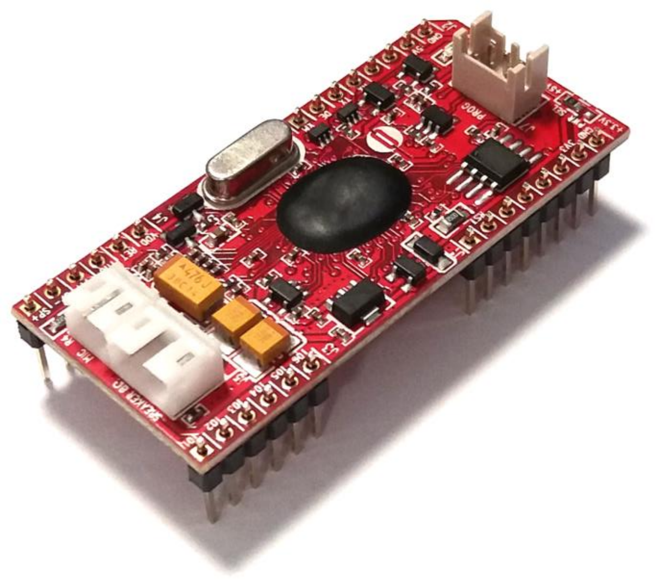
EasyVR 3 (Plus)

User Manual

Release 1.0.17



www.veear.eu

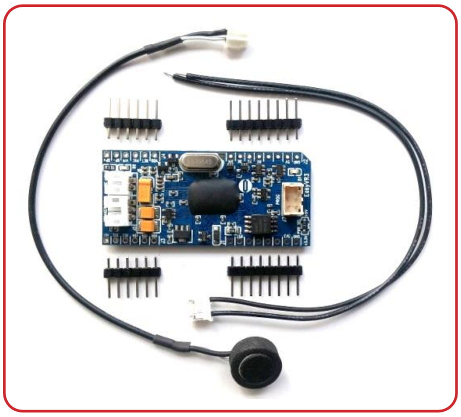
Table of Contents

Document History Information

Revision	Date	Description
1.0	2015/01/27	• Initial draft
1.0.3	2015/02/09	• New drawings and updated descriptions
1.0.4	2015/03/19	• Added new pictures and minor updates
1.0.5	2015/03/25	• Updated pictures and quick-start sections
1.0.6	2015/03/30	• Added programming and library chapters • Added PC software description • Updated pictures and layout
1.0.7	2015/03/31	• Minor corrections
1.0.8	2015/04/01	• Updated custom data screenshots and description
1.0.9	2015/04/02	• Added chapter for QuickUSB adapter
1.0.10	2015/04/22	• Updated mechanical drawing of module
1.0.11	2015/06/05	• Added note about soldering headers • Removed old logo from drawings
1.0.12	2015/07/02	• Updated QuickT2SI screenshots • Added notes about message recording functions
1.0.13	2016/03/10	• Added J7 pin-out numbering on module picture
1.0.14	2016/08/02	• Fixed protocol description • Added missing protocol elements • Updated Arduino library documentation • Updated EasyVR Commander with new interface elements • Updated product features
1.0.15	2017/03/20	• Added configuration table for EasyVR Shield operating modes • Added advanced topic for remote connections • Updated quick start guides • Updated products pictures
1.0.16	2018/04/05	• Updated EasyVR features to latest firmware revision • Added firmware upgrade notice
1.0.17	2019/01/28	• Updated documentation for EasyVR 3+

EasyVR 3 Module

Product Description



EasyVR 3 is a multi-purpose speech recognition module designed to easily add versatile, robust and cost effective speech recognition capabilities to almost any application.

The EasyVR 3 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, curtains or kitchen appliances, or adding “hearing” to the most popular robots on the market.

It can be easily plugged into a solder-less breadboard or standard prototyping board, and it is compatible with the mikroBUS™ specifications (see www.mikroe.com/mikrobus).

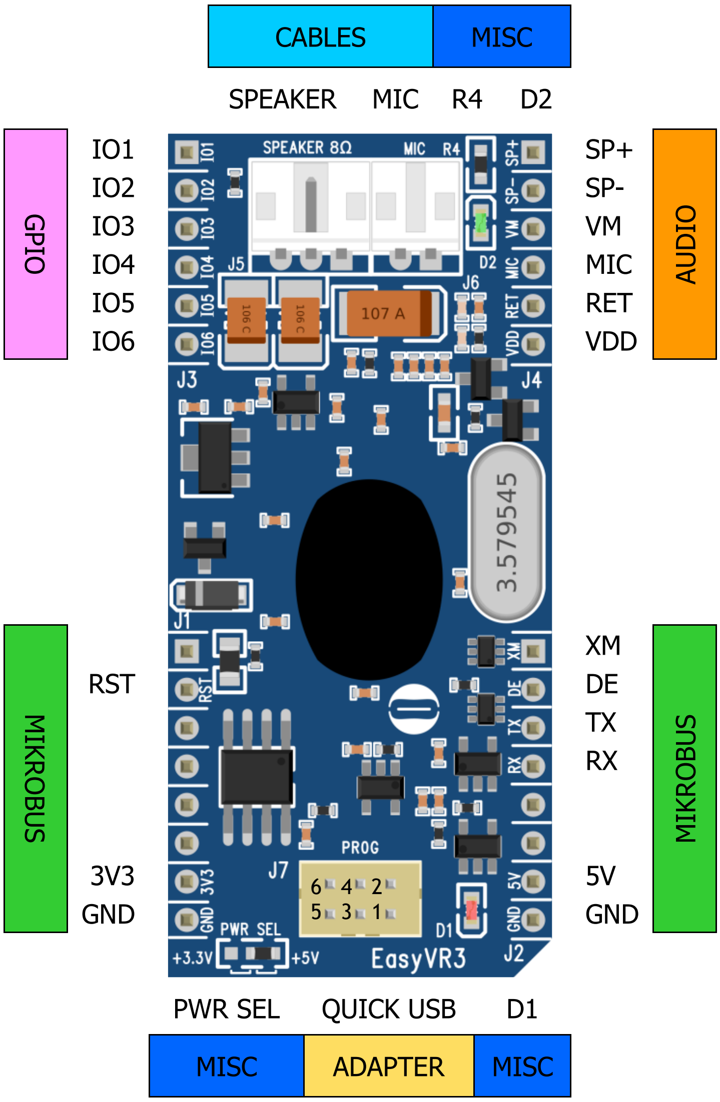
Separate male headers are provided inside the package, along with a microphone cable assembly and speaker wires (loudspeaker not included).

EasyVR 3 Features

- Up to 28 custom Speaker Independent (SI) command vocabularies[1].
  Supported Languages:
    - US English
    - British English
    - French
    - German
    - Italian
    - Japanese
    - Korean
    - Mandarin
    - Spanish
- Up to 64 (256 on EasyVR 3+) user-defined Speaker Dependent (SD) or Speaker Verification (SV) commands, that can be trained in ANY language, divided into maximum 16 groups (up to 32 SD or 5 SV commands each).
- A selection of built-in Speaker Independent (SI) commands for ready-to-run basic controls, in the following languages:
    - English (US)
    - Italian
    - German
    - French
    - Spanish
    - Japanese
- SonicNet™ technology for wireless communications between modules or any other sound source (Audio CD, DVD, MP3 Player).
- Up to around 21 minutes of pre-recorded sounds or speech[2].
- Up to about 120 (137 on EasyVR 3+) seconds of live message recording and playback.
- Real-time Lip-sync capability.
- DTMF tone generation.
- Differential audio output that directly supports 8Ω speakers.
- Easy-to-use Graphical User Interface to program Voice Commands and audio.
- Standard UART interface (powered at 3.3V - 5V).
- Simple and robust documented serial protocol to access and program through the host board.
- 6 General purpose I/O lines that can be controlled via UART commands.

[1] A *QuickT2SI™ Lite* license (sold separately) is required to enable creation of Speaker Independent vocabularies (maximum 12 commands per set). No license required to use SI grammars.
[2] At the maximum compression rate.

Technical specifications



The outer headers J1 and J2 are the mikroBUS™ interface connectors, providing selectable 3.3V/5V power input to the module and voltage translated digital I/O lines, including: UART receive/transmit lines and control pins.

The header J3 provides configurable I/O expansion lines (inputs with weak internal pull-up by default), powered at the internal logic voltage VDD.

The header J4 contains the main analog signals, such as microphone signals and amplified DAC outputs, which are also available on the internal right angle connectors J5 and J6.

The module can also be operated through the programming connector J7 alone, by using the *QuickUSB Adapter Cable*.

Pin assignment

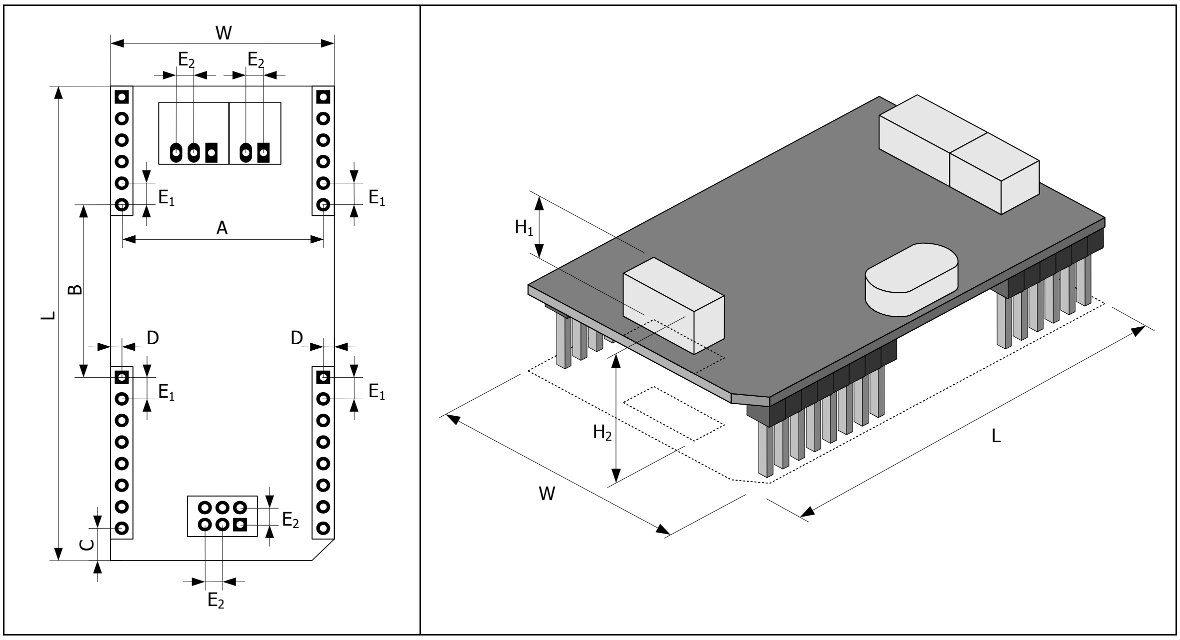
Group	Name	Number	Pin	Type	Description
MIKROBUS	J1	1	-	-	(Not connected)
		2	RST	I	Active low asynchronous reset (internal pull-up)
		3-6	-	-	(Not connected)
		7	3V3	I	3.3V DC power input
		8	GND	-	Ground
	J2	1	XM	I	Boot select (internal pull-down)
		2	DE	O	(Reserved)
		3	TX	O	Serial Data Transmit
		4	RX	I	Serial Data Receive
		5-6	-	-	(Not connected)
		7	5V	I	5.0V DC power input
		8	GND	-	Ground
GPIO	J3	1	IO1	I/O	General purpose I/O (VDD logic levels)
		2	IO2	I/O	General purpose I/O (VDD logic levels)
		3	IO3	I/O	General purpose I/O (VDD logic levels)
		4	IO4	I/O	General purpose I/O (VDD logic levels)
		5	IO5	I/O	General purpose I/O (VDD logic levels)
		6	IO6	I/O	General purpose I/O (VDD logic levels)
AUDIO	J4	1	SP+	O	Differential audio output (can directly drive 8Ω speaker)
		2	SP-	O	
		3	VM	O	Microphone power (to support custom microphones)
		4	MIC	I	Microphone audio input
		5	RET	-	Microphone return (analog ground)
		6	VDD	O	Internal logic voltage (for reference only)
CABLES	J5	1	SP-	O	Differential audio output (can directly drive 8Ω speaker)
		3	SP+	O	
		2	-	-	(Not connected)
	J6	1	MIC	I	Microphone audio input
		2	RET	-	Microphone return (analog ground)
ADAPTER	J7	1	RX_P	O	Programming cable serial data receive
		2	RTS_P	I	Programming cable request to send (reset/boot control)
		3	GND	-	Programming cable ground
		4	5V_P	I	Programming cable 5V DC power output
		5	TX_P	I	Programming cable serial data transmit
		6	CTS_P	O	Programming cable clear to send (tied to ground)

Note: The General Purpose I/O lines (J3.1-6) are at nominal 3.0VDC level. Do not connect higher voltages directly to these pins!

Settings and indicators

Group	Name	Type	Description
MISC	**PWR SEL**	3-Way Jumper (SMD 0603)	Select power input and voltage level between +3.3V and +5V with a zero Ohm resistor or solder bridge
	D1	LED	Red light indicator, normally ON when the board is powered, briefly blinking on serial data received
	D2	LED	Green light indicator, turns ON when the module is listening to its audio input
	R4	Resistor (SMD 0603)	Microphone gain resistor, default is 1.2kΩ

Physical dimensions



Symbol	Parameter	Units (mm / Inches)	
W	Width	25.4	1.000
L	Length	56.4	2.220
H_1	Height (without outer strips J1-J4)	9.5	0.375
H_2	Height (with outer strips J1-J4)	17.0	0.670
E_1	Connector pitch and pin spacing (of outer strips J1-J4)	2.54	0.100
E_2	Connector pitch (of inner connectors J5-J7)	2.00	0.079
A	Headers horizontal spacing	22.86	0.900
B	Headers vertical spacing	20.32	0.800
C	Header vertical offset	3.81	0.150
D	Header horizontal offset	1.27	0.050

Recommended Operating Conditions

Symbol	Parameter	Min	Typ	Max	Unit
5V	DC Power Input (Host) = V_{SEL}	3.15	5.0	5.5	V
3V3		3.15	3.3	5.5	V
5V_P	DC Power Input (Programming cable)	4.0	5.0	5.5	V
Ta	Ambient Operating Temperature Range	0	25	70	°C

Power Supply Requirements

Symbol	Parameter	Min	Typ	Max	Unit
I_{SLEEP}	Sleep current (V_{SEL} = 5.0V)		6		mA
I_{OPER}	Operating current (V_{SEL} = 5.0V)		25	35	mA
I_{AUDIO}	Audio playback current (with 8Ω speaker)		175	250	mA (RMS)
I_{TOT}	Total current consumption (excluding I/O)		25	285	mA (RMS)
I_{PEAK}	Peak supply current (excluding I/O)		400		mA

Electrical Characteristics

These are applicable to pins RX, TX_P.

Symbol	Parameter	Min	Typ	Max	Unit
V_{IH}	Input High Voltage	2.1		5.5	V
V_{IL}	Input Low Voltage	0.0		0.9	V
I_{IL}	Input Leakage Current ($0 < V_I < 5.5V$)		-65		μA

These are applicable to pins TX, DE.

Symbol	Parameter	Min	Typ	Max	Unit
V_{OH}	Output High Voltage (I_{OH} = -0.3 mA, V_{SEL} = 3.3V)	2.6		3.3	V
	Output High Voltage (I_{OH} = -0.3 mA, V_{SEL} = 5.0V)	4.3		5.0	V
V_{OL}	Output Low Voltage (I_{OL} = 5 mA)	0.0		0.2	V

These are applicable to pin XM.

Symbol	Parameter	Min	Typ	Max	Unit
V_{IH}	Input High Voltage	1.4	(0.8)	5.5	V
V_{IL}	Input Low Voltage	0.0	(0.7)	0.5	V
I_{IN}	Input Current ($0 < V_I < 3.3V$)	0	0.2	0.4	mA
	Input Current ($0 < V_I < 5.5V$)	0	0.5	0.7	mA

These are applicable to pin RST.

Symbol	Parameter	Min	Typ	Max	Unit
V_{IH}	Input High Voltage	2.1		5.5	V
V_{IL}	Input Low Voltage	0.0		0.6	V
I_{IL}	Input Leakage Current ($0 < V_I < 5.5V$)		-85		μA

These are applicable to pin RX_P.

Symbol	Parameter	Min	Typ	Max	Unit
V_{OH}	Output High Voltage (I_{OH} = -5 mA)	2.4		3.0	V
V_{OL}	Output Low Voltage (I_{OL} = 8 mA)	0.0		0.6	V

These are applicable to pins IO1 – IO6.

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_I < 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		3.0	V
V_{OL}	Output Low Voltage (I_{OL} = 8 mA)		0.0		0.6	V

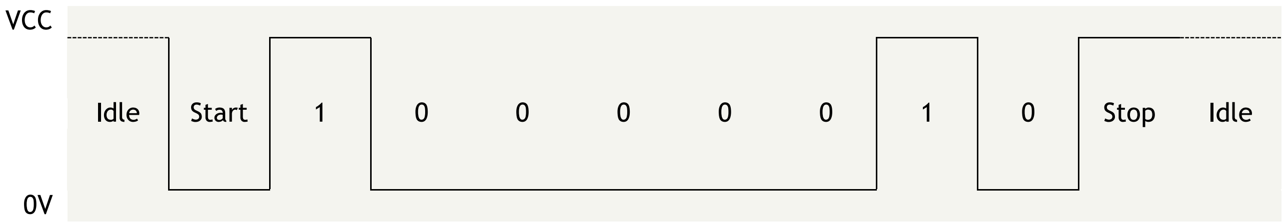
Serial Interface

The EasyVR 3 communicates 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 RX, while the transmitter output data line is TX. 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

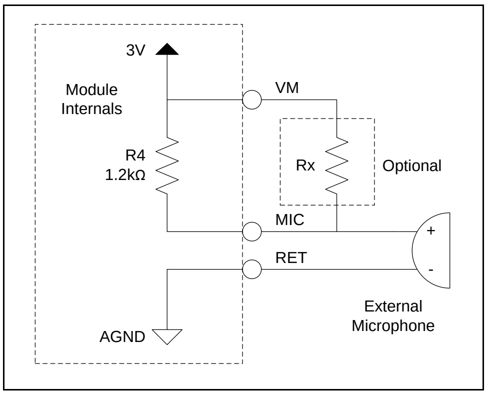
The microphone provided with the EasyVR 3 module is an omnidirectional electret condenser microphone (Horn EM9745P-382):

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

The microphone circuit is optimized for use at ARMS_LENGTH (default, about 60cm) or FAR_MIC distance settings.

If you use a microphone with different specifications the recognition accuracy may be adversely affected. Differences in rated load impedance and sensitivity can be compensated to a certain extent by changing the microphone gain. This can be done in several ways:

- Replacing the internal gain resistor R4 (1.2kΩ)
- Adding an external resistor Rx going in parallel with R4 (it can only reduce gain, useful for HEADSET distance settings)
- Removing the internal resistor R4 and using only the external resistor Rx



Microphone circuit

Modifying gain resistance

You can calculate the overall microphone gain resistance using the formula below:

$$R_s = I \times 10^{\frac{G-S}{20}}$$

Rs is the optimal microphone gain resistance
I is the impedance rating of the microphone
G is the desired overall system gain, defined as follows:

1. If the module is configured for HEADSET microphone distance (typically a few centimeters from the user's mouth), then the overall system gain should be -49 dB (0dB=1v/Pa@1KHz);
2. If the module is configured for ARMS_LENGTH microphone distance (typically 60-90 cm from the user's mouth – this is the default setting of EasyVR), then the overall system gain should be -44 dB;
3. If the module is configured for FAR_MIC microphone distance (up to about 3 meters from the user's mouth), then the overall system gain should be -43 dB.

S is the sensitivity rating of the microphone you want to use, and it is specified in –dB in the microphone's specification[3].

[3] Converting µBars to Pascal: microphone manufacturers specify the sensitivity referencing to µBars or Pascal. If the microphone sensitivity is referenced to µBars, simply add 20 dB to the rating. For example, -58 dB/µBars + 20dB = -38 dBV/Pa.

Examples

1) The optimal gain resistance for the bundled microphone at ARMS_LENGTH distance is:

$$R_s = 2200 \times 10^{\frac{-44-(-38)}{20}} = 1103$$

Use the closest standard 5% resistor to Rs. In this example, it would be 1.1 kΩ. The EasyVR uses a 1.2 kΩ resistor to allow use of “FAR” settings without replacing the internal resistor.
Sometimes you might also need to compensate some gain loss for a voltage lower than the microphone ratings (using a larger resistor value sets a higher input gain).

2) The gain resistance for the bundled microphone at HEADSET distance would be:

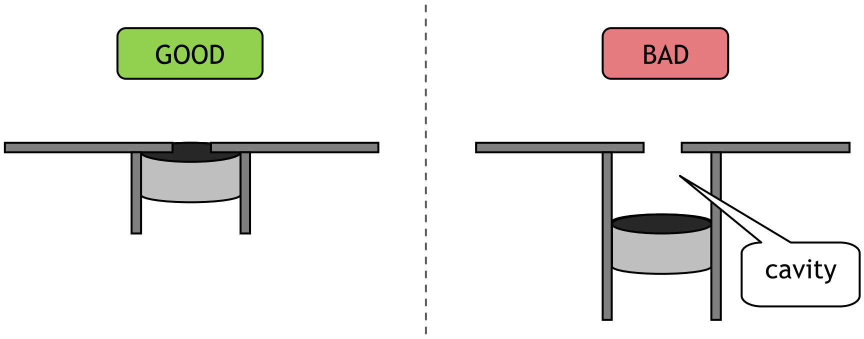
$$R_s = 2200 \times 10^{\frac{-49-(-38)}{20}} = 620$$

In this case you may just add an external 1.2 kΩ resistor to get a gain resistance of 600 Ω (close enough).

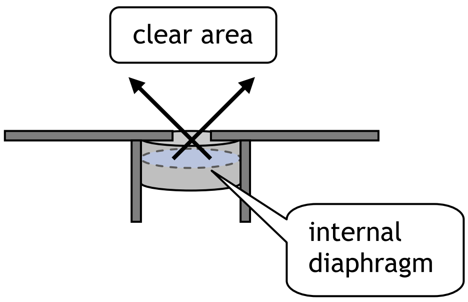
Positioning guidelines

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 airspace can lead to acoustic resonance, which can reduce recognition accuracy.

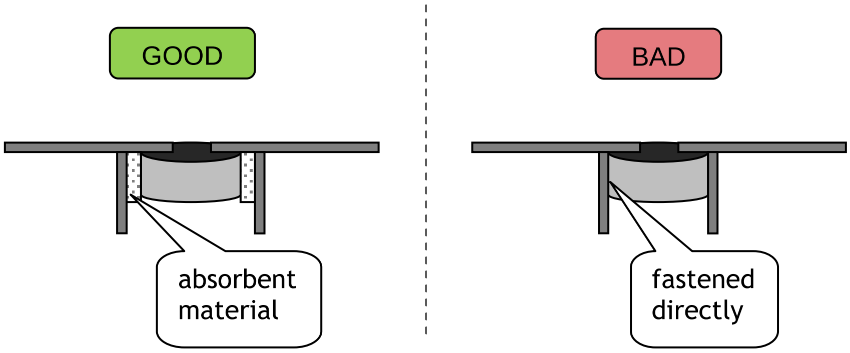


2. **No Obstructions, Large Hole** - The area in front of the microphone element must be kept clear of obstructions to avoid interference with recognition. The diameter of the hole in the housing in front of the microphone should be at least 5 mm. Any necessary plastic surface in front of the microphone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



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.

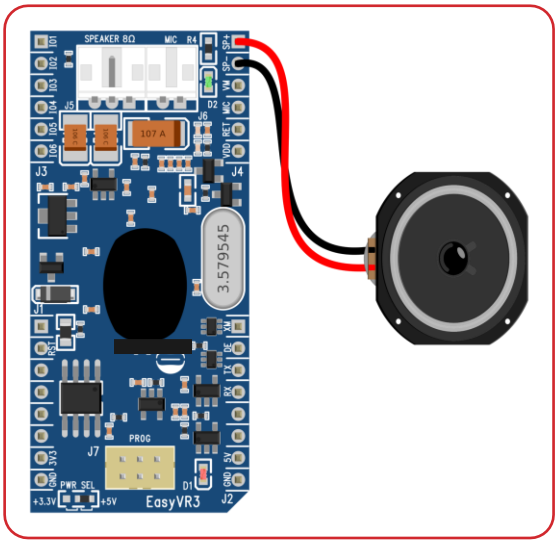


4. **Distance** - If the microphone is moved from 15 cm to 30 cm from the speaker's mouth, the signal power decreases by a factor of four. The difference between a loud and a soft voice can also be more than a factor of four. Although the internal preamplifier of the EasyVR compensates for a wide dynamic 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 3 audio output interface is capable of directly driving an 8Ω speaker. It can also be connected to an external audio amplifier to drive lower impedance loudspeakers.

Note: Connecting speakers with 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 sensitivity and the desired output volume (usually in the order of 1-10kΩ).

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

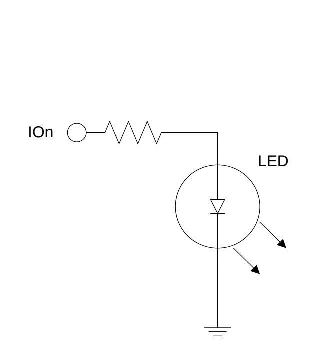
General Purpose I/O

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

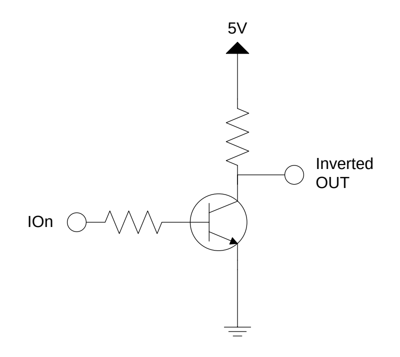
The six I/O pins IO1–IO6 are connected directly to the embedded microcontroller on the EasyVR module, so they are referenced to the internal 3.0V regulated power supply VDD. If you need to interface to circuits using a different supply, there are a number of solutions you can adopt. Some of these are outlined below (here IOn indicates any one of the six I/O pins of the EasyVR).

Use a pin as an output

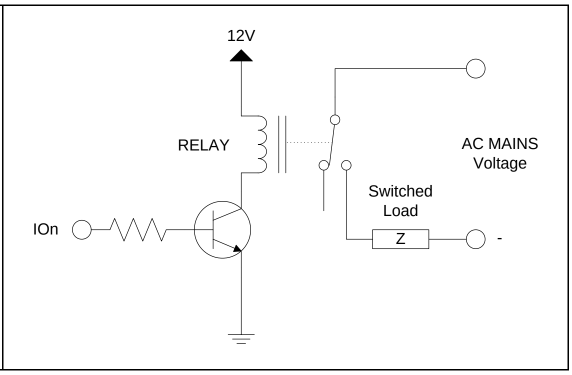
All the I/O pins are inputs with weak internal pull-up after power on. You must explicitly configure a pin before you can use it as an output (see the example code *Use general purpose I/O pins*).



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:

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

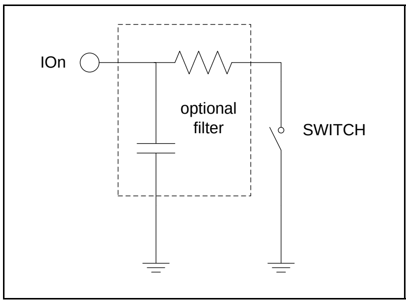
Where V_{LED} is the LED forward voltage, as reported on the LED datasheet, at the driving current I_{OH} (see section *Electrical Characteristics*). Let's assume a typical low-current LED has a V_F=1.8V at 5mA, the resistor value is:

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

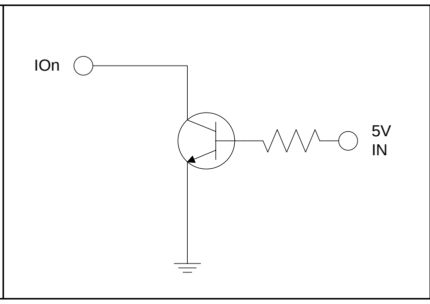
Now stay on the safe side and choose a slightly larger resistor, such as 150Ω.
If you want to drive higher current LEDs, you need a circuit like the second one, where you put the LED between the output resistor and the collector of the NPN transistor.

Use a pin as an input

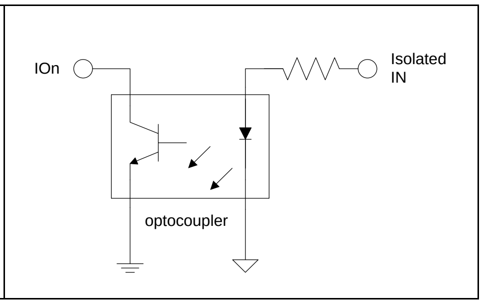
All the I/O pins are inputs with weak internal pull-up after power on or reset. You may also configure the pin to have a strong pull-up or no pull-up at all (see the example code *Use general purpose I/O pins*).



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.

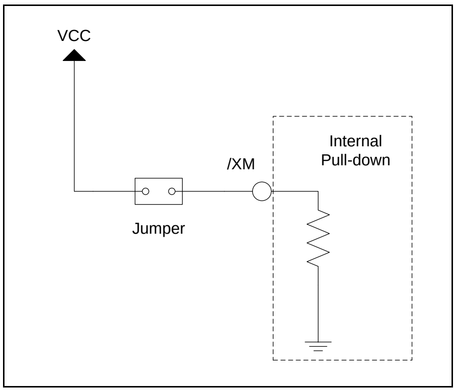
Again, you should refer to the manufacturer's datasheet when interfacing any external components and to calculate required resistors values or other passive components.

Flash Update

The EasyVR module includes a boot loader that allows to update the firmware and to download new sound tables or custom grammars 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 (or switch) taking the signal to a suitable pull-up resistor.

To download a firmware update, a sound table or a custom grammar 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).

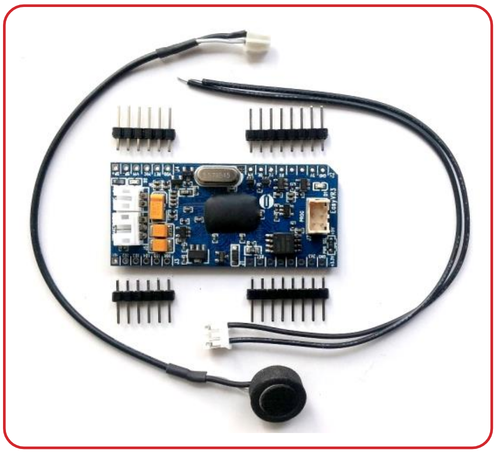


Boot mode selection circuit

To learn how to download new sound tables or custom grammars to your EasyVR 3 module, have a look at the section *Using Custom Data*.

Quick start guide for using the module

Assembly notes



The EasyVR 3 is provided with separate standard 2.54mm-pitch male headers that can be used to connect the module to a breadboard, prototyping board, custom boards or carrier boards like the EasyVR Shield 3.

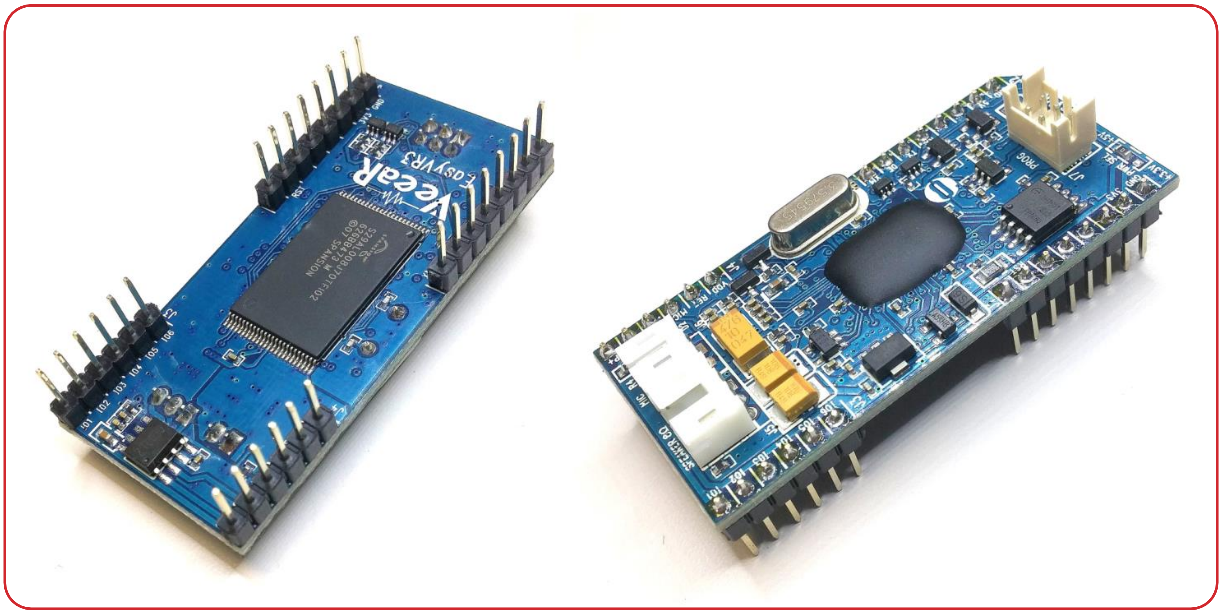
When male headers are necessary, make sure they are well soldered on the module in order to prevent electrical issues.

Some practical guides to hand soldering:

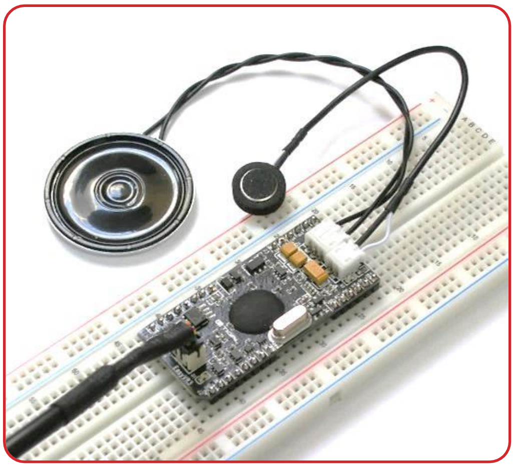
- Adafruit Guide To Excellent Soldering
- Sparkfun How to Solder: Through-Hole Soldering

When assembling the module, especially in preparation for the EasyVR Shield 3, make sure you insert male headers from the bottom side and solder them on the top side, and that all the headers are straight and vertical for a smooth insertion on the carrier board.

The end result should be similar to the following pictures:



EasyVR 3 as a Development Board



The EasyVR 3 module has been designed to allow use as a standalone development board when combined with a USB-Serial adapter.

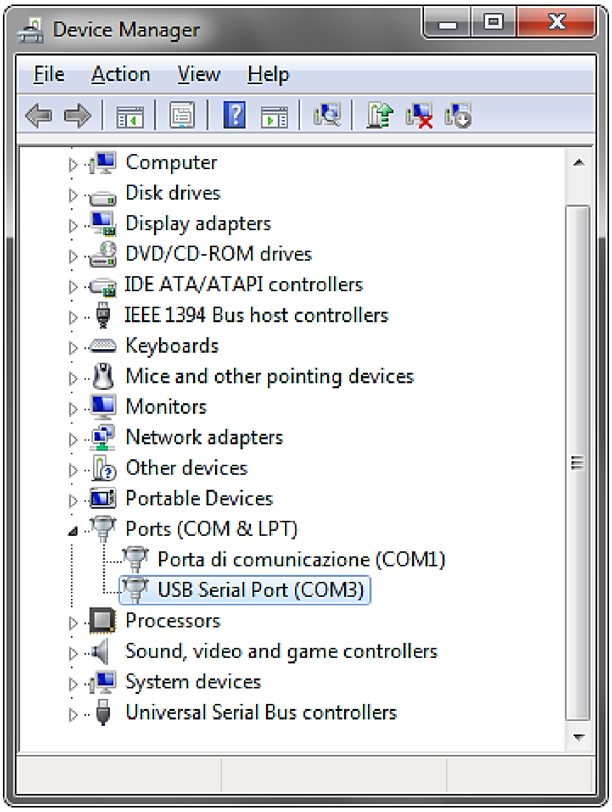
The QuickUSB adapter cable can be used to program voice commands and sound outputs directly to an EasyVR 3 module and quickly test its functions from your PC.

Just connect the microphone and an 8Ω speaker to the module, plug-in the adapter cable and you are ready to go.

The EasyVR 3 boot mode is managed automatically through the serial handshake lines and you don't need to set any jumper.

Getting started

1. Connect the microphone to the 2-way socket MIC (J6)
2. Connect an 8Ω speaker to the 3-way socket SPEAKER (J5)
3. Connect a QuickUSB cable to the 3x2 pins socket (J7)
4. Plug the USB end of the adapter cable to your PC.
   The first time it may take some time to install the required drivers (see Software Setup)
5. If your installation is successful you will see a new virtual COM port in your Device Manager:



(The actual COM port number may vary)

6. Now start the EasyVR Commander software
7. Choose your COM Port and click connect
8. Then enjoy your EasyVR!

Serial Adapter Interface

Connector J7 is a 6-pin socket specifically designed for the QuickUSB serial adapter cable, but another adapter may also be connected to this port, as long as it uses the same connector type, pin assignment and electrical specifications.

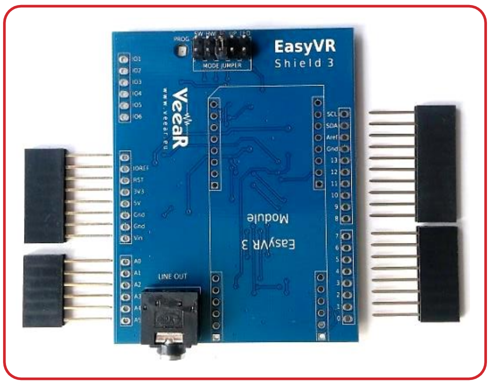
Pin	Name	Type[4]	Notes
1	RX_P	I	Adapter should have TTL/LVTTL compatible inputs (V_{IH} = 2.0V)
2	RTS_P	O	Adapter outputs can have 3.3V or 5V levels RTS handshake is required for automatic reset and boot mode control
3	GND	-	Common ground
4	5V_P	O	Adapter should provide a 5V DC power output for the module (see *Recommended Operating Conditions* and *Power Supply Requirements*)
5	TX_P	O	Adapter outputs can have 3.3V or 5V levels
6	CTS_P	I	CTS is tied to GND on the module

Connector type is Hirose DF11 Series (female on the adapter cable, male on the module).

[4] Please note that pin direction here is referring to the external adapter

EasyVR Shield 3 for Arduino

Product description



The EasyVR Shield 3 is an adapter board for the EasyVR 3 module, designed to simplify its use among the Arduino community.

The Shield is compatible with any Arduino board using UNO-R3 Shield headers, running at either 3.3V or 5V levels, by using the IOREF pin to select the EasyVR operating voltage.

It is also backward compatible with earlier Arduino boards that don't have the IOREF pin, which are using 5V I/O levels by default.

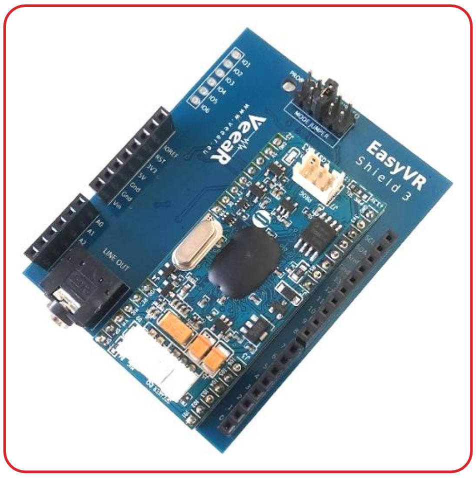
If your board does not have the IOREF pin but it is running at 3.3V, you can still operate the EasyVR Shield 3 correctly if you manually connect pins IOREF and 3V3 together, for example with a jumper wire.

The board comes with separate Arduino stackable headers for the Shield interface. The EasyVR 3 module is also provided separately.

Note: The EasyVR 3 module and all stackable headers must be soldered before use!

EasyVR Shield 3 Features

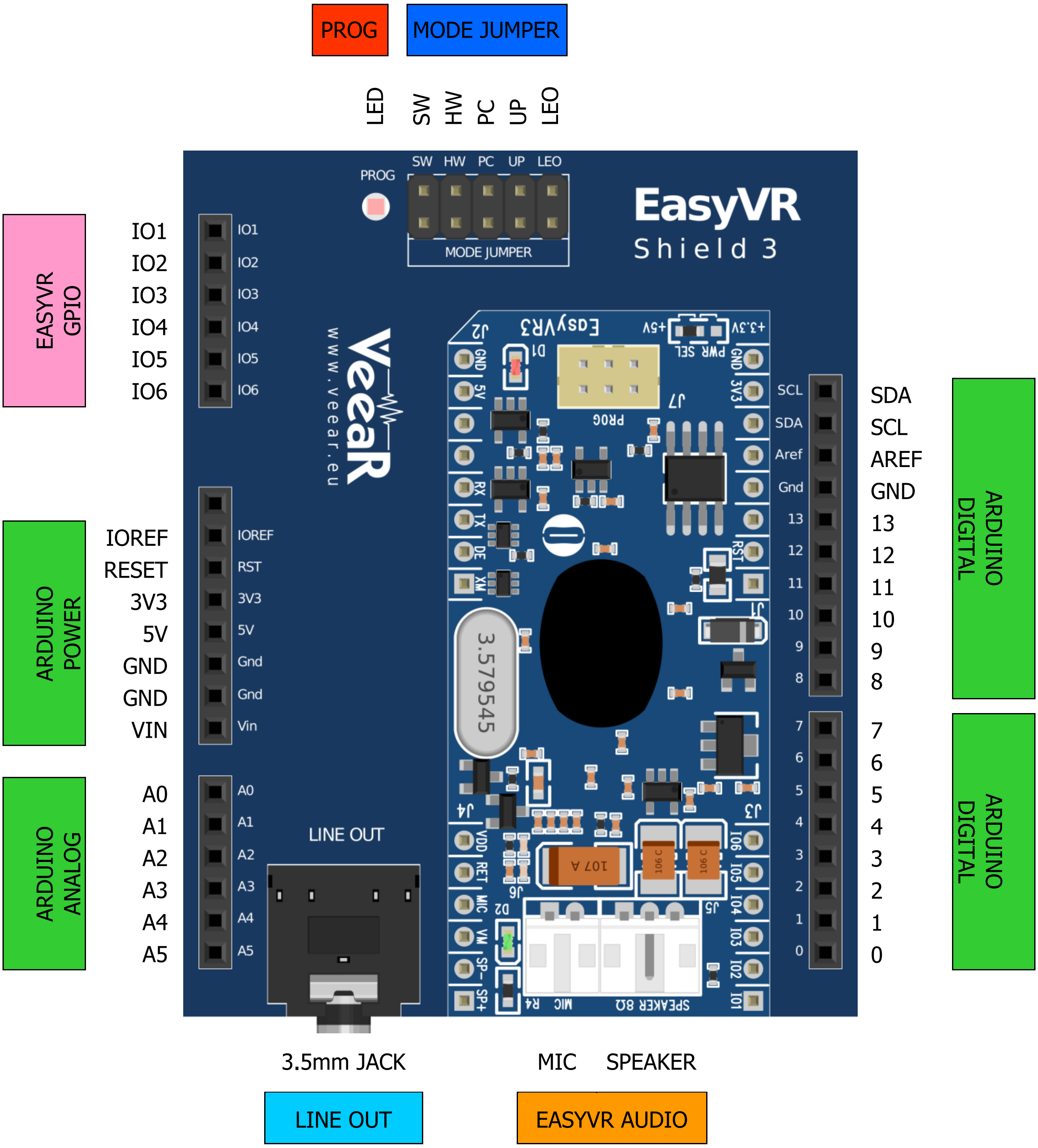
- Compatible with Arduino boards that have the 1.0 Shield interface (UNO R3) including, but not limited to:
    - Arduino Zero
    - Arduino Uno
    - Arduino Mega
    - Arduino Leonardo
    - Arduino Due
- Supports 5V and 3.3V main boards through the IOREF pin (defaults to 5V if this pin is absent)
- Supports direct connection to the PC on main boards with a separate USB/Serial chip and a special software-driven "bridge mode" on boards with only native USB interface, for easy access and configuration with the EasyVR Commander
- Enables different modes of serial connection and also flash updates to the embedded EasyVR module (through the Mode Jumper)
- Supports remapping of serial pins used by the Shield (in SW mode)
- Provides a 3.5mm audio output jack suitable for headphones or as a line out



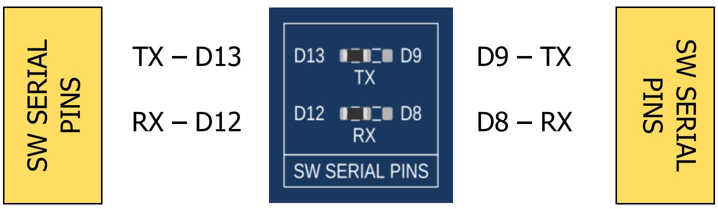
EasyVR Shield 3 fully assembled

Technical specifications

Board overview



(Top View)



(Detail - Bottom View)

Pin assignment

Group	Pin	Description
ARDUINO HEADERS	-	Arduino UNO-R3 Shield interface, pass-through connectors (Pins 0-1 are in use when J12 is set to UP, PC, HW or LEO) (Pins 12-13 or 8-9 are in use when J12 is set to SW)
EASYVR AUDIO	-	Audio cables connectors of the EasyVR 3 module (microphone and speaker)
LINE OUT	-	3.5mm stereo/mono jack (16Ω - 32Ω headphones or line-level output)
MODE JUMPER	**SW**	Arduino Software Serial (connected to pins 12-13 or 8-9)
	HW	Arduino Hardware Serial (connected to pins 0-1)
	PC	PC Mode (Arduino disabled, EasyVR in command mode)
	UP	Update Mode (Arduino disabled, EasyVR in boot mode)
	LEO	Leonardo Update (Arduino enabled, EasyVR in boot mode)
PROG	-	Red light indicator for Flash programming modes (UP and LEO)
SW SERIAL PINS	**RX**	Use resistor to select Software Serial RX pin: 12 or 8
	TX	Use resistor to select Software Serial TX pin: 13 or 9
EASYVR GPIO	**IO1**	General purpose I/O as found on the embedded EasyVR 3 module (referenced at the internal VDD logic level - see note below)
	IO2	
	IO3	
	IO4	
	IO5	
	IO6	

Note: The General Purpose I/O lines (IO1-IO6) are at nominal 3.0VDC level. Do not connect higher voltages directly to these pins!

Mode Jumper settings

This jumper selects the operating mode of the EasyVR Shield and it can be placed in one of four positions:

- **SW – Software Serial mode**
  Use it for controlling the EasyVR module from your Arduino sketch through a software serial port (using pins 12-13). You can also connect the EasyVR Commander in this mode, provided that the running sketch implements bridge mode (see the Arduino library examples).
- **HW – Hardware Serial mode**
  Use it for controlling the EasyVR module from your Arduino sketch through the hardware serial port (using pins 0-1).
- **PC – PC Connection mode**
  Use it for direct connection with the EasyVR Commander. In this mode, the Arduino controller is held in reset and only the embedded USB/Serial adapter is used.
- **UP – Flash Update mode**
  Use it for firmware updates or to download sound table data and custom grammars to the on-board flash memory from the EasyVR Commander. In this mode, the Arduino controller is held in reset and only the embedded USB/Serial adapter is used. The EasyVR module is set in boot mode.
- **LEO – Leonardo Update mode**
  This is similar to the regular *Flash Update* mode, for Arduino boards that don't have a separate USB/Serial adapter, such as Arduino Leonardo. The EasyVR module is set in boot mode, but the Arduino controller is not reset and it must be running the special "bridge" sketch.

Software Serial Pins settings

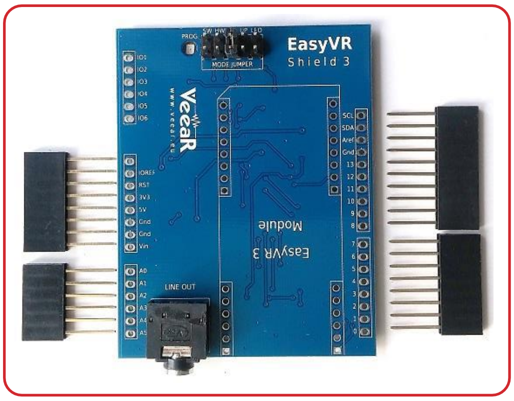
On the bottom side of the board there are two SMD resistors that you can move to select the two pins of Arduino that the EasyVR will be connected to when in Software Serial mode (*Mode Jumper* on **SW**).

- **RX – Software Serial Receiver pin**
  - D12 – Use digital pin 12 as serial receiver (default)
  - D8 – Use digital pin 8 as serial receiver
- **TX – Software Serial Transmitter pin**
  - D13 – Use digital pin 13 as serial transmitter (default)
  - D9 – Use digital pin 9 as serial transmitter

The choice of pins 12-13 is maintained for backward compatibility with the previous hardware revisions of the EasyVR Shield. However those pins may also be used for the SPI interface, so another choice of pins 8-9 is provided. If you want to use different pins make sure the receiver pin supports change interrupts.

Quick start guide for using the Shield

Assembly notes



The EasyVR Shield 3 is provided with separate through-hole headers to connect the Shield to Arduino boards and other microcontroller-based boards adopting an Arduino compatible interface.

The EasyVR 3 module is provided separately and it should be fully assembled before it is soldered on top of the Shield. Make sure they are both well soldered in order to prevent electrical issues (see highlighted areas in the picture below).

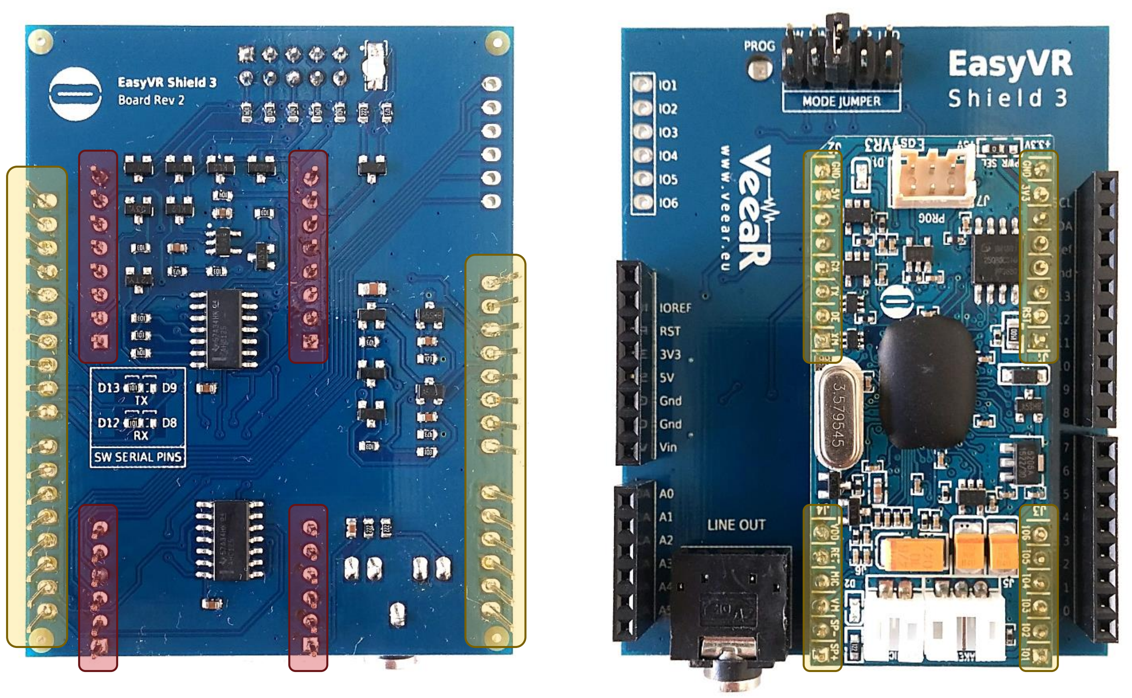
Some practical guides to hand soldering:

- Adafruit Guide To Excellent Soldering
- Sparkfun How to Solder: Through-Hole Soldering

Before you mount the EasyVR 3 module on the top, make sure you have assembled the pass-through headers on the EasyVR Shield 3 in the correct position and orientation. It is easier if you place all the connectors through the Shield with their plastic housings faced down on your desk, so that they can stand straight and vertical, and allow soldering of the metal leads easily from the back side of the board, which is now facing up.

As the last step, insert an already assembled EasyVR 3 module on its reserved footprint on the Shield, make sure it is plugged all the way in, and then complete soldering of male headers on the bottom side.

The end result should be similar to the following pictures:



Note: Pay attention to the tail of male headers (red areas) on the bottom of the EasyVR 3 module when you plug the Shield on a base board and make sure they are not touching other metal parts or connectors. If necessary you can trim the tails to prevent undesired contacts.

Prepare the software

1. Install the EasyVR Commander, available on the VeeaR website:
   http://www.veear.eu/downloads/
2. Install a recent version of the Arduino IDE, available from the official Arduino community:
   https://www.arduino.cc/en/main/software
3. Install the EasyVR Arduino libraries[5] on your PC.
   You can use the "Library Manager" in recent versions of the Arduino IDE to automatically download and install the latest version of the library.
   In alternative, to perform a manual install see:
   https://www.arduino.cc/en/Guide/Libraries

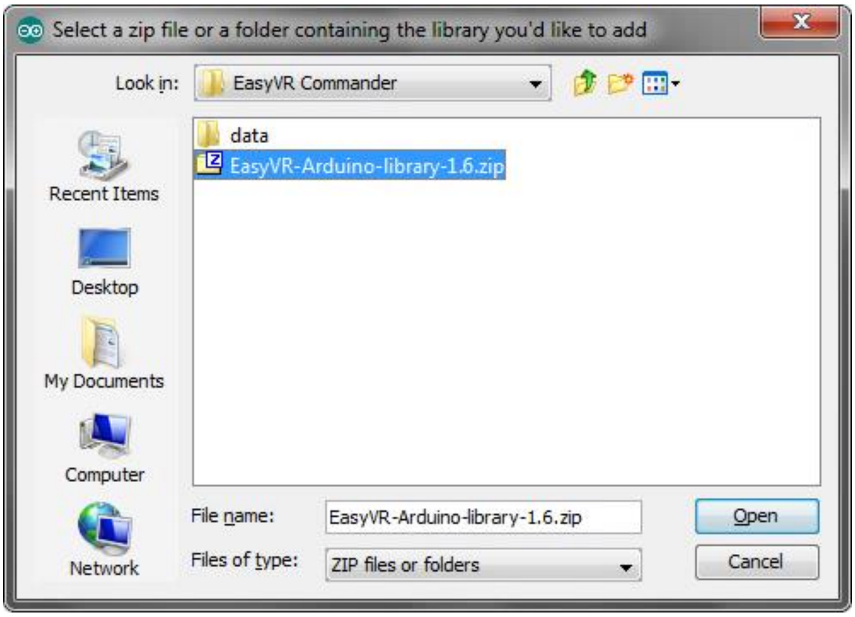


Figure 1 - Manual installation of Arduino library

Prepare the hardware

1. Insert the EasyVR Shield on top of your Arduino board
2. If you want audio output, either wire an 8Ω speaker into the *SPEAKER* connector (J5) on the EasyVR module or connect headphones or amplified speakers to the *LINE OUT* 3.5mm audio jack on the Shield
3. Connect the supplied microphone to the *MIC* connector (J6) on the EasyVR module
4. Make sure the *Mode Jumper* (J7) is in the correct position (see table below)
5. Connect a USB cable from your PC to the appropriate USB port for your Arduino board (see table below)

[5] The Arduino library archive file can also be found in the EasyVR Commander program folder.

Shield configuration table

Each Arduino board requires the correct combination of jumper settings and USB port, according to the operating mode and usage of the Shield.

The EasyVR Shield can be configured to support several operating modes using the *Mode Jumper* (J7) and the special "bridge mode" software provided by the EasyVR Arduino library and included in all the example sketches.

There are three main operating modes:

1. **Arduino Sketch**
   The Shield can connect the EasyVR module to either the first Hardware Serial port on pins 0/1 or a Software Serial port on pins 8/9 or 12/13 (default).
   The example sketches will auto-configure the library to use the default settings in the table below, but customization is possible within some constraints.
   The Mode Jumper can be in the HW or SW position (sketch running normally).
2. **EasyVR Commander**
   Some Arduino main boards have a second controller with a USB/Serial adapter that can be used for direct connection to the EasyVR module from the PC.
   Some boards have a native USB interface, or have a serial adapter which is not routed to the first Hardware Serial port. In those cases the "bridge" code running on the Arduino controller can soft-connect the EasyVR module to the PC.
   The Mode Jumper can be in the PC position (sketch not running) or the HW or SW position (sketch running the "bridge" software).
3. **Download / Update**
   This is similar to the above, except that the EasyVR module is configured to enter "boot mode" at power on or reset.
   The Mode Jumper can be in the UP position (sketch not running) or LEO position (sketch running the "bridge" software).

Each board should also be connected to the correct on-board USB port (when more are available), which is used by the example sketches to provide feedback on the "Monitor Window" and to run the "bridge" software for the EasyVR Commander.

Supported combinations are detailed in the table below:

	Arduino Sketch		EasyVR Commander		Download / Update	
Main Board	**Mode Jumper**	**USB Port**	**Mode Jumper**	**USB Port**	**Mode Jumper**	**USB Port**
Arduino Zero / M0 Pro	HW	Programming	HW	Programming	LEO	Programming
Arduino Uno / Mega / 2009	SW	Monitor	SW / PC	Monitor	UP	Monitor
Arduino Leonardo / Due	HW	Native	HW	Native	LEO	Native
Arduino Due	HW	Native	PC	Programming	UP	Programming

Test the Shield on Arduino

1. Select your Arduino board from Arduino IDE menu *"Tools"* > *"Board"* and the correct serial port (both choices must match the on-board USB port you have connected)
2. Open and upload the example sketch *"TestEasyVR"* from the Arduino IDE menu
3. Open the *"Serial Monitor"* window at 9600 bps
4. Send a question mark "?" (without quotes) and after a few seconds you should receive an "EasyVR detected" message, along with a summary of the module configuration
5. See the comments on top of the sketch for usage details and other tests you can perform

Other examples are also included with the library, to test special features of the EasyVR module.

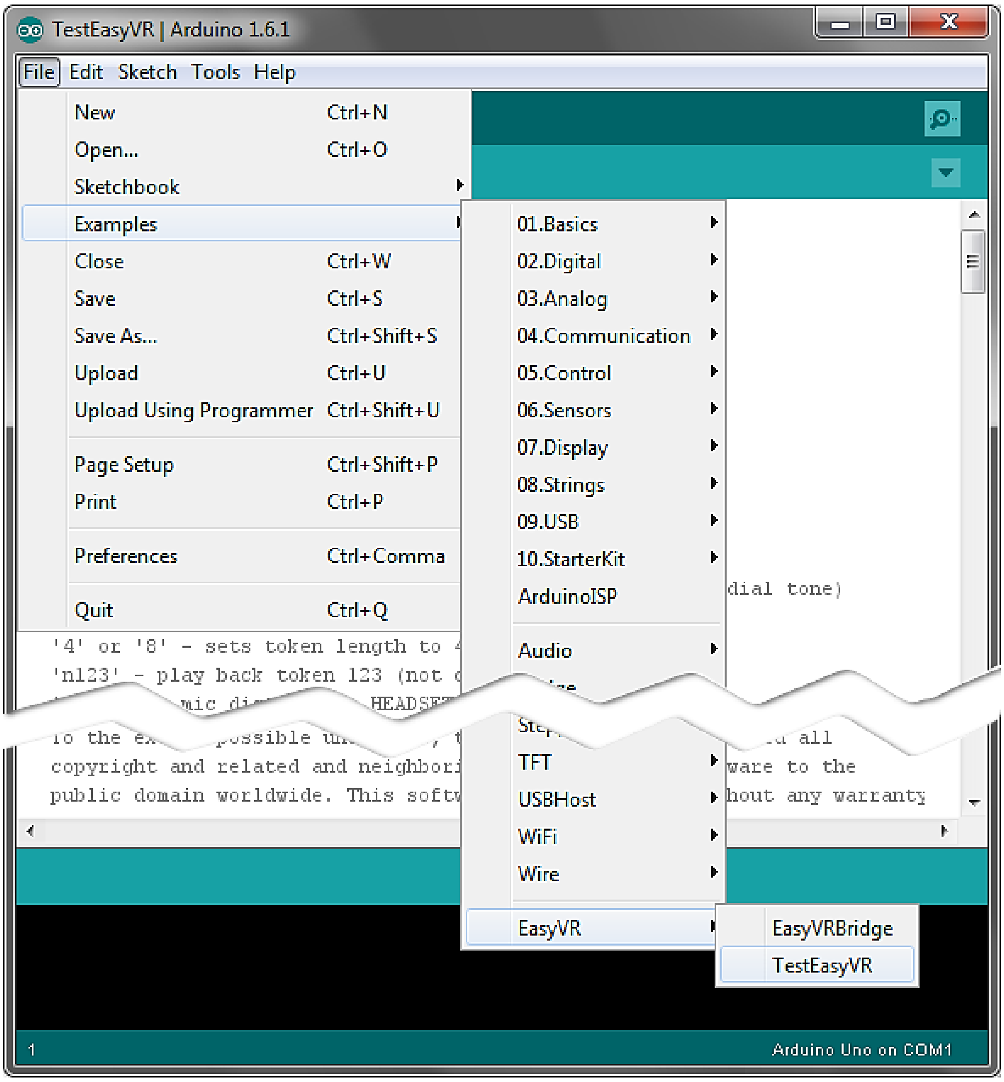


Figure 2 - Finding example sketches for the EasyVR

Test the Shield from the EasyVR Commander

1. Select your Arduino board from Arduino IDE menu *"Tools"* > *"Board"* and the correct serial port (both choices must match the on-board USB port you have connected)
2. Open and upload the example sketch *"TestEasyVR"* or *"EasyVRBridge"* from the Arduino IDE menu *"File"* > *"Examples"* > *"EasyVR"*
3. Make sure the *"Serial Monitor"* window is not open in the Arduino IDE
4. Open the EasyVR Commander and connect to the same serial port used by Arduino

When the EasyVR Commander is connected, you can also generate a template code for Arduino that will use the provided libraries (see *EasyVR Arduino Library Documentation*). All you need is to write actions for each recognized command.

Download custom data or Firmware update

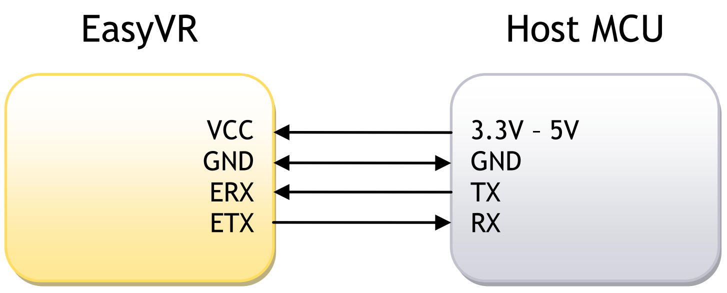
1. Follow the same steps for the EasyVR Commander above, but make sure to adjust the Mode Jumper settings for your main board (see table above) and that the *PROG* red LED is on
2. While the EasyVR Commander is disconnected choose *"Update Custom Data"* from the *"File"* menu or *"Update firmware"* from the *"Help"* menu

EasyVR Programming

Communication Protocol

Introduction

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, and 100 ms - 1 s. That accounts for slower or faster host systems and therefore suitable also for software-based serial communication (bit-banging) or communication proxies/bridges.

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 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 the 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

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

Some commands share the same lower case letter, because there were no command identifiers available when the protocol has been expanded, but the first argument is used to discriminate.

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, STS_AWAKEN (if sleeping)	

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_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_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_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_MIC_DIST

'k' (6Bh)	Set the microphone operating distance
[1]	Fixed to (-1)
[2]	Distance settings (1-3): 1 = “headset” (around 5cm from speaker’s mouth) 2 = “arm’s length” (default setting, from about 50cm to 1m) 3 = “far mic” (up to around 3m)
Expected replies: STS_SUCCESS	

CMD_TRAILING

't' (74h)	Set trailing silence for SI recognition
[1]	Fixed to (-1)
[2]	Amount of silence at the end of an utterance (0-31): 0 = 100ms … in steps of 25ms 31 = 875ms
Expected replies: STS_SUCCESS	

CMD_FAST_SD

'f' (66h)	Set fast operating mode for SD/SV recognition
[1]	Fixed to (-1)
[2]	Operating mode (0 = normal/default, 1 = fast/low-latency)
Expected replies: STS_TOKEN, STS_TIMEOUT	

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 word set
[1]	Word set 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 '`') EasyVR Commander encodes digits 0-9 as A-J prefixed by '^'
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 (erase commands/groups and messages)
'R' (52h)	Confirmation character
Expected replies: STS_SUCCESS	

CMD_RESET_SD

'r' (72h)	Reset only commands and groups
'D' (52h)	Confirmation character
Expected replies: STS_SUCCESS	

CMD_RESET_RP

'r' (72h)	Reset only message recordings
'M' (52h)	Confirmation character
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' (77h)	Wave table entry playback
[1-2]	Two positive values that form a 10-bit index to the sound table (index = [1] * 32 + [2], 0 = built-in "beep", 1-1023 = sound index)
[3]	Playback volume (0-31, 0 = min volume, 15 = full scale, 31 = double gain)
Expected replies: STS_SUCCESS, STS_ERROR	

CMD_PLAY_DTMF

'w' (77h)	Play a DTMF key tone or dial tone
[1]	Fixed to (-1)
[2]	Index of phone tone to play (0-9 for digits, 10 for '*' key, 11 for '#' key and 12-15 for extra keys 'A' to 'D', -1 for the dial tone)
[3]	Tone duration minus 1 (0-31 in 40ms units for keys, in seconds for the dial tone)
Expected replies: STS_SUCCESS	

CMD_DUMP_SX

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

CMD_DUMP_SI

'z' (7Ah)	Read custom and built-in grammars data
[1]	Index of SI grammar to read (0-31) or (-1) to get the total count of SI grammars (including the first 4 built-in word sets)
Expected replies: STS_GRAMMAR, STS_COUNT	

CMD_SEND_SN

'j' (6Ah)	Send a SonicNet™ token
[1]	Length of token (4 or 8 in bits)
[2-3]	Two positive values that form an 8-bit token index (index = [2] * 32 + [3], 0-15 for 4-bit tokens or 0-255 for 8-bits tokens)
[4-5]	Two positive values that form a 10-bit delay for token output since the next sound playback (delay = [4] * 32 + [5], 0 = send immediately, 1-1023 = delay in units of 27.46ms)
Expected replies: STS_SUCCESS	

CMD_RECV_SN

'f' (66h)	Receive a SonicNet™ token
[1]	Length of token (4 or 8 in bits)
[2]	Rejection level (0-2 = higher values mean fewer results, 1 = default)
[3-4]	Two positive values that form a 10-bit timeout for token detection (timeout = [3] * 32 + [4], 0 = wait forever, 1-1023 = timeout in units of 27.46ms)
Expected replies: STS_TOKEN, STS_TIMEOUT	

CMD_LIPSYNC

'l' (6Ch)	Start real-time lip-sync
[1]	Fixed to (-1)
[2-3]	Activation threshold (10-bit value = [2] * 32 + [3]): 270 = default setting
[4-5]	Timeout option (8-bit value = [4] * 16 + [5]): 0 = no timeout (can be interrupted) 1-255 = duration in seconds
Expected replies: STS_LIPSYNC	

CMD_RECORD_RP

'r' (72h)	Record a message
[1]	Fixed to (-1)
[2]	Message index (0-31)
[3]	Data format (8)
[4]	Timeout option (0-31): 0 = no timeout (can be interrupted) 1-31 = duration in seconds
Expected replies: STS_SUCCESS, STS_ERROR	

CMD_PLAY_RP

'p' (70h)	Play a message recording
[1]	Fixed to (-1)
[2]	Message index (0-31)
[3]	Playback options (bit-mask): Bit 2 (4) = playback speed (0 = normal, 1 = fast) Bit 1-0 (0-3) = volume attenuation (0 = normal, 1 = -2.2dB, 2 = -4.5dB, 3 = -6.7dB)
Expected replies: STS_SUCCESS, STS_ERROR	

CMD_ERASE_RP

'e' (65h)	Erase a message recording
[1]	Fixed to (-1)
[2]	Message index (0-31)
Expected replies: STS_SUCCESS, STS_ERROR	

CMD_VERIFY_RP

'v' (76h)	Verify file-system integrity for message recordings
[1]	Fixed to (-1)
[2]	Type of operation: 0 = check only 1 = check and fix errors
Expected replies: STS_SUCCESS, STS_ERROR	

CMD_SERVICE + SVC_EXPORT_SD

'~' (7Eh)	Service protocol expansion
'X' (58h)	Export command raw data
[1]	Group index (0 = trigger, 1-15 = generic, 16 = password)
[2]	Command position (0-31)
Expected replies: STS_SERVICE + SVC_DUMP_SD	

CMD_SERVICE + SVC_IMPORT_SD

'~' (7Eh)	Service protocol expansion
'I' (49h)	Import command raw data
[1]	Group index (0 = trigger, 1-15 = generic, 16 = password)
[2]	Command position (0-31)
[3-514]	Raw command data (encoded as hex nibbles – byte = [n] * 16 + [n+1])
[515-518]	Data checksum (16-bit sum of all 256 bytes, starting at 1234h)
Expected replies: STS_SUCCESS, STS_INTERR (if checksum fails)	

CMD_SERVICE + SVC_VERIFY_SD

'~' (7Eh)	Service protocol expansion
'V' (56h)	Verify command raw data
[1]	Group index (0 = trigger, 1-15 = generic, 16 = password)
[2]	Command position (0-31)
Expected replies: STS_SUCCESS, STS_RESULT, STS_SIMILAR, STS_ERROR	

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 or total number of SI grammars
[1]	Integer (0-31 = command/grammar count, -1 = 32 commands/grammars)
In reply to: CMD_COUNT_SD, CMD_DUMP_SI	

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 '`') EasyVR Commander encodes digits 0-9 as A-J prefixed by '^'
In reply to: CMD_DUMP_SD	

STS_ERROR

'e' (65h)	Signal recognition error
[1-2]	Two positive values that form an 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_LIPSYNC

'l' (6Ch)	Lip-sync streaming data
[1-N]	Mouth position (0-31): 0 = fully closed 31 = fully open **Note:** New values are available at request around every 27ms until timeout occurs or command is interrupted. A new status is sent at the end (see STS_TIMEOUT, STS_INTERR).
In reply to: CMD_LIPSYNC	

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, etc.	

STS_RESULT

'r' (72h)	Recognized 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)	Recognized 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 = VRbot, 1-7 = EasyVR 2, 8-15 = EasyVR 3, 16+ = EasyVR 3 Plus)
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 positive 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	

STS_GRAMMAR

'z' (7Ah)	Provide custom grammar data
[1]	Some flags for this grammar (currently16 is returned for trigger grammars, 0 for commands)
[2]	Number of commands in this grammar (0-31)
[3]	Length of first command label (0-31)
[4-n]	Text of first command label (ASCII characters from 'A' to '`')
…	Repeat last two fields for all the commands in this grammar
In reply to: CMD_DUMP_SI	

STS_TOKEN

'f' (66h)	Detected a SonicNet™ token
[1-2]	Two positive values that form the index of a received token (index = [1] * 32 + [2], 0-15 for 4-bit tokens or 0-255 for 8-bits tokens)
In reply to: CMD_RECV_SN	

STS_SERVICE + SVC_DUMP_SD

'~' (7Eh)	Service protocol expansion
'D' (44h)	Provide command raw data
[1-512]	Raw command data (encoded as hex nibbles - byte = [n] * 16 + [n+1])
[513-516]	Data checksum (16-bit sum of all 256 bytes, starting at 1234h)
In reply to: CMD_SERVICE + SVC_EXPORT_SD	

Communication Examples

These are some examples of actual command and status characters 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
# (uses 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 or custom 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') * 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'
IF count = -1 THEN count = 32
```

Read a user defined command group

```
# 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 word set), command index and language identifier to use with the communication protocol.

		Language						
		0	1	2		3	4	5
Trigger Word set	**Command Index**	**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

Below the list of the most useful error codes that may be returned by training or recognizing 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
31h	ERR_RP_BAD_LEVEL	play - illegal compression level
38h	ERR_RP_NO_MSG	play, erase, copy - msg doesn't exist
39h	ERR_RP_MSG_EXISTS	rec, copy - msg already exists
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_CUSTOM_NOTA	recognized SI word is not in vocabulary
81h	ERR_CUSTOM_INVALID	invalid data (for memory check)

The first group of codes (03h – 07h) is 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) indicates 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 of codes (31h – 39h) deals with errors in message recording, playback or erase.

The fifth group (4Ah – 4Eh) deals with errors in the compressed sound data, either because the wrong version of the QuickSynthesis™ tool has been used to generate the sound table or because a not supported compression scheme has been selected (or data is generically invalid).

The last group (80h – 81h) contains error codes of the EasyVR module itself. Error code (80h = NOTA) 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 can be ignored. Error code (81h = INVALID) is a generic error reported after data validation and currently used for file-system check-up (message recordings).

Protocol header file

This file “protocol.h” can be used with applications written in the C language. You can download a recent copy from the VeeaR website.

```
#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_MIC_DIST    'k' // set microphone (<1>=-1) distance <2>
#define CMD_LEVEL       'v' // set sd level <1>
#define CMD_VERIFY_RP   'v' // verify filesystem (<1>=-1) with flags <2> (0=check only, 1=fix)
#define CMD_LANGUAGE    'l' // set si language <1>
#define CMD_LIPSYNC     'l' // start real-time lipsync (<1>=-1) with threshold <2-3>, timeout <4-5>
#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_TRAILING    't' // set trailing (<1>=-1) silence <2> (0-31 = 100-875 milliseconds)
#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_DUMP_RP     'd' // dump message (<1>=-1) at pos <2>
#define CMD_ERASE_SD    'e' // reset command at group <1> pos <2>
#define CMD_ERASE_RP    'e' // erase recording (<1>=-1) at 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_PLAY_RP     'p' // play recording (<1>=-1) at pos <2> with flags <3>
#define CMD_MASK_SD     'm' // get active group mask
#define CMD_RESETALL    'r' // reset all memory (commands/groups and messages), with <1>='R'
#define CMD_RESET_SD    'r' // reset only commands/groups, with <1>='D'
#define CMD_RESET_RP    'r' // reset only messages, with <1>='M'
#define CMD_RECORD_RP   'r' // record message (<1>=-1) at pos <2> with bits <3> and timeout <4>
#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_PLAY_DTMF   'w' // play (<1>=-1) dial tone <2> for duration <3>
#define CMD_DUMP_SX     'h' // dump wave table entries
#define CMD_DUMP_SI     'z' // dump si settings for ws <1> (or total ws count if -1)
#define CMD_SEND_SN     'j' // send sonicnet token with bits <1> index <2-3> at time <4-5>
#define CMD_RECV_SN     'f' // receive sonicnet token with bits <1> rejection <2> timeout <3-4>
#define CMD_FAST_SD     'f' // set sd/sv (<1>=-1) to use fast recognition <2> (0=normal/default, 1=fast)

#define CMD_SERVICE     '~' // send service request
#define SVC_EXPORT_SD   'X' // request export of command <2> in group <1> as raw dump
#define SVC_IMPORT_SD   'I' // request import of command <2> in group <1> as raw dump
#define SVC_VERIFY_SD   'V' // verify training of imported raw command <2> in group <1>

#define STS_SERVICE     '~' // get service reply
#define SVC_DUMP_SD     'D' // provide raw command data <1-512> followed by checksum <513-516>

#define STS_MASK        'k' // mask of active groups <1-8>
#define STS_COUNT       'c' // count of commands <1> (or number of ws <1>)
#define STS_AWAKEN      'w' // back from power down mode
#define STS_DATA        'd' // provide training <1>, conflict <2>, command 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_LIPSYNC     'l' // lipsync stream follows
#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' // table entries count <1-2> (10-bit), table name <3-35> (counted string)
#define STS_GRAMMAR     'z' // si grammar: flags <1>, word count <2>, labels... <3-35> (n counted strings)
#define STS_TOKEN       'f' // received sonicnet token <1-2>
```

```
#define STS_MESSAGE     'g' // message status <1> (0=empty, 4/8=bits format), length <2-7>

// 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
```

A better source of information and a reference protocol implementation for the C/C++ language can be found in the Arduino Library source.

EasyVR Arduino Library

The EasyVR library implements the serial communication protocol to manage the EasyVR module and the EasyVR Shield from Arduino boards and compatible controllers and it enables a quick access to all the EasyVR features.

Installation

To install the EasyVR library on your Arduino IDE use the menu `Sketch` > `Include Library` > `Add .ZIP Library` and open the release archive.
You can also use `Sketch > Include Library > Manage Libraries...` and look for "EasyVR" to download and install the latest release.

Examples

You can easily open the example sketches included with the EasyVR library from inside the Arduino IDE, using the menu `File` > `Examples` > `EasyVR` and choosing one of the available sketches.

EasyVR library settings

Macros

- #define EASYVR_RX_TIMEOUT
- #define EASYVR_STORAGE_TIMEOUT
- #define EASYVR_WAKE_TIMEOUT
- #define EASYVR_PLAY_TIMEOUT
- #define EASYVR_TOKEN_TIMEOUT

Detailed Description

By assigning new values to these symbols you can alter the default settings used by the library implementation.
These settings are available for completeness. The default values should be appropriate for normal use cases.

Macro Definition Documentation

#define EASYVR_RX_TIMEOUT

Receive timeout (in ms). The maximum time that is spent waiting for a reply from the EasyVR module.

#define EASYVR_STORAGE_TIMEOUT

Reply timeout for storage operations (in ms). The maximum time that is spent waiting for a reply after a command that involves write access to the EasyVR internal storage.

#define EASYVR_WAKE_TIMEOUT

Wakeup maximum delay (in ms). The maximum time that the EasyVR module can spend for waking up from idle states.

#define EASYVR_PLAY_TIMEOUT

Playback maximum duration (in ms). The maximum time that is spent waiting for a synchronous playback operation to complete. Asynchronous playback is not affected.

#define EASYVR_TOKEN_TIMEOUT

Token maximum duration (in ms). The maximum time that is spent by the EasyVR module for sending a SonicNet token and reply.

EasyVR Class Reference

Public Types

- enum ModuleId { VRBOT, EASYVR, EASYVR2, EASYVR2_3, EASYVR3, EASYVR3_1, EASYVR3_2, EASYVR3_3, EASYVR3_4, EASYVR3_5, EASYVR3PLUS }
- enum Language { ENGLISH, ITALIAN, JAPANESE, GERMAN, SPANISH, FRENCH }
- enum Group { TRIGGER, PASSWORD }
- enum Wordset { TRIGGER_SET, ACTION_SET, DIRECTION_SET, NUMBER_SET }
- enum Distance { HEADSET, ARMS_LENGTH, FAR_MIC }
- enum Knob { LOOSER, LOOSE, TYPICAL, STRICT, STRICTER }
- enum Level { EASY, NORMAL, HARD, HARDER, HARDEST }
- enum TrailingSilence { TRAILING_MIN, TRAILING_DEF, TRAILING_MAX, TRAILING_100MS, TRAILING_200MS, TRAILING_300MS, TRAILING_400MS, TRAILING_500MS, TRAILING_600MS, TRAILING_700MS, TRAILING_800MS }
- enum CommandLatency { MODE_NORMAL, MODE_FAST }
- enum Baudrate { B115200, B57600, B38400, B19200, B9600 }
- enum WakeMode { WAKE_ON_CHAR, WAKE_ON_WHISTLE, WAKE_ON_LOUDSOUND, WAKE_ON_2CLAPS, WAKE_ON_3CLAPS }
- enum ClapSense { CLAP_SENSE_LOW, CLAP_SENSE_MID, CLAP_SENSE_HIGH }
- enum PinConfig { OUTPUT_LOW, OUTPUT_HIGH, INPUT_HIZ, INPUT_STRONG, INPUT_WEAK }
- enum PinNumber { IO1, IO2, IO3, IO4, IO5, IO6 }
- enum SoundVolume { VOL_MIN, VOL_HALF, VOL_FULL, VOL_DOUBLE }
- enum SoundIndex { BEEP }
- enum GrammarFlag { GF_TRIGGER }
- enum RejectionLevel { REJECTION_MIN, REJECTION_AVG, REJECTION_MAX }
- enum MessageSpeed { SPEED_NORMAL, SPEED_FASTER }
- enum MessageAttenuation { ATTEN_NONE, ATTEN_2DB2, ATTEN_4DB5, ATTEN_6DB7 }
- enum MessageType { MSG_EMPTY, MSG_8BIT }
- enum LipsyncThreshold { RTLS_THRESHOLD_DEF, RTLS_THRESHOLD_MAX }
- enum ErrorCode { ERR_DATACOL_TOO_LONG, ERR_DATACOL_TOO_NOISY, ERR_DATACOL_TOO_SOFT, ERR_DATACOL_TOO_LOUD, ERR_DATACOL_TOO_SOON, ERR_DATACOL_TOO_CHOPPY, ERR_DATACOL_BAD_WEIGHTS, ERR_DATACOL_BAD_SETUP, ERR_RECOG_FAIL, ERR_RECOG_LOW_CONF, ERR_RECOG_MID_CONF, ERR_RECOG_BAD_TEMPLATE, ERR_RECOG_BAD_WEIGHTS, ERR_RECOG_DURATION, ERR_T2SI_EXCESS_STATES, ERR_T2SI_BAD_VERSION, ERR_T2SI_OUT_OF_RAM, ERR_T2SI_UNEXPECTED, ERR_T2SI_OVERFLOW, ERR_T2SI_PARAMETER, ERR_T2SI_NN_TOO_BIG, ERR_T2SI_NN_BAD_VERSION, ERR_T2SI_NN_NOT_READY, ERR_T2SI_NN_BAD_LAYERS, ERR_T2SI_TRIG_OOV, ERR_T2SI_TOO_SHORT, ERR_RP_BAD_LEVEL, ERR_RP_NO_MSG, ERR_RP_MSG_EXISTS, ERR_SYNTH_BAD_VERSION, ERR_SYNTH_ID_NOT_SET, ERR_SYNTH_TOO_MANY_TABLES, ERR_SYNTH_BAD_SEN, ERR_SYNTH_BAD_MSG, ERR_CUSTOM_NOTA, ERR_CUSTOM_INVALID, ERR_SW_STACK_OVERFLOW, ERR_INTERNAL_T2SI_BAD_SETUP }
- enum BridgeMode { BRIDGE_NONE, BRIDGE_NORMAL, BRIDGE_BOOT, BRIDGE_ESCAPE_CHAR }

Public Member Functions

- EasyVR (Stream &s)
- bool detect ()
- bool stop ()
- int8_t getID ()
- bool setLanguage (int8_t lang)
- bool setTimeout (int8_t seconds)
- bool setMicDistance (int8_t dist)
- bool setKnob (int8_t knob)
- bool setTrailingSilence (int8_t dur)
- bool setLevel (int8_t level)
- bool setCommandLatency (int8_t mode)
- bool setDelay (uint16_t millis)
- bool changeBaudrate (int8_t baud)
- bool sleep (int8_t mode)

- bool addCommand (int8_t group, int8_t index)
- bool removeCommand (int8_t group, int8_t index)
- bool setCommandLabel (int8_t group, int8_t index, const char *name)
- bool eraseCommand (int8_t group, int8_t index)
- bool getGroupMask (uint32_t &mask)
- int8_t getCommandCount (int8_t group)
- bool dumpCommand (int8_t group, int8_t index, char *name, uint8_t &training)
- int8_t getGrammarsCount (void)
- bool dumpGrammar (int8_t grammar, uint8_t &flags, uint8_t &count)
- bool getNextWordLabel (char *name)
- void trainCommand (int8_t group, int8_t index)
- void recognizeCommand (int8_t group)
- void recognizeWord (int8_t wordset)
- bool hasFinished ()
- int8_t getCommand ()
- int8_t getWord ()
- int16_t getToken ()
- int16_t getError ()
- bool isTimeout ()
- bool isAwakened ()
- bool isConflict ()
- bool isMemoryFull ()
- bool isInvalid ()
- bool setPinOutput (int8_t pin, int8_t config)
- int8_t getPinInput (int8_t pin, int8_t config)
- void detectToken (int8_t bits, int8_t rejection, uint16_t timeout)
- void sendTokenAsync (int8_t bits, uint8_t token)
- bool sendToken (int8_t bits, uint8_t token)
- bool embedToken (int8_t bits, uint8_t token, uint16_t delay)
- void playSoundAsync (int16_t index, int8_t volume)
- bool playSound (int16_t index, int8_t volume)
- bool dumpSoundTable (char *name, int16_t &count)
- bool playPhoneTone (int8_t tone, uint8_t duration)
- bool resetAll (bool wait=true)
- bool resetCommands (bool wait=true)
- bool resetMessages (bool wait=true)
- bool checkMessages ()
- bool fixMessages (bool wait=true)
- void recordMessageAsync (int8_t index, int8_t bits, int8_t timeout)
- void playMessageAsync (int8_t index, int8_t speed, int8_t atten)
- void eraseMessageAsync (int8_t index)
- bool dumpMessage (int8_t index, int8_t &type, int32_t &length)
- bool realtimeLipsync (int16_t threshold, uint8_t timeout)
- bool fetchMouthPosition (int8_t &value)
- bool exportCommand (int8_t group, int8_t index, uint8_t *data)
- bool importCommand (int8_t group, int8_t index, const uint8_t *data)
- void verifyCommand (int8_t group, int8_t index)
- int bridgeRequested (Stream &port)
- void bridgeLoop (Stream &port)

Detailed Description

An implementation of the EasyVR communication protocol.

Member Enumeration Documentation

enum ***ModuleId***

Module identification number (firmware version)

Enumerator

VRBOT Identifies a VRbot module
EASYVR Identifies an EasyVR module
EASYVR2 Identifies an EasyVR module version 2
EASYVR2_3 Identifies an EasyVR module version 2, firmware revision 3
EASYVR3 Identifies an EasyVR module version 3, firmware revision 0
EASYVR3_1 Identifies an EasyVR module version 3, firmware revision 1
EASYVR3_2 Identifies an EasyVR module version 3, firmware revision 2
EASYVR3_3 Identifies an EasyVR module version 3, firmware revision 3
EASYVR3_4 Identifies an EasyVR module version 3, firmware revision 4
EASYVR3_5 Identifies an EasyVR module version 3, firmware revision 5
EASYVR3PLUS Identifies an EasyVR module version 3+, firmware revision 0

enum ***Language***

Language to use for recognition of built-in words

Enumerator

ENGLISH Uses the US English word sets
ITALIAN Uses the Italian word sets
JAPANESE Uses the Japanese word sets
GERMAN Uses the German word sets
SPANISH Uses the Spanish word sets
FRENCH Uses the French word sets

enum ***Group***

Special group numbers for recognition of custom commands

Enumerator

TRIGGER The trigger group (shared with built-in trigger word)
PASSWORD The password group (uses speaker verification technology)

enum ***Wordset***

Index of built-in word sets

Enumerator

TRIGGER_SET The built-in trigger word set
ACTION_SET The built-in action word set
DIRECTION_SET The built-in direction word set
NUMBER_SET The built-in number word set

enum ***Distance***

Microphone distance from the user's mouth, used by all recognition technologies

Enumerator

HEADSET Nearest range (around 5cm)
ARMS_LENGTH Medium range (from about 50cm to 1m)
FAR_MIC Farthest range (up to 3m)

enum ***Knob***

Confidence thresholds for the knob settings, used for recognition of built-in words or custom grammars (not used for the mixed trigger group)

Enumerator

LOOSER Lowest threshold, most results reported
LOOSE Lower threshold, more results reported
TYPICAL Typical threshold (default)
STRICT Higher threshold, fewer results reported
STRICTER Highest threshold, fewest results reported

enum ***Level***

Strictness values for the level settings, used for recognition of custom commands (not used for the mixed trigger group)

Enumerator

EASY Lowest value, most results reported
NORMAL Typical value (default)
HARD Slightly higher value, fewer results reported
HARDER Higher value, fewer results reported
HARDEST Highest value, fewest results reported

enum ***TrailingSilence***

Trailing silence settings used for recognition of built-in words or custom grammars (including the mixed trigger group), in a range from 100ms to 875ms in steps of 25ms.

Enumerator

TRAILING_MIN Lowest value (100ms), minimum latency
TRAILING_DEF Default value (400ms) after power on or reset
TRAILING_MAX Highest value (875ms), maximum latency
TRAILING_100MS Silence duration is 100ms
TRAILING_200MS Silence duration is 200ms
TRAILING_300MS Silence duration is 300ms
TRAILING_400MS Silence duration is 400ms
TRAILING_500MS Silence duration is 500ms
TRAILING_600MS Silence duration is 600ms
TRAILING_700MS Silence duration is 700ms
TRAILING_800MS Silence duration is 800ms

enum ***CommandLatency***

Latency settings used for recognition of custom commands or passwords (excluding the mixed trigger group)

Enumerator

MODE_NORMAL Normal settings (default), higher latency
MODE_FAST Fast settings, better response time

enum ***Baudrate***

Constants to use for baudrate settings

Enumerator

B115200 115200 bps
B57600 57600 bps
B38400 38400 bps
B19200 19200 bps
B9600 9600 bps (default)

enum ***WakeMode***

Constants for choosing wake-up method in sleep mode

Enumerator

WAKE_ON_CHAR Wake up on any character received
WAKE_ON_WHISTLE Wake up on whistle or any character received
WAKE_ON_LOUDSOUND Wake up on a loud sound or any character received
WAKE_ON_2CLAPS Wake up on double hands-clap or any character received
WAKE_ON_3CLAPS Wake up on triple hands-clap or any character received

enum ***ClapSense***

Hands-clap sensitivity for wakeup from sleep mode. Use in combination with WAKE_ON_2CLAPS or WAKE_ON_3CLAPS

Enumerator

CLAP_SENSE_LOW Lowest threshold
CLAP_SENSE_MID Typical threshold
CLAP_SENSE_HIGH Highest threshold

enum ***PinConfig***

Pin configuration options for the extra I/O connector

Enumerator

OUTPUT_LOW Pin is an output at low level (0V)
OUTPUT_HIGH Pin is an output at high level (3V)
INPUT_HIZ Pin is an high impedance input
INPUT_STRONG Pin is an input with strong pull-up (~10K)
INPUT_WEAK Pin is an input with weak pull-up (~200K)

enum ***PinNumber***

Available pin numbers on the extra I/O connector

Enumerator

IO1 Identifier of pin IO1
IO2 Identifier of pin IO2
IO3 Identifier of pin IO3
IO4 Identifier of pin IO4 [only EasyVR3]
IO5 Identifier of pin IO5 [only EasyVR3]
IO6 Identifier of pin IO6 [only EasyVR3]

enum ***SoundVolume***

Some quick volume settings for the sound playback functions (any value in the range 0-31 can be used)

Enumerator

VOL_MIN Lowest volume (almost mute)
VOL_HALF Half scale volume (softer)
VOL_FULL Full scale volume (normal)
VOL_DOUBLE Double gain volume (louder)

enum ***SoundIndex***

Special sound index values, always available even when no soundtable is present

Enumerator

BEEP Beep sound

enum ***GrammarFlag***

Flags used by custom grammars

Enumerator

GF_TRIGGER A bit mask that indicate grammar is a trigger (opposed to commands)

enum ***RejectionLevel***

Noise rejection level for SonicNet token detection (higher value, fewer results)

Enumerator

REJECTION_MIN Lowest noise rejection, highest sensitivity
REJECTION_AVG Medium noise rejection, medium sensitivity
REJECTION_MAX Highest noise rejection, lowest sensitivity

enum ***MessageSpeed***

Playback speed for recorded messages

Enumerator

SPEED_NORMAL Normal playback speed
SPEED_FASTER Faster playback speed

enum ***MessageAttenuation***

Playback attenuation for recorded messages

Enumerator

ATTEN_NONE No attenuation (normalized volume)
ATTEN_2DB2 Attenuation of -2.2dB
ATTEN_4DB5 Attenuation of -4.5dB
ATTEN_6DB7 Attenuation of -6.7dB

enum ***MessageType***

Type of recorded message

Enumerator

MSG_EMPTY Empty message slot
MSG_8BIT Message recorded with 8-bits PCM

enum ***LipsyncThreshold***

Threshold for real-time lip-sync

Enumerator

RTLS_THRESHOLD_DEF Default threshold
RTLS_THRESHOLD_MAX Maximum threshold

enum ***ErrorCode***

Error codes used by various functions

Enumerator

ERR_DATACOL_TOO_LONG too long (memory overflow)
ERR_DATACOL_TOO_NOISY too noisy
ERR_DATACOL_TOO_SOFT spoke too soft
ERR_DATACOL_TOO_LOUD spoke too loud
ERR_DATACOL_TOO_SOON spoke too soon
ERR_DATACOL_TOO_CHOPPY too many segments/too complex
ERR_DATACOL_BAD_WEIGHTS invalid SI weights
ERR_DATACOL_BAD_SETUP invalid setup
ERR_RECOG_FAIL recognition failed
ERR_RECOG_LOW_CONF recognition result doubtful
ERR_RECOG_MID_CONF recognition result maybe
ERR_RECOG_BAD_TEMPLATE invalid SD/SV template
ERR_RECOG_BAD_WEIGHTS invalid SI weights
ERR_RECOG_DURATION incompatible pattern durations
ERR_T2SI_EXCESS_STATES state structure is too big
ERR_T2SI_BAD_VERSION RSC code version/Grammar ROM dont match
ERR_T2SI_OUT_OF_RAM reached limit of available RAM
ERR_T2SI_UNEXPECTED an unexpected error occurred
ERR_T2SI_OVERFLOW ran out of time to process
ERR_T2SI_PARAMETER bad macro or grammar parameter
ERR_T2SI_NN_TOO_BIG layer size out of limits
ERR_T2SI_NN_BAD_VERSION net structure incompatibility
ERR_T2SI_NN_NOT_READY initialization not complete
ERR_T2SI_NN_BAD_LAYERS not correct number of layers
ERR_T2SI_TRIG_OOV trigger recognized Out Of Vocabulary
ERR_T2SI_TOO_SHORT utterance was too short
ERR_RP_BAD_LEVEL play - illegal compression level
ERR_RP_NO_MSG play, erase, copy - msg doesn't exist
ERR_RP_MSG_EXISTS rec, copy - msg already exists
ERR_SYNTH_BAD_VERSION bad release number in speech file
ERR_SYNTH_ID_NOT_SET (obsolete) bad sentence structure
ERR_SYNTH_TOO_MANY_TABLES (obsolete) too many talk tables
ERR_SYNTH_BAD_SEN (obsolete) bad sentence number
ERR_SYNTH_BAD_MSG bad message data or SX technology files missing

ERR_CUSTOM_NOTA none of the above (out of grammar)
ERR_CUSTOM_INVALID invalid data (for memory check)
ERR_SW_STACK_OVERFLOW no room left in software stack
ERR_INTERNAL_T2SI_BAD_SETUP T2SI test mode error

enum ***BridgeMode***

Type of Bridge mode requested

Enumerator

BRIDGE_NONE Bridge mode has not been requested
BRIDGE_NORMAL Normal bridge mode (EasyVR baudrate 9600)
BRIDGE_BOOT Bridge mode for EasyVR bootloader (baudrate 115200)
BRIDGE_ESCAPE_CHAR Special character to enter/exit Bridge mode

---

Constructor & Destructor Documentation

EasyVR *(Stream & s)*

Creates an EasyVR object, using a communication object implementing the #Stream interface (such as #HardwareSerial, or the modified #SoftwareSerial and #NewSoftSerial).

Parameters:

s	the Stream object to use for communication with the EasyVR module

---

Member Function Documentation

bool detect ()

Detects an EasyVR module, waking it from sleep mode and checking it responds correctly.

Return values:

true	if a compatible module has been found

bool stop ()

Interrupts pending recognition or playback operations.

Return values:

true	if the request is satisfied and the module is back to ready

int8_t getID ()

Gets the module identification number (firmware version).

Return values:

integer	is one of the values in ModuleId

bool setLanguage (int8_t lang)

Sets the language to use for recognition of built-in words.

Parameters:

lang	(0-5) is one of values in Language

Return values:

true	if the operation is successful

bool setTimeout (int8_t seconds)

Sets the timeout to use for any recognition task.

Parameters:

seconds	(0-31) is the maximum time the module keep listening for a word or a command

Return values:

true	if the operation is successful

bool setMicDistance (int8_t dist)

Sets the operating distance of the microphone. This setting represents the distance between the microphone and the user's mouth, in one of three possible configurations.

Parameters:

dist	(1-3) is one of values in Distance

Return values:

true	if the operation is successful

bool setKnob (int8_t knob)

Sets the confidence threshold to use for recognition of built-in words or custom grammars.

Parameters:

knob	(0-4) is one of values in Knob

Return values:

true	if the operation is successful

bool setTrailingSilence (int8_t dur)

Sets the trailing silence duration for recognition of built-in words or custom grammars.

Parameters:

dur	(0-31) is the silence duration as defined in TrailingSilence

Return values:

true	if the operation is successful

bool setLevel (int8_t level)

Sets the strictness level to use for recognition of custom commands.

Parameters:

level	(1-5) is one of values in Level

Return values:

true	if the operation is successful

bool setCommandLatency (int8_t mode)

Enables or disables fast recognition for custom commands and passwords. Fast SD/SV recognition can improve response time.

Parameters:

mode	(0-1) is one of the values in CommandLatency

Return values:

true	if the operation is successful

bool setDelay (uint16_t millis)

Sets the delay before any reply of the module.

Parameters:

millis	(0-1000) is the delay duration in milliseconds, rounded to 10 units in range 10-100 and to 100 units in range 100-1000.

Return values:

true	if the operation is successful

bool changeBaudrate (int8_t baud)

Sets the new communication speed. You need to modify the baudrate of the underlying Stream object accordingly, after the function returns successfully.

Parameters:

baud	is one of values in Baudrate

Return values:

true	if the operation is successful

bool sleep (int8_t mode)

Puts the module in sleep mode.

Parameters:

mode	is one of values in WakeMode, optionally combined with one of the values in ClapSense

Return values:

true	if the operation is successful

bool addCommand (int8_t group, int8_t index)

Adds a new custom command to a group.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group

Return values:

true	if the operation is successful

bool removeCommand (int8_t group, int8_t index)

Removes a custom command from a group.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group

Return values:

true	if the operation is successful

bool setCommandLabel (int8_t group, *int8_t* index, *const char ** name*)*

Sets the name of a custom command.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group
name	is a string containing the label to be assigned to the specified command

Return values:

true	if the operation is successful

bool eraseCommand (int8_t group, *int8_t* index*)*

Erases the training data of a custom command.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group

Return values:

true	if the operation is successful

bool getGroupMask (uint32_t & mask*)*

Gets a bit mask of groups that contain at least one command.

Parameters:

mask	is a variable to hold the group mask when the function returns

Return values:

true	if the operation is successful

int8_t getCommandCount (int8_t group*)*

Gets the number of commands in the specified group.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups

Return values:

integer	is the count of commands (negative in case of errors)

bool dumpCommand (int8_t group, *int8_t* index, *char ** name, *uint8_t &* training*)*

Retrieves the name and training data of a custom command.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group
name	points to an array of at least 32 characters that holds the command label when the function returns
training	is a variable that holds the training count when the function returns. Additional information about training is available through the functions isConflict() and getWord() or getCommand()

Return values:

true	if the operation is successful

int8_t getGrammarsCount (void)

Gets the total number of grammars available, including built-in and custom.

Return values:

integer	is the count of grammars (negative in case of errors)

bool dumpGrammar (int8_t grammar, *uint8_t &* flags, *uint8_t &* count*)*

Retrieves the contents of a built-in or a custom grammar. Command labels contained in the grammar can be obtained by calling getNextWordLabel()

Parameters:

grammar	(0-31) is the target grammar, or one of the values in Wordset
flags	is a variable that holds some grammar flags when the function returns. See GrammarFlag
count	is a variable that holds the number of words in the grammar when the function returns.

Return values:

true	if the operation is successful

*bool getNextWordLabel (char ** name*)*

Retrieves the name of a command contained in a custom grammar. It must be called after dumpGrammar()

Parameters:

name	points to an array of at least 32 characters that holds the command label when the function returns

Return values:

true	if the operation is successful

void trainCommand (int8_t group, *int8_t* index*)*

Starts training of a custom command. Results are available after hasFinished() returns true.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group

Note:

The module is busy until training completes and it cannot accept other commands. You can interrupt training with stop().

void recognizeCommand (int8_t group*)*

Starts recognition of a custom command. Results are available after hasFinished() returns true.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups

Note:

The module is busy until recognition completes and it cannot accept other commands. You can interrupt recognition with stop().

void recognizeWord (int8_t wordset*)*

Starts recognition of a built-in word. Results are available after hasFinished() returns true.

Parameters:

wordset	(0-3) is the target word set, or one of the values in Wordset, (4-31) is the target custom grammar, if present

Note:

The module is busy until recognition completes and it cannot accept other commands. You can interrupt recognition with stop().

bool hasFinished ()

Polls the status of on-going recognition, training or asynchronous playback tasks.

Return values:

true	if the operation has completed

int8_t getCommand ()

Gets the recognised command index if any.

Return values:

(0-31)	is the command index if recognition is successful, (-1) if no command has been recognized or an error occurred

int8_t getWord ()

Gets the recognised word index if any, from built-in sets or custom grammars.

Return values:

(0-31)	is the command index if recognition is successful, (-1) if no built-in word has been recognized or an error occurred

int16_t getToken ()

Gets the index of the received SonicNet token if any.

Return values:

integer	is the index of the received SonicNet token (0-255 for 8-bit tokens or 0-15 for 4-bit tokens) if detection was successful, (-1) if no token has been received or an error occurred

int16_t getError ()

Gets the last error code if any.

Return values:

(0-255)	is the error code, (-1) if no error occurred

bool isTimeout ()

Retrieves the timeout indicator.

Return values:

true	if a timeout occurred

bool isAwakened ()

Retrieves the wake-up indicator (only valid after hasFinished() has been called).

Return values:

true	if the module has been awakened from sleep mode

bool isConflict ()

Retrieves the conflict indicator.

Return values:

true	is a conflict occurred during training. To know what caused the conflict, use getCommand() and getWord() (only valid for triggers)

bool isMemoryFull ()

Retrieves the memory full indicator (only valid after addCommand() returned false).

Return values:

true	if a command could not be added because of memory size constaints (up to 32 custom commands can be created)

bool isInvalid ()

Retrieves the invalid protocol indicator.

Return values:

true	if an invalid sequence has been detected in the communication protocol

bool setPinOutput (int8_t pin, *int8_t* config*)*

Configures an I/O pin as an output and sets its value

Parameters:

pin	(1-3) is one of the values in PinNumber
config	(0-1,5-6) is one of the output values in PinConfig (OUTPUT_LOW, OUTPUT_HIGH) or Arduino style HIGH and LOW macros

Return values:

true	if the operation is successful

int8_t getPinInput (int8_t pin, *int8_t* config*)*

Configures an I/O pin as an input with optional pull-up and return its value

Parameters:

pin	(1-3) is one of the values in PinNumber
config	(2-4) is one of the input values in PinConfig (INPUT_HIZ, INPUT_STRONG, INPUT_WEAK)

Return values:

integer	is the logical value of the pin

void detectToken (int8_t bits, *int8_t* rejection, *uint16_t* timeout*)*

Starts listening for a SonicNet token. Manually check for completion with hasFinished().

Parameters:

bits	(4 or 8) specifies the length of received tokens
rejection	(0-2) specifies the noise rejection level, it can be one of the values in RejectionLevel
timeout	(1-28090) is the maximum time in milliseconds to keep listening for a valid token or (0) to listen without time limits.

Note:

The module is busy until token detection completes and it cannot accept other commands. You can interrupt listening with stop().

void sendTokenAsync (int8_t bits, uint8_t token)

Starts immediate playback of a SonicNet token. Manually check for completion with hasFinished().

Parameters:

bits	(4 or 8) specifies the length of trasmitted token
token	is the index of the SonicNet token to play (0-255 for 8-bit tokens or 0-15 for 4-bit tokens)

Note:

The module is busy until playback completes and it cannot accept other commands. You can interrupt playback with stop().

bool sendToken (int8_t bits, uint8_t token)

Plays a SonicNet token and waits for completion.

Parameters:

bits	(4 or 8) specifies the length of trasmitted token
token	is the index of the SonicNet token to play (0-255 for 8-bit tokens or 0-15 for 4-bit tokens)

Return values:

true	if the operation is successful

bool embedToken (int8_t bits, uint8_t token, uint16_t delay)

Schedules playback of a SonicNet token after the next sound starts playing.

Parameters:

bits	(4 or 8) specifies the length of trasmitted token
token	is the index of the SonicNet token to play (0-255 for 8-bit tokens or 0-15 for 4-bit tokens)
delay	(1-28090) is the time in milliseconds at which to send the token, since the beginning of the next sound playback

Return values:

true	if the operation is successful

Note:

The scheduled token remains valid for one operation only, so you have to call playSound() or playSoundAsync() immediately after this function.

void playSoundAsync (int16_t index, int8_t volume)

Starts playback of a sound from the sound table. Manually check for completion with hasFinished().

Parameters:

index	is the index of the target sound in the sound table
volume	(0-31) may be one of the values in SoundVolume

Note:

The module is busy until playback completes and it cannot accept other commands. You can interrupt playback with stop().

bool playSound (int16_t index, *int8_t* volume*)*

Plays a sound from the sound table and waits for completion

Parameters:

index	is the index of the target sound in the sound table
volume	(0-31) may be one of the values in SoundVolume

Return values:

true	if the operation is successful

Note:

To alter the maximum time for the wait, define the EASYVR_PLAY_TIMEOUT macro before including the EasyVR library.

*bool dumpSoundTable (char ** name, *int16_t &* count*)*

Retrieves the name of the sound table and the number of sounds it contains

Parameters:

name	points to an array of at least 32 characters that holds the sound table label when the function returns
count	is a variable that holds the number of sounds when the function returns

Return values:

true	if the operation is successful

bool playPhoneTone (int8_t tone, *uint8_t* duration*)*

Plays a phone tone and waits for completion

Parameters:

tone	is the index of the tone (0-9 for digits, 10 for '*' key, 11 for '#' key and 12-15 for extra keys 'A' to 'D', -1 for the dial tone)
duration	(1-32) is the tone duration in 40 milliseconds units, or in seconds for the dial tone

Return values:

true	if the operation is successful

bool resetAll (bool wait = *true)*

Empties internal memory for custom commands/groups and messages.

Parameters:

wait	specifies whether to wait until the operation is complete (or times out)

Return values:

true	if the operation is successful

Note:

It will take some time for the whole process to complete (EasyVR3 is faster) and it cannot be interrupted. During this time the module cannot accept any other command. The sound table and custom grammars data is not affected.

bool resetCommands (bool wait = `true`)

Empties internal memory for custom commands/groups only. Messages are not affected.

Parameters:

wait	specifies whether to wait until the operation is complete (or times out)

Return values:

true	if the operation is successful

Note:

It will take some time for the whole process to complete (EasyVR3 is faster) and it cannot be interrupted. During this time the module cannot accept any other command. The sound table and custom grammars data is not affected.

bool resetMessages (bool wait = `true`)

Empties internal memory used for messages only. Commands/groups are not affected.

Parameters:

wait	specifies whether to wait until the operation is complete (or times out)

Return values:

true	if the operation is successful

Note:

It will take some time for the whole process to complete (EasyVR3 is faster) and it cannot be interrupted. During this time the module cannot accept any other command. The sound table and custom grammars data is not affected.

bool checkMessages ()

Performs a memory check for consistency.

Return values:

true	if the operation is successful

Note:

If a memory write or erase operation does not complete due to unexpected conditions, like power losses, the memory contents may be corrupted. When the check fails getError() returns ERR_CUSTOM_INVALID.

bool fixMessages (bool wait = `true`)

Performs a memory check and attempt recovery if necessary. Incomplete data will be erased. Custom commands/groups are not affected.

Parameters:

wait	specifies whether to wait until the operation is complete (or times out)

Return values:

true	if the operation is successful

Note:

It will take some time for the whole process to complete (several seconds) and it cannot be interrupted. During this time the module cannot accept any other command. The sound table and custom grammars data is not affected.

void recordMessageAsync (int8_t index, int8_t bits, int8_t timeout)

Starts recording a message. Manually check for completion with hasFinished().

Parameters:

index	(0-31) is the index of the target message slot
bits	(8) specifies the audio format (see MessageType)
timeout	(0-31) is the maximum recording time (0=infinite)

Note:

The module is busy until recording times out or the end of memory is reached. You can interrupt an ongoing recording with stop().

void playMessageAsync (int8_t index, int8_t speed, int8_t atten)

Starts playback of a recorded message. Manually check for completion with hasFinished().

Parameters:

index	(0-31) is the index of the target message slot
speed	(0-1) may be one of the values in MessageSpeed
atten	(0-3) may be one of the values in MessageAttenuation

Note:

The module is busy until playback completes and it cannot accept other commands. You can interrupt playback with stop().

void eraseMessageAsync (int8_t index)

Erases a recorded message. Manually check for completion with hasFinished().

Parameters:

index	(0-31) is the index of the target message slot

bool dumpMessage (int8_t index, int8_t & type, int32_t & length)

Retrieves the type and length of a recorded message

Parameters:

index	(0-31) is the index of the target message slot
type	(0,8) is a variable that holds the message format when the function returns (see MessageType)
length	is a variable that holds the message length in bytes when the function returns

Return values:

true	if the operation is successful

Note:

The specified message may have errors. Use getError() when the function fails, to know the reason of the failure.

bool realtimeLipsync (int16_t threshold, uint8_t timeout)

Starts real-time lip-sync on the input voice signal. Retrieve output values with fetchMouthPosition() or abort with stop().

Parameters:

threshold	(0-1023) is a measure of the strength of the input signal below which the mouth is considered to be closed (see LipsyncThreshold, adjust based on microphone settings, distance and background noise)
timeout	(0-255) is the maximum duration of the function in seconds, 0 means infinite

Return values:

true	if the operation is successfully started

bool fetchMouthPosition (int8_t & value)

Retrieves the current mouth position during lip-sync.

Parameters:

value	(0-31) is filled in with the current mouth opening position

Return values:

true	if the operation is successful, false if lip-sync has finished

*bool exportCommand (int8_t group, int8_t index, uint8_t * data)*

Retrieves all internal data associated to a custom command.

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group
data	points to an array of at least 258 bytes that holds the command raw data

Return values:

true	if the operation is successful

*bool importCommand (int8_t group, int8_t index, const uint8_t * data)*

Overwrites all internal data associated to a custom command. When commands are imported this way, their training should be tested again with verifyCommand()

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group
data	points to an array of at least 258 bytes that holds the command raw data

Return values:

true	if the operation is successful

void verifyCommand (int8_t group, int8_t index)

Verifies training of a custom command (useful after import). Similarly to trainCommand(), you should check results after hasFinished() returns true

Parameters:

group	(0-16) is the target group, or one of the values in #Groups
index	(0-31) is the index of the command within the selected group

int bridgeRequested (Stream & port)

Tests if bridge mode has been requested on the specified port

Parameters:

port	is the target serial port (usually the PC serial port)

Return values:

non	zero if bridge mode should be started

Note:

The EasyVR Commander software can request bridge mode when connected to the specified serial port, with a special handshake sequence.

void bridgeLoop (Stream & port)

Performs bridge mode between the EasyVR serial port and the specified port in a continuous loop. It can be aborted by sending a question mark ('?') on the target port.

Parameters:

port	is the target serial port (usually the PC serial port)

EasyVR Commander

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

You 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 QuickUSB cable, an adapter board or a microcontroller host board with a running “bridge” program[6] 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, and then go with the “Connect” command.

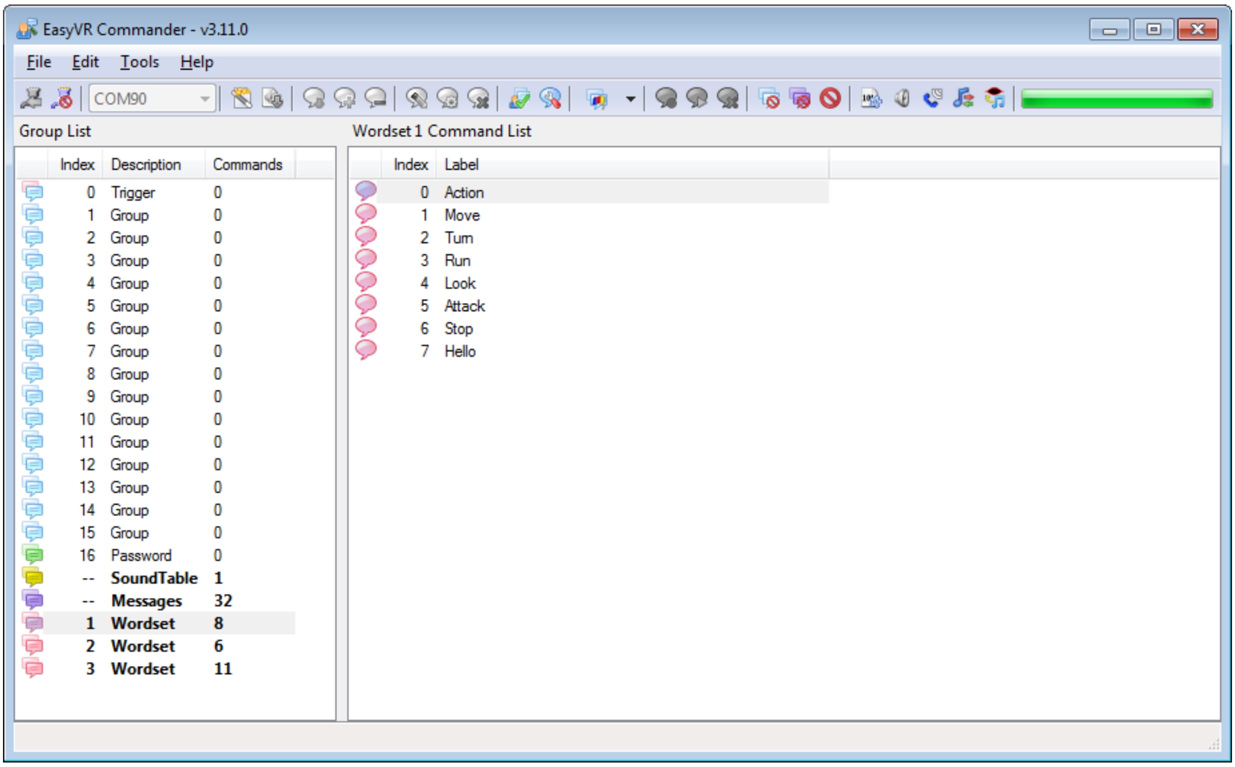


Figure 3 – Main application window

There are five kinds of commands in the software (see Figure 3 and Figure 7):

- *Trigger* - 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, organized in subsets
- *Password* - special group of "voice passwords" (up to 5), using Speaker Verification (SV) technology
- *Wordset* - built-in set of SI commands (for instance in *Figure 3* above, the Wordset 1 is selected)
- *Grammar* - custom set of SI commands (created with *QuickT2SI™ Lite* software).

There are also two other categories of data shown:

- *SoundTable* - list of compressed audio samples (prompts, sounds, etc.) loaded into the module
- *Messages* - run-time sound recordings stored on internal memory

[6] 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.

Remote Connections (Advanced Topic)

The EasyVR Commander can also connect to remote systems, typically Linux PCs or Linux embedded systems (such as BeagleBone, Raspberry Pi, etc.), that expose a remote EasyVR module through the network.

To enable this feature (available since v3.12.0) you need to configure the remote system and manually edit the EasyVR Commander settings.

On Linux based systems, you can use either “socat” or “ser2net” utilities to map the serial port where the EasyVR module is physically connected to a TCP port of your choice.

> **Attention!** The examples below have been tested with specific versions of the hardware and the software; they might not work for your own combination.
> Since modifications to the system configuration are required, this topic is recommended for expert users only!

Configuring the Remote System

Example 1: BeagleBone Black (Rev C)

- Connect the EasyVR 3 module to UART1 on the BeagleBone expansion headers:

EasyVR 3 Module	BeagleBone Black	
GND	DGND	(P9.1)
5V or 3V3 (depending on PWR jumper)	VDD_3V3	(P9.3)
TX	UART1_RXD	(P9.26)
RX	UART1_TXD	(P9.24)

  In alternative, you may use an adapter board for mikroBUS interface, like the *“mikroBUS Cape”* by MikroElektronika, just make sure you set the PWR jumper to the +3V3 position on the EasyVR 3 module and that you plug it in socket 2 or 3 (other sockets use a different UART).

- Install a recent Debian image (e.g. Jessie) on the SD Card or internal e-MMC memory
- Run the following commands on a shell prompt (comments start with #):

```
# install ser2net
sudo apt-get install ser2net

# permanently map TCP port 6666 to UART1
echo 6666:raw:0:/dev/ttyO1:9600 8DATABITS NONE 1STOPBIT | sudo tee -a /etc/ser2net.conf

# enable UART1 pin mux
echo cape_enable=bone_capemgr.enable_partno=BB-UART1 | sudo tee –a /boot/uEnv.txt

# reboot
sudo reboot
```

Example 2: Raspberry Pi Model B+

- Connect the EasyVR 3 module to the Raspberry Pi expansion header:

EasyVR 3 Module	Raspberry Pi	
GND	GND	(Pin 9)
5V or 3V3 (depending on PWR jumper)	3.3V	(Pin 1)
TX	UART_RX	(Pin 10)
RX	UART_TX	(Pin 8)

  In alternative, you may use an adapter board for mikroBUS interface, like the *“PI2 click Shield”* by MikroElektronika, just make sure you set the PWR jumper to the +3V3 position on the EasyVR 3 module.

- Install a recent Debian image (e.g. Jessie) on the SD Card or internal e-MMC memory
- Run the following commands on a shell prompt (comments start with #):

```
# install ser2net (and samba to broadcast hostname)
sudo apt-get install ser2net samba

# permanently map TCP port 5555 to UART
echo 5555:raw:0:/dev/serial0:9600 8DATABITS NONE 1STOPBIT | sudo tee -a /etc/ser2net.conf

# enable serial interface without login shell (menu Interfacing Options > Serial > No > Yes)
sudo raspi-config

# reboot
sudo reboot
```

Configuring the EasyVR Commander

- Locate the folder where the EasyVR Commander settings are stored. Open any Explorer folder, copy-paste this text in the address bar and press Enter:

```
%LocalAppData%\VeeaR\EasyVR Commander
```

- Open the file “Settings.ini” and append the following section at the end:

```
[AddressList]
Count=2
Address0=TCP:raspberrypi,5555
Address1=TCP:beaglebone,6666
```

  Those lines reflect the above examples of remote systems configuration and expect that your PC and the remote systems are on the same local network or anyway that the remote systems can be reached via TCP connection to the specified symbolic addresses (numerical IP addresses are also supported).
- Restart the EasyVR Commander for the new settings to become effective. Now you can choose to connect to one of the remote systems you just added, as if they were regular serial ports.

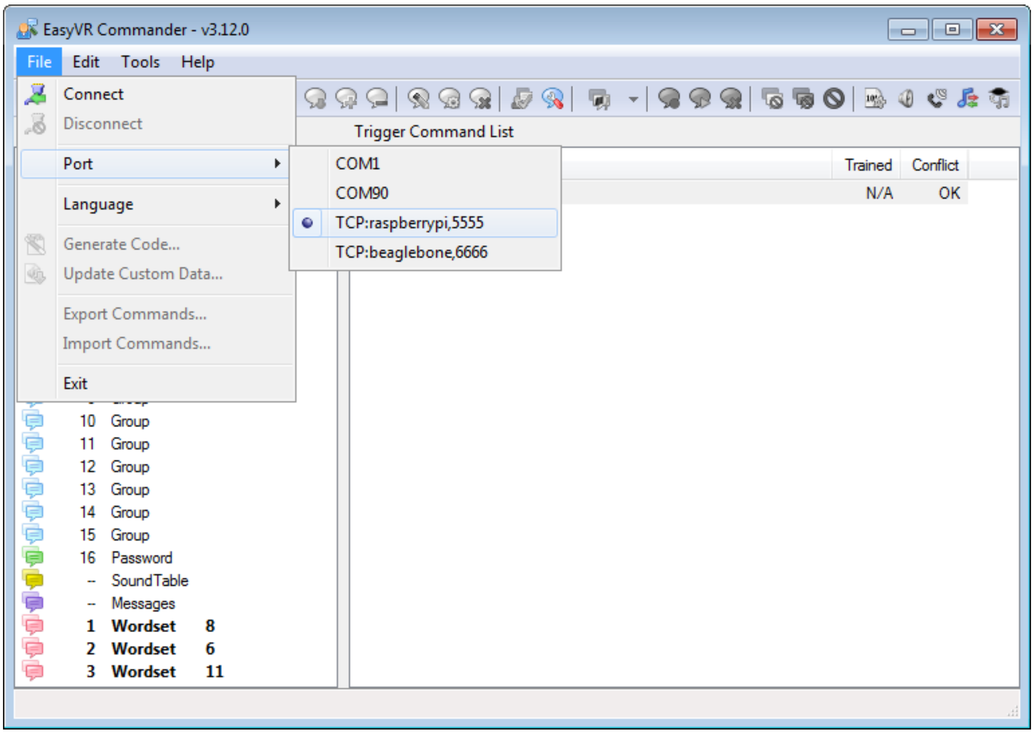


Figure 4 - Remote connections now available

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.

You 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 5) when the *"Train Command"* action is invoked.

> **Note:** Only Latin characters and digits can be used for labels, as well as the underscore character.

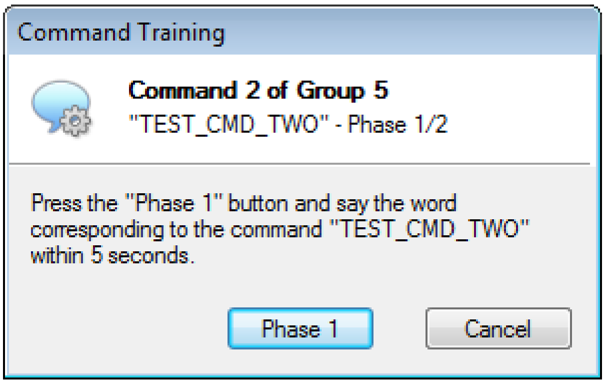


Figure 5 - Guided training dialog

After clicking on “Phase 1” or “Phase 2” buttons, remember you have 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.

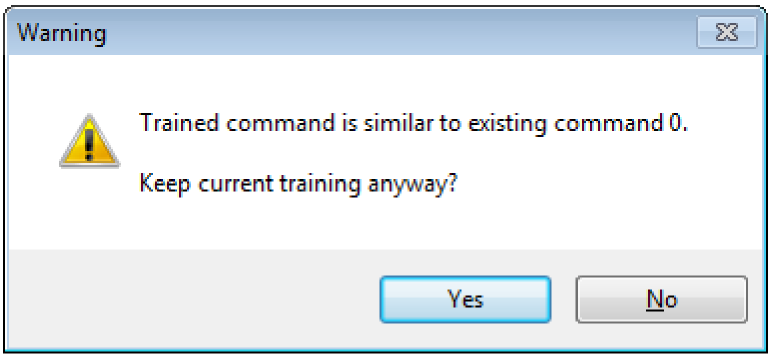


Figure 6 - Alert dialog in case of training conflicts

The software will also alert if a command is too similar to an existing one by specifying the index of the conflicting command in the *"Conflict"* column. For example, in the following *Figure 7* 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.

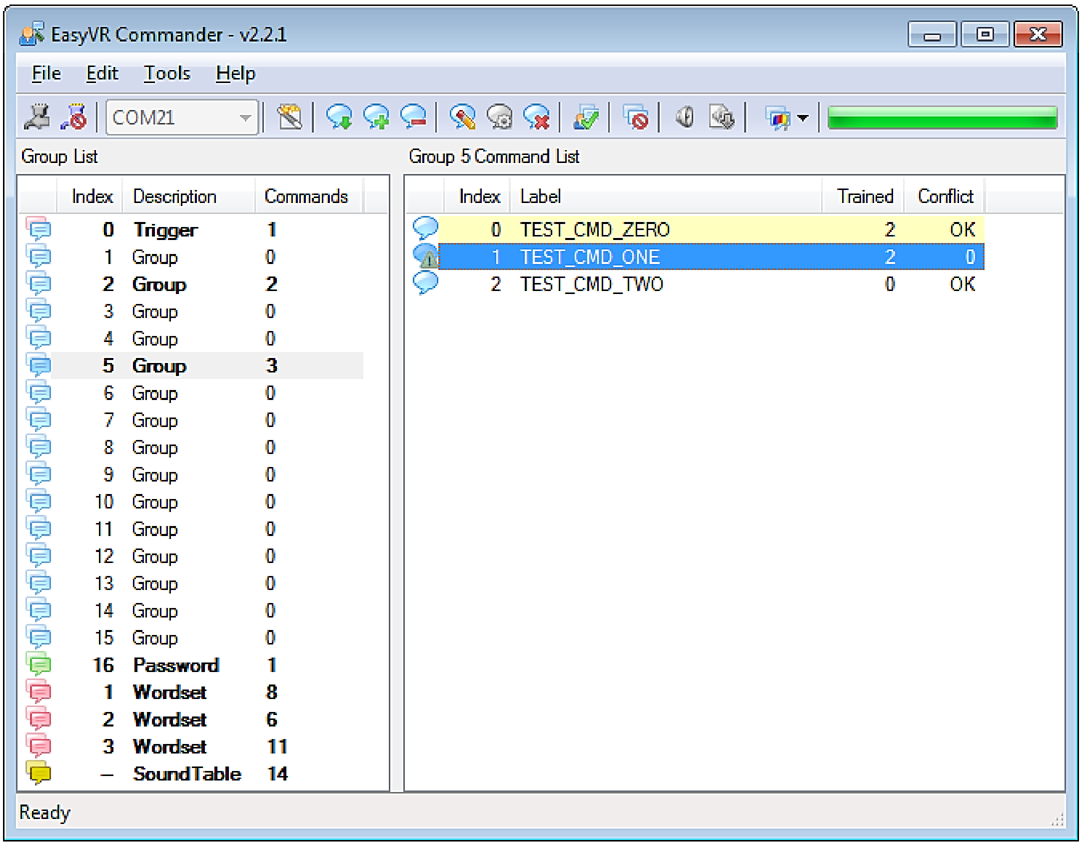


Figure 7 – Showing conflicting commands

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 need to erase the previous training first.

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

Recognition Settings

The module comes programmed with some default settings that can affect voice recognition. These parameters can be altered in those cases where the default values do not offer the best performance.

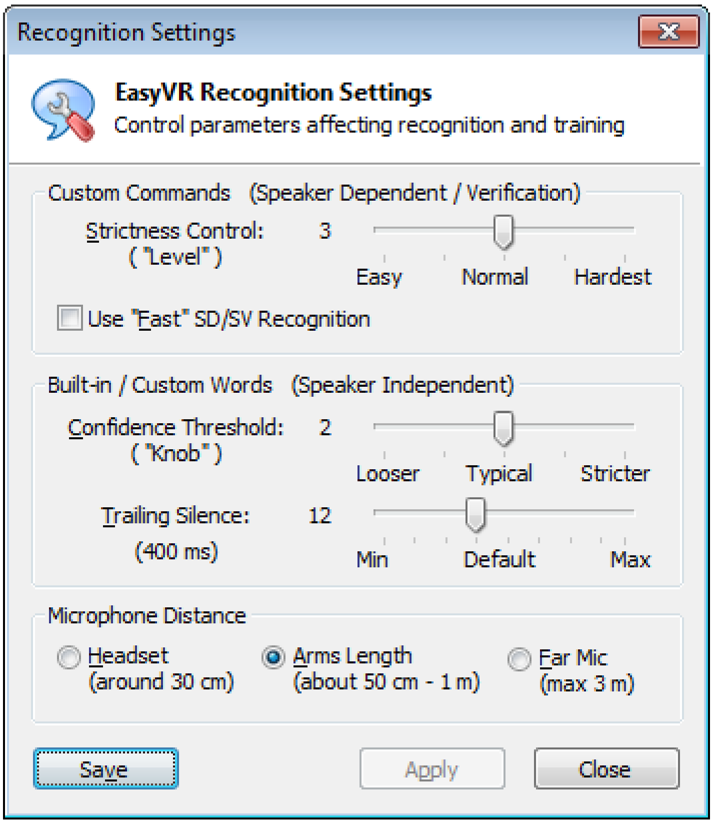


Figure 8 – Interface for changing recognition settings

The "*Level*" and "*Knob*" parameters affect the way recognition results are evaluated and reported, each one for a different kind of voice recognition algorithm (Speaker Dependent/Verification and Speaker Independent, respectively).

Both these values are used for a sort of acceptance threshold: each word or command recognized is assigned a score by the algorithm, which is compared to the threshold.

In some situations the algorithm may flag a correct result as an error or a low confidence result. In those cases you may try to lower the threshold and allow more results to be reported as correct. The drawback is that even words that were correctly refused before now might also be accepted.

The vice-versa is also true: you can increase the threshold to avoid some incorrect words to be reported as good, but then you may also lose a few correct results. So, in the end, you need to find the best compromise.

Ticking *"Use "Fast" SD/SV Recognition"* will use a faster algorithm for SD and SV recognition.
In a similar way, faster SI recognition can be obtained by lowering the *Trailing Silence* (default is 400 ms).

The last parameter affects the internal microphone pre-amplifier and AGC (Automatic Gain Control) stages and is an indication of the expected operating distance of the microphone from the speaker's mouth.

Note: The EasyVR module is optimized for the default distance setting "*Arms Length*".
Any other settings may require hardware modifications to the on-board gain resistor.

To change the recognition settings of the currently connected EasyVR device press the “*Apply*” button. The window is non-modal, so you can test the effects of your changes while leaving it open.

The “*Save*” button makes the EasyVR Commander remember your settings and automatically apply them to every connected device. The module itself does not store any option.

Phone Tones Generation (DTMF)

The EasyVR module is also capable of generating DTMF sounds. This feature can be tested by using the “*Dial Tones*” command in the “*Tools*” menu.

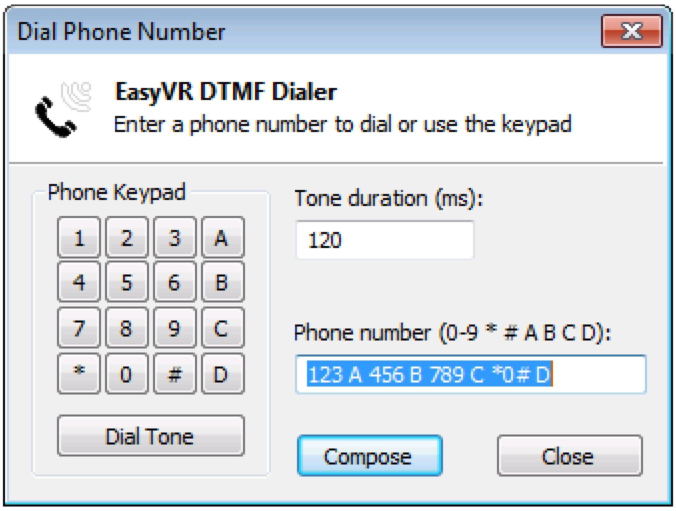


Figure 6 – Interface for generating phone tones

The tone duration can be specified in increments of 40 ms (milliseconds). The dial tone has a fixed duration of 3 seconds (its duration can be modified when programming the EasyVR).

Testing SonicNet™

Another feature available from the "*Tools*" menu is the "*SonicNet*", a wireless communication protocol based on transmission and detection of special sequences of tones, called "tokens".
Two kinds of tokens can be selected: a short version, with up to 16 different tokens, and a long version that provides up to 256 tokens.

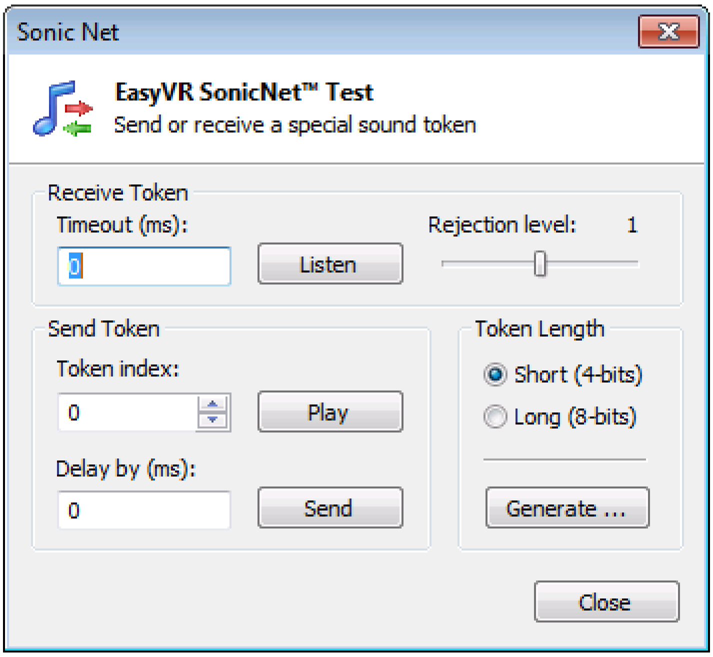


Figure 7 – Interface for testing SonicNet features

The EasyVR module can listen for incoming tokens continuously, or for as long as about 28 seconds (specified with a granularity of around 27.5 ms). Another parameter for token detection is the rejection level that specifies the receiver sensitivity: higher rejection means lower sensitivity that is a lower detection rate, and vice-versa.
When the timeout parameter is set to 0, the module will listen continuously and you can use the "*Play*" button to send a token from your PC soundcard and the "*Stop*" button to stop listening.

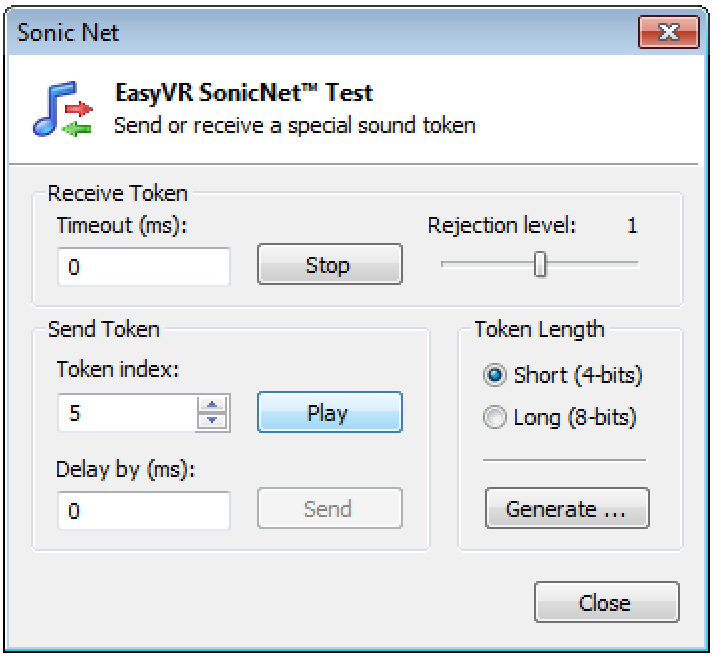
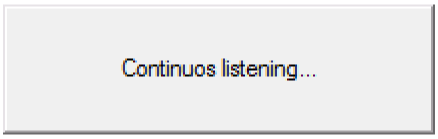


Figure 9 - Modified interface during continuous listening

A pop-up window will display the current state of token detection:



Tokens may also be transmitted from the module with the "*Send*" button. An optional delay parameter can be used to indicate that the token will be mixed with the next sound played from the Soundtable, after the specified amount of time since the playback begins. In this case the SonicNet dialog will close to let you choose a sound to play back.

Note: If you want to mix tokens with a compressed audio sample, you must use a compression scheme with a sample rate of 9.3 kHz when building the Soundtable in the QuickSynthesis™ tool.

If the delay is 0, the token is sent out immediately. Other values can be specified up to around 28 seconds of delay (with a granularity of around 27.5 ms).

Finally, you can also export all the tokens of the specified length to some folder on your PC as Wave files (.WAV format) by using the "*Generate…*" button. You can then use those files to embed SonicNet™ tokens into other software or external sound sources (such as portable players, CDs or DVDs, etc…)

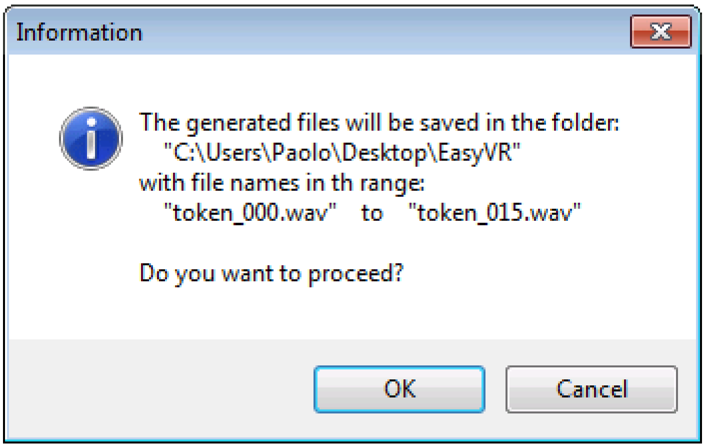


Figure 10 - Export of 4-bit tokens

Real-Time Lip-Sync

Another feature available from the "*Tools*" menu is the "*Real-Time Lip-Sync*". In fact, starting from Firmware revision 4, you can use this feature to let your "animatronic" characters "speak" moving their lips. Clicking on "*Real-Time Lip-Sync*" will open the following window:

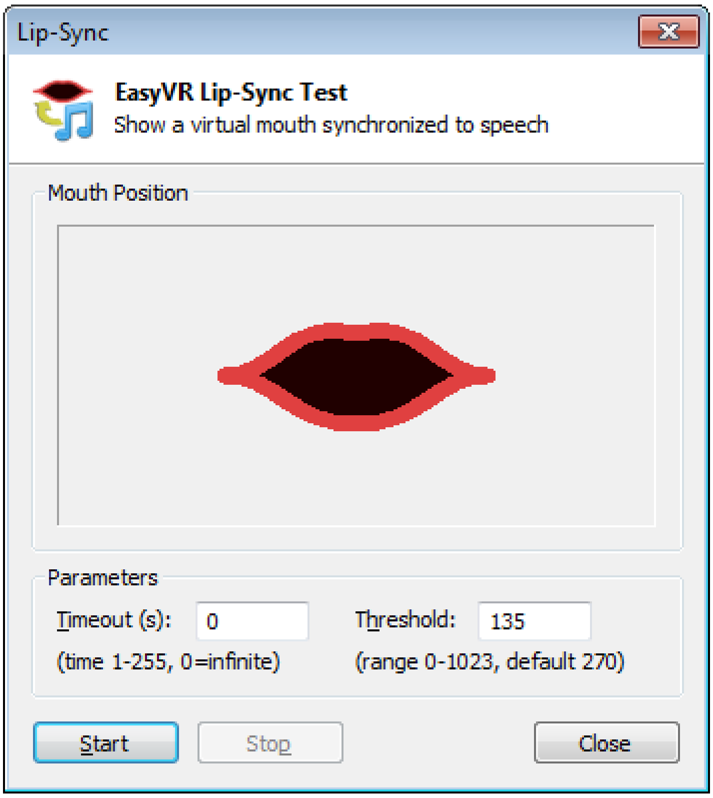


Figure 11 - Lip-Sync interface with animated mouth

If you click on "Start" the virtual mouth on the window will start moving in sync with your voice.
In real *animatronic* characters you can use a servo to move their lips.

You can specify a maximum duration (timeout) or leave it running until you click on "Stop". You may also specify a threshold for the input signal power, below which the mouth is considered closed (silence).

Import and Export of Custom Commands

Starting with firmware revision 4 of the EasyVR 3 module, it is possible to export all the data, including training, of the existing commands and passwords into a file. You can then use that file to import all the commands at once into another module or just as a backup for your commands.

While the module is connected, click on "Export Commands…" under the "File" menu and write the name of the file that will contain the exported commands. In the same way you can select "Import Commands…" and then choose the file with the commands you would like to import.

Note: During the Import procedure all the existing commands will be discarded and replaced by the ones contained in the specified file!

During the Import/Export process some feedback is provided with pop-up windows and with the progress bar on the main window.

Using Custom Data

Sound Table

The EasyVR module can play one of the sounds or sentences saved on its 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™ tool. Please refer to this application's own manual for details about the creation of a sound table. Let's summarize the basic steps here:

- Prepare the audio files you want to include in the sound table in WAV format, uncompressed 16-bit 22050Hz mono. To create the sound files you may use a free software like Audacity for example (http://audacity.sf.net)
- Open Sensory's QuickSynthesis™ 5 and create a new project, specifying "RSC4 family"
- Add your WAV files and specify one of the supported compression scheme (see table below)
- Optionally add sentences, by combining basic WAV sounds. That allows you to save memory when you have speech audio files, if they share some pieces (like "You said" + "One", "You said" + "Two", and so on)
- Build the project with QuickSynthesis™ and use default settings ("*Build linkable module*", "*Load in CONST space*", "*Load above or at: 0*"). You will be asked to recompress new or modified sound files, just confirm and proceed
- Now save your project and build it once again, so that the EasyVR Commander will see that your build is up to date.

The audio compression formats supported by the EasyVR module (from highest to lowest compression rate):

Compression Scheme	Available Time (8kHz 15% silence)	Available Time (9.3kHz 15% silence)
SX-2	20.9 minutes	18.1 minutes
SX-3	18.4 minutes	15.9 minutes
SX-4	16.3 minutes	14.1 minutes
SX-5	14.7 minutes	12.6 minutes
SX-6	13.4 minutes	11.6 minutes
4-bit ADPCM	209 seconds	N/A
8-bit PCM	108 seconds	92 seconds

For audio file containing speech, the SX-3 compression is usually a good choice. If you need higher quality try lower compression rates. Please note that due to the sampling rate used, the audio files cannot contain very high frequencies (less than half the sampling rate).

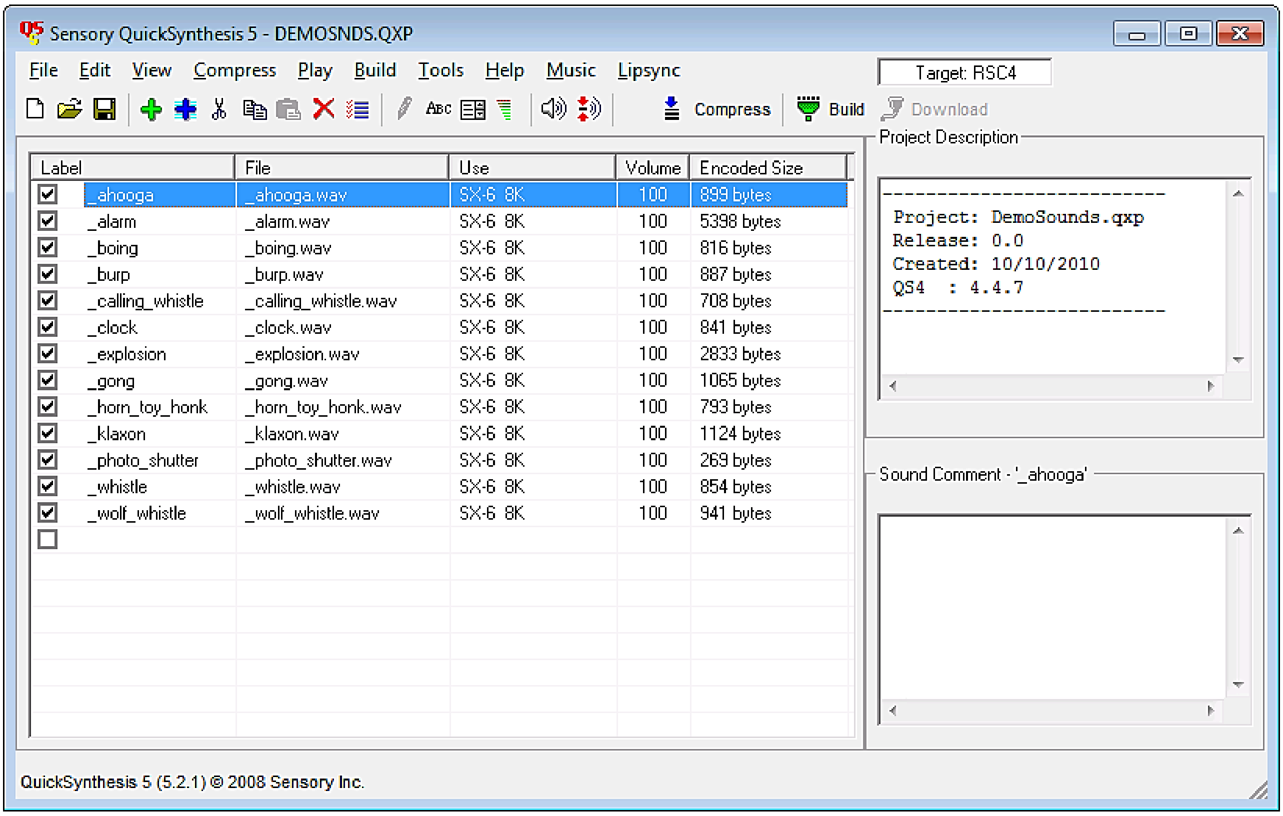


Figure 12 - External tool for creating a Soundtable

Note: Only one Soundtable can be downloaded to the EasyVR module, so make sure you include all the sounds you want to use in a single project.

Speaker Independent Custom Vocabularies

The set of built-in Speaker Independent recognition vocabularies can be expanded with custom grammars, that you can create with the QuickT2SI™ tool (a separate license is required to use the software).

When you create a QuickT2SI™ project, you are presented with a list of words or short phrases (also called “commands”) and an optional trigger word/phrase. The so-called “trigger” is a special set that contains only one word or phrase, with an improved recognition performance, that is used as an entry point for any vocal interaction with a device that is continuously listening to the user’s voice.

If you need to use a trigger word, it is important to carefully choose it so that it has good performance, with very few unintended activations and a high recognition rate. When the user says the trigger word followed by a command, the system can discard unintended activations when the trigger is not followed by a command within a short amount of time (usually around 3 seconds). Moreover, there is only one trigger word to listen to, instead of a list of several commands, so the chance to pick up a random command from background noise or talk is also lower, when using a trigger word.

For assistance on using the QuickT2SI™ Software, please refer to the software help file.

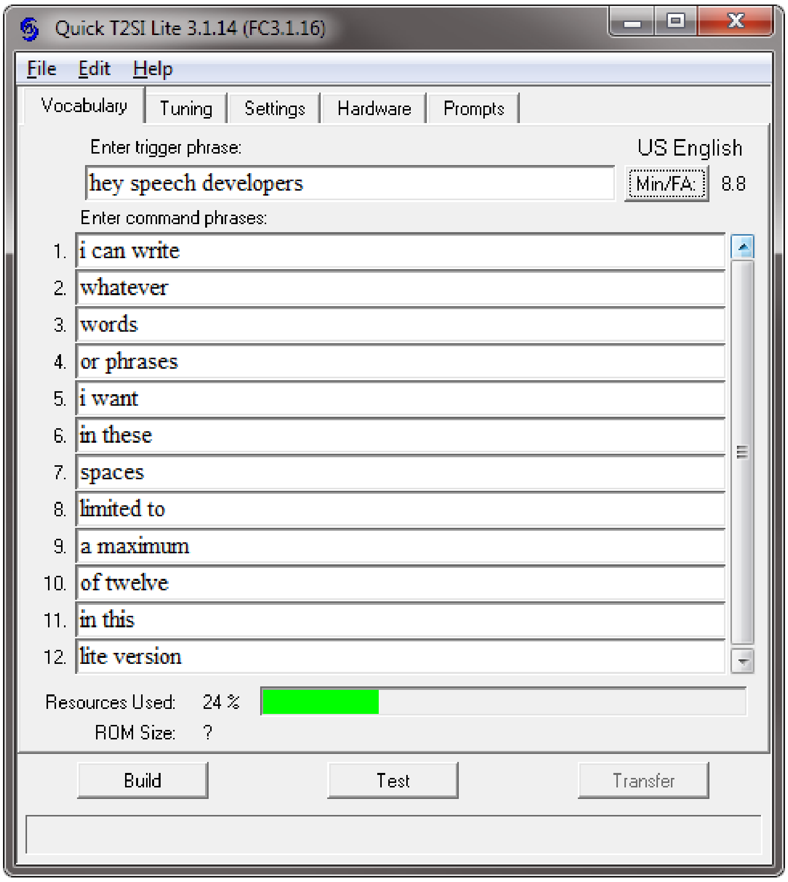


Figure 13 - External tool for custom vocabularies

Several projects can also be combined together if they are using the same acoustic model (language data) using the Acoustic Model Combiner included with the tool. This is useful if you have many command vocabularies, in order to save space in the EasyVR memory.

Updating Custom Data

Once the sound table and/or custom recognition grammars have been created, they can be processed by the EasyVR Commander and downloaded to the module. Note that you 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 Custom Data"* is enabled, either on the toolbar or the *"File"* menu, and it can be used to start the update process. First you are required to list all the QuickSynthesis™ and QuickT2SI™ projects you want to use. A new file containing the specified custom data will be generated and the contents will be displayed, so that you can verify them before updating the module.

Note: The projects must have been built already with the QuickSynthesis™ or the QuickT2SI™ tool, before the custom data generation can be completed successfully. If a recent build is not available you will receive a warning message, the project files can be opened in their respective tools 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 "Refresh" button. If the process completes successfully, the *"Download"* button will be enabled and the flash update process can start.

The default format of generated data is suitable for the EasyVR 3. For previous versions of the module or the shield please make sure to check the option *"Old Format (EasyVR 2.0)"*.

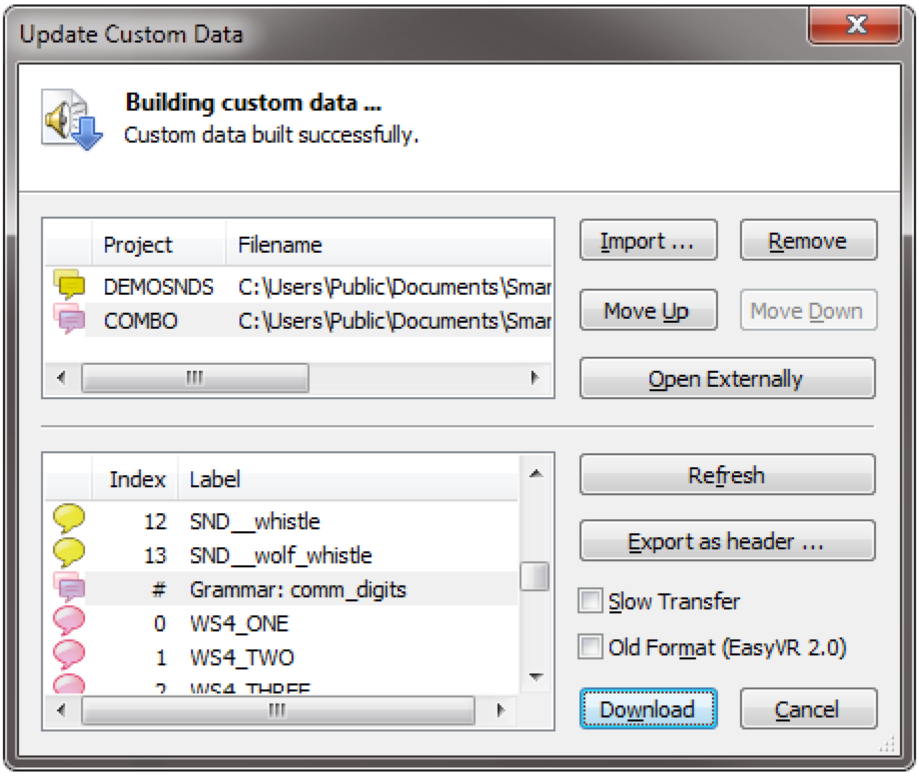


Figure 14 - Interface to build and download custom data

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 bps, but the option “*Slow transfer*” can be used to reduce it to 115200, for better compatibility with slower serial adapters[7]. One adapter that can go to full speed is the QuickUSB cable. Otherwise any USB/Serial adapter with TTL/CMOS interface can be used for updating the flash. The EasyVR Shield can be used for the download, provided that the mode jumper is in UP or LEO position.

Note: Every download will overwrite the previously transferred custom data.

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*.

Custom grammars will be displayed just after the built-in word sets and they work exactly the same way. Trigger words, when specified, will have their own vocabulary with only one entry. You can test and use the custom trigger and command grammars as you do with the built-in ones.

Note: The built-in trigger word set is handled in a special way, as it is active also when recognizing from the first user defined command group. This is the only case where SD and SI commands are mixed together and does not apply to custom trigger vocabularies.

[7] Arduino UNO (and other boards with USB/Serial adapter based on ATMEGA8U2) need the option “*Slow transfer*” enabled

Message Recording

Starting from firmware Revision 1 of the EasyVR 3 module, it's possible to record up to 32 messages. The communication protocol and Arduino library have been updated accordingly (see related chapters).

The first time you update the module with a firmware with message recording capability, the serial flash memory of the module has to be formatted. The EasyVR Commander will automatically identify a module with the memory to be formatted and will alert you with the following window:

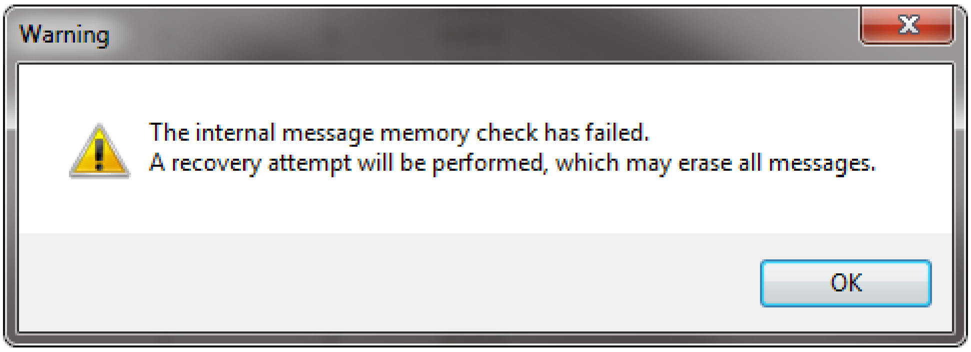


Figure 15 - Alert for corrupted message storage

The same window could appear in case the memory is corrupted for any reason (i.e. the module has been switched off while recording or erasing a message).

With EasyVR Commander v3.10.0 and above you can also record, play and erase messages by clicking on the specific buttons, after selecting the "Messages" group in the Group List.
In order to record a message you have to press the "record button" to start recording and press it again to stop.

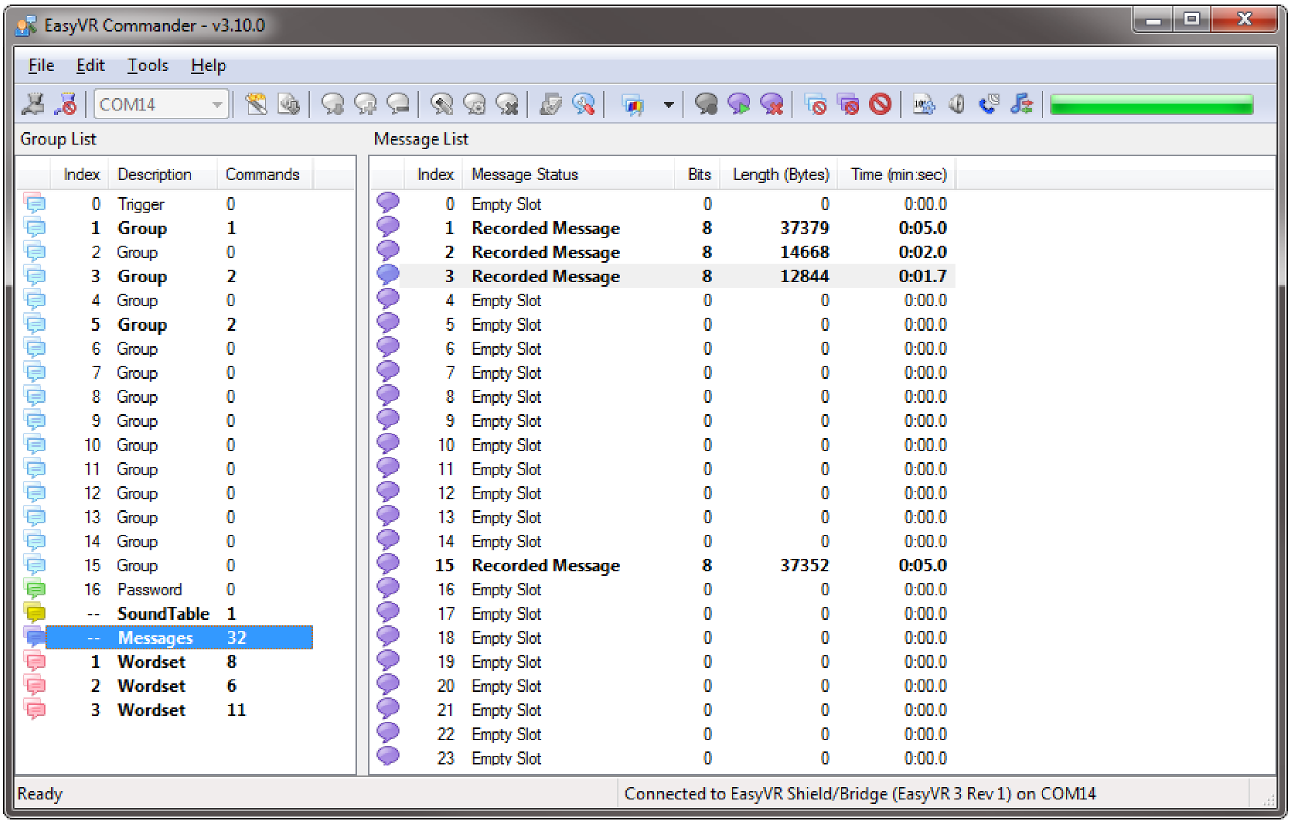


Figure 16 - User interface for messages

Updating Firmware

The EasyVR firmware can be updated in a similar way to custom data by using the command "Update Firmware..." from the "Help" menu. Note that you must first disconnect from the module and do the steps required to start it in "boot-mode" (see the section *Flash Update*).

Firmware files can be found in the EasyVR Commander installation folder (default *C:\Program Files (x86)\VeeaR\EasyVR Commander*), for instance file "EasyVR3_FW_Rev0.EVRFW" is the EasyVR 3 firmware Revision 0.
Please be sure to get the most up-to-date EasyVR Commander in order to have to most up-to-date available firmware as well.

The specified file will be verified as an official firmware release and basic version information will be displayed. If the firmware passes the verification step, then the "Download" button will be enabled.

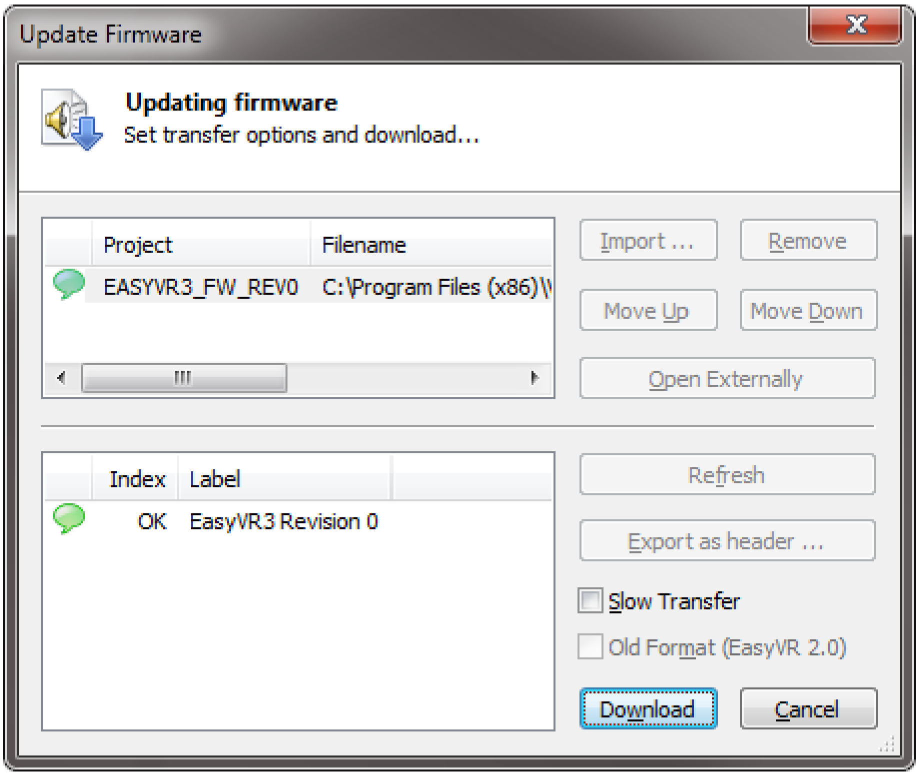


Figure 17 - Interface for updating EasyVR firmware

Note: After a new firmware is downloaded to the module, the custom data already present (sound table and grammars) is erased and it must be downloaded again if necessary.

Important Upgrade Notice

Since EasyVR 3 Firmware Revision 5 the internal memory layout for user data has changed, in order to accommodate space for 32 more SD commands, increasing the total maximum number of commands to 64. The space previously reserved for live message recordings has therefore been reduced by around 20%.

When upgrading firmware to Revision 5 or later (and also if going back to a previous version) you should follow these steps:

- Use the "*Export Commands*" function to save a backup of your SD/SV commands (optional);
- Erase all data from user memory, with the "*Reset All*" command (all the recorded messages and user commands will be deleted);
- Perform the firmware upgrade to Revision 5;
- Use the "*Import Commands*" feature to restore commands from the previous backup (optional).

QuickUSB Adapter Cable

Product Description

The QuickUSB is an USB-to-UART adapter cable, easy to use and supported on all the major operating systems.

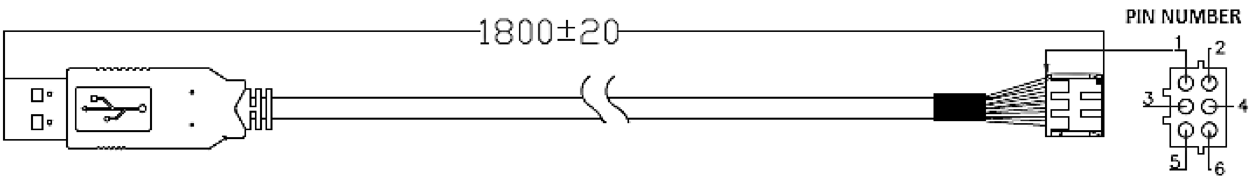
It plugs into a standard USB port and brings all the UART signals to a convenient 6-pin 2mm pitch female connector (Hirose DF11 Series).

QuickUSB Features

- USB 2.0 Full Speed interface
- Full UART (RX/TX and RTS/CTS) at 3.3V
- Data transfer rates from 300 bps to 3 Mbps
- Extended operating temperature range: -40°C to 85°C
- Cable Length: 1.8 m

Technical Specifications

Drawings and Schematics

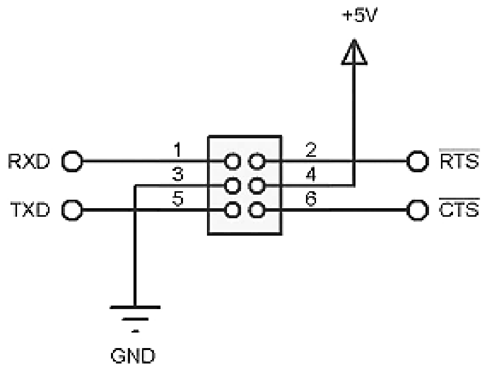


A SIDE

USB Type A Plug
(male connector)

B SIDE

Hirose DF11-6DS-2C
(female connector)



Pin Description

Pin	Type	Name	Description
1	I	RXD	Asynchronous Data Receive line
2	O	RTS#	Request To Send flow control line
3	GND	GND	Power and signal ground
4	Power	VCC	5V power output (USB VBUS)
5	O	TXD	Asynchronous Data Transmit line
6	I	CTS#	Clear To Send flow control line

Operating Conditions

Symbol	Parameter	Min	Typ	Max	Unit
VCC	DC Power Output (VBUS) *	4.5	5.0	5.5	V
Ta	Ambient Operating Temperature Range	-40	25	85	°C

(*) Output current might be limited by USB Host and internal adapter settings

Electrical Characteristics

These are applicable to pins RXD, CTS.

Symbol	Parameter	Min	Typ	Max	Unit
V_{IH}	Input High Voltage	1.5		3.3	V
V_{IL}	Input Low Voltage	0.0		1.0	V

These are applicable to pin TXD, RTS.

Symbol	Parameter	Min	Typ	Max	Unit
V_{OH}	Output High Voltage (I_{OH} = -1 mA)	2.2	2.7	3.2	V
V_{OL}	Output Low Voltage (I_{OL} = 2 mA)	0.3		0.5	V

Quick Start Instructions

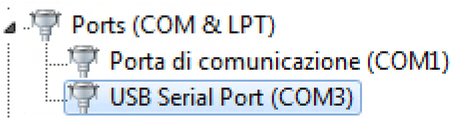
Software Setup

The first time you want to use the QuickUSB you need to install the required software drivers for your Operating System. Please follow this procedure (one-time only):

1. Plug the USB end of the cable to your PC and leave the other end unconnected
2. If you have a recent OS version please allow the system to search for updated drivers:
    a. If the setup was successful, unplug the cable, the procedure is complete
    b. If the setup failed, unplug the cable and continue to the next step
3. Download the latest official drivers for your OS version from the FTDI VCP Drivers page
4. Run the driver setup package, or follow any alternate installation procedure accompanying the download
5. After the software setup is complete go back to step 1 above, the OS should now be able to find the required drivers automatically

Using the Adapter

Once the required software drivers are correctly installed and configured by the Operating System, you should not need to repeat the above procedure anymore.



Just plug the cable on the target board first, then plug the USB connector to a free port on your PC.

The adapter will be visible as a new USB Serial Port device and a virtual COM port on Windows. This COM port number is the one to use with the EasyVR Commander.

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 FAQ for a possible solution: http://www.veear.eu/faq

If you cannot find an existing solution on the FAQ, please contact us using the contact form on our website at http://www.veear.eu/support. The more detail you provide, the better support we can give.

All VeeaR branded boards and software are manufactured by RoboTech srl

RoboTech srl assumes no responsibility for any errors, which may appear in this manual. Furthermore, RoboTech srl reserves the right to alter the hardware, software, and/or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. RoboTech srl products are not authorized for use as critical components in life support devices or systems.