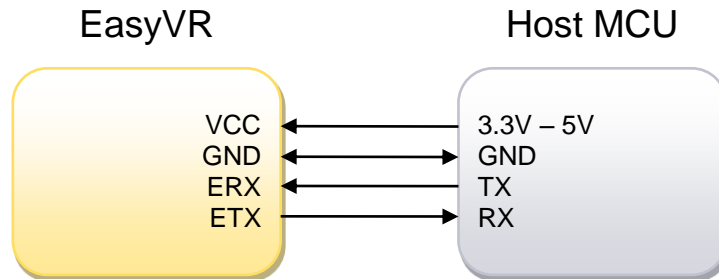
- Command and status characters, respectively on the TX and RX lines, chosen among lower-case letters.
- Command arguments or status details, again on the TX and RX lines, spanning the range of capital letters.

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 EasyVR 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).

Since the EasyVR serial interface also is software-based, a very short delay might be needed before transmitting a character to the module, especially if the host is very fast, to allow the EasyVR to get back listening to a new character.

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 signalled 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	40	41	42	43	...	59	5A	5B	5C	5D	5E	5F	60
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, training or playback 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
	  4 = Spanish
	  5 = French

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	

CMD_QUERY_IO

'q' (71h)	Configure, query or modify general purpose I/O pins
[1]	Pin number (1 = pin IO1, 2 = pin IO2, 3 = pin IO3)
[2]	Pin mode (0 = output low, 1 = output high, 2 = input*, 3 = input strong**, 4 = input weak***) * High impedance input (no pull-up) **Strong means ~10K internal pull-up ***Weak means ~200K internal pull-up (default after power up)
Expected replies: STS_SUCCESS (mode 0-1), STS_PIN (mode 2-4)	

CMD_PLAY_SX

'w' (6Eh)	Wave table entry playback
[1-2]	Two 5-bit values that form a 10-bit index to the sound table (index = [1] * 32 + [2])
[3]	Playback volume (0-31, 0 = min volume, 15 = full scale, 31 = double gain)
Expected replies: STS_SUCCESS, STS_ERROR	

CMD_DUMP_SX

'h' (68h)	Read wave table data
Expected replies: STS_TABLE_SX, STS_OUT_OF_MEM	

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 (error = [1] * 16 + [2], see appendix)
In reply to: CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD, CMD_PLAY_SX	

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, recognition or playback	

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, CMD_QUERY_IO, CMD_PLAY_SX	

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 error (no more room for commands or sound table not present)
In reply to: CMD_GROUP_SD, CMD_DUMP_SX	

STS_ID

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

STS_PIN

'p' (70h)	Provide pin input status
[1]	Logic level (0 = input low, 1 = input high)
In reply to: CMD_QUERY_IO	

STS_TABLE_SX

'd' (64h)	Provide sound table data
[1-2]	Two 5-bit values that form a 10-bit count of entries in the sound table (count = [1] * 32 + [2])
[3]	Length of table name (0-31)
[4-n]	Text of table name (ASCII characters from 'A' to '`')
In reply to: CMD_DUMP_SX	

Communication Examples

These are some examples of actual command and status strings exchanged with the EasyVR 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 PROTOCOL HEADER FILE 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.

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 ' '
id = RECEIVE
IF id = 'A' THEN
    # it's a VRbot
ELSE IF id = 'B' THEN
    # it's an EasyVR
ELSE
    # next generation?
END IF

# 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 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
```

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 'Q'  # name length (16 characters, digits count twice)
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
```

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
```

Recognition of an SD command

```
# start recognition in group 1
SEND 'd'
SEND 'B'
# wait for reply:
result = RECEIVE

IF result = 'r' 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
```

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
```

In the code above the superscripts render as: 2^4, 2^8, and 2^{24}.

```
# 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
```

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'
```

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
```

Use general purpose I/O pins

```
# set IO1 pin to logic low level
SEND 'q'
SEND 'B'
SEND 'A'
IF RECEIVE = 'o' THEN OK ELSE ERROR

# set IO2 pin to logic high level
SEND 'q'
SEND 'C'
SEND 'B'
IF RECEIVE = 'o' THEN OK ELSE ERROR

# set IO2 pin as input with strong pull-up and read state
SEND 'q'
SEND 'C'
SEND 'D'
IF NOT RECEIVE = 'p' THEN ERROR
# ack and read logic level
SEND ' '
pin_level = RECEIVE - 'A'

# set IO3 pin as high impedance input (reading state is optional)
SEND 'q'
SEND 'D'
SEND 'C'
IF NOT RECEIVE = 'p' THEN ERROR
```

Use custom sound playback

```
# play a beep at full volume (works with any or no table)
SEND 'w'
SEND 'A'
SEND 'A'
SEND 'P'
IF RECEIVE = 'o' THEN OK ELSE ERROR

# play entry 13 at half volume
SEND 'w'
SEND 'A'
SEND 'N'
SEND 'H'
 IF RECEIVE = 'o' THEN OK ELSE ERROR
```

```
# play entry 123 (=3*32+26) at max volume
SEND 'w'
SEND 'A' + 3
SEND 'A' + 26
SEND 'A' + 31
IF RECEIVE = 'o' THEN OK ELSE ERROR
```

Read sound table

```
# dump sound table
SEND 'h'
IF NOT RECEIVE = 'h' THEN ERROR
# read count of entries and name length
SEND ' '
count = (RECEIVE - 'A') * 32
SEND ' '
count = count + (RECEIVE - 'A')
SEND ' '
length = RECEIVE - 'A'
# read name text
FOR i = 0 TO length - 1
 SEND ' '
 label[i] = RECEIVE
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.

Trigger Wordset	Command Index	Language					
		0	**1**	**2**		**3**	**4**	**5**
		English (US)	**Italian**	**Japanese**	**Rōmaji**	**German**	**Spanish**	**French**
0	**0**	robot	robot	ロボット	*robotto*	roboter	robot	robot
1	**0**	action	azione	アクション	*acution*	aktion	acción	action
	1	move	vai	進め	*susu-me*	gehe	muévete	bouge
	2	turn	gira	曲がれ	*magare*	wende	gira	tourne
	3	run	corri	走れ	*hashire*	lauf	corre	cours
	4	look	guarda	見ろ	*miro*	schau	mira	regarde
	5	attack	attacca	攻撃	*kougeki*	attacke	ataca	attaque
	6	stop	fermo	止まれ	*tomare*	halt	para	arrête
	7	hello	ciao	こんにちは	*konnichiwa*	hallo	hola	salut
2	**0**	left	a sinistra	左	*hidari*	nach links	a la izquierda	à gauche
	1	right	a destra	右	*migi*	nach rechts	a la derecha	à droite
	2	up	in alto	上	*ue*	hinauf	arriba	vers le haut
	3	down	in basso	下	*shita*	hinunter	abajo	vers le bas
	4	forward	avanti	前	*mae*	vorwärts	adelante	en avant
	5	backward	indietro	後ろ	*ushiro*	rückwärts	atrás	en arrière
3	**0**	zero	zero	ゼロ	*zero*	null	cero	zéro
	1	one	uno	一	*ichi*	eins	uno	un
	2	two	due	二	*ni*	zwei	dos	deux
	3	three	tre	三	*san*	drei	tres	trois
	4	four	quattro	四	*yon*	vier	cuatro	quatre
	5	five	cinque	五	*go*	fünf	cinco	cinq
	6	six	sei	六	*roku*	sechs	seis	six
	7	seven	sette	七	*nana*	sieben	siete	sept
	8	eight	otto	八	*hachi*	acht	ocho	huit
	9	nine	nove	九	*kyu*	neun	nueve	neuf
	10	ten	dieci	十	*jyuu*	zehn	diez	dix

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
4Ah	ERR_SYNTH_BAD_VERSION	bad release number in speech file
4Eh	ERR_SYNTH_BAD_MSG	bad data in speech file or invalid compression
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 EasyVR 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 fourth group (4Ah – 4Eh) deals with errors in the compressed sound data, either because the wrong version of the QuickSynthesis^{TM} tool has been used to generate the sound table or because a not supported compression scheme has been selected (or data is generically corrupt).

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.

Protocol header file

This file "protocol.h" can be used with applications written in the C language.

```c
#ifndef PROTOCOL_H
#define PROTOCOL_H

#define CMD_BREAK        'b'   // abort recog or ping
#define CMD_SLEEP        's'   // go to power down
#define CMD_KNOB         'k'   // set si knob <1>
#define CMD_LEVEL        'v'   // set sd level <1>
#define CMD_LANGUAGE     'l'   // set si language <1>
#define CMD_TIMEOUT      'o'   // set timeout <1>
#define CMD_RECOG_SI     'i'   // do si recog from ws <1>
#define CMD_TRAIN_SD     't'   // train sd command at group <1> pos <2>
#define CMD_GROUP_SD     'g'   // insert new command at group <1> pos <2>
#define CMD_UNGROUP_SD   'u'   // remove command at group <1> pos <2>
#define CMD_RECOG_SD     'd'   // do sd recog at group <1> (0 = trigger mixed si/sd)
#define CMD_ERASE_SD     'e'   // reset command at group <1> pos <2>
#define CMD_NAME_SD      'n'   // label command at group <1> pos <2> with length <3> name <4-n>
#define CMD_COUNT_SD     'c'   // get command count for group <1>
#define CMD_DUMP_SD      'p'   // read command data at group <1> pos <2>
#define CMD_MASK_SD      'm'   // get active group mask
#define CMD_RESETALL     'r'   // reset all commands and groups
#define CMD_ID           'x'   // get version id
#define CMD_DELAY        'y'   // set transmit delay <1> (log scale)
#define CMD_BAUDRATE     'a'   // set baudrate <1> (bit time, 1=>115200)
#define CMD_QUERY_IO     'q'   // configure, read or write I/O pin <1> of type <2>
#define CMD_PLAY_SX      'w'   // wave table entry <1-2> (10-bit) playback at volume <3>
#define CMD_DUMP_SX      'h'   // dump wave table entries


#define STS_MASK         'k'   // mask of active groups <1-8>
#define STS_COUNT        'c'   // count of commands <1>
#define STS_AWAKEN       'w'   // back from power down mode
#define STS_DATA         'd'   // provide training <1>, conflict <2>, label <3-35> (counted string)
#define STS_ERROR        'e'   // signal error code <1-2>
#define STS_INVALID      'v'   // invalid command or argument
#define STS_TIMEOUT      't'   // timeout expired
#define STS_INTERR       'i'   // back from aborted recognition (see 'break')
#define STS_SUCCESS      'o'   // no errors status
#define STS_RESULT       'r'   // recognised sd command <1> - training similar to sd <1>
#define STS_SIMILAR      's'   // recognised si <1> (in mixed si/sd) - training similar to si <1>
#define STS_OUT_OF_MEM   'm'   // no more available commands (see 'group')
#define STS_ID           'x'   // provide version id <1>
#define STS_PIN          'p'   // return pin state <1>
#define STS_TABLE_SX     'h'   // provide table count <1-2> (10-bit), name <3-35> (counted string)

// protocol arguments are in the range 0x40 (-1) to 0x60 (+31) inclusive
#define ARG_MIN     0x40
#define ARG_MAX     0x60
#define ARG_ZERO    0x41
#define ARG_ACK     0x20       // to read more status arguments
#endif //PROTOCOL_H
```

EasyVR Commander

The EasyVR Commander software can be used to easily configure your EasyVR module connected to your PC through an adapter board, or by using the microcontroller host board with the provided "bridge" program (available for ROBONOVA controller board, Arduino 2009/UNO, Parallax Basic Stamp).

The user can define groups of commands or passwords and generate a basic code template to handle them. It is required to edit the generated code to implement the application logic, but the template contains all the functions or subroutines to handle the speech recognition tasks.

Getting started

Connect the adapter board or a microcontroller host board with a running "bridge" program[1] to your PC, then check that all devices are properly turned on and start the EasyVR Commander.
Select the serial port to use from the toolbar or the "File" menu, then go with the "Connect" command.
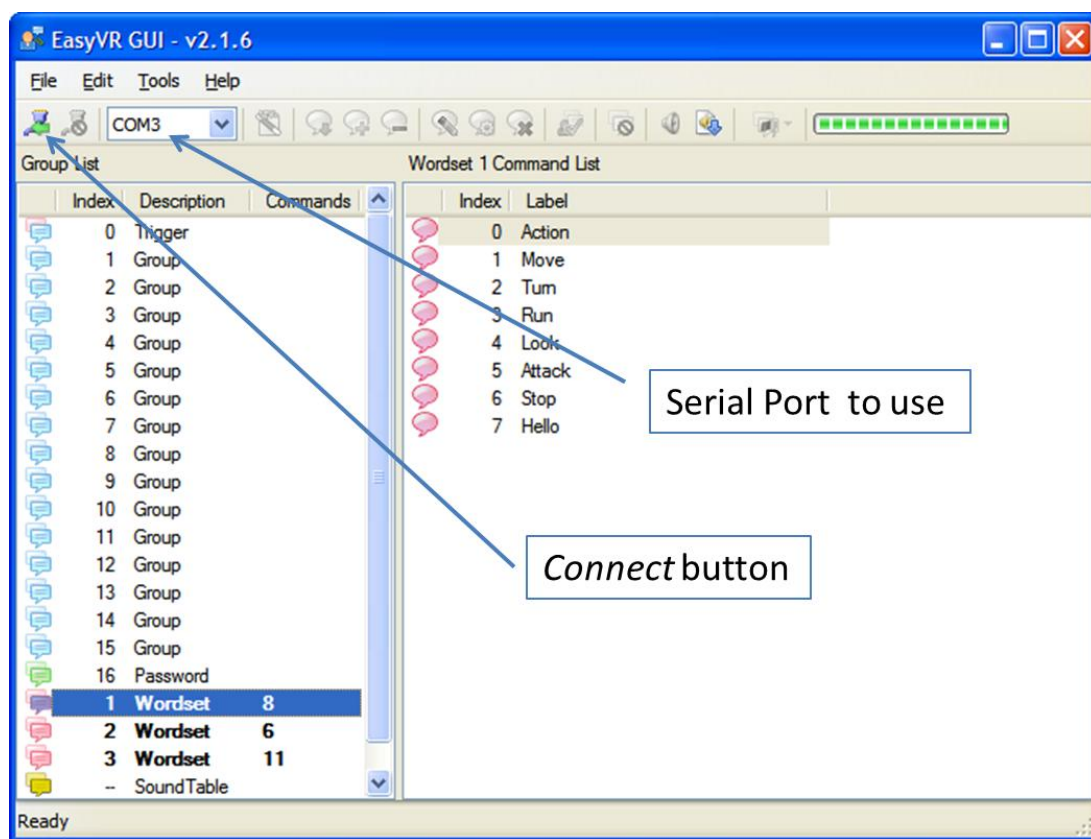
Figure 1 – Main application window

There are four kinds of commands in the software (see Figure 1 and Figure 3):

- **Trigger** - is a special group where you have the built-in SI trigger word "Robot" and you may add one user-defined SD trigger word. Trigger words are used to start the recognition process
- *Group* - where you may add user-defined SD commands
- *Password* - a special group for "vocal passwords" (up to five), using Speaker Verification (SV) technology
- *Wordset* - built-in set of SI commands (for instance in Figure 1 above, the Wordset 1 is selected)

---

[1] On some systems the EasyVR Commander can automatically upload the "bridge" program to the host board once connected. That applies to Robonova controller board and Parallax Basic Stamp.

Speech recognition

The recognition function of the EasyVR works on a single group at a time, so that users need to group together all the commands that they want to be able to use at the same time.

When EasyVR Commander connects to the module, it reads back all the user-defined commands and groups, which are stored into the EasyVR module non-volatile memory.

The user can add a new command by first selecting the group in which the command needs to be created and then using the toolbar icons or the *"Edit"* menu.

A command should be given a label and then it should be trained twice with the user's voice: the user will be guided throughout this process (see Figure 2) when the *"Train Command"* action is invoked.
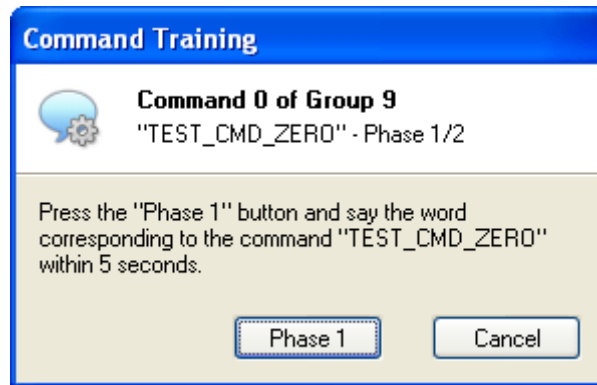
Figure 2 – Guided training dialog

Note: after clicking on Phase 1 or Phase 2 buttons, remember to start speaking only when you see this little window:
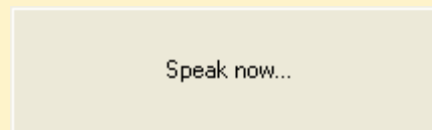
If any error happens, command training will be cancelled. Errors may happen when the user's voice is not heard correctly, there is too much background noise or when the second word heard is too different from the first one.

The software will also alert if a command is too similar to another one by specifying the index of the conflicting command in the *"Conflict"* column. For example, in the following Figure 3 the command "TEST_CMD_ONE" sounds too similar to "TEST_CMD_ZERO" (i.e. they have been trained with a similar pronunciation).

Note: TEST_CMD_ZERO and TEST_CMD_ONE are just examples of labels, you should use label names that reflects the real command that you are going to train.
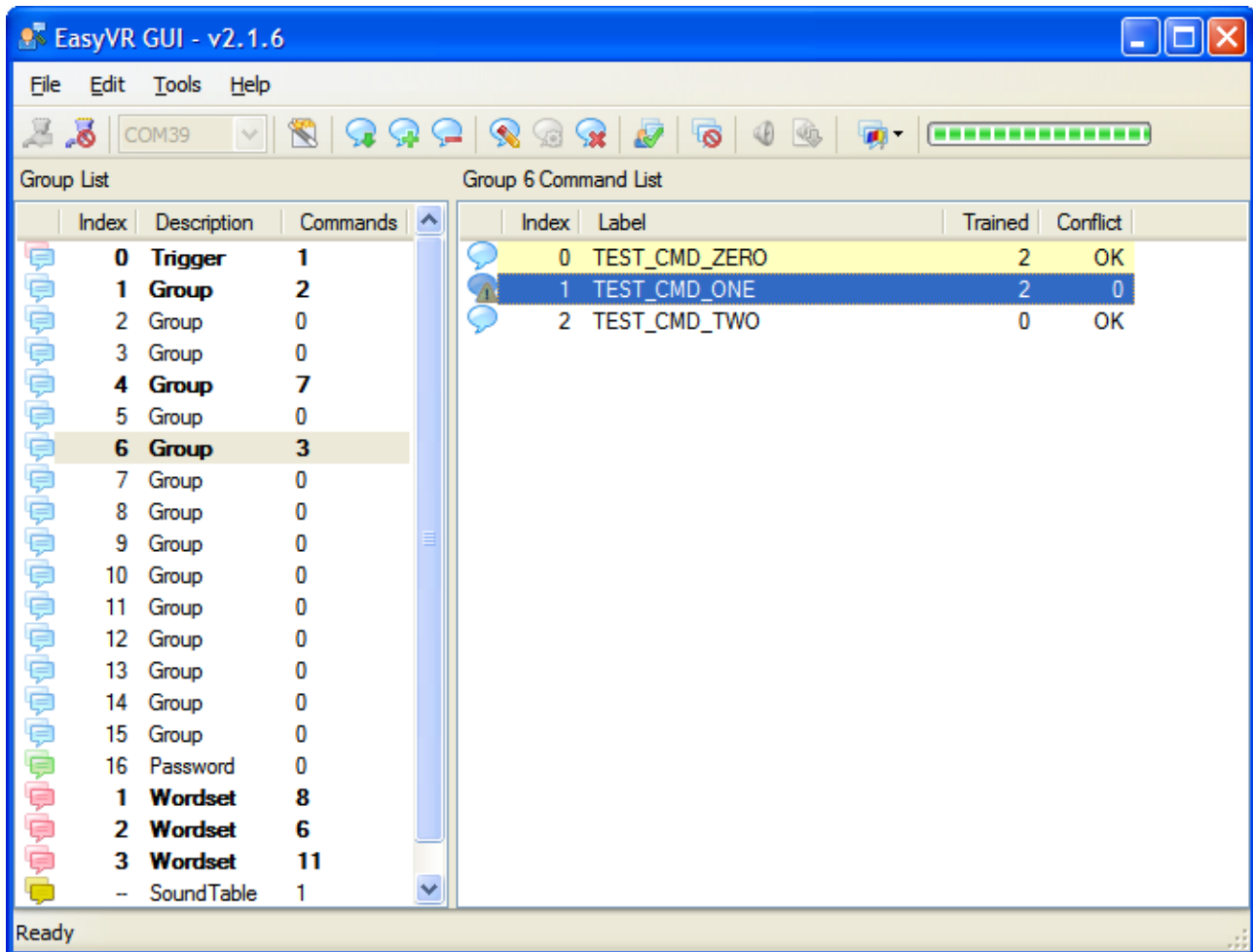
Figure 3 – Conflicting commands

The current status is displayed in the EasyVR Commander list view where groups that already contain commands are highlighted in bold.

The selected group of commands can also be tested, by using the icon on the toolbar or the *"Tools"* menu, to make sure the trained commands can be recognized successfully.

> **Note:** If you want to re-train a command you have to erase the previous training first.

> **Note:** "Vocal passwords"(group 16) are much more sensitive to environment noise and distance from the microphone: be sure to train and give the password in similar conditions

Download a new Sound Table

The EasyVR can also play one of the sounds or sentences saved on the internal flash memory. A predefined "beep" sound is also always available, even when no sounds have been downloaded to the module.

The custom sounds are organized in a so-called "*sound table*", that users can prepare and build with the special QuickSynthesis^{TM} tool. Please refer to this application's own manual for details about creation of a sound table.

Once the sound table has been created, it can be processed by the EasyVR Commander and downloaded to the module. The user must first disconnect from the module and do the steps required to start it in "boot-mode" (see the section FLASH UPDATE).

Now the command *"Update Sound Table"* is enabled, either on the toolbar or the *"Tools"* menu, and it can be used to start the update process.
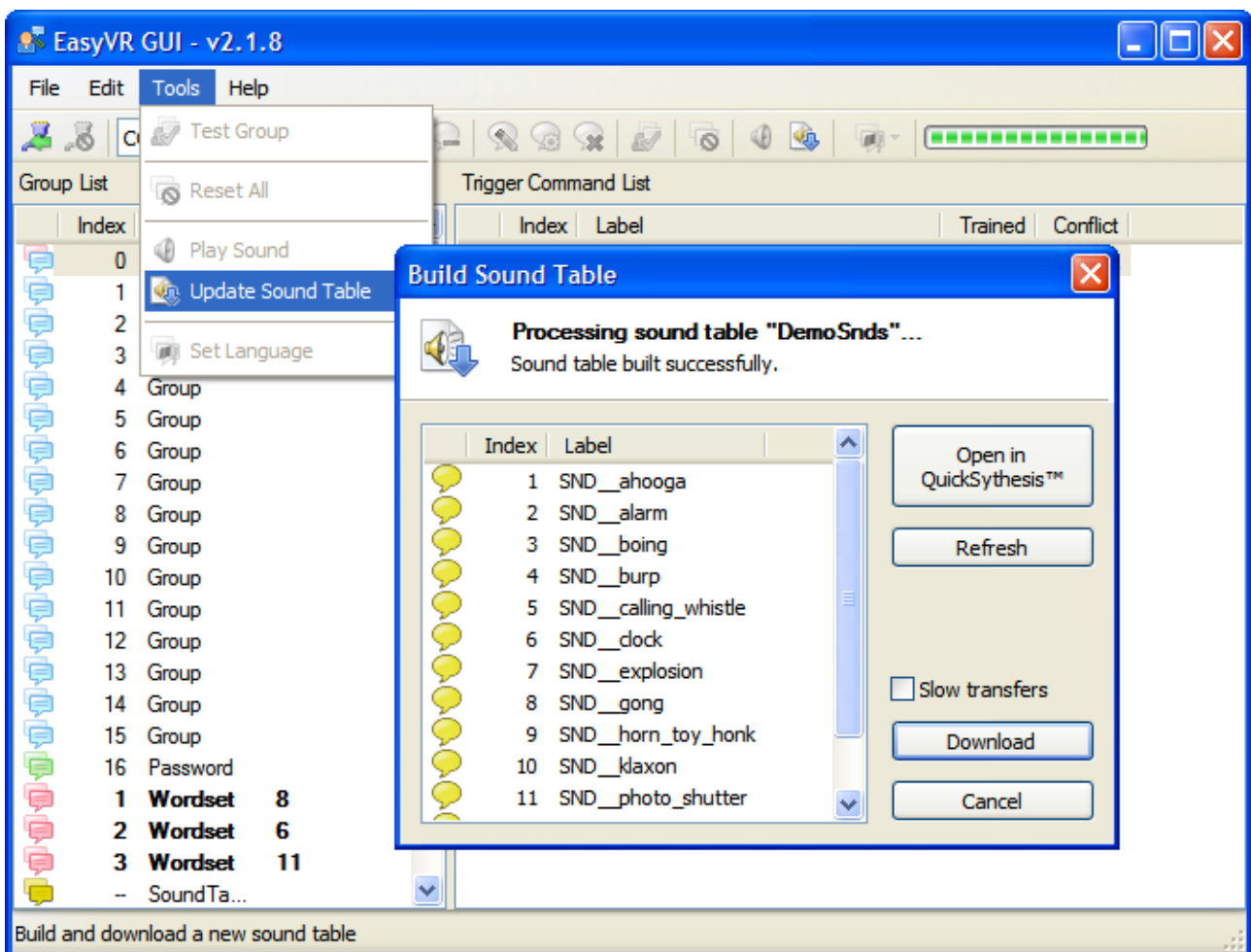

Figure 4

First the user will be prompted to open the QuickSynthesis project file just created and a new sound table will be generated.

The project must have been built already with the QuickSynthesis tool, before the sound table generation can be completed successfully. If a recent build is not available the user will receive a warning message, the project can be opened in QuickSynthesis again and a fresh build started (make sure the project file has been saved before the build).

Once back in the EasyVR Commander the project can be reloaded by pressing the "Refesh" button. If the

process completes successfully, the *"Download"* button will be enabled and the flash update process can start.

The download process will connect at a higher speed to the EasyVR module, so the "bridge" program running on your host device might not work (in particular Robonova and Basic Stamp cannot be used for this purpose) and you might need a true "serial adapter".

The full speed used is 230400 bit/s, but the option "Slow transfer" can be used to reduce it to 115200. One adapter that can go to full speed is the VoiceGP DevBoard. Otherwise any USB/Serial adapter with TTL/CMOS interface can be used for updating the flash.

After the download completes, a new connection can be established with the EasyVR module (in "normal-mode") and the new sounds will be displayed by the EasyVR Commander, in the special group *"SoundTable"* (the last one in the list with a yellow icon).

They can be played back and tested using the "*Play Sound*" command on the toolbar or in the *"Tools"* menu.

See also how to do that in your application in the code example USE CUSTOM SOUND PLAYBACK.

How to get support

Please feel free to contact us with any questions, queries or suggestions.
If your question is about technical support or troubleshooting for one of our products, we kindly ask you to first check our Forum for a possible solution: http://www.veear.eu

If you cannot find an existing solution on the forum, we strongly recommend posting your support request on the forum for as quick a response as possible. The more detail you provide, the better support we can give.

VeeaR © TIGAL KG, all right reserved.

All VeeaR branded boards and software are designed and manufactured by RoboTech srl