EasyTerm
User manual
v2.1.1

Table of Contents

Document versions

Version	Release date	Changes
1.0.0i		• Initial non-public version
1.0.1i	15.7.2022	• Add chapter describing update procedure of FW via USB using STM32CubeProgrammer app • Add description of checkbox labeling feature (text, textOrigin, untickTextColor and tickTextColor parameters) • Add description of different checkbox styling possibilities with examples and new backgroundColor and checkboxWidth parameters) • Add description of text line decoration feature (lineDecoration, lineDecorationColor and lineDecorationThickness parameters) • Improvements, fixes
1.0.2	29.8.2022	• Add description for multi-action feature for button widget (action selectable by user) with multi-label possibility. • Update DAC interface API. • Add 10b addressing parameter to I2C. • Add different bit order feature to SPI (LSb first). • Improvements, fixes • First public release
1.1.0	25.11.2022	• Add initialized parameter to several interfaces • Add I2C scan features • Fixed descriptions – SPI and UART transaction sizes are in words not bytes (as transaction can have more than 8 bits) • Fixed default width/heights descriptions for widgets where automatic dimension calculation applies • Add TextMargin parameter to checkbox
1.2.0	26.01.2023	• Add expression-based read format for interfaces • Add list of error messages with tips and descriptions • Changed UART txrx functionality and improved descriptions on UART transactions. • Add form float parse mask with configurable precision • Add description of command/reply pairing option • **I**nitialized changed to **I**nit • Comm. interfaces **R**ead**F**ormat must be enclosed in quotation marks from now

1.3.0	8.4.2023	• Add ParseMask parameter to slider to readjust it automatically when message is parsed on some value • Button, checkbox and form label text can be multi-line using '\n' (same as text widget) • Add LineSpacing parameter to all widgets with multi-line capability • Add ContinueOnActionError system command that allows to continue processing of multi-command actions when previous command has failed (action parameter of buttons, sliders, checkboxes, trigger or timer) • Add Wait system command to delay EasyTerm from processing other commands in a blocking manner • Slider TrackColor parameter renamed to OutlineColor • Removed multiline parameter from text widget as it is always multi-line when '\n' characters are used • Add ERR-CMD-PARAM_STRING_TOO_LONG error • Add Quick start chapter • Add DTR signaling requirements description to Master connection configuration chapter
2.0.0	2.10.2023	• Add WIFI chapter with related commands and remote control description • Add MODBUS chapter with related commands • Add ActionOnParse command for extracting messages content and executing configured action with it • Add DisableInternalCommandEchoing and UseAlternateHostUart parameters to SYStem command. • Add notes that quotes that are not the beginning or the end of command parameter body must be escaped by '\' • Corrections
2.0.1	4.12.2023	• Add missing WIFI commands • Improvements/additions to WIFI module description chapter • Add factory reset procedure to buttons description in "Hardware description" chapter
2.1.0	24.06.2024	• Add Image widgets chapter • Add hardware ver.1.1 description, images and specifications • Add DisableRepliesToUart, DisableRepliesToUsb and DisableCommMirror parameters to system command • Add important usage tips to Quick Start chapter
2.1.1	14.11.2024	• Add HOST UART interface (top-side connector) vs UART interface (side connector) clarification to Quick Start tips

Safety precautions

To prevent possible electrical shock, fire, personal injury, or damage to the product, carefully read this safety information before attempting to install or use the product. In addition, follow all generally accepted safety practices and procedures for working with and near electricity.

The product has been designed and tested in accordance with the harmonized standard publication EN 61000-4-2,3 and EN 55032. The product left the manufacturer in a safe condition.

This product can be used exclusively in indoor locations (office, home or laboratory). Device certification was not performed in accordance with industrial standards so industrial use is not permitted. Using device for critical applications (those in which failure can cause any type of harm or damage) is strictly forbidden. Keep device out of liquids and extreme conditions such as temperatures below 0˚C or above 70˚C. Operate device in conformity with its specifications - do not exceed voltage ranges, follow correct polarity and avoid mechanical damage by careful handling. Do not disassemble device unless approved by manufacturer. Use power supplies with overload/over-current protections to prevent risk of fire damage in case of device malfunction. These are typically protected PC USB ports, protected USB chargers, regulated power supply units or batteries with embedded over-current protection. Usage with batteries without such protections can result in risk of fire in case of device malfunction (mainly when unprotected Li-Ion battery cells are used).

What is EasyTerm

EasyTerm is a device that offers to embedded developers, testers and hobbyists whole new ways to evaluate, control and develop their projects. EasyTerm is also a multi-tool with rich interfacing capabilities. It is controlled by a user-friendly set of ASCII commands giving user an easy and time-efficient access to various HMI (human-machine-interface) features via LCD display, diverse communication interfaces like UART, SPI and I2C and mixed signal features through GPIOs, ADC, DAC and PWM. Reading external device register, generating PWM signal, converting a voltage, displaying a plot, sending a message are tasks accomplished in a few seconds.

EasyTerm can perform various actions by:

- transmitting commands to EasyTerm via PC or SBC (single-board computer) through USB using terminal application (or CommandBuilder app) or using a programming language like python providing scripting capabilities through pyserial library.
- transmitting commands to EasyTerm via user device provided that it contains an MCU with an accessible UART interface (no library integration is needed)
- executing commands internally by building-up a GUI using HMI features and assigning commands to widget actions. Pressing a button can be the same as receiving one or many commands. Such commands can be parametrized via usage of slider or keypad widgets.

Commands can be composed by user manually or by using CommandBuilder app (no manual or API knowledge is then needed).

EasyTerm also offers low-power consumption mode that is beneficial for logging purposes (LCD turned off) or for interfacing purposes and thus can be power-supplied via batteries when LCD is used infrequently. EasyTerm can then be considered as a logger or as a UART/USB to "some" interface bridge.

You can use HMI features to evaluate your device using EasyTerm by processing incoming messages from the device under evaluation using widgets like forms or plots. You can control your device by composing and transmitting messages using widgets like sliders, keypads, buttons or check-boxes.

You can benefit from the coexistence of HMI and rich interfacing features and thus use the EasyTerm for example as a touch-controlled PWM generator, as a signal generator, as a voltage

measurement tool with plotting capability, etc.. Parameters can be set via sliders or keypads. Results of actions can be displayed inside forms or plots. You can easily turn the EasyTerm into a flexible multi-tool with tons of possibilities. GUI layouts can be saved to memory and recovered after power-up – thus entire functionality can be defined once through PC and then the EasyTerm can be used remotely (as a standalone device performing commands as widgets are handled by you via touch screen).

We are offering example layout scripts downloadable via EasyTerm resources web-page (touch controlled PWM generator, signal generator, SPI master, I2C master, logger etc...). We strongly recommend to try it out as a example and guide how things can be done.

HMI offers these features:

- **Basic widgets** like buttons, checkboxes, sliders, forms, texts with user-predefined actions, behavior and appearance
- **Logging window widgets** (terminal-style logging) with non-volatile recording feature, recover feature, timestamps, color-styling.
- **Plot widgets** with different modes of operation, automatic scaling and offset rejection. Tracking or manual cursors. For both float-encoded and integer-encoded values.
- **Keyboard widgets** for filling parts of commands/messages with hexadecimal, decimal or string parameters
- Multi-screen support of widget layouts that can be saved and recovered easily.

See **HMI description** chapter for more details and examples

External interfaces offers these features:

- **GPIOs** easily configurable as inputs, outputs or as interfacing peripherals. Configurable pull-ups, pull-downs. Selectable open-drain or open collector output type.
- **ADC** for conversions of external channel voltage and battery voltage. Supported are one-shot conversions and continuous conversions with configurable sample rate and sample count.
- **DAC** for constant voltage conversion, user-predefined pattern generation (AWG) with configurable samples, boundaries and sample rate, function generation (AFG – sine, triangle, saw) with configurable frequency and amplitude.
- **PWM** with configurable duty cycle, frequency and polarity. Supports various parameters configuration methods (duty cycle configurable by percents or by defining phase times, rate

configurable via frequency or period). Supports lockable phase time or frequency for automatic parameters calculation.

- **TRIGGER** with configurable action (command(s)) that are executed when configured signal edge is detected on trigger pin.
- **UART** with configurable baud-rate, TX and RX level inversions. Transmitting byte sequence or ASCII strings, configurable RX timeout and on-background reception.
- **SPI** master with configurable frequency, polarity and phase. Configurable dummy byte and ignore byte count for read transactions. Manual or automatic chip select assertion.
- **I2C** master with configurable frequency and slave address. User-friendly write-and-read operations.
- **3.3V power supply** for user purposes that can be turned on/off via command
- All interfaces has configurable read data formats (hexadecimal, decimal, binary), byte swapping, byte separation for purposes of greater readability

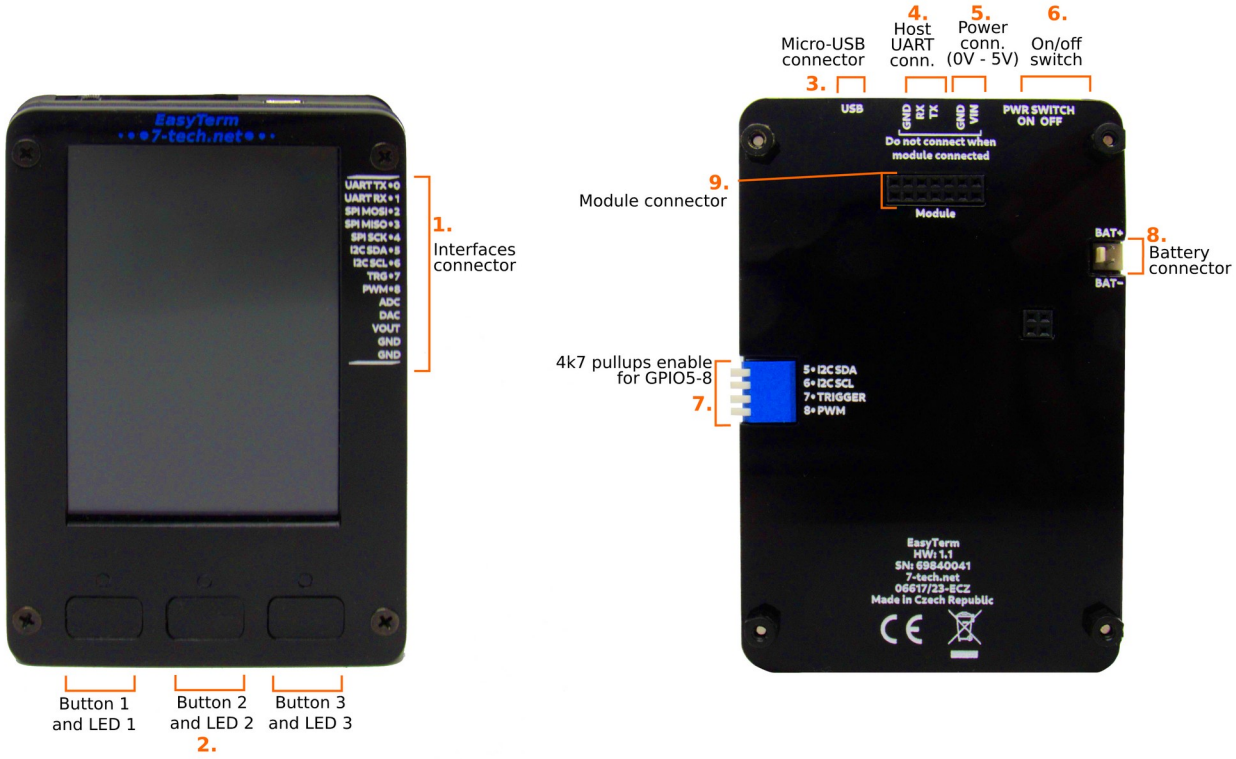
EasyTerm is a device that

- offers a quick and easy way to implement various HMI layouts providing control and evaluation of user device or internal interfacing features
- offers various interfaces for diverse tasks that are in many cases very time consuming and not very convenient (for example read registers from some SPI or I2C IC, set GPIO pin level, read voltage, generate voltage, generate PWM...). For these simple tasks this device can replace the need for many bulky devices and thus helps to keep table tidy and work effective.
- offers long-term logging to non-volatile memory for future analysis through logWindow widget or through host (e.g. PC terminal application)

EasyTerm is not a device that

- can be used for building fancy graphical user interfaces with spectacular effects. EasyTerm is focused on productivity and user comfort.
- can be used for advanced tasks like high accuracy or high speed measurements.

Hardware description



Identifier	Name	Description
1.	**Interfaces connector**	Interfaces connector – 14 pin connector with access to various interfaces. Each pin can be configured as an input, output or peripheral (ADC and DAC are peripheral-only). Possible peripheral functions are UART, SPI, I2C, PWM, TRIGGER, ADC, DAC and VOUT offering 3.3V voltage output that can be activated via command.
2.	**Buttons and LEDs**	Three mechanical buttons. Their actions can be set via specific system commands. Default actions are: Button 1 – Move to screen page with lower index, button 2 – move to screen page with higher index, button 3 – toggle between sleep mode and awake mode (turns LCD on or off). Factory reset can be performed by holding all 3 buttons when device is off and then turning device on while still holding all 3 buttons. Buttons then can be released.

3.	**Micro-USB connector**	Micro-USB connector. Device supports VCP class. Commands can be send via this interface from various terminal applications or user applications. If device is in bootloader mode, USB is supporting DFU (device firmware update) protocol. Device can be powered through this connector.
4.	**Host UART connector**	UART connector that can be connected to master (often some micro-controller or FTDI chip). Commands can be send via this interface from various terminal applications, user applications or through user device from its embedded FW.
5.	**Power connector**	Power supply connector. Input voltage range is 2.5V - 5V. Polarity protection is integrated. Do not excess voltage limits even when on/off switch is in off position! Device can be powered through this connector.
6.	**On/off switch**	Toggle switch that turns device on/off. Some circuitry is powered even when device is off consuming <1uA current.
7.	**External 4k7 pull-ups switch**	Dip switch for connecting or disconnecting individual 4.7 kOhm pull-ups to G5, G6 (I2C) and G8 (TRG) and G9 (PWM). These pull-ups are recommended for usage with I2C interface as command switchable pull-ups (Gx PinPull command) tends to have higher resistance that is not suitable for such purpose.
8.	**Battery connector**	Connect batteries to this connector. Input voltage range is 2.5V – 5V. Device can be powered through this connector. Do not excess voltage limits even when on/off switch is in off position!
9.	**Module connector**	Connect WIFI module to this connector. Connect when EasyTerm is turned off. Connector has dual rows – it is for redundancy – it does no matter to which row you will connect the module.

Specifications

Parameter	Value			Units	Note
	Min	Typ	Max		
LCD resolution		320x480		pixels	3.5" LCD screen with capacitive touch panel
Power-supply voltage (V_{IN} connector, V_{BAT} connector)	2.5		5.5	V	Do not exceed specified range even when on/off switch is in off position.
Supply current (no sleep mode)	-	125	-	mA	Typical value. Device powered by voltage of 3.3V. LCD brightness set to 50%.
Supply current (sleep level 0)	-	2 (after 30s)	-	mA	When powered by 3.3V and measured after 30s after sleep level 0 entered (touch controller reduced its scanning frequency).
Supply current (sleep level 1)	-	540	-	uA	When powered by 3.3V.
Supply current (sleep level 2)	-	85	-	uA	When powered by 3.3V.
Additional consumption when VOUT (3.3V) PSU enabled	-	65	-	uA	When powered by 3.3V.
ADC inaccuracy	-	0.5	5	mV	Tested across several units in whole code range after factory calibration. Typical value is result of averaged errors (absolute values) across whole range. Maximum value is maximum error across whole code range with some margin.
DAC inaccuracy	-	0.5	5	mV	Tested across several units in whole code range after factory calibration. Typical

					value is result of averaged errors (absolute values) across whole range. Maximum value is maximum error across whole code range with some margin.
ADC sample frequency for continuous conversions	1 Hz	–	2 MHz	–	
DAC sample frequency for continuous conversions	1 Hz	–	2 MHz	–	
Maximum ADC input voltage	3.25	3.3	3.35	V	Not causing data saturation.
Maximum DAC output voltage	3.25	3.3	3.35	V	
GPIO output high-level voltage	3.25	3.3	3.35	V	No load.
Maximum allowed voltage on all extension connector pins	–	5	5.5	V	
VOUT voltage value	3.25	3.3	3.35	V	No load.
VOUT maximum load current	–	250	–	mA	Approx. 100mV drop is observed at 250mA load, 50mV drop at 100mA

WIFI module description

Communication via internet is supported by using optional WIFI module consisting typically of:

- Version 1: WEMOS D1 MINI development board with ESP8266EX chip and adapter specifically designed to be used as WEMOS D1 MINI base board
- Versions 2 and 3: "Whole unit " module (black) containing ESP32C6 with thinner profile and adapter to be able to place module underneath the EasyTerm.

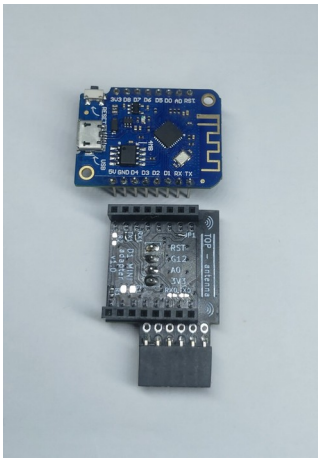
It allows several features dramatically increasing flexibility of EasyTerm:

- Remote control – control EasyTerm via the internet. It uses MQTT under the hood. User configures command and reply topics that can be used to communicate via many applications offering MQTT client capabilities – CmdBuilder, Node-RED, various Android apps etc...
- HTTP GET/POST operations – controlling and monitoring various smart IOT devices and web accessible resources
- MQTT publish/subscribe – ideal for smart home/lab/process ecosystem

Modules contains connectors that allows connection to EasyTerm through HOST UART and powering EasyTerm via VEXT connector. Power scheme is flexible:

- User may power WIFI module via USB cable and it will power EasyTerm via VEXT connection. Connecting EasyTerm via USB at the same time is safe thanks to power management circuitry.
- User may power EasyTerm via all means possible (USB, VBAT connector) and then user may power WIFI module through EasyTerm VOUT connector after executing "SYS VoutState=1" command. This method allows to disable WIFI module power completely by disabling VOUT through "SYS VoutState=0" command and thus conserve power instead of performing WIFI Sleep command.

Module version 1 consists of WIFI adapter (i. e. baseboard) and WEMOS D1 mini module. Adapter maintains connection between HOST UART and VEXT connectors and WIFI module. Adapter has one of pin sockets locked by dummy pin for correct connectors alignment.



WIFI adapter and module

Connected module and adapter

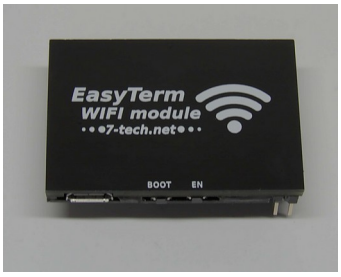
EasyTerm with WIFI

Module version 2 (black) is thinner alternative to version 1 and it is not separable (no longer an adapter with base board but rather a whole unit). Correct connection with EasyTerm is also enforced via dummy pin lock. It comes with simple adapter to be able to place WIFI module underneath the EasyTerm and thus be hidden.

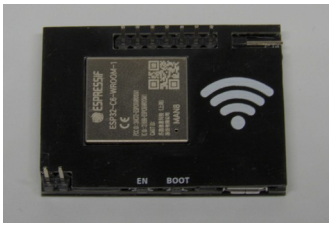
Connected module

Adapter to place module underneath the EasyTerm

Module version 3 (black) is the latest version compatible only with EasyTerm HW version 1.1 due to its connector on the back. It can be connected only to the backside of EasyTerm. Power is now directly applied through that connector (external power-supply or VOUT power wiring through cable is no longer needed).



Module front



Module back

EasyTerm with module

See WIFI interface chapter for more information.

Possible power configurations

Device can be powered from three power inputs. Voltage applied to these inputs must be in range between 2.5V – 5V. Do not excess this range even when on/off switch is in off position (power supply ORing circuitry is still operating).

- V_{IN} connector (5.)
- Micro-USB connector (3.)
- Battery connector (8.)

Voltage (in a specified range 2.5V - 5V) can be applied to more than one of these connectors at the same time (even via all three connectors). Device has special circuit (aka power-supply ORing circuit) that prevents current flow from power supply connector with applied voltage to connectors with applied lower-value voltage. If device is connected through USB, current will be drawn only from this connector. If USB cable is not connected, the device will by default prioritize power from V_{IN} connector over battery connector. All of these connectors are polarity-protected. Short-circuit may however happen when EasyTerm is powered by several non-isolated power-supplies when polarity on one connector was reversed (current will flow from positive PSU rail of one supply to GND rail of second power-supply).

Master connection configurations

Device can be controlled remotely via ASCII commands via two ways

- USB connector – Virtual COM port (3.)

  typical case: Master is a PC or a single board computer (SBC)

- UART connector (4.)

  typical case: Master is an embedded device

EasyTerm can be controlled via these two interfaces – there is no switching performed. Device listens at both of these connectors at the same time. If USB cable is disconnected, EasyTerm will switch this interface to low-power mode to conserve power. On GNU/linux systems there may be additional required steps to be able to communicate through USB such as adding user to "uucp" group (e.g. Arch linux) or "dialout" group (many other distributions such as Ubuntu).

Example of such command is:

```
sudo usermod -aG dialout username
```

Windows shall install VCP driver automatically after connection of EasyTerm to PC via USB cable.

Common terminal applications can be used such as EasyTerm's CmdBuilder, Script Communicator, Putty, Hercules etc. EasyTerm uses DTR signaling when communicating via USB to detect if COM port is opened or closed. In most cases DTR is asserted correctly (as it should be...) by default even when no specific flow control is used. Please make sure that this signaling is enabled otherwise EasyTerm wont sent replies. This feature can be enabled in commonly used applications as follows:

Putty terminal – works out of box

Script communicator – DTR checkbox must be checked (default state)

Hercules – DTR checkbox checked (not default state)

Termite – DTR/DSR selected as a Flow control (not default state)

Library like **pySerial** (python serial library) works out of box and **Node-RED** serial nodes as well.

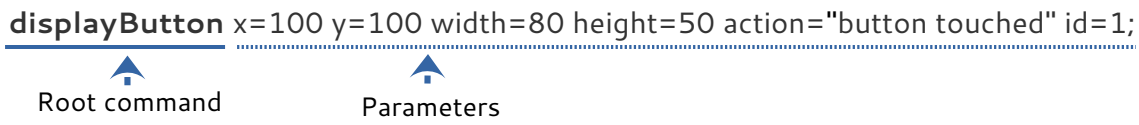
Please contact us in case of any problems.

Command protocol description

Device control is performed via usage of specific ASCII commands. They were designed to be user-friendly, easily understandable and self documenting.

Command has two part – first part – the root command and a seconds part – parameters or sub-commands.

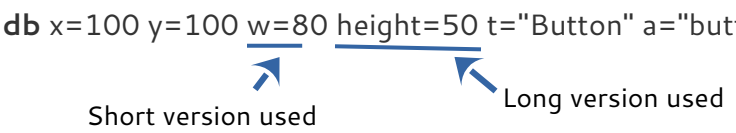
For example, lets demonstrate it on display button root command (long version is demonstrated)



Sending this command causes button with id=1 to be displayed on specified coordinates with specified size. On click, it will transmit „button touched" string to master.

Both root commands and parameter are case **insensitive**

Many root commands and parameters has also a **short version** that is more compact but less self-documenting. Long and short versions can be combined:



Commands are separated from each other by any of these means:

- **time space** between reception each of them (currently approx. 20ms)
- character **;**
- **LF** (aka line-feed) special character. This character is specified as ‚\n' in many programming languages
- **CR** (carriage-return) special character. This character is specified as ‚\r' in many programming languages

Multiple commands can be sent immediately after each other using these separation methods. So lets display two buttons. Commands will be separated by „;' character. Clicking on the second button will remove the first one (action is **R**emove**B**utton root-command). It is also possible to use CR or LF special characters – specified as '\n' or '\r' (see first button action command for example) – quotes will prevent evaluating them as command separators.

db x=100 y=100 w=80 h=50 t="1" a="button touched\n" id=1; **db** x=10 y=10 t="2" a="rb id=1" id=2;

Many parameters are optional as you can see that we skipped width and height parameters for second button.

Some root commands (mainly those dedicated to interfacing) have also **readable parameters**. Lets configure I2C address and read it back immediately by appending question mark after equals mark.

```
I2C address=A8 address=?;
```

EasyTerm will respond:

```
I2C a=A8\r\n
```

Commands/reply pairing:

Commands can be paired with replies via usage of command identifier enclosed in "[]" brackets. This identifier can be a common string or number encoded in string. It may be needed:

1) User (for example) reads multiple registers from some I2C circuit and needs to parse its replies to separate form widgets. As EasyTerm replies with I2C wrrd=... in both cases, user can differentiate between replies using command identifiers – for example:

`I2C wrrd=0A,1 [REG_0x0A];` and `I2C wrrd=0B,1 [REG_0x0B]`

Thanks to that user can set forms parse masks as:

`df id=0 pm="I2C wrrd=%s" [REG_0x0A];` and `df id=0 pm="I2C wrrd=%s" [REG_0x0B];`

Other possibility is to configure interface **R**ead**F**ormat to be expression based with labels.

2) It will be used for JSON-RPC compatibility layer in the future (automation etc...).

Low-power capability

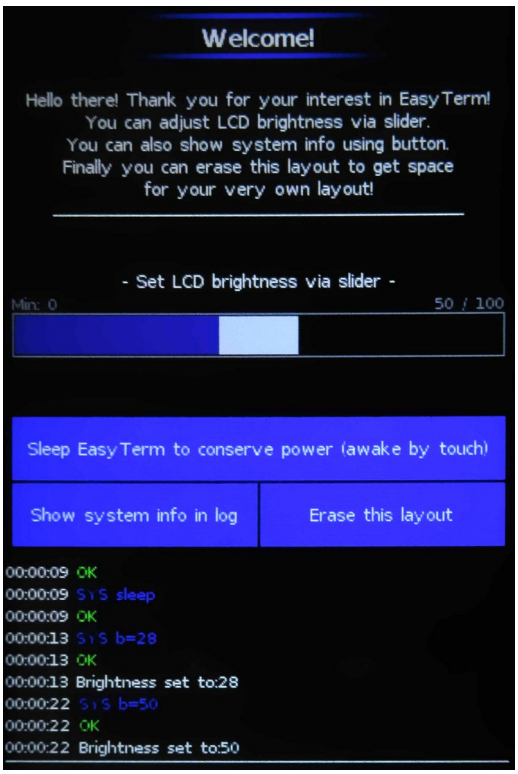
EasyTerm allows 3 different sleep levels. They are configured via system command parameter (**S**leep**L**evel - see system command description). See specifications for current consumptions of these levels.

- Level 0: LCD backlight is turned off and LCD is turned to low-power mode. EasyTerm can be waken-up by detecting touch, by pressing push buttons (default configuration – see push button chapter) or by executing wake command. Capacitive touch panel is active and analog features are powered on.
- Level 1: Same as level 0 but capacitive touch panel is disabled (touching LCD touch-panel will not wake device). EasyTerm can be waken by pressing push buttons (default configuration – see push button chapter) or executing wake command.
- Level 2: Same as level 1 but analog features are disabled as well (ADC and DAC interface wont be functional). This is the most conservative mode.

Sleep can be entered via SYS SLEEP command (see system command description). Sleep can be exited via SYS WAKE command. EasyTerm can enter sleep mode automatically by setting wake-timeout parameter (see system command description). When USB cable is inserted the power consumption during sleep mode is not reduced as much as when USB cable is disconnected – device maintains USB connection.

Quick start

Please read this carefully before starting to use EasyTerm. It summarizes basic information and tips to be effective with EasyTerm. The most quick way to try-out EasyTerm is to connect it via micro-USB cable which provides both power and communication. Do not forget to turn on/off switch to "on" position (labeled as "6" in Hardware description chapter). Welcome screen shall be displayed (see image below). User can display FW version via button "Show system info in log", change current LCD brightness or enable sleep mode. This layout should be erased via button "Erase this layout" to create space for user layouts and widgets. User can perform same action via PC using any terminal application and transmitting the same commands as those assigned to widgets. Do not forget to open correct port (baud-rate does not matter in case of USB communication). Please see "Master connection configurations" chapter in case of any problems. User can also communicate via HOST UART connector (4.) - using either some sort of USB<->UART bridge or through any embedded device with accessible UART interface. Default BaudRate is 38400 bauds/s. Powering is possible through USB or via any of the two connectors - VIN (5.) or BAT (8.).



Welcome screen layout

Here are some **very important tips** – these are not automatically clear without carefully studying manual so please bear them in mind, they will improve your experience with EasyTerm significantly:

- For common text-based UART communication with permanent reception the HOST UART interface is typically what you need due to its operation simplicity. The UART interface (do not mistaken them - on the side of EasyTerm) is more targeted for specific IC protocols/binary streams etc.
- Please take a look at the System command. It is important command for configuring crucial EasyTerm behavior. Other commands are feature specific so you can study them later only when you need them.
- It is recommended to always display Log window widget wherever possible (as it is used for example in "welcome" screen layout). It shows the overview of executed commands and EasyTerm replies. It can be used as a "debug" window when trying out various widgets and observing EasyTerm behavior. It also avoids the need of PC terminal application when EasyTerm is used standalone (without PC).
- When displaying widgets we also suggest to set "id" parameter. When you decide that widget should be placed somewhere else or you want to alter its dimension/behavior you can just alter the specific parameter of composed command and re-transmit the command (with the same id). This will automatically remove displayed widget from screen and displays it again (with updated properties - it will be replaced). This way you don't have to remove widget via specific command prior to displaying them again – that will be very time consuming.
- Commands displaying widgets don't have to be re-transmitted again every time EasyTerm is powered-on. You can store current "layout" to memory using SaveLayout command. You can also set System CommandOnInit to perform LoadLayout command – this way widgets will be restored after EasyTerm is powered-on.
- You can enter command also via EasyTerm screen by displaying a button with action parameter set for example as a="%s". This will invoke keypad when button is touched so user can fill any command. It is handy in situations when EasyTerm is used standalone (without PC) and user want to input command that is not assigned to any HMI widget yet.
- **Advanced usage:** user can perform several complex operations natively in EasyTerm. There is an example to periodically read SHT40 temperature/humidity sensor, automatically convert read bytes to temperature and humidity values by using

ReadFormat expressions and automatically execute specific command to publish these values over MQTT using **ActionOnParse** command when temperature and humidity values are reported. It is however recommended to perform complex operations outside of EasyTerm (for example in Node-RED flow) and use EasyTerm as a simple as possible command ↔ reply endpoint and leave data processing and logic to suitable technology (for example Node-RED).

1) Enable VOUT (3V3) to power SHT40, initialize I2C peripheral of EasyTerm, configure I2C address and set ReadFormat using expression.
sys vs=1; i2c i=1 a=44 rf="ef2(temperature:-45+(175(($0*256)+$1)/65535), humidity:-6+((125*$3*256+$4)/65535))";*

2) Temperature and humidity will be converted by sending "FD" byte over I2C to SHT40. After approx. 10ms data can be read as a reply for read operation of 6 bytes. Assign all of these commands to timer command action parameter so it will be executed periodically (and automatically) each second.
TIM a="I2C wr=FD+SYS w=10+I2C rd=6" p=1000 s=1;

3) EasyTerm will reply following message: "I2C rd=temperature:X,humidity:Y\r\n" each second as a response to I2C rd command triggered by timer and thanks to the configured ReadFormat expression. User can display these value for example in form widgets having ParseMask parameters set as pm="temperature:%s," for temperature and pm="humidity:%s\r\n" for humidity.

4) Replies "temperature:X,humidity:Y\r\n" can also be used to trigger any command when matching some parse mask – for example to automatically execute a command to publish temperature over MQTT. ActionOnParse command can be used for that. If ParseMask parameter of such command will match the reply, values will be extracted from reply and injected to Action parameter – which can be also any command. Filled Action will be then executed (e.g. *WIFI mp="TOPIC",0,0,"23.56"*).
AOP0 pm="temperature:%s," a="WIFI mp=\"TOPIC\",0,0,\"%s\"";

HMI commands overview

EasyTerm offers LCD with touch panel. User can build HMI with a variety of widgets focused on control and evaluation.

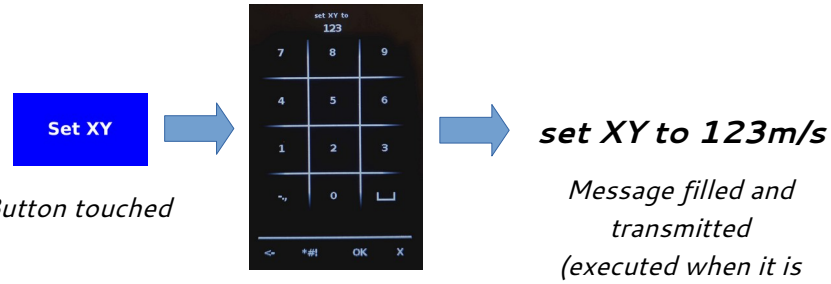
- **Widgets focused on control:** Buttons, checkboxes, sliders, keyboards
- **Widgets focused on logging and evaluation:** Forms, logging windows, plots
- **Design widgets:** Texts, graphics

Widgets focused on control can be considered as initiators. These widgets offers action parameter. When such parameter is not a known EasyTerm command, it is considered a message and it is transmitted through HOST UART and USB (when connected). This parameter can contain special placeholder marks (%d, %s...) that can be „replaced" by user via keyboard or slider – this allows message parametrization. When it is known command it can be parametrized as well and then executed by EasyTerm.

Widgets focused on evaluation has parse mask parameter with placeholder marks that defines a region in received messages that is extracted out. Value or sub-strings can be then displayed inside form widgets or plots (plot allows parsing only string-encoded values).

Message/command parametrization – button example

We've displayed a button with action defined as generic message **a="set XY to %dm/s"**. When button is hit, keyboard is invoked (**„%d"** placeholder was detected) and user can replace **„%d"** placeholder mark. Complete string is then sent to master.



Button touched

Numpad invoked
User enters „123"

Message filled and transmitted (executed when it is a command)

User can also invoke string keyboard and fill commands freely. Entire action parameter can be defined only by keyboard (user can display button with a="%s" and fill entire command manually via invoked keypad). There is also **„%_"** placeholder that is automatically filled via previously filled

value via previous placeholder. It can be used for example to synchronize slider or form widget when multiple value input methods are used, for example in case of a button with action **„a="some value=%d+es v=%_ id=0"“**

user entered "10" via invoked keyboard → EasyTerm will send message "some value=10" and then execute command to edit slider widget (identified by id=0) value to be "10" as well.

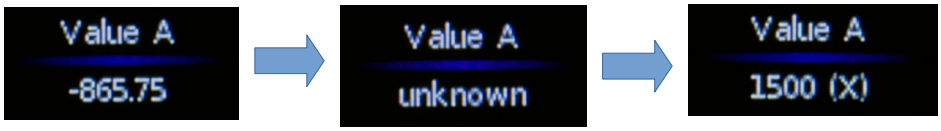
Command parametrization exists also for **slider widget**. Placeholder is filled by value that depends on actual slider handle position (between configured min a max parameters).

Command reply or received message parsing – form example

For example lets assume that some user's master sends periodically this type of messages (3 of them are shown):

Value A:-865.75,value B:55 ➡ *Value A:unknown,value B:8* ➡ *Value A:1500 (X),value B:6*

By defining form parse mask to **pm="Value A:%s,"**(where „%s" is string placeholder mark defining a region between Value A: and ','), EasyTerm will update form value on each message having this format:



See Form widget for more details.

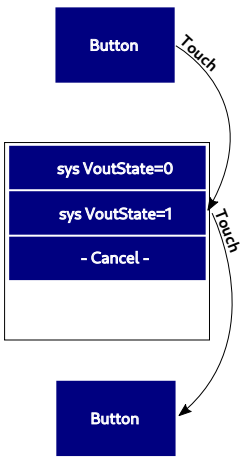
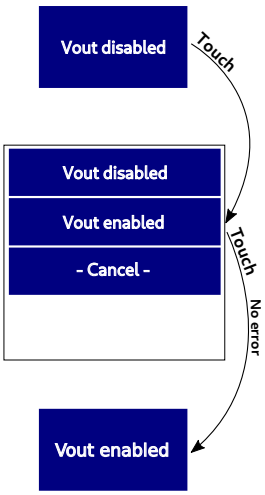
The same applies for plot widget where only „%d" and „%f" parsing placeholders can be used.

Button widget

Related root commands: DisplayButton, EditButton, RemoveButton, ListButtons

Related help commands: Display**B**utton, **E**dit**B**utton, **R**emove**B**utton, **L**ist**B**uttons

Button is a configurable widget with assigned action that is performed after touch. If action is a command then it is evaluated, otherwise it is considered as a generic string and it is sent to master via HOST_UART and USB (if connected). Multiple action commands or messages can be defined by separating them with '+' delimiter. Multi-action button(user select which action to execute) can be used by defining multiple actions separated by "|" (aka pipe) delimiter. Actions can be parametrized by keypad by using '%d', '%s' or '%x' marks. Action can be command that refer to the same button – this way button can edit itself after touch. See example commands below.

Button with simple text and action	Button with simple text ("Button" and multiple actions separated by "|" i.e. "sys VoutState=0|sys VoutState=1")	Button with multiple texts ("Vout disabled|Vout enabled") and multiple actions ("sys VoutState=0|sys VoutState=1")
Button		

Display button command

Root command: **D**isplay**B**utton

Help command: **D**isplay**B**utton?

Description: This command displays button.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=100	0
y	-	Y coordinate	y=100	0
w	**W**idth	Button width. Automatically determined when not specified.	w=80	Determined by label width. No smaller than 60 px.
h	**H**eight	Button height.	h=50	40
c	**C**olor	Button color specified as r,g,b where each value is in percents (0-100)	c=0,0,100 (blue button)	0,0,100 (blue)
t	**T**ext	Button label text. Enclose it in quotation marks. When multi-action button is used (action parameter contains '/' delimiters between multiple actions) then when text have the same amount of '/' delimiters then the list of actions displayed are labeled according to these '/' separated segments. See examples above. Can be multi-line (by using '\n' characters for new-line)	t="this is button"	"" (empty)
f	**F**ont	Button label text font. Number specifies font height (10,14,18,22) . After font height number the style can be specified (b – bold, i – italics, bi – bold italics)	f=14b (14 pixel height, bold style)	14

tc	**T**ext **C**olor	Button label text color specified as r,g,b where each value is in percents (0-100)	tc=100,0,0 (red text)	100,100,100 (white)
ls	**L**ine **S**pacing	Spacing between lines when multi-line text is used.	ls=5	0
a	**A**ction	Action enclosed in quotation marks that will be performed when button is touched. Can be command or non-command string. Non-command strings are resent to master via host UART or USB, commands are evaluated. Multiple commands must be separated with ,+'. Next command is not executed when previous has failed. If action contains %d, %s or %x placeholder marks then keyboard will be invoked after button touch. Via keyboard user can fill(replace) these placeholders. Using %_ placeholder will use the previously entered value from previous placeholder (for example a="PWM dc=%d+es v=%_ id=0" will invoke keyboard once to enter PWM duty cycle parameter and then adjust slider to the same value (slider is synchronized and can be used as an alternative value input method)). When multiple commands are separated by '|' then the list of these commands is displayed and user chooses command (or message) that shall be executed (or sent). When text have also multiple segments separated by '|' then list of these actions is labeled according to that. CR and LF special characters can be used with '\r' or '\n' respecively. See example.	a="button touch\n"	"" (empty)
id	-	Button identifier (0-254). ID is used for EditButton command and RemoveButton command to reference a button. If DisplayButton command contains ID that is already used for other button, this button will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID

sp	**S**creen**P**age	Screen page. If not specified, button will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Blue button with text "Click me!" that will send „button touched\n" to master.

```
db x=10 y=10 w=80 h=50 a="button touched\n" t="Click me!" c=0,0,100 tc=100,100,100
id=0;
```

Red button that will send „button removed!" to master and remove itself.

```
db x=10 y=10 w=80 h=50 a="button removed!+rb id=0" t="button" c=100,0,0
tc=100,100,100 id=0;
```

Button that will display text widget that will be defined by user via keyboard. See text widget for its parameters.

```
db x=10 y=10 w=80 h=50 a="dt a=center t="text entered:%s"" t="Set text" id=0;
```

Edit button command

Root command: **E**dit**B**utton

Help command: **E**dit**B**utton**?**

Description: This command edits button with a specific ID. ID is not optional and MUST be specified.

Parameters can be any of Display button command. Only parameters that are specified are edited (updated).

Remove button command

Root command: **R**emove**B**utton

Help command: **R**emove**B**utton**?**

Description: This command removes button with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	id	Button ID to be removed	id=0	(must be specified)

List buttons command

Root command: **L**ist**B**uttons

Help command: **L**ist**B**uttons?

Description: This command lists info of all buttons on specified screen page. If screen page is not specified, info of all buttons is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	**S**creen**P**age	Screen page	id=0	None – lists all buttons if screen page is not specified

Checkbox widget

Related root commands: DisplayCheckbox, EditCheckbox, RemoveCheckbox, ListCheckboxes

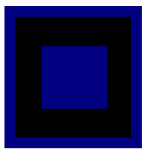
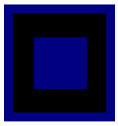
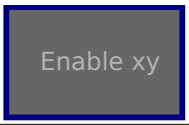
Related help commands: **D**isplay**C**heckbox**?**, **E**dit**C**heckbox**?**, **R**emove**C**heckbox**?**, **L**ist**C**heckbox**?**

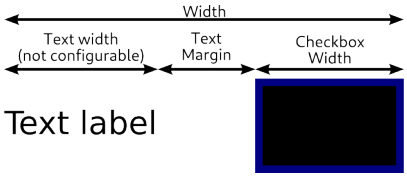
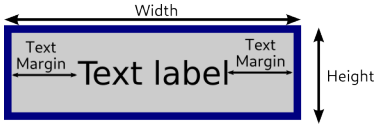
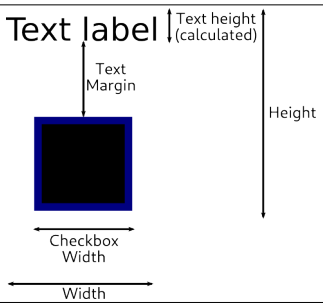
Checkbox is a widget that performs one of the two predefined actions based on current checkbox state (ticked/unticked). Thus action is defined for checkbox tick and checkbox untick. By default checkbox is ticked or unticked immediately on touch. Master acknowledgement is however possible.

Master acknowledgement mechanism is possible by specifying tick and untick phrase and phrase timeout. If checkbox is touched, action is performed but checkbox state is changed only if tick/untick phrase is received from master before phrase timeout expires. This mechanism makes sure that checkbox state is changed only if master registered requested action and confirmed that via mentioned acknowledgement phrase. If phrase timeout is set to 0 device waits forever and thus master can change checkbox state by sending tick/untick phrase without user interacting with LCD.

If action is a command then it is evaluated, otherwise it is considered as a generic string and sent to master via host UART and USB. Multiple actions can be performed.

Checkbox graphics examples:

	Enable xy 	Ticked  Unticked 
Basic checkbox	Checkbox with **T**ext parameter	Checkbox with **O**utline**T**hickness set to 1, **T**ext**O**rigin set to **C**enter, **I**nner**S**ize set to 100, different **T**ick**T**ext**C**olor and **U**ntick**T**ext**C**olor color values and gray **B**ackground**C**olor

Text**O**rigin set to **L**eft. When **T**ext**M**argin and **W**idth are configured then checkbox width occupies remaining space. When **W**idth and **C**heckbox**W**idth is configured then text margin is automatically determined.	
Text**O**rigin set to **C**enter. Here **H**eight and **T**ext**M**argin are configured. Width is automatically determined.	
Text**O**rigin set to **T**op, **H**eight, **T**ext**M**argin and **C**heckbox**W**idth are configured checkbox height equals to remaining space.	

Display checkbox command

Root command: **D**isplay**C**heckbox

Help command: **D**isplay**C**heckbox**?**

Description: This command displays checkbox.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=100	0
y	-	Y coordinate position	y=100	0
w	**W**idth	Overall checkbox widget width - if not set it will be calculated automatically based on TextMargin, text width and TextOrigin.	w=80	50

h	**H**eight	Checkbox height. If not set it will be calculated automatically based on TextMargin, text height and TextOrigin.	h=50	50
tm	**T**ext**M**argin	Used only when text is used as a label to determine space between checkbox and text label. Suitable for automatic width/height calculations. When TextOrigin is center it determines overall checkbox size as text is in center of checkbox.	tm=5	2
cw	**C**heckbox**W**idth	Used to determine width of checkbox "box" graphics. Ignored when TextOrigin is center. Can be used to set width of checkbox when top or bottom TextOrigin is used as overall widget widget may be determined by text width otherwise. Tip: When CheckboxWidth with Width and TextOverlay set to left or right is used then multiple checkboxes can be displayed below each other with aligned checkbox and its label.		
c	**C**olor	Checkbox color specified as r,g,b where each value is in percents (0–100)	c=0,0,100 (blue button)	0,0,100 (blue)
bc	**B**ackground**C**olor	Background color (area of unticked checkbox or area between outline and ticked box)	c=0,0,100 (blue background)	Same as current screen
is	**I**nner**S**ize	Size of a tick mark in percents relative to inner space (0–100)	is=80	81
ot	**O**utline**T**hickness	Thickness of checkbox outline graphics in pixels	ot=4	4

s	**S**tate	Initial checkbox state (0–unticked, 1–ticked)	s=1	0
ta	**T**ick**A**ction	Action enclosed in quotation marks that will be performed when checkbox is touched. Can be command or string. Strings are sent back to master, commands are evaluated. Multiple actions are separated with ‚+'. Next command is not executed when previous has failed. If action contains %d, %s or %x characters then after checkbox touch keyboard will be displayed for user to fill these placeholders.	ta="set something on"	„" (empty)
ua	**U**ntick**A**ction	Same as TickAction parameter but for untick.	ua="set something off"	„" (empty)
tp	**T**ick**P**hrase	If tick phrase is specified, tick acknowledgement mechanism is applied and checkbox is ticked only if this tick phrase is received before PhraseWaitTime expires.	tp="OK"	„" (empty)
up	**U**ntick**P**hrase	If untick phrase is specified, untick acknowledgement mechanism is applied and checkbox is unticked only if this untick phrase is received before PhraseWaitTime expires.	up="OK"	„" (empty)
pwt	**P**hrase**W**ait**T**ime	Time in milliseconds that device wait for tick or untick phrase reception (if they are specified). If phrase timeout is set to 0, device waits forever and master can change checkbox state even without user interacting with LCD.	pwt=500	0

t	**T**ext	Text of checkbox label (optional). Can be multi-line (by using '\n' characters for new-line)	t="Enable something"	„" (empty)
f	**F**ont	Checkbox label text font. Number specifies font height (10,14,18,22) . After font height number the style can be specified (b – bold, i – italics, bi – bold italics)	f=14b (14 pixel height, bold style)	14
ttc	**T**ick**T**ext**C**olor	Color of checkbox text when ticked.	utc=100,100,100 (white)	100,100,100 (white)
utc	**U**ntick**T**ext**C**olor	Color of checkbox text when not ticked.	utc=50,50,50 (gray)	100,100,100 (white)
to	**T**ext**O**rigin	Position of text label. Can be any of these values: **T**op - Above checkbox **L**eft – On checkbox side **R**ight – On checkbox side **B**ottom – Below checkbox **C**enter – In the center of checkbox (recommeded to set **I**nner**S**ize to 100 on differentiate text color via **T**ick**T**ext**C**olor and **U**ntick**T**ext**C**olor parameters)	to=l	Top
ls	**L**ine**S**pacing	Spacing between lines when multi-line text is used.	ls=5	0
id	-	Checkbox identifier (0-254). ID is used for EditCheckbox command and RemoveCheckbox command to reference a checkbox. If DisplayCheckbox command contains ID that is already used for another checkbox, this checkbox will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID

sp	**S**creen**P**age	Screen page. Checkbox will be displayed on actual screen page if screen page is not specified.	sp=1	Actual screen page

Examples:

Checkbox that will send „ticked!\n" on touch to master when ticked and „unticked!\n" when unticked. It will change its state immediately.

```
dc x=10 y=100 ta="ticked!\n" ua="unticked!\n" id=0;
```

Checkbox that will send „ticked!\n" to master and waits 5 s to receive acknowledgement message „acknowledge" from master to change checkbox state. On untick it changes its color to green. If master wont reply „acknowledge" in time the checkbox will remain unticked.

```
dc x=10 y=100 ta="ticked!\n" ua="ec id=0 c=0,100,0" tp="acknowledge" pwt=5000
id=0;
```

Edit checkbox command

Root command: **E**dit**C**heckbox

Help command: **E**dit**C**heckbox**?**

Description: This command edits checkbox with a specific ID. ID is not optional and MUST be specified.

Parameters can be any of Display checkbox command. Only parameters that are specified are edited (updated).

Remove checkbox command

Root command: **R**emove**C**heckbox

Help command: **R**emove**C**heckbox**?**

Description: This command removes checkbox with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	Button ID to be removed	id=0	(Must be specified)

List checkboxes command

Root command: **L**ist**C**heckboxes

Help command: **L**ist**C**heckboxes**?**

Description: This command lists info of all checkboxes on specified screen page. If screen page is not specified, info of all checkboxes is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	**S**creen**P**age	Screen page	sp=0	None – lists all checkboxes if screen page is not specified

Text widget

Related root commands: DisplayText, EditText, RemoveText, ListTexts

Related help commands: DisplayText**?**, EditText**?**, RemoveText**?**, ListTexts**?**

Basic text widget. Alignment and multi-line text is supported - character ‚+' is then used as a line break.

Display text command

Root command: **D**isplay**T**ext

Help command: **D**isplay**T**ext**?**

Description: This command displays text.

Parameters				
Short	**Long**	**Desciption**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position. See **A**lign parameter for x coordinate meaning.	x=100	0
y	-	Y coordinate	y=100	0
bw	**B**ackground**W**idth	Width of text background. When calculated text width is wider than specified BackgroundWidth, the BackgroundWidth will adjust automatically so text wont overflow.	bw=200	Same as calculated text width
bh	**B**ackground**H**eight	Height of text background. When calculated text height is taller than specified BackgroundHeight, the BackgroundHeight will adjust automatically so text wont overflow.	h=50	Same as calculated text height
tc	**T**ext**C**olor	Text color specified as r,g,b where each value is in percents (0-100)	tc=0,100,0 (green)	100,100,100 (white)

bc	**B**ackground**C**olor	Text background color specified as r,g,b where each value is in percents (0-100)	bc=100,0,0 (red)	Same as screen background color
t	**T**ext	Text Can be multi-line (by using '\n' characters for new-line)	t="hello"	„" (empty)
f	**F**ont	Text font. Number specifies font height (10,14,18,22) . After font height number the style can be specified (b – bold, i – italics, bi – bold italics)	f=14b (14 pixel height, bold style)	14 (14 pixel height, no style)
tc	**T**ext**C**olor	Button label text color specified as r,g,b where each value is in percents (0-100)	tc=100,0,0 (red text)	100,100,100 (white)
ls	**L**ine**S**pacing	Spacing between lines when multi-line text is used.	ls=5	0
a	**A**lign	Text alignment. Supported values are l/left, c/center, r/right. **Left alignment:** x coordinate specifies position of text start. **Center alignment:** x coordinate specifies position of text center **Right alignment:** x coordinate specifies position of text end.	a=center	left
ld	**L**ine**D**ecoration	Line decoration across whole **B**ackground**W**idth dimension. Can be any of these values: **N**one – no line **U**nderline – line under text **L**ine – line on text sides base on **A**lign parameter **O**verline – line above text (make sure that **B**ackground**W**idth is set on value higher than is the width of text else there will be no space on line decoration)	ld=l	None

ldc	**L**ine**D**ecoration**C**olor	Color of line decoration	ldc=0,0,100 (blue)	100,100,100 (white)
id	-	Text identifier (0-254). ID is used for EditText command and RemoveText command to reference a text. If specified ID is already used for another text, this text will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID
sp	**S**creen**P**age	Screen page. If not specified, text will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Blue multi-line red text with white background

```
dt y=170 t="Hello!\nHow are you?" m=1 tc=100,0,0 bc=100,100,100 id=0;
```

Single-line centered to middle of screen

```
dt x=160 a=center y=170 t="1+5 equals 6" m=0 id=0;
```

Edit text command

Root command: **E**dit**T**ext

Help command: **E**dit**T**ext**?**

Description: This command edits text with a specific ID. ID is not optional and MUST be specified.

Parameters can be any of DisplayText command. Only parameters that are specified are edited (updated).

Remove text command

Root command: RemoveText

Help command: RemoveText?

Description: This command removes text with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	Text ID to be removed	id=0	(Must be specified)

List texts command

Root command: ListTexts

Help command: ListTexts?

Description: This command lists info of all texts on specified screen page. If screen page is not specified, info of all texts is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	ScreenPage	Screen page	sp=0	None – lists all texts if screen page is not specified

Form widget

Related root commands: DisplayForm, EditForm, RemoveForm, ListForms

Related help commands: **D**isplay**F**orm**?**, **E**dit**F**orm**?**, **R**emove**F**orm**?**, **L**ist**F**orms**?**

Form is a widget that contains label and value. This value is parsed from incoming messages and can be string or number. For example if master is sending periodic messages like „Battery is 3.2V" . If parse mask is „Battery is %s", form will display 3.2V. If master sends „Battery is disconnected", form will display „disconnected". This widget is ideal for observing some changing value or text instead of looking on scrolling lines on terminal application.

DisplayForm command

Root command: **D**isplay**F**orm

Help command: **D**isplay**F**orm**?**

Description: This command displays a form.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=100	0
y	-	Y coordinate	y=100	0
w	**W**idth	Form width. Automatically determined when not specified.	w=100	Determined by label width, form design and font used
h	**H**eight	Form height. Automatically determined when not specified.	h=50	Determined by design and font used

pm	**P**arse**M**ask	Mask that is used to parse-out contents from received messages or EasyTerm replies. Eg if master sends „Battery=2.5V" message and parse mask is defined as "Battery=%sV" the value parsed-out will be string containing number between ‚=' and ‚V' boundaries. In this case defining parse mask as "=%sV" will bring same results. Please bear in mind that CR and LF are also valid characters that may be present in message to be parsed. So to parse EasyTerm's own CRLF terminated reply (e.g ADC voltage) use correct mask such as "ADC v=%s\r" or "ADC v=%d\r". Format specifier can be: **%s** – mostly used – it just extracts sub-string between specified boundaries **%d** – tries to convert sub-string to decimal. Sub-string is discarded when not a number. **%x** – converts parsed-out integer to hexadecimal. Only integer must be present between boundaries. **%b** - converts parsed-out integer to hexadecimal. Only integer must be present between boundaries. **%f** – tries to parse sub-string as a float value. User can change precision between 1 to 9 places after dot. For example for precision of two places use **%.2f**	Pm="Battery=%sV"	"" (empty) Must be specified
tc	**T**ext**C**olor	Form color specified as r,g,b where each value is in percents (0-100)	c=0,0,100 (blue text)	100,100,100 (white)
t	**T**ext	Form label text. Enclose it in quotation marks. Can be multi-line (by using '\n' characters for new-line)	t="value is:"	„" (empty)

f	**F**ont	Form font. Number specifies font height (10,14,18,22) . After font height number the style can be specified (b – bold, i – italics, bi – bold italics)	f=14b (14 pixel height, bold style)	14
ls	**L**ine**S**pacing	Spacing between lines when multi-line text is used.	ls=5	0
ft	**F**orm**T**ype	Form type. Can be any of these values: Stripe(s) – text on top, value on bottom, separated by stripe VerticalTable(vt) – Text on top, value on bottom, table-like outline HorizontalTable(ht) – Text on left, value on right, table-like outline ValueOnly(vo) - No text label, only value	v=valueOnly	v=stripe
tc	**T**ext**C**olor	Form text color specified as r,g,b where each value is in percents (0-100)	tc=100,0,0 (red text)	100,100,100 (white)
bc	**B**ackground**C**olor	Form background color. Specified as r,g,b where each value is in percents (0-100)	bc=100,100,100 (white)	Same as screen background
gc	**G**raphic**C**olor	Color of graphic elements of form. Specified as r,g,b where each value is in percents (0-100)	gc=100,0,0 (red graphic)	0,0,100 (blue graphic)
gt	**G**raphic**T**hickness	Thickness of graphic elements in pixels.	gt=3 (3 pixel thick graphic)	1
sw	**S**tripe**W**idth	Width of form stripe. Only for stripe variant. Value is in percents (0-100).	sw=50 (stripe width is half of form width)	80
so	**S**eparator **O**ffset	Offset position of separator relative to form width. Value is in percents. Only for horizontal-table form.	25 (separator is positioned near ¼ of form width)	50

id	-	Form identifier (0-254). ID is used for EditForm command and RemoveForm command to reference a form. If DisplayForm command contains ID that is already used for other form, this form will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID
sp	**S**creen**P**age	Screen page (0-254). If not specified, form will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Form labeled "Voltage" that will display string after "voltage=" mark. If master sends for example message: „Measured voltage=55mV" then „55mV" will be displayed in form.

```
df y=220 w=100 t="Voltage" pm="voltage=%s" gc=100,0,0 id=0;
```

User wants the same but he wants to see that value in hexadecimal format. User must specify parse mask so that only integer will be parsed-out. When master sends the same value „Measured voltage=55mV" then „37" will be displayed (as it is hexadecimal representation of decimal number „55").

```
df y=220 w=100 t="Voltage (hex)" pm="voltage=%xmV" gc=100,0,0 id=0;
```

EditForm command

Root command: **E**dit**F**orm

Help command: **E**dit**F**orm**?**

Description: This command edits form with a specific ID. ID is not optional and MUST be specified.

Parameters can be any of Display form command. Only parameters that are specified are edited (updated).

RemoveForm command

Root command: RemoveForm

Help command: RemoveForm?

Description: This command removes form with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	Form ID to be removed	id=0	(Must be specified)

ListForms command

Root command: ListForms

Help command: ListForms?

Description: This command lists info of all forms on specified screen page. If screen page is not specified, info of all forms is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	ScreenPage	Screen page	sp=0	None – lists all forms if screen page is not specified

Slider widget

Related root commands: DisplaySlider, EditSlider, RemoveSlider, ListSliders

Related help commands: DisplaySlider?, EditSlider?, RemoveSlider?, ListSliders?

Display slider command

Root command: DisplaySlider

Help command: DisplaySlider?

Description: This command displays a slider. Action has to be specified and must contain ‚%d' (for integer type slider) or ‚%f' (for float type slider) value placeholder. Depending on activation type, EasyTerm will fill this value placeholder by value that is currently set and perform predefined action. Action can be a command or a common string. String is sent back to master, commands are executed.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=100	0
y	-	Y coordinate	y=100	0
w	**W**idth	Slider width	w=80	100
h	**H**eight	Slider Height	h=50	50
oc	**O**utline**C**olor	Slider outline color specified as r,g,b where each value is in percents (0-100)	oc=0,0,100 (blue)	50,50,50 (gray)
hc	**H**andle**C**olor	Slider handle color specified as r,g,b where each value is in percents (0-100)	hc=0,0,100 (blue)	80,80,80 (light gray)
fc	**F**illing**C**olor	Slider filling color (color to the left from handle position) specified as r,g,b where each value is in percents (0-100)	fc=0,0,100 (blue)	0,0,100 (blue)

vc	**V**alue**C**olor	Slider text color specified as r,g,b where each value is in percents (0-100)	vc=0,0,100 (blue)	50,50,50 (gray)
vt	**V**alue**T**ype	This parameter defines slider value type. Possible options are „integer" („i") or „float" („f"). It specifies whenever only fixed numbers are determined from current slider's handle position or if floating point number are also permitted.	vt=float	integer
v	**V**alue	This specifies initial slider value (determines initial slider handle position). Value must be between min and max parameters. Floating point value is allowed only if ValueType parameter is set to „float". User can readjust slider position programatically by editing this parameter (es v=50 id=0 edits slider with id "0" to adjust it position to "50"). Slider can be also adjusted by configuring correct ParseMask.	v=50	Halfway between min and max parameters
pm	**P**arse**M**ask	Slider value (handle position) can be adjusted by receiving message and parsing-out respective value. Eg. when slider parse mask is "DAC v=%d\r\n" and EasyTerm receives "DAC v=?" command, EasyTerm replies to master (e.g. "DAC v=500\r\n") and slider is readjusted to value that was read ("500").	pm="valueXY: %d"	"" (empty)
min	-	Minimum value (must be lower value than value parameter if it was set already). Floating point value is allowed only if ValueType parameter is set to „float".	min=20	0

max	–	Maximum value (must be higher value than value parameter if it was set already). Floating point value is allowed only if ValueType parameter is set to „float".	max=80	100
a	**A**ction	Action enclosed in quotation marks that will be performed when depending on activation type. Can be command or string. Strings are sent back to master, commands are evaluated. Multiple commands are separated with ‚+'. Next command is not executed when previous has failed. Action must contain %d (integer value type slider) or %f (float value type slider). This placeholder is then filled by actual slider value.	a="button touched"	"" (empty)
aa	**A**ction**A**ctivation	This defines when action is filled by slider actual value and processed. There are two possible options **OnRelease (or)** – action will be filled and processed after slider handle release **OnDelay (od)** – action will be filled and processed periodically (determined by delay parameter) but only if value differs from previous (slider handle was moved).	aa=OnDelay	OnRelease
d	**D**elay	This parameter is used only if „OnDelay" action activation is configured. It specifies period in milliseconds in which action is performed when slider is continuously moved.	d=500	100

id	-	Slider identifier (0-254). ID is used for EditSlider command and RemoveSlider command to reference a slider. If DisplaySlider command contains ID that is already used for another slider, this slider will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID
sp	**S**creen**P**age	Screen page. If not specified, slider will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Display slider that will complete message „value set=...%" with percent value (0-100) that is set by slider and send it to master on slider handle release. Initial value (slider handle was not moved yet) is set to zero.

```
ds x=10 y=270 w=300 h=70 a="value set=%d%\n" min=0 max=100 value=0 aa=OnRelease
id=0;
```

Display slider that will set EasyTerm LCD backlight brightness level. It will change continuously as user moves slider handler.

```
ds x=10 y=270 w=300 h=70 a="sys b=%d" min=0 max=100 aa=OnDelay d=100 id=0;
```

Edit slider command

Root command: **E**dit**S**lider

Help command: **E**dit**S**lider**?**

Description: This command edits slider with a specific ID. ID is not optional and MUST be specified.

Parameters can be any of Display slider command. Only parameters that are specified are edited (updated).

Remove slider command

Root command: **R**emove**S**lider

Help command: RemoveSlider?

Description: This command removes slider with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	Slider ID to be removed	id=0	(Must be specified)

List sliders command

Root command: ListSliders

Help command: ListSliders?

Description: This command lists info of all sliders on specified screen page. If screen page is not specified, info of all sliders is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	ScreenPage	Screen page	sp=0	None – lists all sliders if screen page is not specified

Log window

Log window is special HMI widget for logging EasyTerm > Master communication. It behaves like standard PC terminal application. User can scroll through messages by scroll gesture. Log window supports timestamps. Timestamp initial time and date can be configured via specific system command. It also supports color differentiating of messages (commands, generic text messages and EasyTerm error messages and replies can have different color for better readability).

You can filter messages to be displayed via **FilterLog** command as it affects both logging to memory and logging to log window. It allows device to filter-out command messages,generic text messages or messages that are parsed-out by any form/plot widget. For logging messages to non-volatile storage see **RecordLog** command instead.

Log window commands

Root commands: DisplayLog, EditLog, RemoveLog

Help commands: DisplayLog?, EditLog?, RemoveLog?

Display log window command

Root command: DisplayLog

Help command: DisplayLog?

Description: This command displays log window. Log window is supported only on portrait-oriented screens. It will occupy whole screen width. Height is configurable. Only one log window instance is possible (there is no ID parameter).

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
y	-	Y coordinate	y=100	0
h	Height	Log window height	h=50	Half of the LCD height
tc	TextColor	Log text color specified as r,g,b where each value is in percents (0-100)	tc=0,0,100 (blue)	100,100,100 (white)
bc	BackgroundColor	Log background color specified as r,g,b where each value is expressed in percents (0-100)	bc=0,0,100 (blue)	0,0,0 (black)
cc	CommandColor	Command message color specified as r,g,b where each value is expressed in percents (0-100)	cc=0,0,100 (blue)	0,0,100 (blue)
ec	ErrorColor	Error message color specified as r,g,b where each value is expressed in percents (0-100)	ec=0,0,100 (blue)	100,0,0 (red)
tsc	TimeStampColor	Timestamp color specified as r,g,b where each value is expressed in percents (0-100)	tsc=0,0,100 (blue)	0,0,100 (blue)

rc	**R**eply**C**olor	Command reply color specified as r,g,b where each value is expressed in percents (0–100)	rc=0,0,100 (blue)	0,0,100 (blue)
sp	**S**creen**P**age	Screen page. If not specified, log will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Log window with white background, black text and green timestamp.

```
dl h=480 bc=100,100,100 tc=0,0,0 tsc=0,100,0;
```

Edit log command

Root command: **E**dit**L**og

Help command: **E**dit**L**og**?**

Description: This command edits log. Parameters can be any of Display log command. Only parameters that are specified are edited (updated).

Remove log command

Root command: **R**emove**L**og

Help command: **R**emove**L**og**?**

Description: This command removes log window.

Plot widgets

Plot widget is a special logging widget. By defining parse mask, EasyTerm will parse-out values from messages based on this parse mask and draw a point inside plot window. User can visualize changing data and observe trend of values. By performing touch on plot window, plot menu is displayed. Plots supports both manual and tracking cursors, automatic range scaling and offset rejection. User configures plot sample period. If EasyTerm received more than one message (which value was parsed-out) between two samples, new plot sample will be an average of all these parsed values. If messages with parsed-out value are received less frequently than plot sample rate frequency, plot samples either keeps previous value (**hold-mode** plot) or it will perform linear approximation on plot samples that occurred between two messages (**approximated-mode** plot).

Samples which values are determined by values parsed-out from received messages are color-differentiated from sample that were calculated.

Plot commands

Root commands: DisplayPlot, EditPlot, RemovePlot, ListPlots

Help commands: DisplayPlot?, EditPlot?, RemovePlot?, ListPlots?

Display plot command

Root command: DisplayPlot

Help command: DisplayPlot?

Description: This command displays a plot. On plot window touch menu is displayed.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=10	9
y	-	Y coordinate position	y=10	0
w	**W**idth	Plot window width. Minimum is 200. Plot width is always in multiples of 20px steps and 2px is reserved by outline. If user configures width to 300px, resulting width will be 280px. User shall set at least 302px for 300px plot with outline.	w=302	302
h	**H**eight	Plot window height. Minimum is 80.	h=200	100
gc	**G**raphics**C**olor	Plot graphics color (menu, text, track) specified as r,g,b where each value is in percents (0-100)	c=100,0,0 (red)	0,0,100 (blue)
bc	**B**ackground**C**olor	Plot background color specified as r,g,b where each value is in percents (0-100)	c=100,100,100 (white)	0,0,0 (black)
asc	**A**pproximated**S**ample**C**olor	Approximated sample color. This color is used for samples that were calculated by approximation between measured (real) samples.	Asc=50,50,50 (gray)	0,0,50 (dark blue)

xs	**XS**cale	X scale index. Determines period of time between samples. Possible values: 0 (200 ms), 1 (500 ms), 2 (1 s), 3 (2 s), 4 (4 s), 5 (8 s), 6 (20 s), 7 (40 s), 8 (60 s)	xs=3 (2 s)	4 (4 s)
ys	**YS**cale	Yscale index. Determines interval of value between Y grid lines. Possible values: 0 (2), 1 (20), 2 (40), 3 (100), 4 (500), 5 (1000), 6 (2000), 7 (10000), 8 (20000)	ys = 3 (100)	4
ays	**A**utomat ic**YS**cale	Enable or disable automatic y-axis scaling. Automatically adjusts y-scale so track will be expanded as much as possible without track clipping.	ays=1	0
aor	**A**utomat ic**O**ffset **R**ejectio n	Enable or disable automatic offset rejection. When enabled and track currently displayed can be moved near plot bottom without causing clipping, some offset will be subtracted to move track to bottom.	aor=1	0
m	**M**ode	Plot mode parameter. Can be „hold" (dummy samples keeps previous value) or „approx" – dummy samples are calculated by using linear approximation after new value is parsed.	m=approx	hold
s	**S**topped	When set to „0", EasyTerm is parsing received messages - assigning values to plot. When set to „1", plot is stopped and no values are assigned.	s=1	0

pm	**P**arse**M**ask	Mask that is used to parse-out contents from received messages or EasyTerm replies. Eg if master sends „Battery=2.5V" message and parse mask is defined as "Battery=%dV" the parsed-out value will be the number between ‚=' and ‚V'. Please bear in mind that CR and LF are also valid characters that may be present in message to be parsed. So to parse EasyTerm's own CRLF terminated reply (e.g ADC voltage) use correct mask such as "ADC v=%d\r". Possible marks are: **%f** – tries to parse extracted sub-string as a float value. Plot will use float values for data series. **%d** – tries to parse extracted sub-string as a decimal value. Plot will use decimal values for data series.	Pm="Battery=%dV"	"" (empty) Must be specified
id	-	Plot identifier (0-254). ID is used for EditPlot and RemovePlot commands to reference a plot. If DisplayPlot commnad contains ID that is already used for other plot, this plot will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID
sp	**S**creen**P**age	Screen page. If not specified plot will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Plot that will extract values after „value for plot=" from received message. Example of message is "value for plot=3.214". Value type will be configured as „float" as „%f" mark is used. Samples between extracted values will be calculated via approximation. Y axis scale index will be 0 (interval of 2 per division).

```
dp x=9 y=0 w=302 h=240 ys=0 pm="value for plot=%f" m=approximated id=0;
```

Plot that will extract values from same message as above. Value type will be configured as „integer" as „%d" mark is used, so only whole numbers are possible. Samples between extracted

values wont be approximated (steps will be observed). Y axis scale index will be 0 (interval of 2 per division).

```
dp x=9 y=0 w=302 h=240 ys=0 pm="value for plot=%d" m=hold id=0;
```

Edit plot command

Root command: **E**dit**P**lot

Help command: **E**dit**P**lot**?**

Description: This command edits plot with a specific ID. ID is not optional and MUST be specified.

Parameters can be any of DisplayPlot command. Only parameters that are specified are edited (updated).

Remove plot command

Root command: **R**emove**P**lot

Help command: **R**emove**P**lot**?**

Description: This command removes plot with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	Plot ID to be removed	id=0	(Must be specified)

List plots command

Root command: **L**ist**P**lots

Help command: **L**ist**P**lots**?**

Description: This command lists info of all plots on specified screen page. If screen page is not specified, info of all plots is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	**S**creen**P**age	Screen page	sp=0	None – lists info of all plots if screen page is not specified

Graphics widget

Related root commands: DisplayGraphics, EditGraphics, RemoveGraphics, ListGraphics

Related help commands: DisplayGraphics?, EditGraphics?, RemoveGraphics?, ListGraphics?

This widget is a simple graphic element – basically a line or rectangle with optional gradient effect (used for example layouts headers). Rectangular-like shapes can be used as backgrounds for another widgets. Images can be displayed by "Image" command counterpart (see respective chapter).

Overlapping - Graphics widgets can be overlapped by other widgets (text widgets, buttons, other images etc...). When overlapping with widgets that can have transparent background and text (text widgets, form widgets) the graphics color under area occupied by the widget should be homogeneous (typical dashboard-like experience) so anti-aliasing will work correctly (EasyTerm will detect color under text at its left-top corner and use it for color blending around text letters).

Display graphics command

Root command: DisplayGraphics

Help command: DisplayGraphics?

Description: This command displays graphics element (only rectangle/line is supported, see image widget for images).

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=100	0
y	-	Y coordinate	y=100	0
w	**W**idth	Width	w=80	100
h	**H**eight	Height	h=50	50
c	**C**olor	Color specified as r,g,b where each value is in percents (0-100)	c=0,0,100 (blue)	0,0,100 (blue)
bc	**B**ackgroundColor	Background color (for gradient purposes) specified as r,g,b where each value is in percents (0-100)	C=0,0,0 (black)	Same as screen background color

gi	**G**radien t**I**ntensit y	Gradient intensity in percents (0–100). Defines steepness of gradient between **B**ackground**C**olor and **C**olor.	gi=10	100 (no gradient is used – **C**olor is used)
gd	**G**radien t**D**irecti on	Starting direction of gradient. Can be one of these options: **L**eft, **R**ight, **S**ides, **C**enter.	gd=**L**eft	**S**ides
ga	**G**radien t**A**xis	Axis on which is gradient applied. Can be one **H**orizontal or **V**ertical	ga=**V**ertical	**H**orizontal
id	-	Graphics element identifier (0–254). ID is used for EditGraphics command and RemoveGraphics command to reference a graphics element. If DisplayGraphics command contains ID that is already used for other graphics widget, this widget will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID
sp	**S**creen**P** age	Screen page. If not specified, graphic element will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Line-like graphical object with gradients (sides by default).

```
dg x=10 y=100 w=300 h=2 gi=5 id=0;
```

EditGraphics command

Root command: **E**dit**G**raphics

Help command: **E**dit**G**raphics**?**

Description: This command edits graphics element with a specific ID. ID is not optional and MUST be specified. Parameters can be any of DisplayGraphics command. Only parameters that are specified are edited (updated).

RemoveGraphics command

Root command: **R**emove**G**raphics

Help command: **R**emove**G**raphics**?**

Description: This command removes graphics widget with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	Graphic element ID to be removed	id=0	(Must be specified)

ListGraphics command

Root command: **L**ist**G**raphics

Help command: **L**ist**G**raphics**?**

Description: This command lists info of all graphics elements on specified screen page. If screen page is not specified, info of all graphics elements is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	**S**creen**P**age	Screen page	sp=0	None – lists all graphics elements if screen page is not specified

Image widget

Related root commands: DisplayImage, EditImage, RemoveImage, ListImages, AddStoredImage, RemoveStoredImage, ClearStoredImages, ListStoredImages

Related help commands: DisplayImage?, EditImage?, RemoveImage?, ListImages?, AddStoredImage?, RemoveStoredImage?, ClearStoredImages?, ListStoredImages?.

Image shall be named and uploaded via CmdBuilder (graphics/image widget tab – command AddStoredImage is used internally during upload). Commands consists of two categories: image storage manipulation for uploading images to EasyTerm FLASH memory, removing images from memory, listing uploaded images and image widget commands to display uploaded images, remove them from screen etc. User can display stored image using DisplayImage command with the same name as used during upload.

Overlapping - Image widgets can be overlapped by other widgets (text widgets, buttons, other images etc...). When overlapping with widgets that can have transparent background and text (text widgets, forms) the image background under area occupied by the widget should be homogeneous (typical dashboard-like experience) so anti-aliasing will work correctly (EasyTerm will detect color under text at its left-top corner and use it for color blending around text letters).

Display image command

Root command: DisplayImage

Help command: DisplayImage?

Description: This command displays an image widget – previously stored in memory referencing the same name identifier.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
x	-	X coordinate position	x=100	0
y	-	Y coordinate	y=100	0
n	Name	Name identifier of stored image.	n="img1"	Must be specified

id	-	Image widget identifier (0-254). ID is used for EditImage command and RemoveImage command to reference a image widget. If DisplayImage command contains ID that is already used for other image widget, this widget will be replaced (ideal for trying different widget positions etc...).	id=1	Lowest possible unused ID
sp	ScreenPage	Screen page. If not specified, graphic element will be displayed on actual screen page.	sp=1	Actual screen page

Examples:

Display image widget.

```
di x=10 y=10 n="myimage1" id=0;
```

EditImage command

Root command: EditImage

Help command: EditImage?

Description: This command edits image widget with a specific ID. ID is not optional and MUST be specified. Parameters can be any of DisplayImage command. Only parameters that are specified are edited (updated).

RemoveImage command

Root command: RemoveImage

Help command: RemoveImage?

Description: This command removes image widget with a specific ID. ID is not optional and MUST be specified.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
id	-	ID of image widget to be removed	id=0	(Must be specified)

ListImages command

Root command: **L**ist**I**mages

Help command: **L**ist**I**mages**?**

Description: This command lists info of all image widgets on specified screen page. If screen page is not specified, info of all displayed image widgets is sent to master.

AddStoredImage command

Root command: **A**dd**S**tored**I**mage

Help command: **A**dd**S**tored**I**mage?

Description: This command adds (uploads) image to image storage inside EasyTerm memory. Do not use this command manually – use a CmdBuilder to upload images. It will convert/compress selected image automatically to EasyTerm image format and perform serial transmission to EasyTerm via its data transfer protocol. It will also automatically detect converted image size, width, height and format. No user intervention is needed. In case you really need custom solution to upload images – contact us.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
w	**W**idth	X coordinate position	w=100	Must be specified
h	**H**eight	Y coordinate	h=100	Must be specified
n	**N**ame	Name identifier. The same name must be used for **D**isplay**I**mage command.	n="img1"	Must be specified
s	**S**ize	File transfer size in bytes (after conversion to EasyTerm image format).	id=1	Lowest possible unused ID
f	**F**ormat	Screen page. If not specified, graphic element will be displayed on actual screen page.	sp=1	Actual screen page

t	**T**imeout	Timeout for upload in seconds. After image upload is done or after timeout expires the EasyTerm will continue with normal operation listening for commands. When upload is done through CmdBuilder this timeout is calculated automatically.	t=30	15

RemoveStoredImage command

Root command: **R**emove**S**tored**I**mage

Help command: **R**emove**S**tored**I**mage**?**

Description: This command remove image specified by name from EasyTerm FLASH memory. It will also free memory for more images.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
n	**N**ame	Name identifier of image to be removed.	`n="img1"`	(Must be specified)

ClearStoredImages command

Root command: **C**lear**S**tored**I**mages

Help command: **C**lear**S**tored**I**mages**?**

Description: This command clears (formats) entire image storage. Can be used to remove all images from image storage at once.

ListSoredImages command

Root command: **L**ist**S**tored**I**mages

Help command: **L**ist**S**tored**I**mages**?**

Description: This command lists all stored images in image storage. It will also show remaining image space in bytes.

Screen page related commands

Related root commands: DisplayScreenPage, DisplayScreenPageLeft, DisplayScreenPageRight, EditScreenPage, ListScreenPage, ClearScreenPage

Related help commands: DisplayScreenPage?, EditScreenPage?, ListScreenPage?, ClearScreenPage?

These commands offers several features like:

- Switching to another screen pages
- Editing specific screen page (changing background color or orientation)
- Clear all objects on specific screen page

DisplayScreenPage command

Root command: DisplayScreenPage

Help command: DisplayScreenPage?

Description: This command displays screen page specified by ScreenPage (it displays all widgets that were inserted to such screen page).

Parameters				
Short	**Long**	**Desciption**	**Example**	**Default value**
sp	ScreenPage	Screen page	sp=2	None – must be specified

DisplayScreenPageLeft command

Root command: DisplayScreenPageLeft

Description: This command displays screen page with lower index than index of actual displayed screen page.

No parameters are possible.

DisplayScreenPageRight command

Root command: **D**isplay**S**creen**P**age**R**ight

Description: This command displays screen page with higher index than index of actual displayed screen page.

No parameters are possible.

EditScreenPage command

Root command: **E**dit**S**creen**P**age

Help command: **E**dit**S**creen**P**age**?**

Description: This command edits screen page specified by ScreenPage. Background or orientation can be modified. Only parameters that are specified are edited (updated).

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
sp	**S**creen**P**age	Screen page	sp=0	None – must be specified
bc	**B**ackgroundColor	Background color specified as r,g,b where each value is in percents (0–100)	bc=0,0,0 (black)	None
o	**O**rientation	Screen orientation. Can be one of these options: landscape (l) – landscape orientation (experimental feature) portrait (p) – portrait orientation	o=portrait	None

ListScreenPage command

Root command: **L**ist**S**creen**P**age

Help command: **L**ist**S**creen**P**age**?**

Description: This command print information about all screen pages.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	**S**creen**P**age	Screen page Either a numeric value representing screen page index or word all to clear all screen pages. When not specified then current screen page will be cleared.	Sp=0 (clear first screen page) sp=all (clear all screen pages)	Current screen page when not specified.

ClearScreenPage command

Root command: **C**lear**S**creen**P**age

Help command: **C**lear**S**creen**P**age**?**

Description: This command removes all widgets from specified screen page.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
sp	**S**creen**P**age	Screen page	sp=0	None – must be specified

Layout commands overview

Related root commands: SaveLayout, LoadLayout, ListLayoutBanks

Related help commands: **S**ave**L**ayout**?**, **L**oad**L**ayout**?**, **L**ist**L**ayout**B**anks**?**

These commands offers saving of all displayed widgets and screen configuration to non-volatile memory bank. User can load this saved layout by specifying memory bank – all widgets are then restored to saved state.

SaveLayout command

Root command: **S**ave**L**ayout

Help command: **S**ave**L**ayout**?**

Description: This command saves all displayed widgets to specified non-volatile memory bank.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
bid	**B**ank**ID**	Identifier of bank to which will be layout written. Valid ID is in range 0 to 4.	bid=1	None – must be specified

LoadLayout command

Root command: **L**oad**L**ayout

Help command: **L**oad**L**ayout**?**

Description: This command loads layout (recovers all saved widgets) from specified non-volatile memory bank.

Widgets are restored to a saved state.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
bid	**B**ank**ID**	Identifier of bank that will be loaded. Valid ID is in range 0 to 4.	bid=1	None – must be specified

ListLayoutBanks command

Root command: **L**ist**L**ayout**B**anks

Help command: **L**ist**L**ayout**B**anks**?**

Description: This command will send info about all layout memory banks to master. These information show how many widgets are saved in specific memory bank or if specific memory bank is empty.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
bid	**B**ank**ID**	Identifier of bank that will be loaded. Valid ID is in range 0 to 4.	bid=1	Info about all layout banks will be sent to master if no bankId is specified.

Interface commands overview

Related root commands: GPIOx, PushButtonx, UART, MODBUS, SPI, I2C, PWM, TRG, ADC, DAC, WIFI

Related help commands: GPIO?, PushButton?, UART?, MODBUS?, SPI?, I2C?, PWM?, TRG?, ADC?, DAC?, WIFI?

EasyTerm supports various configurable interfaces controlled by master.

These interfaces can be divided between communication interfaces and mixed signal interfaces

Communication interfaces

- UART
- MODBUS
- SPI
- I2C
- WIFI

Mixed-signal interfaces

- GPIOs
- Push buttons
- Trigger
- PWM
- ADC
- DAC

Interfaces are accessible through interfaces connector on the side of the device. Push-buttons are a exception as these are located below LCD display.

Each interface pin has configurable function – it can be configured as a input, output or as a peripheral.

Pull-down and pull-up can be enabled for all interface pins.

GPIO interface

Related root command: GPIO**x** (where x is number of interface pin).

Related help command: GPIO**?** (where x is number of interface pin).

GPIO interface offers initial configuration of any interface pin. Interface pin can be configured as input, output or peripheral. Pull-ups or pull-downs can be enabled.

Parameters					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is [illegible]
m	**M**ode	Interface pin mode Can be one of these options: „Disabled" („d") – Pin will be in Hi-Z state „Input" („i") – Pin will be input „Output" („o") – Pin will be output „Peripheral" („p") – Pin will be connected to available peripheral (SPI, ADC...)	●	**Set** G1 m=p **Read** G1 m=? → G1 m=p	Disabled
ot	**O**utput**T**ype	Interface pin output type (only when pin is configured as output) Can be one of these options: PushPull (pp) - push-pull driver OpenDrain (od) – open-drain driver	●	**Set** G1 ot=pp **Read** G1 ot=? → G1 ot=PushPull	PushPull
pp	**P**in**P**ull	Enables internal pull-up/pull-down (not possible if pin mode is „disabled") Can be one of these options: no – no pull-up/pull-down up – pull-up enabled down – pull-down enabled	●	**Set** G1 pp=up **Read** G1 pp=? → G1 pp=up	no

os	**O**utput **S**tate	Set pin output state (only when pin mode is configured as „output")	●	**Set** G1 os=1 **Read** G1 os=? → G1 os=1	0
is	**I**nput **S**tate	Read pin input state (only when pin mode is configured as „input" or „output")	●	**Read** G1 is=? → G1 is=1	-

Example: Set pin 5 to peripheral mode

```
G5 m=p;
```

PushButton interface

Related root command: Push**B**utton**{X}** (where {X} is number of interface pin).

Related help command: Push**B**utton**?**

Parameters					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is
ac	**A**ction**C**lick	Action enclosed in quotation marks that will be performed when push button is pressed momentarily. Can be command or string. Strings are sent to master via host UART or USB, commands are evaluated. Multiple commands must be separated with ‚+'. If action contains %d, %s or %x placeholder marks then keyboard will be invoked after button touch. Via keyboard user can fill(replace) these placeholders.	●	**Set** PushButton1 ac="Push button pressed!" **Read** PushButton1 ac=? → PushButton1 ac="Push button pressed!"	"" (empty)
ah1s	**A**ction**H**old **1s**	Action enclosed in quotation marks that will be performed when push button is pressed for at least 1 second but no longer than 3 seconds. See ActionClick parameter description for more details.	●	**Set** PushButton1 ah1s="Push button was hold!" **Read** PushButton1 ah1s=? → PushButton1 ah1s="Push button was hold!"	"" (empty)

ah3s	**A**ction**H**old 3**s**	Action enclosed in quotation marks that will be performed when push button is pressed for at least 3 seconds. See ActionClick parameter description for more details.	●	**Set** PushButton1 ah3s="Push button was hold!" **Read** PushButton1 ah3s=? → PushButton1 ah3s="Push button was hold!"	""

UART interface

Related root command: UART

Related help command: UART?

Note: This is UART on the side of the unit. Do not mistaken it for HOST UART. For common text-based communication with permanent reception the HOST UART is typically what you need due to its operation simplicity. This UART interface is more targeted for specific IC protocols/binary streams etc.

UART interface supports configurable baud-rate, RX and TX inversion, endianness, parity and word lengths. Data to be transmitted can be defined as a sequence of hexadecimal characters or as ASCII strings. Reception can be performed in query→reply mode by configuring RX timeout and using respective command (txrx). Background reception is supported. EasyTerm will not be blocked after enabling background reception – free to transmit data to prevent loosing data for reception in case of scenario where reception is enabled after transmission. Reception will be ended and received data will be reported after configured *n* words were received or after reception timeout has expired.

Note: Each respective interface pin must be configured to „peripheral" mode, eg.: "G0 m=p; G1 m=p;" or by using its alias "UART i=1".

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)

i	init	Writing „1" to this parameter will initialize UART by setting respective pins to peripheral mode (it is an alias to commands „G0 m=p; G1 m=p;"). Writing „0" will de-initialize UART by setting respective pins to hi-z (disabled) mode (it is an alias to commands „G0 m=d; G1 m=d;").	●	**Set** UART i=1 **Read** UART i=? → UART i=1	0
br	**B**aud**R**ate	UART baud rate. Allowed values are between 1000 and 1000000.	●	**Set** UART br=19200 **Read** UART br=? → UART br=19200	38400
txi	**TX**Inversion	TX pin level inversion Can be one of these options: 0 – Normal 1 - Inverted	●	**Set** UART tx=1 **Read** UART txi=? → UART txi=0	0
rxi	**RX**Inversion	RX pin level inversion Can be one of these options: 0 – Normal 1 - Inverted	●	**Set** UART rx=1 **Read** UART rxi=? → UART rxi=0	0
wl	**W**ord**L**ength	Word length – number of bits inside each word including parity bit if used. Can be any of these values: 7/8/9	●	**Set** UART wl=9 **Read** UART wl=? → UART wl=9	8
p	**P**arity	Type of parity (both for transmission and reception). Can be any of these values: **N**one – no parity bit **O**dd – Odd parity bit **E**ven – Even parity bit Please adjust **W**ord**L**ength accordingly.	●	**Set** UART p=e **Read** UART p=? → UART p=e	None

sb	**S**top**B**its	Number of stop bits. Can be any of these values: 0.5/1/1.5/2	●	**Set** UART sb=1.5 **Read** UART sb=? → UART sb=1.5	1
msbf	**MSBF**irst	If enabled data frame starts with MSB bit first (typical data frames starts with LSB bit first)	●	**Set** UART msbf=1 **Read** UART msbf=? → UART msbf=1	0
rxt	**RXT**imeout	Set RX timeout for reception in milliseconds. Reception will end if this timeout has expired or if requested byte count was received.	●	**Set** UART rxt=500 **Read** UART rxt=? → UART rxt=500	1000
rf	**R**ead**F**ormat	Format of received data in which they are reported by EasyTerm. Format is defined either: **By expression:** Suitable when bytes needs to be recalculated via some formula (e.g. conversion of data bytes to temperature value etc.). Expression result can be represented as a float ("ef") or decimal ("ed"). Float representation can have configured precision by using for example (“ef2") for precision of 2 (2 numbers after dot). Bytes read are referenced by "$i" where i is index of byte read (i.e. "$0" is first byte that was received). Let’s use following formula (for example from some data-sheet) to recalculate received bytes to temperature $Temp = -45 + 175 * \frac{byte[3] \ll 8 + byte[4]}{2^{16} - 1}$ Read format will be following (we want to get number with fractional part (float): UART rf="ef(temp:-45+175*(($3<<8)+$4)/((2**16)-1)". Please use enclosing brackets	●	**Set** UART rf = "x1" **Read** UART rf=? → UART rf="x1"	"x1"

		frequently and make sure there are no space characters. Multiple such expressions with separate labels (convenient for parsing) can be combined: UART rf="ef(FOO:$0*2,BAR:$0+$1)" **By format specifier:** If first character is ‚0', ‚0' is used as prefix Second character specifies format (d–decimal, x–hexadecimal, b–binary) Third character specifies how many bytes is contained in one value (0 for hexadecimal is continuous format). Fourth character define if bytes shall be swapped Fifth character defines how many bytes shall be swapped. Example: (received data are 0x01 0x02 0x03 0x04) 0x1 – read data as 0x01 0x02 0x03 0x04 0x2 – read data as 0x0102 0x0304 x0 – read data as 01020304 x1 – read data as 01 02 03 04 x1s2 – read data as 02 01 04 03 x1s4 – read data as 04 03 02 01 d1 – read data as 1 2 3 4 d2 – read data as 258 772 b1 – read data as 00000001 00000002 ... 0b1 – read data as 0b00000001 0b00000002 ...			
tx	–	Transmit byte sequence in hexadecimal format (010AFF...) or in ASCII format („hello world"). Use quotes when ASCII format is used.		UART tx=010AFF *or* UART tx="hello"	–

rx	–	Receive n words Define n after equals sign. (rx=n). Reception ends if *n* words are received or if RX timeout expires. Warning: This will block EasyTerm until *n* words are received. You probably want to use "brx" and "tx" after that (or txrx as an alias for that).		UART rx=4	–
txrx	–	This is an alias for "brx" and "tx" called in such order. It does following: 1) Enables reception on background (non-blocking) for *n* words. 2) Transmits byte sequence (see "tx" description). See example. Reception ends after *n* words are received or if RX timeout expires.		UART txrx=010A,4 (reception for 4 words is activated and 010A is transmitted).	–
brx	–	Enable background reception to receive n words. Define *n* as parameter value. Reception ends after *n* words are received or if RX timeout expires.		UART brx=4	–

MODBUS interface

Related root command: MODBUS

Related help command: MODBUS?

MODBUS interface reuses UART interface pins and configurable GPIO pin for output enable signal. External UART<->RS-485 bus driver adapter is required. MODBUS interface supports configurable baud-rate, user configurable output enable pin with configurable active level and configurable receive timeout. Common operations such as write/read holding register, write/read coil, read input registers and read input status are supported.

Note: Each respective interface pin must be configured to „peripheral" mode by using command "MODBUS i=1".

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
i	init	Writing „1" to this parameter will initialize MODBUS by setting respective pins (UART_TX, UART_RX) to peripheral mode and initialize MODBUS features. Writing „0" will de-initialize MODBUS by setting respective pins to hi-z (disabled) mode (it is an alias to commands „G0 m=d; G1 m=d;").	●	**Set** MODBUS i=1 **Read** MODBUS i=? → MODBUS i=1	0
br	**B**aud**R**ate	MODBUS baud rate. Allowed values are between 1000 and 1000000. Default is 9600.	●	**Set** MODBUS br=9600 **Read** MODBUS br=? → MODBUS br=9600	9600

sa	**S**lave**A**ddress	Slave device address to communicate to. Use hexadecimal format.	●	**Set** MODBUS sa=5D **Read** MODBUS sa=? → MODBUS sa=5D	00
oeio	**O**utput **E**nable **IO**	Output enable interface pin. Can be any of these values: 2,3,4,5,6,7 or 8	●	**Set** MODBUS oeio=2 **Read** MODBUS oeio=? → MODBUS oeio=2	- (must be configured)
oeal	**O**utput **E**nable **A**ctive**L**ow	Output enable active low. Configures output enable pin level when driver output shall be enabled for transmission and disabled when transmission has ended. 0 – Output is enabled when pin is high-level (for the most of RS485 drivers) 1 – Output is enabled when pin is low-level	●	**Set** MODBUS oeal=1 **Read** MODBUS oeal=? → MODBUS oeal=1	0
daoe	**D**isable **A**utomatic**O**utput**E**nable	Output enable interface pin is not handled automatically when disabled. 0 – Output enable pin is set to active level by EasyTerm automatically for transmission (recommended). 1 – Output enable pin is not toggled automatically (user asserts it by some other way...). Typically it is recommended to keep it in 0 (default state) and thus let EasyTerm handle it automatically.	●	**Set** MODBUS daoe=1 **Read** MODBUS daoe=? → MODBUS daoe=1	0
rxt	**RXT**imeout	Set RX timeout for reception in milliseconds. Reception will end if this timeout has expired or if all expected data were received.	●	**Set** MODBUS rxt=500 **Read** MODBUS rxt=? → MODBUS rxt=500	1000

p	**P**arity	Additional parity bit. Can be any of these values: **N**one/**O**dd/**E**ven	●	**Set** MODBUS p=e **Read** MODBUS p=? → MODBUS p=e	None
whr	**W**rite**H**olding**R**egisters	Write holding registers. Append register address and data (2 bytes per register) as *whr=address_hex,data_hex* Write multiple holding registers function (16) is used when more than 2 bytes of data is specified (2 or more registers will be written) – otherwise write single holding register function (6) is used.		whr=07D7,0A1B (write 1 register) whr=07D7,0A1B0055 (write 2 registers)	-
rhr	**R**ead**H**olding**R**egisters	Read holding registers. Append register address and total number of registers to be read as: *rhr=address_hex,register_count*		rhr=07D7,1 (read 1 register)	-
wc	**W**rite**C**oils	Write coils. Append starting coil address, total number of coils to be written and bytes in hexadecimal format containing bits to be written (ordered from lsb to msb) as: *wc=address_hex,coil_count,data_hex*		wc=001A,3,03 (write 3 coils with values 1,1 and 0).	-
rc	**R**ead**C**oils	Read coils. Append starting coil address and total number of coils to be read as: *rc=address_hex,coil_count*		rc=001A,2	-
ris	**R**ead**I**nput**S**tatus	Read "discrete" input status. Similar to read coils operation. *ris=address_hex,input_count*		ris=20A0,2	-
rir	**R**ead**I**nput**R**egisters	Read input registers. Append register address and total number of registers to be read as: *rc=address_hex,register_count*		rir=20A0,2	-
rf	**R**ead**F**ormat	Format of received data in which they are reported by EasyTerm. Format is defined either: **By expression:** Suitable when bytes needs to be recalculated	●	**Set** MODBUS rf = "x1" **Read** MODBUS rf=?	"x0"

		via some formula (e.g. conversion of data bytes to temperature value etc.). Expression result can be represented as a float ("ef") or decimal ("ed"). Float representation can have configured precision by using for example (“ef2") for precision of 2 (2 numbers after dot). Bytes read are referenced by "$i" where i is index of byte read (i.e. "$0" is first byte that was received). Let’s use following formula (for example from some data-sheet) to recalculate received bytes to temperature $Temp = -45 + 175 * \frac{byte[3] \ll 8 + byte[4]}{2^{16} - 1}$ Read format will be following (we want to get number with fractional part (float): UART rf="ef(temp:-45+175*(($3<<8)+$4)/((2**16)-1)". Please use enclosing brackets frequently and make sure there are no space characters. Multiple such expressions with separate labels (convenient for parsing) can be combined: UART rf="ef(FOO:$0*2,BAR:$0+$1)" **By format specifier:** If first character is ‚0‘, ‚0‘ is used as prefix Second character specifies format (d-decimal, x-hexadecimal, b-binary) Third character specifies how many bytes is contained in one value (0 for hexadecimal is continuous format). Fourth character define if bytes shall be swapped Fifth character defines how many bytes shall be swapped. Example: (received data are 0x01 0x02 0x03 0x04) 0x1 – read data as 0x01 0x02 0x03 0x04 0x2 – read data as 0x0102 0x0304		→ MODBUS rf="x1"	

		x0 – read data as 01020304 x1 – read data as 01 02 03 04 x1s2 – read data as 02 01 04 03 x1s4 – read data as 04 03 02 01 d1 – read data as 1 2 3 4 d2 – read data as 258 772 b1 – read data as 00000001 00000002 ... 0b1 – read data as 0b00000001 0b00000002 ...			

SPI interface

Related root command: SPI

Related help command: SPI?

SPI interface supports configurable frequency, clock polarity and clock phase. Data to be transmitted can be defined as a sequence of hexadecimal characters or as ASCII strings. Receptions can be performed separately or along with transmitted bytes. Bytes to be sent during reception (so called „dummy" bytes) can be also configured (typically 0xFF).

As some devices responds with valid data after some specified number of words of invalid data, user can configure this number so EasyTerm will ignore these words and report only valid data after those ignored.

To make transactions simple as possible, EasyTerm supports automatic ChipSelect assertion. User must configure which pin will be used for this function. Device will set such pin to low level before any transaction (CS assertion) and release it afterwards (CS deassertion). If transactions must not be interrupted by releasing ChipSelect line, use respective command (txrx).

Note: Each respective interface pin must be configured to „peripheral" mode, eg.: "G2 m=p; G3 m=p; G4 m=p;" or using its alias "SPI i=1".

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
i	Init	Writing "1" to this parameter will initialize SPI by setting respective pins to peripheral mode (it is an alias to commands G2 m=p; G3 m=p; G4 m=p;). Writing "0" will de-initialize SPI by setting respective pins to hi-z (disabled) mode (it is an alias to commands G2 m=d; G3 m=d; G4 m=d;).	●	**Set** SPI i=1 **Read** SPI i=? → SPI i=1	0

f	**F**requency	SPI clock frequency. It is possible to use MHz or kHz as postfix (only whole numbers are permitted). Can be one of these values: 40MHz, 20MHz, 10MHz, 5MHz, 2500kHz, 1250kHz, 625kHz or 312500Hz.	●	**Set** SPI f=10MHz **Read** f=? → SPI f=10000000	1MHz
cpol	**C**lock**P**O**L**arity	Clock polarity Can be one of these options: 0 – Normal 1 - Inverted	●	**Set** SPI cpol=0 **Read** SPI cpol=? → SPI cpol=0	0
cpha	**C**lock**P**H**A**se	Clock phase Can be one of these options: 0 – Normal 1 - Inverted	●	**Set** SPI cpha=1 **Read** SPI cpha=? → SPI cpha=0	0
dw	**D**ummy **W**ord	Dummy word that will be transmitted during receive operations	●	**Set** SPI dw=FF **Read** SPI dw=? → SPI dw=FF	FF
acs	**A**utomatic**C**hip**S**elect	Configures automatic chip-select feature. If enabled, EasyTerm will set ChipSelect pin to low level before any transmit/receive transaction. After transaction the ChipSelect pin is set to high level. If disable, manual chip-select mode is selected and CS state is changed via ChipSelect parameter. 0 - Disabled 1 - Enabled	●	**Set** SPI acs=1 **Read** SPI acs=? → SPI acs=0	0
csio	**C**hip**S**elect**IO**	Set chip-select pin number to be used either in automatic or manual chip select mode. Only pins 0,1,5,6,7 or 8 can be used.	●	**Set** SPI csio=5 **Read** SPI csio=? → SPI csio=5	0

cs	**C**hip**S**elect	Set chip-select pin state if AutomaticChipselect feature is disabled (manual chip-select mode). Before configuration the pin mode must be configured to „output" via respective GPIO command. 1 is active level (CS pin goes to low-level), 0 is CS inactive level (CS pin goes to high level).	●	**CS to active** SPI cs=1 **Read** SPI cs=? → SPI cs=0	0
wbrv	**W**ords**B**efore**R**ead**V**alid	Number of received words that are ignored from the the beginning of reception (ie. they will not be present in rx and txrx replies).	●	**Set** SPI wbrv=2 **Read** SPI wbrv=? → SPI wbrv=2	0
wl	**W**ord**L**ength	Number of bits per word (or number of clock periods). For values less than 8 align data to be send on byte boundary. For values higher than 8 align data on 16 bit boundary. I.E.: user wants to transmit 9 bits at once (0x123) so wl shall be 9 and write operation SPI tx=0123.	●	**Set** SPI wl=16 **Read** SPI wl=? → SPI wl=16	8
lsbf	**LSBF**irst	Send least significant bit (lsb) first.	●	**Set** SPI lsbf=1 **Read** SPI lsbf=? → SPI lsbf=1	0
hd	**H**alf**D**uplex	Set half-duplex transmission mode. When enabled, SPI_MOSI pin will be used for both transmission and reception.	●	**Set** SPI hd=1 **Read** SPI hd=? → SPI hd=1	0
rf	**R**ead**F**ormat	Format of received data in which they are reported by EasyTerm. Format is defined either: **By expression:** Suitable when bytes needs to be recalculated via some formula (e.g. conversion of data bytes to temperature value etc.). Expression result can be represented as a float ("ef") or decimal ("ed"). Float representation can have configured precision by using for example	●	**Set** SPI rf="x1" **Read** SPI rf=? → SPI rf="x1"	"x1"

("ef2") for precision of 2 (2 numbers after dot). Bytes read are referenced by "$i" where i is index of byte read (i.e. "$0" is first byte that was received).

Let's use following formula (for example from some data-sheet) to recalculate received bytes to temperature

$$Temp = -45 + 175 * \frac{byte[3] \ll 8 + byte[4]}{2^{16} - 1}$$

Read format will be following (we want to get number with fractional part (float):

SPI rf="ef(temp:-45+175*(($3<<8)+$4)/((2**16)-1)". Please use enclosing brackets frequently and make sure there are no space characters.

Multiple such expressions with separate labels (convenient for parsing) can be combined:

SPI rf="ef(FOO:$0*2,BAR:$0+$1)"

By format specifier:

If first character is ‚0', ‚0' is used as prefix

Second character specifies format (d-decimal, x-hexadecimal, b-binary, c-ASCII characters)

Third character specifies how many bytes is contained in one value (0 for hexadecimal is continuous format).

Fourth character define if bytes shall be swapped

Fifth character defines how many bytes shall be swapped.

Example:

(received data are 0x01 0x02 0x03 0x04)

0x1 – read data as 0x01 0x02 0x03 0x04

0x2 – read data as 0x0102 0x0304

x0 – read data as 01020304

x1 – read data as 01 02 03 04

x1s2 – read data as 02 01 04 03

x1s4 – read data as 04 03 02 01

d1 – read data as 1 2 3 4

d2 – read data as 258 772

b1 – read data as 00000001 00000002 ...

0b1 – read data as 0b00000001 0b00000002 ...

tx	–	Transmit byte sequence in hexadecimal format (010AFF...) or in ASCII format („hello world"). Use quotes when ASCII format is used.		SPI tx=010AFF SPI tx="hello"	–
rx	–	Receive n words. Define n after equals sign. (rx=n). Master will transmit **D**ummy**W**ord via its MOSI pin and receive data on MISO pin (!when **H**alf**D**uplex is enabled then MOSI is used for reception and only clock signal is generated!). Reception ends when n words are received.		SPI rx=4	–
txrx	–	Transmit data sequence (see tx= description) with reception enabled during transmitting (only when **H**alf**D**uplex is disabled). After transmission it receives additional *n* words (see rx operation). Reception ends after n words are received. Use separate "tx" and "rx" when this is not what you want. Example: SPI txrx=010A,4 where 010A are two byte words to be sent and 4 means that 4 words (each 1 byte wide by default) will be received after transmission. This results in 6 bytes reception (when word is 1 byte wide). This operation is primary for automatic CS assertion when user wants to join transmit operation with receive operation without CS toggling between them as many SPI ICs must not be interrupted between read command transmission and reply receive for example).		SPI txrx=010A,4 (transmit 010A with reception active and receive additional 4 words)	–

I2C interface

Related root command: I2C

Related help command: I2C?

I2C interface supports configurable frequency and address. Transactions can be write, read or write-and-read. Data to be written can be defined as a sequence of hexadecimal characters or as ASCII strings.

Write-and-read operation (also called „repeated start") are also supported via specific command (wrrd) and are commonly used for device register read. The master first writes address of register to be read to slave device and then it performs read operation to obtain data from that register.

Note: Each respective interface pin must be configured to „peripheral" mode, eg.: "G5 m=p; G6 m=p;" or by using its alias "I2C i=1". Do not forget on pull-up resistors (either use EasyTerm pullups (piano switch at the bottom side of device) or external). We strongly recommend to connect slave device interface to EasyTerm I2C bus with pull-ups first before applying power to slave device and initializing I2C so that EasyTerm and slave device I2C interfaces wont enter some unknown/failure state. Feel free to measure SDA and SCL pin levels (shall be around 3.3V when not transmitting).

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
i	Init	Writing "1" to this parameter will initialize I2C by setting respective pins to peripheral mode (it is an alias to commands G5 m=p; G6 m=p;). Writing "0" will de-initialize I2C by setting respective pins to hi-z (disabled) mode (it is an alias to commands G5 m=d; G6 m=d;).	●	**Set** I2C i=1 **Read** I2C i=? → I2C i=1	0

f	**F**requency	I2C clock frequency. It is possible to use MHz or kHz as postfix (only whole numbers are permitted). Allowed values are: 10kHz, 100kHz, 400kHz, 1MHz	●	**Set** I2C f=1MHz **Read** I2C f=? → I2C f=100000	100kHz
a	**A**ddress	Slave address to be communicated with. Use hexadecimal format (see example).	●	**Set** I2C a=5A **Read** I2C a=? → I2C a=5A	0
as	**A**ddress **S**can	Locate all responding devices on I2C bus using whole range of 7-bit addresses (0-127). Result is NONE_FOUND or list of hexadecimal values.	●	**Perform and read result** I2C as=? → I2C as=48	-
10ba	**10B**it**A**ddressing	Use 10-bit addressing.	●	**Set** I2C 10ba=1 **Read** I2C 10ba=? → I2C 10ba=1	0
rf	**R**ead**F**ormat	Format of received data in which they are reported by EasyTerm. Format is defined either: **By expression:** Suitable when bytes needs to be recalculated via some formula (e.g. conversion of data bytes to temperature value etc.). Expression result can be represented as a float ("ef") or decimal ("ed"). Float representation can have configured precision by using for example ("ef2") for precision of 2 (2 numbers after dot). Bytes read are referenced by "$i" where i is index of byte read (i.e. "$0" is first byte that was received). Let's use following formula (for example from some data-sheet) to recalculate received bytes to temperature	●	**Set** I2C rf="x1" **Read** I2C rf=? → I2C rf="x1"	"x1"

		$Temp = -45 + 175 * \frac{byte[3] \ll 8 + byte[4]}{2^{16} - 1}$ Read format will be following (we want to get number with fractional part (float): I2C rf="ef(temp:-45+175*(($3<<8)+$4)/((2**16)-1)". Please use enclosing brackets frequently and make sure there is no space characters. Multiple such expressions with separate labels (convenient for parsing) can be combined: I2C rf="ef(FOO:$0*2,BAR:$0+$1)" **By format specifier:** If first character is ‚0', ‚0' is used as prefix Second character specifies format (d–decimal, x–hexadecimal, b–binary) Third character specifies how many bytes is contained in one value (0 for hexadecimal is continuous format). Fourth character define if bytes shall be swapped Fifth character defines how many bytes shall be swapped. Example: (received data are 0x01 0x02 0x03 0x04) 0x1 – read data as 0x01 0x02 0x03 0x04 0x2 – read data as 0x0102 0x0304 x0 – read data as 01020304 x1 – read data as 01 02 03 04 x1s2 – read data as 02 01 04 03 x1s4 – read data as 04 03 02 01 d1 – read data as 1 2 3 4 d2 – read data as 258 772 b1 – read data as 00000001 00000002 ... 0b1 – read data as 0b00000001 0b00000002 ...			
wr	**WR**ite	Write byte sequence to slave device in hexadecimal format (010AFF...) or in ASCII format („hello world"). Use quotes when ASCII format is used.		I2C wr=010AFF wr="hello"	-

rd	**R**ea**D**	Read n bytes Define n after equals sign. (rd=n). Reception ends if n bytes are received.		I2C rd=4	-
wrrd	**WR**ite **R**ea**D**	Write byte sequence (see wr= description) and read n bytes after write is done (generating repeated-start condition). See example. Read ends after n bytes are received. Typically used for reading slave device register which address is written first.		I2C wrrd=010A,4 (write 010A and read 4 bytes)	-

WIFI interface

Related root command: WIFI

Related help command: WIFI?

See **WIFI module description** chapter for more information related to WIFI module hardware.

WIFI functionality currently offers these features:

- Remotely control of EasyTerm via internet. It is performed by utilizing MQTT broker and configuring command (receiving) topic and reply (transmitting) topic. EasyTerm subscribes to command topic and processes incoming messages the same way as they would appear on USB or HOST UART. Replies are then published to reply topic.
- Subscribe to user configurable topics for general (user) purposes.
- Publish to user configurable topics for general (user) purposes.
- Perform HTTP requests such as GET and POST

These functionalities are very flexible and can be used in coexistence with EasyTerm features. User can for example trigger HTTP GET request (via HMI button, manually or by using TIMER command) and let EasyTerm parse its result to form or plot HMI widgets.
User can trigger HTTP GET/POST requests (either with fixed content or with parametrized content via sliders, keypad widgets etc...) and send some data to some server or IOT device. The same can be performed via MQTT by configuring subscriptions or by executing publish commands.

Remote control can be used to communicate via any MQTT client utility. New versions of CmdBuilder has MQTT client embedded so user can send commands the same way as he was used to send commands by using virtual COM port. Another use case are automations – typically by using technologies such as Node-RED/Home Assistant which supports MQTT and offers possibility to perform highly flexible logic flows – using EasyTerm as flexible API endpoint to perform various interface operation and still providing HMI features to provide convenient ways to control or monitor this automation flow.

Note 1: WIFI must be initialized after connecting and powering WIFI module via "WIFI i=1" to allow correct communication. Before that EasyTerm can receive data that can be considered as garbage and also sent messages out via HOST UART (replies, command echoing) that are unknown for WIFI module communication protocol as EasyTerm don't knows about module presence until "WIFI i=1" command is sent. After that EasyTerm changes HOST UART behavior significantly.

Note 2: WIFI module version 1 (WEMOS D1 MINI – ESP8266EX) module currently supports MQTT over TCP (authenticated/not-authenticated), but not TLS. If TLS is really a must for your usage please contact us. We'll try to offer some solution.

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
i	**I**nit	Writing "1" to this parameter will inform EasyTerm about WIFI module presence. It will adjust its behaviour and HOST UART communication methods accordingly. It also sends several commands to WIFI module to correctly initialize it.	●	**Set** WIFI i=1 **Read** WIFI i=? → WIFI i=1	0
scan	scan	Perform scan of APs in vicinity and reports them to master.		**Perform** WIFI scan **Reply** → WIFI AP="ESSID",RSSI,ENCRYPTION_TYPE...	-
-	version	Read HW and FW version from WIFI module.	●	**Read** **WIFI version=?** AT version:... SDK version:... Bin version:... OK	-
cap	**C**onnect**AP**	Connect AP. Command paremeter is in format cap="essid","password" Password is optional in not secured networks. Query ("WIFI cap=?") will print AP connection status.	●	**Perform** WIFI cap="myNetwork","password" **Reply** → WIFI_AP_CONNECTED	-

dap	**D**isconnect**AP**	Disconnect currently connected AP.		**Perform** WIFI dap	-
-	**SLEEP**	Sleep WIFI module. Command can be without value parameter (sleep infinitely) or with value parameter specifying amount of time in milliseconds to sleep. Module may be woken up by itself by time expiration or by connecting EN pin of module with EasyTerm and properly configuring **W**ake**IO** parameter and executing WIFI WAKE command.		**Perform** WIFI sleep (infinite sleep) WIFI sleep=1000 (sleep for 1s)	
-	**WAKE**	Wake WIFI module by toggling EN pin (EasyTerm IO must be connected to module EN pin and this IO must be specified via **W**ake**IO** parameter).		**Perform** WIFI wake	
wio	**W**ake**IO**	EasyTerm IO that is connected with WIFI module EN pin and that will be toggled to wake WIFI module.	●	**Perform** WIFI wio=0 (IO0 is used to toggle WIFI module EN pin).	
hg	**H**ttp**G**et	Perform HTTP GET operation and return server reply. Specify whole URL in quotes as in example.		**Perform** WIFI httpGet="https://timeapi.io/api/Time/current/zone?timeZone=Europe/Amsterdam" **Reply** → WIFI hr="REPLY"	-
hpu hpt hpj hpf	**H**ttp**P**ost**U**rl **H**ttp**P**ost**T**ext **H**ttp**P**ost**J**son **H**ttp**P**ost**F**orm	Perform HTTP POST operation and return server reply. Select command according to payload content-type. Append URL and data as in example.		**Perform** WIFI hpu="URL","DATA" **Reply** → WIFI hr="DATA"	-

ubh	UseBlockingHttp	Perform HTTP operations in a blocking manner (EasyTerm will wait on operation completion). Enabling may cause more sluggish UI experience as EasyTerm will be blocked for some time depending on operation latency.	●	**Set** WIFI ubh=1 **Read** WIFI ubh=? → WIFI ubh=1	0
cmb	**C**onnect**M**qtt**B**roker	Connect to MQTT broker. Command call shall be in following format: WIFI cmb="URL",PORT,"CLIENT_ID","USERNAME","PASSWORD" where CLIENT_ID is free to choose (any) client identifier string. USERNAME and PASSWORD is optional, so valid call is: WIFI cmb="URL",PORT,"CLIENT_ID" *or* WIFI cmb="URL",PORT,"CLIENT_ID","USERNAME","PASSWORD". Query ("WIFI cmb=?") will print broker connection status.	●	**Perform** WIFI cmb="test.mosquitto.org",1883,"FOO" **Reply** → WIFI_MQTT_CONNECTED	-
dmb	**D**isconnect**M**qtt**B**roker	Disconnect MQTT broker.		**Perform** WIFI dmb **Reply** → WIFI_MQTT_DISCONNECTED	-

rct	**R**emote**C**ontrol **T**opics	Command for enabling remote control over internet. Valid call is in following format (example): WIFI rct="CMD_TOPIC","REPLY_TOPIC". EasyTerm listens (subscribes) for commands on CMD_TOPIC and publishes replies to REPLY_TOPIC. Please note that loosing connection to broker may cause EasyTerm to appear sluggish as it wants to transmit replies for every command but fails with timeout everytime.		**Perform** WIFI rct="MY_CMD_TOPIC","MY_REPLY_TOPIC" **Reply** → OK	–
ms	**M**qtt**S**ubscribe	Subscribe to some topic. Multiple calls (multiple subscribed topics) are possible. Append topic and QOS parameter. Query ("WIFI ms=?") will print currently subscribed topics.	●	**Perform** WIFI ms="TOPIC",0 **Reply** → OK **Data received** → WIFI mm="TOPIC","DATA"	–
mu	**M**qtt**U**nsubscribe	Unsubscribe some previously subscribed topic.		**Perform** WIFI mu="TOPIC" **Reply** → OK	–
mp	**M**qtt**P**ublish	Publish data to topic. Append topic, QOS level(0–2), retain (0 or 1) and data.		**Perform** WIFI mp="TOPIC",0,0,"DATA" **Reply** → OK	–
lwt	**L**ast**W**ill**T**estament	Configures last will testament (optional). Append topic, QOS level (0–2), retain (0 or 1) and message.		**Perform** WIFI lwt="TOPIC",0,0,"MESSAGE" **Reply** → OK	–

ac	**A**t **C**ommand	Perform any AT command as specified in Espressif documentation.		**Perform** WIFI ac="AT+GMR" **Reply** → AT command reply	–
–	FACTORYRESET	Perform factory reset of WIFI module. User should not need that in normal cases.		**Perform** WIFI factoryreset	–

Trigger interface

Related root command: TRG

Related help command: TRG?

This interface will cause evaluation of predefined command immediately after configured signal edge is detected on TRIGGER pin.

Note: Respective interface pin must be configured to „peripheral" mode, eg.: "G7 m=p;" or by using its alias "TRG i=1".

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
i	**I**nit	Writing "1" to this parameter will initialize trigger by setting respective pin to peripheral mode (it is an alias to command G7 m=p;). Writing "0" will de-initialize trigger by setting respective pin to hi-z (disabled) mode (it is an alias to command G7 m=d;).	●	**Set** TRG i=1 **Read** TRG i=? → TRG i=1	0
a	**A**ction	Configured command to be evaluated after signal edge is detected on TRIGGER pin.	●	**Set** TRG a="Edge detected!" **Read** TRG a=? → TRG a="Edge detected!"	-

e	**E**dge	Signal edge that causes action command to be evaluated. Can be any of these values: r (rising) – rising edge f (falling) – falling edge b (both) – both edges	●	**Set** TRG e=r **Read** TRG e=? → TRG e=r	–

PWM interface

Related root command: PWM

Related help command: PWM?

This interface supports PWM generation with configurable duty cycle, phase intervals, rate and polarity.

Lock feature is implemented to ease configuration. For example if PWM is configured to generate signal with t1 phase configured to last 10ms and t2 phase configured to last the same time, user can lock t1 parameter. Then if user increase PWM rate (frequency), only time of t2 phase will be automatically decreased. T1 phase will remain unchanged. See parameters overview for different update locks possibilities.

PWM signal can be generated in one-shot manner (one period) or continuously.

Inactive pin state can also be defined.

Note: Respective interface pin must be configured to „peripheral" mode, eg.: "G8 m=p;" or by using its alias "PWM i=1".

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
i	Init	Writing "1" to this parameter will initialize PWM by setting respective pin to peripheral mode (it is an alias to command G8 m=p;). Writing "0" will de-initialize PWM by setting respective pin to hi-z (disabled) mode (it is an alias to command G8 m=d;).	•	**Set** PWM i=1 **Read** PWM i=? → PWM i=1	0

t1	–	T1 phase time. Possible postfixes are: us, ms, s It automatically reconfigure DutyCycle parameter. Maximum value is 20s, minimum value is 1us.	•	**Set** PWM t1=10ms **Read** PWM t1=? → PWM t1=10ms	–
t2	–	T2 phase time. Possible postfixes are: us, ms, s It automatically reconfigure DutyCycle parameter. Maximum value is 20s, minimum value is 1us.	•	**Set** PWM t2=10ms **Read** PWM t2=? → PWM t2=10ms	–
dc	**D**uty**C**ycle	PWM duty cycle in percents. It automatically reconfigures t1 and t2 parameters according to update lock configuration (see UpdateLock parameter).	•	**Set** PWM dc=50 **Read** PWM dc=? → PWM dc=50	–
r	**R**ate	Set rate. Value can be frequency or period (only whole numbers are permitted). PWM clock is 80MHz, rate/duty cycle resolution is decreased with increasing rate frequency. Allowed post-fixes are: us, ms, s, Hz, kHz, MHz	•	**Set** PWM r=10kHz **Read** PWM r=? → PWM r=10kHz	–
p	**P**olarity	PWM signal polarity 0 – low level first 1- high level first	•	**Set** PWM p=1 **Read** PWM p=? → PWM p=0	0

l	**L**ock	Lock one of parameters. Possible values are: **N**one – no lock applied **r** – Rate lock. When one of the phase time parameter is changed (t1 or t2) , the other one is modified automatically to keep the same rate. Eg.: t1 is increased by 2x, t2 will be set to half automatically. **t1 –** T1 phase time lock. When rate is changed, only T2 time will be changed automatically to reach requested rate and T1 is remained unchanged. **t2** – T2 phase time lock. Same as t1 phase time lock but applied on t2.	•	**Set** PWM l=t1 **Read** PWM l=? → PWM l=t1	none
ul	**U**pdate**L**ock	Apply update lock. PWM signal wont be changed (changing parameters wont be propagated) until update lock is released. 0 – no update lock 1 – update lock is applied	•	**Set** PWM ul=1 **Read** PWM ul=? → PWM ul=1	0
is	**I**dle**S**tate	Idle state when PWM generation is inactive (continuous conversion was stopped or one shot conversion has ended). Possible values: low – low level high – high level hiZ – high impedance state	•	**Set** PWM is=high **Read** PWM is=? → PWM is=high	low
os	**O**ne**S**hot	1 – Starts one shot conversion. Generates one period of PWM signal and sets pin to configured disabled state. 0 – Stops one-shot conversion and sets pin to configured disabled state.	•	**Set** PWM os=1 **Read** PWM os=? → PWM os=1	0
cc	**C**ontinuous**C**onversion	1 – Starts continuous conversion. PWM signal will be generated continuously. 0 – Stops continuous conversion and sets pin to configured disabled state.	•	**Set** PWM cc=high **Read** PWM cc=? → PWM cc=1	0

ADC interface

Related root command: ADC

Related help command: ADC?

ADC interface supports conversion of one sample or buffered conversion of multiple sample in **O**ne**S**hot mode (scan **S**ample**C**ount number of samples and stop) or **C**ontinuous**C**onversion (scan **S**ample**C**ount continuously and overwrite samples from oldest (FIFO)).

ADC interface also supports conversion of battery voltage (when EasyTerm is power-supplied through BAT connector).

It is strongly recommended to power EasyTerm by battery or PSU for best analog performance as noisy USB power can induce high noise levels.

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
v	voltage	Read only – convert ADC channel voltage and returns voltage in millivolts.	●	**Read** ADC v=? → ADC v=153	-
b	battery	Read only – convert battery voltage and returns voltage in millivolts.	●	**Read** ADC b=? → ADC b=3156	-

sr	**S**ample**R**ate	Continuous conversion sample rate. Can be set as a frequency or period (only whole numbers are permitted). Allowed post-fixes are: us, ms, s, Hz, kHz, MHz. Sample rate setting has decreasing resolution on higher values. Perform read operation to make sure which value was actually set. Example: First smaller value than 2MHz is 1.9512MHz (~48kHz resolution), first smaller value than 100125Hz is 100kHz (125Hz resolution).	●	**Set** ADC sr=10ms **Read** ADC sr=? → ADC sr=10ms	-
sc	**S**ample**C**ount	Continuous conversion sample count. Number of samples to be converted. Maximum sample count is 512.	●	**Set** ADC sc=100 **Read** ADC sc=? → ADC sc=100	-
os	**O**ne**S**hot	Start one-shot conversion of **S**ample**C**ount number of samples using **S**ample**R**ate. Only converted samples will be present in sample buffer. 1 – Start conversion 0 – No effect/stop conversion Samples can be read from sample buffer (see command below). EasyTerm will automatically set this value to 0 when **S**ample**C**ount number of samples were converted.		**Set** ADC os=1 **Read** ADC os=? → ADC os=0	0
cc	**C**ontinuous**C**onversion	Starts continuous conversion. Only **S**ample**C**ount number of samples are written to buffer and then overwritten from oldest in **S**ample**R**ate rate. 1 – Start continuous conversion 0- Stop continuous conversion. Only converted samples will be present in sample buffer. Samples can be read from sample buffer (see command below)	●	**Set** ADC cc=1 **Read** ADC cc=? → ADC cc=1	0

rb	**R**ead**B**uffe r	Read only – read sample buffer filled with samples from continuous conversion or one-shot conversion. Device will send the same count of samples that were converted.	●	**Read** ADC rb=? → ADC rb=10,11,15,20	-

DAC interface

Related root command: DAC

Related help command: DAC?

DAC interface supports single conversion of constant voltage and one-shot or continuous (repeated) conversions of configured waveforms.

Signal waveforms are either calculated by EasyTerm (sine/triangle/square/sawtooth) or predefined by used (AWG).

When AWG waveform is used the user creates pattern by setting **S**ample**I**ndex parameter (index of sample inside waveform) and **S**ample parameter (value in millivolts that the sample at predefined index will have). **S**ample**I**ndex is auto-incremented after each write to **S**ample parameter. Thus in many cases it is sufficient to make sure that sample index is set to zero and then transfer whole pattern via repeated write to **S**ample parameter.

When non-AWG waveform is used the waveform samples are calculated automatically based on parameters **W**aveform**F**requency, **W**aveform**A**mplitude, **W**aveform**O**ffset and **S**quare**D**uty**C**ycle (square only).

For waveform conversions user can specify starting sample index, ending sample index and sample rate. Count of samples converted equals to **E**nding**S**ample**I**ndex minus the **S**tarting**S**ample**I**ndex. When used along with AWG the sample rate is kept on configured value via **S**ample**R**ate parameter. Non-AWG waveforms also allows usage of **S**tarting**S**ample**I**ndex and **E**nding**S**ample**I**ndex parameters but frequency of such generated pattern will be based on how many samples are generated (**W**aveform**F**requency is valid only when waveform is not cropped via such parameters).

It is strongly recommended to power EasyTerm by battery or PSU for best analog performance as noisy USB power can induce high noise levels.

Parameters/commands					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
v	**V**oltage	Converts constant voltage. Triggered by write to this parameter. Value is in millivolts.	●	**Set** DAC v=1500 **Read** DAC v=? → DAC v=1500	0
si	**S**ample**I**ndex	Only for AWG pattern. Sample index defines the position in sample buffer to which the sample is written to via AddSample command.	●	**Set** DAC si=5 **Read** DAC si=? → DAC si=5	0
s	**S**ample	Write sample to sample buffer to position defined by SampleIndex. Read returns sample that is located on SampleIndex position in sample buffer (warning – **S**ample**I**ndex is incremented on each **S**ample write).	●	**Set** DAC s=1500 **Read** DAC s=? → DAC s=1500	0
rb	**R**ead**B**uffer	Read sample buffer	●	**Read** DAC rb=? → DAC rb=1500, 100, 2000	-
ssi	**S**tarting**S**ample**I**ndex	Starting sample index which defines beginning of waveform to be generated from sample buffer.	●	**Set** DAC ssi=0 **Read** DAC ssi=? → DAC ssi=0	0
esi	**E**nding**S**ampleIndex	Starting sample index which defines end of waveform to be generated from sample buffer.	●	**Set** DAC esi=5 **Read** DAC esi=? → DAC esi=5	Number of samples written - 1

sr	**S**ample**R**ate	Continuous conversion sample rate. Can be set as a frequency or period. Allowed post-fixes are: us, ms, s, Hz, kHz, MHz (only whole numbers are permitted). Minimum value is 1Hz, maximum value is 2MHz. Sample rate setting has decreasing resolution on higher values. Perform read operation to make sure which value was actually set. Should be set only when AWG pattern is used – other patterns sample rate is kept on maximum possible value and frequency is determined by sample count by pattern generator to have the smoothest shape possible. Example: First smaller value than 2MHz is 1.9512MHz (~48kHz resolution), first smaller value than 100125Hz is 100kHz (125Hz resolution).	●	**Set** DAC sr=10ms **Read** DAC sr=? → DAC sr=10ms	-
w	**W**aveform	Waveform to be generated. When AWG is selected, user must supply samples via consesutive write to AddSample parameter. Can be any of these values: AWG, triangle, sine, sawtooth, square. **W**aveform**F**requency and **W**aveform**A**mplitude applies only for non-AWG waveform.	●	**Set** DAC w=triangle **Read** DAC w=? → DAC w=triangle	sine
wa	**W**aveform **A**mplitude	Amplitude of waveform. Used for all waveforms excluding AWG. Can be value from 0 to 3300(+-10).	●	**Set** DAC wa=1000 **Read** DAC wa=? → DAC wa=1000	1000 (mV)
wo	**W**aveform **O**ffset	Offset of waveform. Used for all waveforms excluding AWG. Can be value from 0 to 3300(+-10). High **W**aveform**A**mplitude and **W**aveform**O**ffset values can result in waveform clipping.	●	**Set** DAC wo=1000 **Read** DAC wo=? → DAC wo=1000	0 (mV)

wf	**W**aveform **F**requency	Used for all waveforms excluding AWG. It determines the sample count that is calculated and sample rate to be used (trying to use highest values possible to get the smoothest shape possible). Can be in Hz, kHz and MHz. Maximum value is 100kHz.	●	**Set** DAC wf=1000Hz **Read** DAC wf=? → DAC wf=1000Hz	1000 (Hz)
sdc	**S**quare**D**uty**C**ycle	Used only for square waveform. It defines duty cycle of square in percents.	●	**Set** DAC sdc=85 **Read** DAC sdc=? → DAC sdc=85	50
os	**O**ne**S**hot	Start one-shot (not repeated) conversion of pattern specified in sample buffer between starting sample index and ending sample index. Conversion of pattern is ended after conversion of ending sample. 1 – Start one-shot pattern conversion 0 – Stop ongoing one-shot pattern conversion If parameter is read, 1 is returned if one-shot pattern conversion is ongoing.	●	**Set** DAC os=1 **Read** DAC os=? → DAC os=1	-
cc	**C**ontinuous**C**onversion	Starts continuous conversion of the pattern specified in sample buffer between starting sample index and ending sample index. After reaching ending sample index, pattern continues from starting sample index. 1 – Starts continuous conversion 0- Stops continuous conversion. Only converted samples will be present in sample buffer.	●	**Set** DAC cc=1 **Read** DAC cc=? → DAC cc=1	-

Miscellaneous commands overview

Logging commands overview

Related root commands: RECORDLOG, SAVELOG, LOADLOG, LISTLOGS, PRINTLOG, CLEARLOG, FILTERLOG

Related help commands: RECORDLOG?, SAVELOG?, LOADLOG?, LISTLOGS?, PRINTLOG?, CLEARLOG?, FILTERLOG?

These commands are dedicated for logging communication between master and EasyTerm or for logging replies of executed commands. There is possibility to set the filter to exclude several types of messages (such as commands, replies, parsed messages, generic text messages) from being logged.

EasyTerm logs communication from the moment it was turned on to its RAM memory (aka "runtime logging"). Memory capacity for such logging is limited (~11 KB) and after reaching its capacity oldest messages will be overwritten. For long-term logging there is a feature called "log recorder" that logs communication to the high-capacity FLASH memory consisting of three banks (1 MB each). Recording can be done to individual banks or to multiple banks to get even more capacity.

Logged communication (both runtime or the newest part of log stored in FLASH) can be displayed via log window widget or can be transmitted to master via PrintLog command.
Transmitting will send entire log.

RecordLog command

Root command: RECORDLOG

Help command: RECORDLOG?

Description: This command starts recording of communication between master and EasyTerm.

Memory bank identifier has to be specified.

Parameters					
Short	**Long**	**Description**	**Example**	**Readable**	**Default value**
bid	**BankID**	Identifier of bank to which log will be recorded. Valid ID is in range 0 to 2. It can be specified by single number (using one bank) or as an interval (using more banks) to record log sequentially for longer time. See example. When read is performed the EasyTerm respond with bank(s) that are recorded to.	**Write** RECORDLOG bid=1 (using single bank) bid=0-2 (using all banks) **Read** RECORDLOG bid=? → RECORDLOG bid=0	●	None – must be specified

RecordLogStop command

Root command: RECORDLOGSTOP

Help command: RECORDLOGSTOP?

Description: This command stops recording of communication between master and EasyTerm.

No parameters

SaveLog command

Root command: SAVELOG

Help command: SAVELOG?

Description: This command will save log of communication received to logWin widget to a specific bank.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
bid	**B**ank**ID**	Identifier of bank to which log will be written. Valid ID is in range 0 to 2. Only exact bank index (no interval) is permitted as log stored in RAM cant exceed bank memory.	SL bid=1	None – must be specified

LoadLog command

Root command: LOADLOG

Help command: LOADLOG?

Description: This command will load log from specified memory bank. This log can be displayed in log window widget.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
bid	**B**ank**ID**	Identifier of bank from which log will be read. Valid ID is in range 0 to 2. Only exact bank index (no interval) is permitted as log loaded to RAM cant exceed bank memory.	LL bid=1	None – must be specified

ListLogs command

Root command: LISTLOGS

Help command: LISTLOGS?

Description: This command will send info of log that is saved in specified bank to master.

If no bank ID is specified, info about all logs from all banks is sent to master.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value**
bid	**B**ank**ID**	Identifier of bank. Information about log present in this bank will be send to master. Valid ID is in range 0 to 2.	ListLogs bid=1	Info about logs from all banks will be send to master if no bankId is specified.

PrintLog command

Root command: PRINTLOG

Help command: **PRINTLOG?**

Description: This command will send all logged messages from specified bank to master. Messages can be transmitted with time and/or date. When BankId parameter is omitted, actual logged communication in RAM will be printed-out.

Parameters				
Short	**Long**	**Description**	**Example**	**Default value** (when parameter is optional)
bid	**B**ank**ID**	Identifier of bank when user wants to print-out stored log. Messages written to this bank will be sent to master. Valid ID is in range 0 to 2. Use a single number – when LOG was recorded to multiple banks (iterval), use starting bank index (first number from interval). When not specified, actual log from RAM will be printed out.	PrintLog bid=1	None – When not specified, RAM log is printed-out.
pt	**P**rint**T**ime	When set to 1, log will be transmitted to master along with time.	PrintLog pt=1	0
pd	**P**rint**D**ate	When set to 1, log will be transmitted to master along with date	PrintLog pd=1	0

ClearLog command

Root command: CLEARLOG

Help command: **CLEARLOG?**

Description: This command will erase RAM log buffer. It will also clear log widget window.

No parameters

FilterLog command

Root command: FILTERLOG

Help command: FILTERLOG?

Description: This command configures filter to filter-out messages from logging of communication between EasyTerm and master. Both logging to memory feature and log window widget are affected.

Parameters					
Short	Long	Description	Example	Readable	Default value
ec	**E**xclude**C**ommands	When enabled all messages that were evaluated as EasyTerm commands wont be logged.	FilterLog ec=1 (commands are excluded)	●	0
ep	**E**xclude**P**arsed	When enabled all messages whose content was parsed by form widget or plot widget wont be logged.	FilterLog ep=1 (parsed-out messages are excluded)	●	0
et	**E**xclude**T**ext	When enabled all generic text messages wont be logged.	FilterLog et=1 (generic text messages are excluded)	●	0
er	**E**xclude**R**eplies	When enabled all device replies wont be logged.	FilterLog er=1 (replies are excluded)	●	0

System commands overview

These commands is dedicated to access various system features. It is possible to set EasyTerm to low-power consumption mode, adjust backlight, enable voltage output, set date and time, set command executed on device initialization, set periodic command execution etc.

Sys command

Root command: **SYS**tem

Help command: **SYS**tem?

Description: This command offer access to various system parameters.

Parameters					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
br	**B**aud**R**ate	Host UART baud rate. When written via host UART, EasyTerm will respond with acknowledge on current baud rate and after that new baud rate will be configured. Other devices communicating with EasyTerm shall reconfigure its baud rate as well when it differs.	●	**Write** SYS br=9600 **Read** SYS br=? → sys br=9600	38400
dbr	**D**efault**B**aud**R**ate	Non-volatile version of command above. Sets default baud-rate that is used after device power-up or reset. If user loses track about currently configured speed, it can be restored via USB (some terminal application), LoadFactoryConfig command or via holding all mechanical buttons during power-up (factory reset).	●	**Write** SYS dbr=9600 **Read** SYS dbr=? → sys dbr=9600	38400

t	**T**ime	Parameter for configuration of EasyTerm time. Format is: „hours:minutes:seconds“.	●	**Write** SYS t=20:45:00 **Read** SYS t=? → sys t=20:45:00	0:0:0
d	**D**ate	Parameter for configuration of EasyTerm date. Format is: „day.month.year“.	●	**Write** SYS d=20.6.2000 **Read** SYS d=? → sys d=20.6.2000	
b	**B**rightness	LCD backlight brightness in a range of 0–100.	●	**Write** SYS b=50 **Read** SYS b=? → sys b=50	50
db	**D**efault**B**rightness	Non-volatile version of command above. Sets default brightness that is used after device power-up or reset. LCD backlight brightness in a range of 0–100.	●	**Write** SYS db=50 **Read** SYS db=? → sys db=50	50
ofd	**O**ptical**F**eedback**D**isable	Disable or enable optical feedback state via LEDs. When is set to 0 (not disabled), green LED blinks on message received through host UART or USB, blue LED blinks on command execution, red LED blinks on error.	●	**Write** SYS ofd=1 **Read** SYS ofd=? → sys ofd=1	0
dofd	**D**efault**O**ptical**F**eedback**D**isable	Non-volatile version of command above. Sets if optical feedback is disabled by default after device power-up or reset.	●	**Write** SYS dofd=1 **Read** SYS dofd=? → sys dofd=1	0

vs	**V**out**S**tate	Enable or disable 3.3V voltage output through Vout connector.	●	**Write** SYS vs=1 **Read** SYS vs=? → sys vs=1	0
dvs	**D**efault**V**out**S**tate	Non-volatile version of command above. Sets default vout state that is used after device power-up or reset.	●	**Write** SYS dvs=1 **Read** SYS dvs=? → sys dvs=1	0
i	**I**nfo	Get system information. Read only. Device responds with FW, Bootloader and HW versions	●	**Read** SYS i=? → sys i= FW VERSION: x.y BOOTLOADER VERSION: x.y HW VERSION: x.y SERIAL NUMBER: x	-
hardReset	**H**ard**R**eset	Hard reset sub-command. Triggers device reset (similar to power on-off cycling). If device is connected to host PC via USB, communication will be lost (reopening COM port may be needed)		SYS hardreset	-
softReset	**S**oft**R**eset	Soft reset sub-command. Triggers device soft-reset. Device state will be set to state similar after power-cycling. USB communication wont be lost.		SYS softreset	-
wake	-	Wake sub-command. Device leaves low power mode (enables LCD backlight and LCD circuitry).		SYS wake	-
sleep	-	Sleep sub-command. Device enters low power mode (see SleepLevel on sleep mode).		SYS sleep	-

sl	**S**leep**L**evel	Can be a number 0-2 configuring sleep level. The higher number the less current is drawn when sleep mode is activated. 0: LCD backlight is turned off 1: Same as level 0 but capacitive touch panel is disabled as well 2: Same as level 1 but analog features are disabled as well See low-power capability chapter for more information.	●	**Write** SYS sl=1 **Read** SYS sl=? → sys sl=1	0
w	**W**ait	Wait for time specified in miliseconds (blocking operation - EasyTerm halts for specified time).		SYS w=1000	
ast	**A**utomatic**S**leep **T**imeout	When set to non-zero value, sleep „state" will be entered automatically after configured time. This time interval is measured from the beginning when these events occurs: There was a touch evaluated on LCD display Mechanical button was touched Wake sub-command was received. Value is in milliseconds.	●	**Write** SYS ast=5000 **Read** SYS ast=? → sys ast=5000	0
dast	**D**efault**A**utomatic**S**leep**T**imeout	Non-volatile version of command above. Automatic sleep timeout will be set on this value after device is powered on or reset is invoked.	●	**Write** SYS dast=5000 **Read** SYS dast=? → sys dast=5000	0
wos	**W**ake**O**r**S**leep	Device will wake up if current state is „sleep". Device will enter sleep mode otherwise.		SYS wos	-
rfc	**R**estore**F**actory **C**onfig	Restore factory configuration and save it as default. Same as factory reset performed via mechanical buttons (see **Hardware description** chapter).		SYS rfc	-

coi	**C**ommand**O**n**I**ni t	Command/s that are executed after device power-on or device reset. Example can be loading (recovering) HMI layout that was saved previously. Parameter string must be enclosed in double-quotes and multiple commands separated with ‚+' symbol. Non-volatile parameter.	●	**Write** SYS coi="ll bid=0" **Read** SYS coi=? → sys coi="ll bid=0"	
sa	**S**uppress**A**ckno wledge	Do not transmit acknowledge messages ("OK") after successfully processing a command to master via USB or HOST UART.	●	**Write** SYS sa=1 **Read** SYS sa=? → sys sa=1	0
dsa	**D**efault**S**uppres s**A**cknowledge	Non-volatile version of command above. Sets initial suppress acknowledge parameter after device power-up or reset.	●	**Write** SYS dsa=1 **Read** SYS dsa=? → sys dsa=1	0
se	**S**uppress**E**rrors	Do not transmit error messages after failure occurred when processing a command to master via USB or HOST UART.	●	**Write** SYS se=1 **Read** SYS se=? → sys se=1	0
dse	**D**efault**S**uppres s**E**rrors	Non-volatile version of command above. Sets initial suppress errors parameter after device power-up or reset.	●	**Write** SYS dse=1 **Read** SYS dse=? → sys dse=1	0
prd	**P**arse**R**epliesDi sable	Disable parsing of replies that are transmitted to master after command execution. Parsing of replies is for example when form is displayed and parsing ADC v=%d which is reply on ADC v=? command.	●	**Write** SYS prd=1 **Read** SYS prd=? → sys prd=1	0

dprd	**D**efault**P**arse**R**eplies**D**isable	Non-volatile version of command above. Sets initial parse replies disable parameter after device power-up or reset.	●	**Write** SYS dprd=1 **Read** SYS dprd=? → sys dprd=1	0
sn	**S**erial**N**umber	Serial number of device.	●	**Read** SYS sn=? → sys sn=6984xxxx	-
coae	**C**ontinue**O**n**A**ction**E**rror	Continue on action error. When set to 1 then if e.g. button action parameter contains multiple commands and some of them fails, others are also processed. Default is "0" and commands after failed one are not executed.	●	**Write** SYS coae=1 **Read** SYS coae=? → sys coae=1	0
dcoae	**D**efault**C**ontinue**O**n**A**ction**E**rror	Non-volatile version of command above.	●	**Write** SYS dcoae=1 **Read** SYS dcoae=? → sys dcoae=1	0
dice	**D**isable**I**nternal**C**ommand**E**choing	Commands executed internally (by interacting with HMI widgets, buttons, timer etc...) are not echoed via USB and UART when set to 1.	●	**Write** SYS dice=1 **Read** SYS dice=? → sys dice=1	0
ddice	**D**efault**D**isable**I**nternal**C**ommand**E**choing	Non-volatile version of command above.	●	**Write** SYS ddice=1 **Read** SYS ddice=? → sys ddice=1	0
dcm	**D**isable**C**omm**M**irror	Messages received via HOST UART are not mirrored to USB and messages received via USB are not mirrored to HOST UART when set to 1.	●	**Write** SYS dcm=1 **Read** SYS dcm=? → sys dcm=1	0

ddcm	**D**efault**D**isable **C**omm**M**irror	Non-volatile version of command above.	●	**Write** SYS ddcm=1 **Read** SYS ddcm=? → sys ddcm=1	0
drvu	**D**isable**R**eplies **V**ia**U**art	EasyTerm will not transmit replies via UART when set to 1.	●	**Write** SYS drvu=1 **Read** SYS drvu=? → sys drvu=1	0
drvv	**D**isable**R**eplies **V**ia**V**cp	EasyTerm will not transmit replies via VCP (USB) when set to 1.	●	**Write** SYS drvv=1 **Read** SYS drvv=? → sys drvv=1	0

Timer command

Root command: TIMer

Help command: TIMer?

Description: This command allows to set-up periodic command execution.

Parameters					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)
s	**S**tate	This parameter enable/disable periodic command execution.	●	**Write** TIM s=1 **Read** TIM s=? → sys tim=1	0
p	**P**eriod	Period of periodic command execution is milliseconds.	●	**Write** TIM p=100 **Read** TIM p=? → TIM p=100	1000
a	**A**ction	Action command that will be executed periodically.	●	**Write** TIM a="hello world" **Read** TIM a=? → tim a="hello world"	""

ActionOnParse command

Root command: **A**ction**O**n**P**arse

Help command: **A**ction**O**n**P**arse?

Description: This commands will parse received messages or own replies on parse mask. If parse mask matches message contents, part of message is extracted and configurable action is executed (%s mark inside action is replaced by extracted part of message).

Example:

Received message: Foo 123 bar

Example action: WIFI mp=\"MY_TOPIC\",0,0,\"%s\"

Expected behavior: EasyTerm will extract 123 from received message "Foo 123 bar" and publish this part (123) to MY_TOPIC using **M**QTT **P**ublish command.

Parameters					
Short	**Long**	**Description**	**Readable**	**Example**	**Default value** (when parameter is optional)

pm	**P**arse**M**ask	Mask that is used to parse-out contents from received messages or EasyTerm replies. Eg if master sends „Battery=2.5V" message and parse mask is defined as "Battery=%sV" the value parsed-out will be string containing number between ‚=' and ‚V' boundaries. In this case defining parse mask as "=%sV" will bring same results but may apply to a broader set of messages. Please bear in mind that CR and LF are also valid characters that may be present in message to be parsed. So to parse EasyTerm's own CRLF terminated reply (e.g ADC voltage) use correct mask such as "ADC v=%s\r". Format specifier can be: **%s** – extracts sub-string between specified boundaries	●	**Write** AOP pm="foo %s bar" **Read** AOP pm=? → AOP pm="foo %s bar"	""
a	**A**ction	Action command that will be executed when received message/reply matches configured **P**arse**M**ask. User can add '%s' mark inside action. Mark will be replaced by extracted message/reply content.	●	**Write** AOP a="WIFI mp=\"MY_TOPIC\", 0,0,\"%s\"" **Read** AOP a=? → AOP a="..."	""

Firmware update

Device features FW update capability through internal bootloader. Bootloader is a small piece of firmware that coexists with EasyTerm firmware and that can be activated by user. Bootloader is responsible for communicating with specific PC software to allow update of EasyTerm´s firmware. Bootloader is entered by holding **button 1** and then turning EasyTerm on via **mechanical switch**. When bootloader is activated, first (green) LED at the bottom of device will be turned on. Bootloader will be automatically entered after power-on if previous update has failed thus allowing user to repeat update procedure (device cannot be so-called "bricked").

There are two ways to update FW:

- Via USB by using recommended PC tool supporting DFU (aka Device Firmware Update) protocol. Recommended is using dfu-util (lightweight solution) or STM32CubeProgrammer (not so lightweight). They are compatible with GNU/Linux and windows operating systems.
- Via host UART by using recommended python script.

Bootloader status indication	
Optical indication	**Meaning**
LED 1 (green) on	Bootloader is active – awaiting FW update file
LED 1 (green) blinking	Bootloader is installing/validating received update
LED 2 (blue) blink	Device has received packet from host PC containing either erase command or a part of transmitted FW file
LED 3 blinking (red)	Update has failed. Switch device off and on again and repeat update procedure. If problem remains, try using different update method (via USB or UART).

Errors indication	
Optical indication	**Meaning**
LED 3 (red) 1 blink	Invalid FW package (file) was transferred via PC host. Please make sure you are using correct file and repeat update procedure.
LED 3 (red) 2 blinks	CRC check of transferred FW file failed. Reason can be data corruption during transfer or update was incomplete (interrupted during update procedure). Repeat update procedure.
LED 3 (red) 3 blinks	Transferred file decryption has failed. Repeat update procedure.
LED 3 (red) 4 blinks	Transferred file authentication has failed. Repeat update procedure.

Update via USB using dfu-util

Dfu-util is a simple multi-platform PC software. It is released for GNU/Linux and Windows OS.

On Windows OS there can be some driver-related manual intervention needed – it will be found out after step 4. On GNU/Linux driver shall be part of Linux kernel.

For GNU/Linux it may be downloadable via package manager, for Windows OS the URL for releases is here: https://dfu-util.sourceforge.net/releases/ - latest version for Windows OS is currently 0.9.

1. Connect EasyTerm via USB to host PC

2. Switch to bootloader mode as described above, check if LED 1 (green) is on (bootloader mode activated).

3. Execute this command via command line tool. In Windows OS please change directory to the one containing DFU-UTIL.

```
dfu-util -l
```

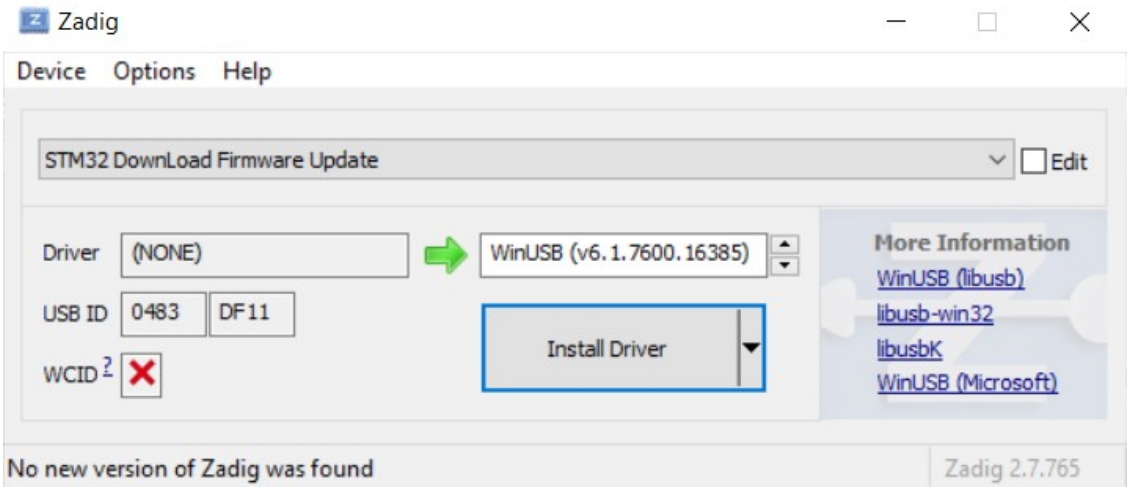
4. Some device shall be listed as shown in example output below – in case of other result on Windows OS (typically error message containing "Cannot open DFU device 0483:df11") please proceed with driver installation via *Zadig* application as shown bellow.

```
dfu-util 0.10

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2020 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Found DFU: [0483:df11] ver=0200, devnum=22, cfg=1, intf=0, path="1-
2", alt=0, name="@Internal Flash   /0x08000000/016*2Ka,240*02Kg",
serial="208B37864B53"
```

In case no DFU device is found or some error is reported and user is using Windows OS, please make sure that EasyTerm bootloader is active and EasyTerm is connected to PC via USB cable. If checked and still no device appears or there is an error stating "Cannot open DFU device 0483:df11" after performing step 3. please download Zadig application from https://zadig.akeo.ie . After running this application click "Options" and then "List all devices". DFU device (STM32 DownLoad Firmware Update) shall be selectable through drop-down list. Please select it and select WinUsb as a driver. Then click on button to proceed (can be labeled as "Install Driver" or "Replace" or otherwise depending on situation). Please see image below.



After that power down EasyTerm and re-enter bootloader once more. Please repeat step 3 and proceed.

6. Remember 'alt' parameter value. In example above the 'alt' parameter value is 0.

7. Start update via this command (replace number after --alt with the same number as found in step 6.

```
dfu-util --alt ALT_NUMBER -D easyTermFW_AAxBB.bin -s 0x8008000
```

Example command with correct progress for our situation (FW version is 01x02 and alt is 0) is shown below:

```
dfu-util --alt 0 -D easyTermFW_01x02.bin -s 0x8008000
```

```
Opening DFU capable USB device...
ID 0483:df11
Run-time device DFU version 011a
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 011a
Device returned transfer size 1024
DfuSe interface name: "Internal Flash   "
Downloading element to address = 0x08008000, size = 444288
Erase     [=========================] 100%       444288 bytes
Erase    done.
Download  [=================        ]  69%       307200 bytes
```

8. Updating shall begin and progress shall be displayed.

We have found out that clicking inside Windows terminal window during download pauses its progress. Hitting any keyboard button resumes it.

9. Wait until download is complete.

10. Restart the EasyTerm (toggle mechanical on/off switch). EasyTerm will proceed with installation (see bootloader LED indication description) and then start updated FW automatically.

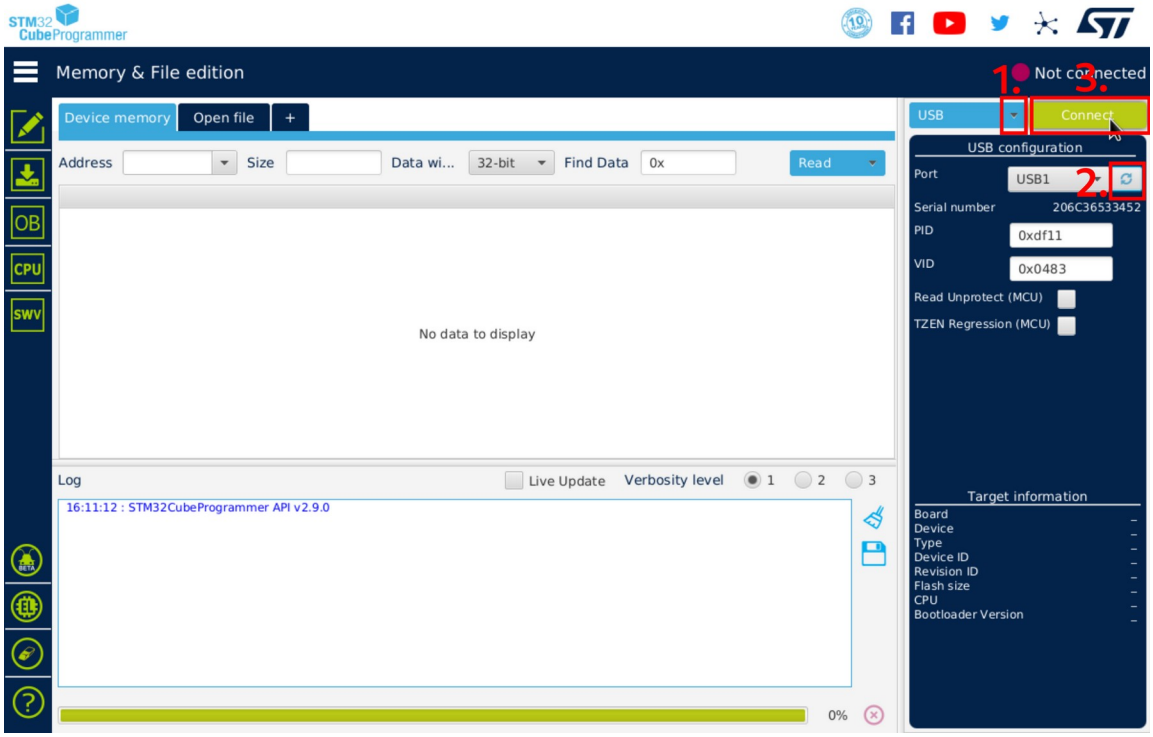
Update via USB using STM32CubeProgrammer

This procedure is suitable for developers that have already installed STM32CubeProgrammer from STMicroelectronics (most often embedded developers using STM32 architecture). It is downloadable from st.com website (requires registration).

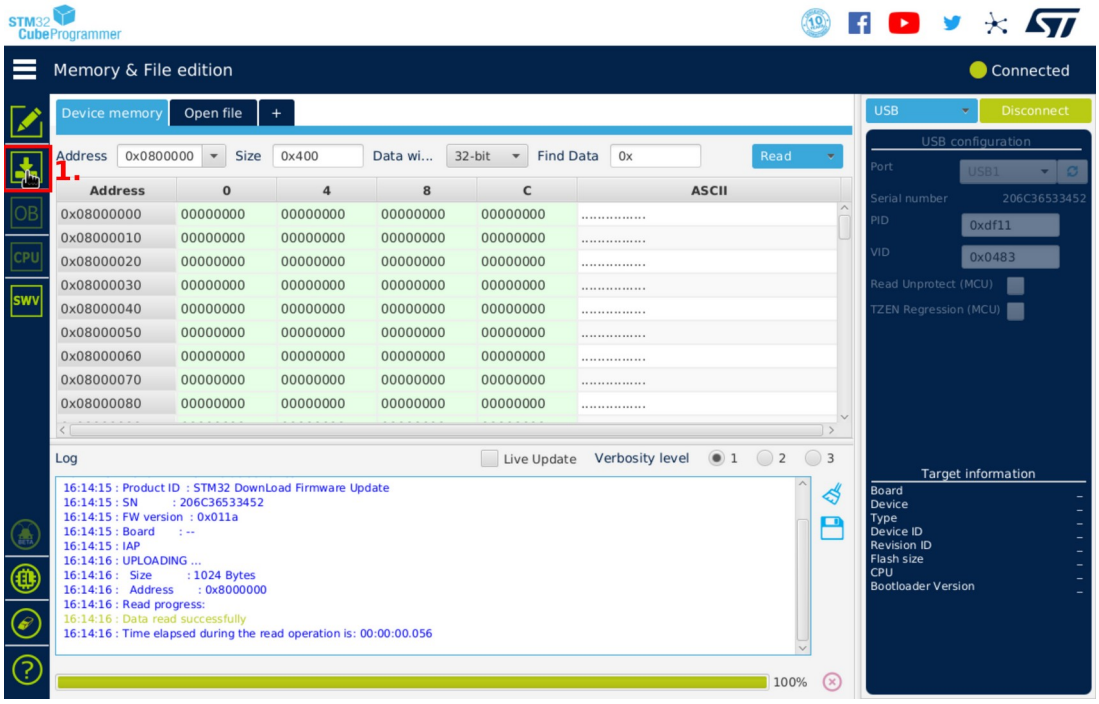
1. Connect EasyTerm via USB to host PC

2. Switch to bootloader mode as described above, check if LED 1 (green) is on (bootloader mode activated).

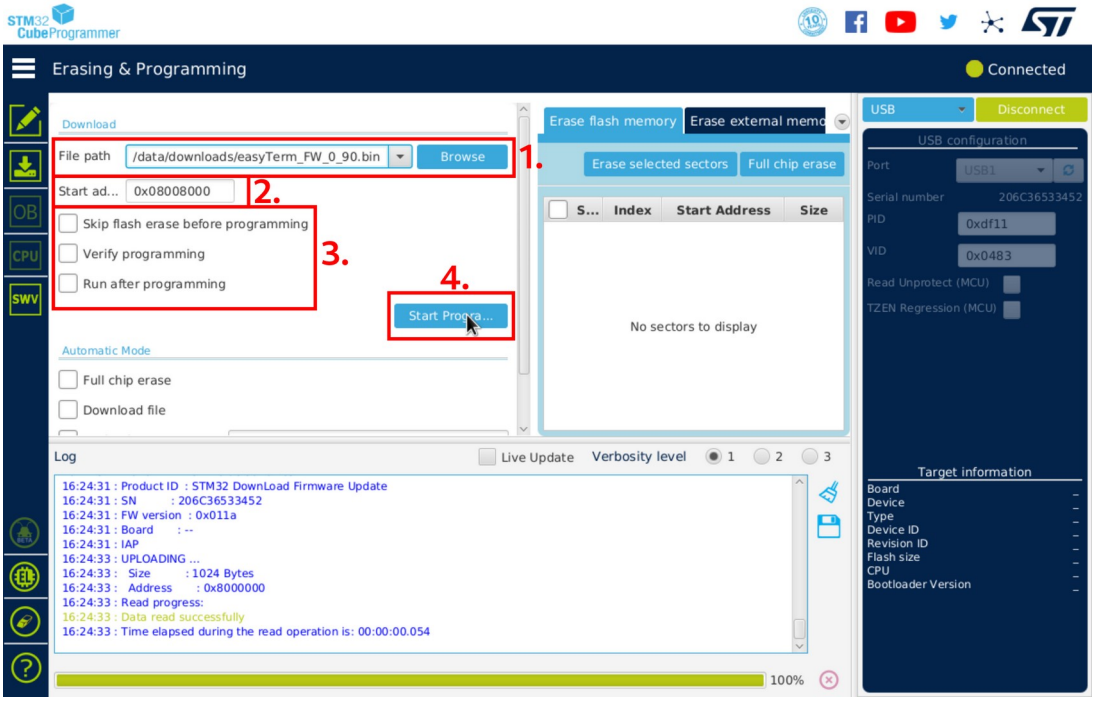
3. Run STM32CubeProgrammer application. Select USB as an interface (1.), click on refresh icon (2.) – USB port shall be filled. Then click on "Connect" button (3.).



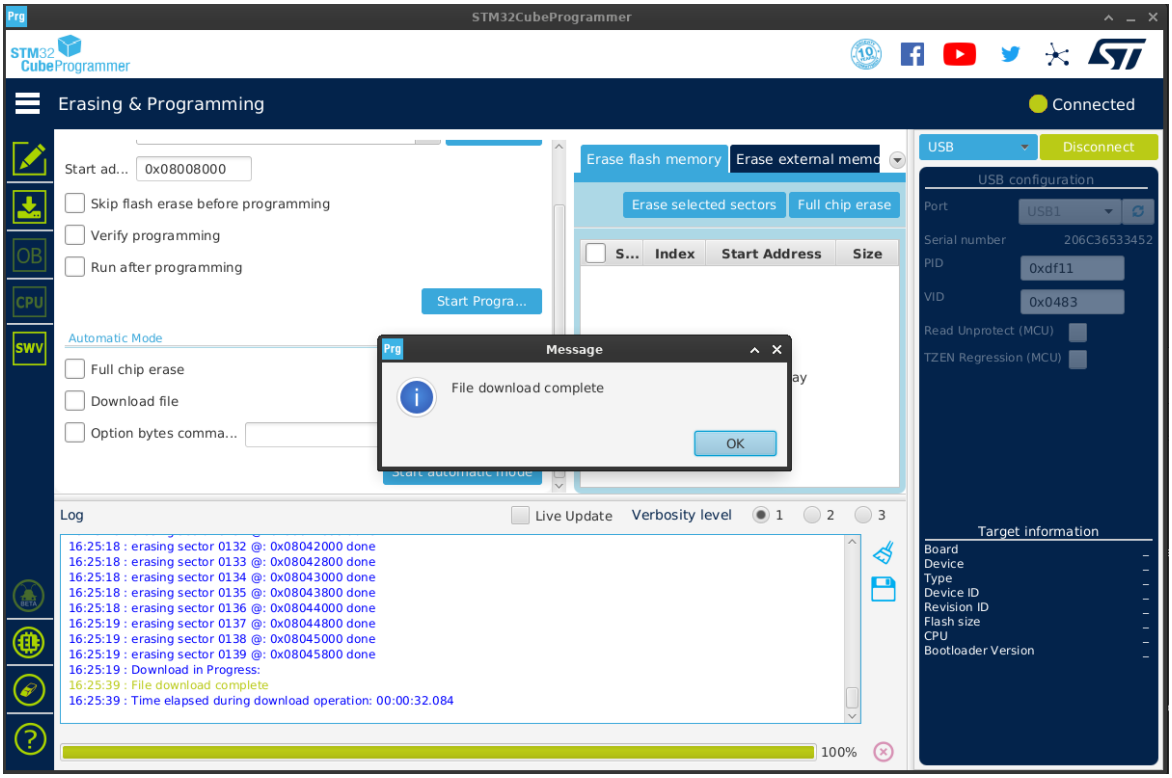
4. Switch to Erasing & Programming



3. Select firmware file (1.). Set start address to 0x08008000 (2.). Untick all boxes (3.). Click on "Start programming" button (4.).



4. Programming shall last for several seconds. After completed, dialog as in picture below shall be displayed.



5. Restart the EasyTerm (toggle mechanical on/off switch). EasyTerm will proceed with installation (see bootloader LED indication description).

Update via UART using yModem protocol

This update solution was implemented as a backup solution when update over USB is failing for some reason. We don't have any reports on such cases so we are pretty confident that update via USB will work as expected.

~~We´ve created simple python script that is the most recommended way for updating via UART. There are two distributions of such script downloadable HERE.~~

Please let us know via support@7-tech.net in case you need these scripts so we can elaborate on this issue and help you with solving problems with USB updates in the first place. We will handle these scripts if this problem cannot be solved promptly. User must have a USB to UART bridge adapter to perform this type of update.

- **yModemUpdate.py** – simple python script recommended when user have installed python3 and serial library on system you or wish to install them.
- **yModemUpdateBundle** – recommended when user does not have python3 or pyserial library installed on system and does not want to install it or have some other issues using yModemUpdate.py script. This package is compiled via PyInstaller that bundles yModemUpdate script with python distribution and all needed libraries to one **.exe** file. Hence the greater size.

1. Connect EasyTerm via UART to USB bridge to host PC and connect UART wires correctly to HOST_UART connector (TX of bridge to RX of EasyTerm and RX of bridge to TX of EasyTerm).

2. Switch to bootloader mode as described above, check if LED 1 (green) is on (bootloader mode activated).

When using yModemUpdate.py script file, run with following command (and replace FILE and PORT correctly).

When using python script in GNU/Linux or Windows:

```
python yModemUpdater.py -i FILE.bin -p PORT
```

Or when using bundle in GNU/Linux

```
sudo chmod +x yModemUpdaterBundle
./yModemUpdaterBundle -i FILE.bin -p PORT
```

Or when using bundle in Windows OS

```
./yModemUpdaterBundle -i FILE.bin -p PORT
```

So for example when using python script in GNU/Linux and if port is /dev/ttyACM0 and FW version is 01x02:

```
python yModemUpdater.py -i easyTerm_01x02.bin -p /dev/ttyACM0
```

For example when using python script in Windows OS if port is COM5 and FW version is 01x02

```
python yModemUpdater.py -i easyTerm_01x02.bin -p COM5
```

Process shall be shown as in example bellow:

```
Binary file size is: 444288 Bytes
Level 1 - waiting for transmission request from device
Correct byte (C character) was received
Level 2 - Transmission of record 0
Level 3 - Transmission of data has begun
Packet 1 out of 434 sent
Packet 2 out of 434 sent
Packet 3 out of 434 sent
Packet 4 out of 434 sent
Packet 5 out of 434 sent
...
```

After transmission done EasyTerm shall restart itself and installation shall start automatically (see bootloader LED indication description). Updated FW will be started after.

Errors

This chapter contains errors that are reported by EasyTerm in certain conditions (typically as a response to invalid command). Please see descriptions and precautions in order to understand what is an issue and what adjustments are needed to be done.

Error message	Description	Precautions
ERR-GENERIC	Unknown error (shall not appear)	
System command related errors		
ERR-SYS-RX_BUFF_OVERFLOW	EasyTerm has receive buffer so commands that are sent via master are not lost when it is processing older commands. However when commands are transmitted too frequently for too long then the receive buffer overflow may occur and new commands can get lost.	Please reduce frequency of command transmission to EasyTerm, insert some pause after several command to let EasyTerm process whole buffer or use request-reply communication scheme (send command and wait for reply) to prevent RX buffer overflow.
ERR-SYS-KEYPAD_ACTIVE	Several operations are forbidden when keypad is displayed (command/message filling) such as changing the screen page and sleep entering.	Finish or cancel inputting via keypad before entering sleep or changing screen page.
Command related errors		
ERR-CMD-INV_PARAM	Unknown parameter for respective root command.	Please check command parameters.
ERR-CMD-INV_PARAM_BODY	Invalid parameter value (string instead number etc.)	Please check command parameters.
ERR-CMD-VALUE_OUT_OF_RANGE	Parameter value is out of range.	Please check command parameters.
ERR-CMD-PARAM_STRING_TOO_LONG	Parameter string is too long.	Please reduce parameter string length.
Log command related errors		
ERR-LOG-NO_LOG_RECORDED	Operation on log is requested but no log recording was executed previously (there is no such log).	Start log recording prior this operation.

ERR-LOG-NO_LOG_BANK_SPECIFIED	Operation like saving or loading is missing log BankID parameter.	Please set BankID parameter.
GUI command related errors		
ERR-GUI-MAX_OBJ_CNT	No more GUI objects can be displayed. FW currently supports maximum of 80 objects displayed in all screen pages (sum).	Please reduce number of GUI object by removing some of them.
ERR-GUI-OBJ_OVERLAP	GUI object to be rendered is overlapping existing one.	Adjust coordinates or dimensions or use another screen page.
ERR-GUI_OBJECT_INVALID_DIMENSIONS	Not supported dimensions for respective object.	Check width and height parameters and adjust them.
ERR-GUI-OBJ_OUTSIDE_SCREEN	Object position and dimensions will cause reaching screen boundaries.	Adjust object position and dimensions so it will fit screen.
ERR-GUI-SCREEN_OUT_OF_RANGE	Screen page index is out of range. Currently 16 screen pages are supported.	Adjust screen page index or do not try to proceed to next screen via button as there is no more screens in that direction.
ERR-GUI-NO_OBJECT_SPECIFIED	Operation requires object ID to be specified.	Please specify GUI object ID for respective operation (EditButton for example).
ERR-GUI-NO_SUCH_OBJECT	There is no GUI object with such object ID.	Adjust object ID parameter.
ERR-GUI-NO_TEXT_SPECIFIED	Object cannot exist without text.	Set text parameter.
ERR-GUI-TEXT_OVERFLOWS_OBJECT	Text is wider than object .	Increase object dimensions or shorten text.
ERR-GUI-NO_PARSE_MASK_SPECIFIED	Object is requesting ParseMask parameter to be set.	Set ParseMask parameter.
ERR-GUI-INVALID_PARSE_MASK	Object ParseMask parameter is invalid.	Fix object ParseMask parameter.
ERR-GUI_SLIDER_VALUE_OUT_OF_RANGE	Intention to set current slider value manually failed as value is exceeding slider min or max parameter values.	Adjust slider value parameter or slider min/max parameters.

ERR-GUI-LANDSCAPE_LOGWIN_UNSUPPORTED	LogWin cannot be displayed in landscape orientation.	Please adjust screen orientation to be portrait or display LogWin on other portrait screen page.
GPIO command related errors		
ERR-GPIO_INVALID_CONFIGURATION	User requested operation on GPIO pin that is unsupported in current configuration. Example is reading pin state when GPIO is used for peripheral (SPI etc.) or setting OutputState when PinMode in Input.	Check pin current configuration and adjust it so that operation will succeed.
PWM command related errors		
ERR-PWM_DUTY_NOT_SET	Some operations like changing PWM rate requires that user has configured PWM duty cycle or T1 and T2 phases.	Set duty cycle or T1 and T2 phases.
ERR-PWM_FREQUENCY_NOT_SET	PWM can generate signal if frequency is set or can be calculated internally by configuring T1 and T2 phase.	Please set frequency or T1 and T2 phases or set any phase with duty cycle.
ERR-PWM_LOCK_ERROR	Locking some PWM parameter can fail in certain conditions: 1) Rate is locked, user wants to change T1 phase by some value but T2 cant be changed by same step to keep rate constant and vice versa (T1 change step is bigger than T2 value). 2)T1 or T2 phase is locked, rate is increased but not locked phase cannot decrease more as it is already at the minimum value.	Check and adjust parameter changes or disable lock.
I2C command related errors		
ERR-I2C-ARBITRATION_LOST	Issue with I2C transactions.	Reconnect slave device (all conductors to prevent back-powering through I2C bus) and re-init I2C via command.
ERR-I2C-SDA_STUCK	Slave device holds I2C SDA low. Typically caused when there are no	EasyTerm will try to release I2C SDA by triggering

	pull-ups used or when slave device is awaiting additional bytes to be read (multi-byte register).	additional clock periods on SCLK sig-nal (via GPIO toggling). Please check your pull-up connection and check if transaction is performed is consensus with slave device datasheet.
ERR-I2C_NOT_ACKNOWLEDGED	Slave device didn't perform acknowledgement.	Check connection between master and slave device, if slave device is powered, if address is correct and presence of pull-ups.
ERR-I2C_GENERIC_ERROR	Unknown error (shall not occur).	
SPI command related errors		
ERR-SPI-CHIPSELECT_INVALID_IO	Chip select IO can be set on GPIO that is not reserved for another purpose.	Change CSIO parameter value.
ERR-SPI-CHIPSELECT_NOT_CONFIGURED	Chip-select IO must be configured for automatic or manual chip-select toggling.	Configure CSIO parameter.
ERR-SPI-UNSUPPORTED_IN_HALF_DUPLEX_MODE	Operation like SPI txrx=... is unsupported in half-duplex mode.	Do not use operations that are unsupported in half-duplex mode.
UART command related errors		
ERR-UART-RECEIVE_TIMEOUT	UART txrx and rx operations are specified with word count EasyTerm shall receive. Reception is performed with timeout configured via RXTimeout parameter. This error is reported when timeout is reached (requested number of words were not received in time).	Adjust RXTimeout parameter or check why device is not transmitting all words to EasyTerm.
MODBUS command related errors		
ERR-MODBUS-UNEXPECTED_SLAVE_REPLY	Slave device responded with data that were not expected as reply for master's request.	Check master's MODBUS integration or physical layer integrity.
ERR-MODBUS-REPLY_TIMEOUT	Reply from slave was not received	Check if slave device is active,

	in time.	its address is correctly configured and if it is responding.
ERR-MODBUS-OUTPUT_ENABLE_IO_NOT_SET	Output enable IO was not set.	Configure OEIO parameter.
ERR-MODBUS-DATA_NOT_MULTIPLE_OF_16BITS	Data are not multiple of 2 bytes.	Check and adjust data count of data bytes to be a multiple of 2 bytes (16 bits).
ERR-MODBUS-ILLEGAL_FUNCTION	Function code is not supported by slave device.	Check function code support on slave device.
ERR-MODBUS-ILLEGAL_DATA_ADDRESS	Invalid register/coil address.	Check register/address map of slave device if the access is not out-of-range.
ERR-MODBUS-ILLEGAL_DATA_VALUE	Not supported value to be written.	Check possible values for that register of slave device.
ERR-MODBUS-SLAVE_DEVICE_FAILURE	Slave failure.	Check slave device.
ERR-MODBUS-SLAVE_BUSY	Slave is busy performing some action.	Wait some time and try to repeat the request.
ERR-MODBUS-INVALID_REPLY_CRC	Reply CRC does not match.	Check integrity of the bus and communication parameters.
ADC command related errors		
ERR-ADC_SAMPLE_COUNT_NOT_SET	Sample count must be set when converting to buffer.	Set SampleCount parameter.
ERR-ADC_SAMPLE_RATE_NOT_SET	Sample rate must be set when converting to buffer.	Set SampleRate parameter.
DAC command related errors		
ERR-DAC_VOLTAGE_HIGHER_THAN_MAXIMUM	Due to variation of reference voltage the DAC may not go all the way up to requested DAC voltage. E.g. when reference voltage is 2.997, DAC v=3000 will report such error.	Decrease the DAC voltage value.
ERR-DAC_SAMPLERATE_NOT_SET	SampleRate must be configured when conversions from buffer is requested.	Set SampleRate parameter.
Common interface commands related errors		

ERR-IF-INVALID_IO_CONFIGURATION	Interfaces like I2C, SPI, UART, PWM and TRG requires that respective GPIOs are configured to peripheral mode.	Please configure respective GPIOs to peripheral mode or use alias (e.g. I2C i=1).
ERR-IF-UNSUPPORTED_IN_SLEEP_LEVEL_2	Analog interfaces like ADC and DAC are unsupported in sleep level 2.	Change sleep level when you want to use analog interfaces.
Hex-parsing related errors		
ERR-HEX-MAX_BYTE_COUNT_REACHED	Data buffer was filled.	Split transactions to several sub-transaction so that buffer will be able to accommodate all data.
ERR-HEX-ODD_NIBBLE_COUNT	Hexadecimal string must have even number of characters.	Adjust hexadecimal string to have even number of characters.
ERR-HEX-NOT_PARSEABLE_CHAR	Hexadecimal string must have characters in range between 0-F.	Adjust hexadecimal string to have only valid characters.